Beginner Microbiome Analysis 2025

Contents

Part I

Introduction

Chapter 1

Welcome

Welcome to CBW's Beginner Microbiome Analysis Workshop!

1.1 AWS/UNIX primer

You can find the schedule and materials for the AWS/UNIX primer here.

1.2 Course Schedule

Time (PST)	Day 1	Time (PST)	Day 2
8:30	Arrivals & Check-in	8:30	Arrivals
9:00	Welcome (Coordinator)	9:00	Module 3 Lecture
9:30	Module 1 Lecture		
10:30	Break (30min)	10:00	Break (30min)
11:00	Module 1 Lab	10:30	Module 3 Lab
13:00	Lunch (1h)	12:30	Class photo + Lunch (1h)
14:00	Module 2 Lecture	13:30	Module 4 Lecture
15:00	Break (30min)	14:30	Break (30min)
15:30	Module 2 Lab	15:00	Module 4 Lab
		17:00	Survey and Closing Remarks
17:30	Finished	17:30	Finished

1.3 Meet Your Faculty

Coming soon!

Part II

Modules

Chapter 2

Module 1

2.1 Lecture

2.2 Lab: Introduction to sequencing data analysis

This tutorial is part of the 2025 CBW Beginner Microbiome Analysis (held in Vancouver, BC, May 26-27).

Author: Robyn Wright

2.2.1 Introduction

In this module, we're going to be learning the basics of moving around on the command line, creating directories, zipping and unzipping folders, making tar archives and installing programs on a server. We'll also be downloading files to the server, investigating the format of fasta and fastq files, and looking at the quality of sequenced samples.

NOTE!

Throughout this module, there are some questions aimed to help your understanding of some of the key concepts. You'll find the answers at the bottom of this page, but no one will be marking them. **You might want to keep the answers open in another tab!**

2.2.2 1.1 Log into your AWS instance

Hopefully you were able to attend the session on Friday where we went over this, but I've added the instructions below as a reminder. You can also follow the instructions here.

2.2.2.1 Mac/Linux

First off, you'll need to save the .pem key file that you've been given.

Next, you'll need to change directory to where your key file is stored. Locate the file in your Finder, right click the file, hold the option key and click on "Copy CBW.pem as pathname".

Then, go to your Terminal window (search for "Terminal" in Launchpad).

You can just remove the CBW.pem from the end to get the file path. E.g., my full path name that I copied is `/Users/robynwright/Downloads/CBW.pem`, so:

```
cd /Users/robynwright/Downloads/
```

Then run:

```
chmod 600 CBW.pem
```

Make sure that you replace `CBW.pem` with whatever the file name is that you saved it as!

Now we can connect to the server:

```
ssh -i CBW.pem ubuntu@##.uhn-hpc.ca
```

where ## is your assigned student number. Again, make sure you replace CBW.pem with whatever the file name is that you chose!

It will then probably ask you if you're sure you want to connect. Type "yes" and press enter. Now you should see something like `(base) ubuntu@ip-10-0-1-199:~$` to the left of your cursor. Great, you're connected!

2.2.2.2 Windows

First off, you'll need to save the .ppk key file that you've been given.

For Windows, we'll be logging in with Putty. So next, go to the "Session" category and fill in the "Host Name (or IP address)" field with:

`ubuntu@##.uhn-hpc.ca`

where ## is your assigned student number.

Now, in the "Connection" category, click the + next to "SSH". Click the + next to "Auth" and then click "Credentials". In the "Private key file for authentication" field, click browse and fine

In the left hand categories, in the Connection category next to SSH click on the +. Click on + next to Auth and then click Credentials. In the private-key file for authentication field, hit browse and find the CBW.ppk file that you downloaded.

Now, click on the "Session" category again. In the "Saved sessions" field, type "Amazon node" and click Save.

Now that Putty is set up, all you have to do is start putty and double-click on "Amazon node" to login. Now you should see something like `(base) ubuntu@ip-10-0-1-199:~$` to the left of your cursor. Great, you're connected!

2.2.3 1.2 Creating directories and moving around on the command line

Now that we're logged in, we're going to get familiar with what's on the server. First of all, type `ls` and press enter. You should see five things printed out:
`CourseData R anaconda3 bin workspace`

These are all directories/folders on the server. Think of these like you would the directories on your own computer, e.g. "My Documents" and "Downloads". The `ls` command is just telling the server to list out the files/directories in the directory that we're currently in.

Next, type in `pwd` and press enter. The `pwd` command tells you which directory you're currently in, so we can see that we're in `/home/ubuntu`.

Now we're going to change directory. We can change to any directory that exists here, but since we're mainly going to be working from `workspace`, we'll change to there. Type in `cd workspace` and press enter (you will always need to press enter to run a command). If you type in `pwd` again, you should see that you've changed directory. If you type in `ls`, you should see that the directory is empty. We're going to create directories for each module that we do, so we'll make one called `bmb_module1`:
`mkdir bmb_module1`
Note that it doesn't really matter what we call this directory, but if you name it something different then you'll need to remember that when we run things later on so it's easier to keep it consistent.

Now let's change to the directory we just made:
`cd bmb_module1`
If you use the `ls` command, you should see that it's empty. Now what happens if we want to go back to the `workspace` directory? We can use `cd ..`, which will take us up one level. Try that and then use the `ls` command again. You should see the folder that you made. If you use `cd` on it's own, then it'll take you back to where you started. Try it and then use the `pwd` command. Now to get back to the directory we created for this module, use: `cd ~/workspace/bmb_module1`. Note that any time you log out from the server (or lose connection/time out and get logged out), you will need to change back to the directory that you're working from. Use the `pwd` command once more to check that you're in the right place (`/home/ubuntu/workspace/bmb_module1`).

2.2.4 1.3 Use wget to download files

Now we're going to download some files to use. There are a lot of different data repositories available online, but we're going to use some data from the Human Microbiome Project. This repository is quite straightforward because all we need is a link to the data and we can download it, but other repositories require their own programs and file formats for downloading which can make them quite complicated (and frustrating, if you don't know what you're doing!) to use. You can see the webpage that we're taking the files from here.

Now we'll download the files. There are a few different commands that we can use for downloading files, so you might have seen others, or may see others in the future, but for now we'll be using a command called `wget`. If you just type in `wget` on its own then you should see some information about it. You'll see that it gives an error message, because we haven't also given it a URL, but it also tells us that we can run `wget --help` to see more options. Try running that. If you scroll back up to where you ran it, you'll see a huge list of options that you could give `wget` depending on what you want to do. We won't be using most of these options for now, but you can usually add `--help` to whatever command you are trying to run to get some more information about what information it is expecting from you (these are called "arguments").

We're going to download three files like so:

```
wget https://g-227ca.190ebd.75bc.data.globus.org/ibdmdb/raw/HMP2/16S/2018-01-08/206534
wget https://g-227ca.190ebd.75bc.data.globus.org/ibdmdb/raw/HMP2/16S/2018-01-08/206536
wget https://g-227ca.190ebd.75bc.data.globus.org/ibdmdb/raw/HMP2/16S/2018-01-08/206538
```

You should see some progress bars come up, but these files aren't very big so they shouldn't take very long to download. Now use `ls` again to see the files. You should see:

```
206534.fastq.gz  206536.fastq.gz  206538.fastq.gz
```

2.2.5 1.4 Move these files to the directories

When we downloaded these, we didn't make a directory to put them in, so let's do that now so that we can tidy them up a bit:

```
mkdir test_data
```

And then we can move them to this directory we've just made using the `mv` command. The `mv` command is expecting the name of the file that we want to move, and then the directory/path that we want to move this file to as arguments:

```
mv 206534.fastq.gz test_data/
```

If you use the `ls` command again now, you will see that there's only two files (along with the `test_data` directory). You can also use the `ls` command on the `test_data` directory: `ls test_data`, and you should see the file that we moved into there.

Often, we might have a lot of files to move and we don't want to have to move them one by one. If we want to move multiple files of the same type, we can do that like this:

```
mv *.fastq.gz test_data/
```

The asterisk (`*`) acts as a wildcard, and anything that ends in `.fastq.gz` will be moved using this command. If you take a look in `test_data` again using the `ls` command, you should see all three files in there now, and not in your current directory. Another useful thing that you can do with the `ls` command is add another argument to get some information about the files: `ls -lh test_data/`. When you run this, you should see who has permission to read/write to the files, the author and the owner of the files, the size of the files, and when they were created. The `-l` is what is telling this to give you the information, and adding the `h` converts this into "human" readable format. Try using it without the `h` - you'll see that the file sizes are in bytes instead of megabytes (MB/M).

2.2.6 1.5 Zip and unzip these files

You might have noticed the `.gz` on the end of the files indicating that they're zipped (or compressed). Sometimes when we run bioinformatics programs they can uncompress the files within the program, but other times we need to uncompress (unzip) them first. These are quite small files so you might think it's unnecessary to zip/compress them, but often we have thousands of sequencing data files and they can each be hundreds of gigabytes (GB) or even terabytes

(TB) in size, so it becomes quite important to keep them compressed until we need them.

We'll use the `gunzip` command to unzip them. Try typing in `gunzip test_data/20` and then pressing the tab button. If you press it a couple of times, you should see a list of your options come up. The tab button can be really useful for completing file paths for you and saving you a lot of typing! Continue typing and choose one of the files to unzip (e.g. type in `8` and then press tab again and then enter). Now if you run `ls -lh test_data` again, you should see that the file you unzipped no longer has the `.gz` on the end, and it's much larger in size now.

We'll zip the file back up for now: `gzip test_data/20` - if you press tab to complete again (and enter), you should find that this will auto-fill with the file that you unzipped, because it's the only file type that the command is able to work with.

What happens if you try to run this on a file that is already zipped? `gzip test_data/206538.fastq.gz` It should tell you that it's unable to run because `.gz` is already on that file.

Let's unzip all of the files now: `gunzip test_data/*` See that we can use the asterisk (`*`) again as a wildcard and it will unzip every file in the `test_data` directory. Take a look at the directory with `ls` or `ls -lh` if you like. Remember to add the directory name!

2.2.7 1.6 Create new tar archive of files

There are several different ways of compressing files - there is `gzip`/`gunzip` that we just used, but we can also package up multiple files inside a directory together. We'll be using the `tar` command for this, and as you can see if you run `tar --help`, you'll see that there are lots of options available for this. Let's try it out with the `test_data` directory: `tar -czvf test_data.tar.gz test_data/` Here we gave the command several arguments: `-czvf` (see below), `test_data.tar.gz` (the file name that we want our tar archive to have) and `test_data/` (the name of the directory that we want to compress. `-czvf` is a short way of giving several arguments to the command: `c` for "create" (creates a new archive), `z` for "gZip" (this tells tar to write/read through gzip), `v` stands for "verbose" (meaning that it will print out information about what it is doing), and `f` for "file" or "archive".

If you now run `ls -lh`, you should see that the tar archive (`test_data.tar.gz`) is a smaller size than the 3 files would be together (check by running `ls -lh test_data`). You can also take a look at what's in the tar archive with the `less` command: `less test_data.tar.gz`. Press `q` (for "quit") when you're ready to exit.

Usually we'll make a tar archive because we want to keep our files but save some space, so let's delete the original folder: `rm -r test_data/`. Hopefully by now you're getting the hang of how these commands work. The `rm` command is for removing files - you should always be really careful when using it because it won't ask you if you're sure like your regular computer would, and most servers don't have a "recycle bin" for the files to be sent to, so if you remove them, they're gone for good. The `-r` argument is for removing a directory rather than just a file.

2.2.8 1.7 Unzip tar archive

If we need to use the data that we zipped into the tar archive again, we'll need to unzip - or extract - it.

To unzip the tar archive, we can do that like so: `tar -xvf test_data.tar.gz`. Note that we just replaced the `cz` with `x` for "extract". You should be able to see the files back in `test_data` with the `ls` command.

2.2.9 1.8 Look at fasta and fastq files with less

Now we're going to take a look at these files. Let's look at `206538` first: `less test_data/206538.fastq`. You can scroll through the file, and remember to press `q` when you want to stop looking at the file. If you want to look at it again, press the up arrow key to run the `less` command again. You can always press the up arrow to go back through the commands that you've run previously. If you've typed something wrong and want to start again, press `ctrl+c` to get back to a blank command prompt.

You should have noticed that this file had the `.fastq` extension. Let's copy across the same files in `fasta` format: `cp -r ~/CourseData/MIC_data/bmb_module1/test_data_fasta/ .` Here, we're using the `cp` command to copy across some files that I already set up earlier. The `~` is usually a shortcut for your home directory to save us typing it each time, and the rest is directories that we've already set up. The `-r` argument is the same as above for `rm` - it means that we're taking a directory and not a file, and then the `.` shows that we want to copy the data into the directory that we are currently in. We could replace it with another file path if we wanted.

Take a look at the same file in `fasta` format: `less test_data_fasta/206538.fasta`

You can also download these files by going to http://#.uhn-hpc.ca/ (where ## is your personal number). Navigate to the correct directory and then right click the files > save link as > choose location > choose name. You'll be able to open them with a text editor like TextEdit (Mac) or Notepad (Windows).

Most servers aren't set up with a handy webpage where you can browse your files, like this one is. In that case, you can use the `scp` (**s**erver **c**o**p**y) command to copy them across to your own computer.

You would do that by going to a new Terminal window. Navigate to the location of your .pem file. To download the whole folder, you can run:

```
scp -r -i CBW.pem ubuntu@##.uhn-hpc.ca:/home/ubuntu/workspace/bmb_module1/test_data_fas
scp -r -i CBW.pem ubuntu@##.uhn-hpc.ca:/home/ubuntu/workspace/bmb_module1/test_data/ .
```

Make sure that you replace the ## with your instance number!

Now you can open the files with a text editor like TextEdit (Mac) or Notepad (Windows). You should see that the fastq file has 4 lines for each sequence, while the fasta only has two. The fasta has a first line that starts with ">" that contains the sequence name and description, and the second line contains the actual DNA sequence (DNA in this case, but this file format could also contain RNA or protein sequences). fastq files start the same, with the first line containing the sequence name and description (but starting with an "@" symbol instead), and the second containing the sequence. The third line then contains a "+" character, and the fourth contains quality information about each base of the sequence (and should contain the same number of characters as the sequence). You can read more about the quality information that the symbols encode here.

To count the number of lines in a file, we can use the `less` command again, with some additional information: `less test_data/206538.fastq | wc -l`

You should see that this file contains 63,048 lines.

QUESTION!

Question 1: How many sequences does this mean that there are?

Remember that you can just use `less test_data/206538.fastq` to look at the file.

Now do the same for the fasta file: `less test_data/206538.fasta | wc -l`.

You should see that the fasta file contains half the number of lines as the fastq file. Sometimes you'll find that the sequences with a fasta file are split across multiple lines, in which case, simply counting the number of lines wouldn't work, so there are also other ways to count the number of sequences in a file, and these can be adapted for other purposes, too. E.g.: `grep -c ">" test_data_fasta/206538.fasta` - the `grep` command pulls out every occurrence of a phrase (or "string", as it's usually called in programming) and the

`-c` argument tells it to count these. What happens if you don't use the `-c` argument? Why do you think this happened?

In most programming languages, you have "positional" arguments and "named" arguments. Positional arguments need to be included in the proper position, or order. The order of positional arguments is defined within the program. Named or keyword arguments are given or passed to the program only after the name is given. In the case above, the `-c ">"` is a named argument and the file name `test_data_fasta/206538.fasta` is a positional argument.

QUESTION!

Question 2: What happens if you try to do the same thing with "@" for the fastq file? Why is this?

Remember to look at the fastq file with the `less` command for clues, or try running it without the `-c` argument.

2.2.10 1.9 Installing programs to the server

Now we're going to learn how to install programs to the server. A lot of the commands we have just used (like `grep` and `less`) are standard ones that will be installed on most servers, but frequently we will need to install other programs (usually called "packages"), too. The packages that we use are often not as stable as those that we download and use on our laptops (like Microsoft Word or Adobe Acrobat Reader) and so they sometimes depend on a particular version of another package. It frequently takes more time to install packages than it does to run them, and any bioinformatician will tell you how frustrating it can be. Anaconda can help to manage this, although it doesn't overcome these problems entirely!

Note that Anaconda is already installed here so you don't need to run this, but **if you want to install it for yourself, you can do so like this**:

```
curl -O https://repo.anaconda.com/archive/Anaconda3-2024.10-1-Linux-x86_64.sh
bash Anaconda3-2024.10-1-Linux-x86_64.sh
#hold down enter key until you get to the end of the agreement, or press q
#type yes
#confirm location by pressing enter
#yes
#now close and reopen the window - you'll need to log back in the same way as you did before!
```

2.2.11 1.10 Conda environments

Anaconda, or conda, allows us to have separate "environments" for installing packages into. This means that if one package requires version 3.1 of another package, but another requires version 2.9, they won't interfere with each other. Often when we're installing new packages or starting a new project, we'll make a new environment. This also helps us to keep track of which versions of a package we've used for a specific project. The environment is essentially a directory that contains a collection of packages that you've installed, so that other packages know where to access the package that they need to use. We're going to make a new environment to install some packages into:

```
conda create -n bmb_module1
```

You'll see that here we're using the `conda` command first, and then giving it the `create` and `-n bmb_module1` arguments. We could call this environment anything we like, but it's best to make this descriptive of what it is so that when we collaborate with others or share our code, it'll be obvious what this environment is for.

You'll need to press `y` at some point, to confirm that you want to install new packages.

Now we can "activate" this environment like this: `conda activate bmb_module1`. Any time you are logged out and you log back in, you'll need to reactivate the environment if you want to be working from it. If you want to see the other environments that are installed and available, you can run `conda info --envs`. You'll see that we've already installed a lot of environments that we'll be using over the next few days. While it would be great to be able to get you to install all of this for yourself, it can take quite a long time to install some packages so we've set most of them up already. If you want to see how we did that, you can see most of that here.

2.2.12 1.11 Install fastqc and multiqc

Now we'll install the packages that we want to use. Usually if there's a package that you're interested in, for example we'll be using one called "fastqc", you can just google "conda install fastqc" and you should see an anaconda.org page as one of the top hits, telling you how to install it. Sometimes you'll also see bioconda documentation, or a "package recipe" and this might give more details if you're struggling to install it. We'll install fastqc like this:

```
conda install bioconda::fastqc
```

You'll need to confirm that you want to install things with `y` at some point. If you forgot to activate the environment (see above), then you'll get an error that you don't have permissions to do this!

You can test to see whether it got installed by typing `which fastqc` - this should show you the location that it's installed in, and it should be in `/home/ubuntu/CourseData/MIC_data/.conda/envs/bmb_module1/bin/fastqc`.

Now we'll install the second package that we need:

```
conda install bioconda::multiqc
```

Confirm this again with `y`

As you might have guessed from the "qc" in both of these names, we'll be using them for Quality Control of the sequence data.

2.2.13 1.12 Perform quality control on fastq files

First we'll be running fastqc, and to do that, we'll first make a directory for the output to go: `mkdir fastqc_out`

Now we'll run fastqc:

```
fastqc -t 4 test_data/*.fastq -o fastqc_out
```

Here the arguments that we're giving `fastqc` are: - `-t 4`: the number of threads to use. Sometimes "threads" will be shown as `--threads`, `--cpus`, `--processors`, `--nproc`, or similar. Basically, developers of packages can call things whatever they like, but you can use the help documentation to see what options are available. We're using 4 here because that's the maximum that we have available. See below (`htop`) for how we find out about how many we have available. - `test_data/*.fastq`: the fastq files that we want to check the quality of. - `-o fastqc_out`: the folder to save the output to.

2.2.14 1.13 htop - looking at the number of processes we have available or running

Try running `htop`. This is an interactive viewer that shows you the processes that are running on your computer/server. There are a lot of different bits of information that this is showing us - you can see all of that here, but the key things for us are: - The CPUs (labelled 0, 1, 2, 3 at the top left) - this shows the percentage of the CPU being used for each core, and the number of cores shown here is the number of different processes/threads that we have available to us. In our case, this is 4. - Memory - this is the amount of memory, or

RAM, that we have available to us. You'll see that it is ~16GB - this is similar to many laptops now, but many servers that you'll use or have access to for bioinformatics analysis will have much more than a standard computer. For example, one of the Langille lab servers has ~1.5 TB RAM. The larger your dataset, or the deeper your sequencing depth, the more RAM you are likely to need. - The processes (at the bottom) - you can see everything that is running under a PID (Process ID). This is useful when you're using a shared server to see who is running what, particularly for when you're wanting to run something that will use a lot of memory or will take a long time and you want to check that it won't bother anyone else.

When you're done looking at this, press `F10` (on a Mac this is `fn`+`F10`) to exit from this screen.

2.2.15 1.14 Back to the quality control

Now take a look at one of the .html files in `bmb_module1/fastqc_out/` (note that you'll need to download it as you did above, and if you still have the webpage open, you will need to refresh it).

Next we'll run multiqc. The name suggests it might be performing QC on multiple files, but it's actually for combining the output together of multiple files, so we can run it like this:

```
multiqc fastqc_out --filename multiqc.html
```

So we've given as arguments: - `fastqc_out`: the folder that contains the fastqc output. - `--filename multiqc.html`: the file name to save the output as.

Now look at `multiqc.html`.

There are some questions here to help you look at the files and interpret these:

QUESTION!

- **Question 3**: What is the GC% of the samples?
- **Question 4**: What % of the samples are duplicate reads? Is this what you expected?
- **Question 5**: Now look at the Sequence Counts section. Which sample has the most reads?
- **Question 6**: How many unique and duplicate reads are in sample 206536?
- **Question 7**: Look at the Sequence Quality Histograms. Do these seem good to you? Why or why not? Does this seem normal?

- **Question 8**: Look at the top overrepresented sequence. If you want to see what it is, paste it into the "Enter accession number(s), gi(s), or FASTA sequence(s)" box here and click on the blue "BLAST" button at the bottom of the page.

2.2.16 1.15 Quality control on the data that we'll be using

Next, if you have time, you can try running the quality control on the data that we're going to be using for the other modules.

First of all, you can create a symlink to the data like this:

```
ln -s ~/CourseData/MIC_data/bmb_raw_data/16S_raw_data/ .
```

Creating a symlink is a bit like creating a desktop shortcut for something - we don't need to create a copy of it, but we just want easy access to it. You'll see that the format of the command is really similar to some of the copy (`cp`) or move (`mv`) commands that we used previously. If you use the `ls` command now, you should see that the directory that we've created the symlink to is a different colour to the other directories and files in the folder. These colours vary depending on the settings you choose in your console window, but they are usually different to indicate that this is a link.

Now we'll create a directory for the fastqc output:

```
mkdir fastqc_out_16S
```

And run fastqc again:

```
fastqc -t 4 16S_raw_data/*.fastq.gz -o fastqc_out_16S
```

Note that fastqc can run on zipped or unzipped files.

And finally, run multiqc again:

```
multiqc fastqc_out_16S --filename multiqc_16S.html
```

Note that files with _R1 are forward reads and those with _R2 are reverse.

QUESTION!

- **Question 9**: What do you notice between the R1 and R2 files? Why do you think that might be?
- **Question 10**: Do all of the samples have the same number of reads in the forward and reverse files?
- **Question 11**: When you look at the Sequence Quality Histograms you'll see two clusters. If you hover over the lines you can see the sample names. What do you notice?
- **Question 12**: Have a look again at the top overrepresented sequence. What is it?

2.2.17 1.16 Install QIIME2

For the final part of this module, we're going to install the program (QIIME2) that we'll be using for the rest of today and some of tomorrow.

First, we'll deactivate the environment that we've been using:

```
conda deactivate
```

If you've found this easy so far and want more of a challenge, try doing this for yourself by following the conda instructions on the webpage here. Make sure that you're using the latest version, 2025.4. **Hint**: You want to install the base distribution and the servers we're using are Linux.

Otherwise, follow the instructions here.

Download the files that we'll need:

```
wget https://raw.githubusercontent.com/qiime2/distributions/refs/heads/dev/2025.4/ampli
```

Create an environment using this file:

```
conda env create -n qiime2-amplicon-2025.4 --file qiime2-amplicon-ubuntu-latest-conda.y
```

Note that where previously, we created an environment and then installed packages, this time we're creating an environment using a file that tells conda exactly how to create that environment and which packages should be installed within it.

Once we've made it, we can activate this new environment:

```
conda activate qiime2-amplicon-2024.10
```

And now we can remove the file that we used to install it:

```
rm qiime2-amplicon-ubuntu-latest-conda.yml
```

2.2.18 Answers

Question 1: How many sequences does this mean that there are? Remember that you can just use `less test_data/206538.fastq` to look at the file.
Each sequences is spread across four different lines, so there are 63,048/4 = 15,762 sequences.

Question 2: What happens if you try to do the same thing with "@" for the fastq file? Why is this? Remember to look at the fastq file with the `less` command for clues, or try running it without the `-c` argument.
The number of "@" in the fastq file is much more than the number of lines. This is because the "@" symbol is also used in the quality information. We can get round this by using part of the sample name, *e.g.,* `grep -c "@206534" test_data/206534.fastq`.

Question 3: What is the GC% of the samples?
51%

Question 4: What % of the samples are duplicate reads? Is this what you expected?
In the "General Statistics" section, we can see that ~97% of the reads are duplicated. Looking in the "Sequence Counts" section and hovering over each sample will show us how many of the reads are unique. This makes sense, because the reads are from PCR-amplified samples so we are expecting most to occur more than once.

Question 5: Now look at the Sequence Counts section. Which sample has the most reads?
206354.

Question 6: How many unique and duplicate reads are in sample 206536?
752 and 29,883.

Question 7: Look at the Sequence Quality Histograms. Do these seem good to you? Why or why not? Does this seem normal?
The sequence quality here is all really high! These are all good sequences, but this isn't normal. This is because the samples that are available for download from the HMP website have already been quality filtered.

Question 8: Look at the top overrepresented sequence. If you want to see what it is, paste it into the "Enter accession number(s), gi(s), or FASTA sequence(s)" box here and click on the blue "BLAST" button at the bottom of the page.

All of the top hits are Bacteroides (finegoldii, caccae, stercoris, etc). These samples are from HMP2 IBD gut samples, so this seems normal!

Question 9: What do you notice between the R1 and R2 files? Why do you think that might be?
There are higher numbers of duplicated reads in the reverse (R2) read files. This is usually due to PCR amplification biases and sequencing artifacts.

Question 10: Do all of the samples have the same number of reads in the forward and reverse files?
Yes! This is good, because they should all be paired reads. If they were different then this would indicate that something in our sequencing had gone wrong.

Question 11: When you look at the Sequence Quality Histograms you'll see two clusters. If you hover over the lines you can see the sample names. What do you notice?
The group of lines with lower mean quality scores corresponds to the reverse (R2) reads. The quality for reverse reads from a MiSeq sequencing run is always a little lower than for the forward reads.

Question 12: Have a look again at the top overrepresented sequence. What is it?
Bradyrhizobium sp. is the top hit (that isn't all N's), and the second hit is also *Bradyrhizobium* sp.

Chapter 3

Module 2

3.1 Lecture

3.2 Lab: Marker gene profiling

This tutorial is part of the 2025 CBW Beginner Microbiome Analysis (held in Vancouver, BC, May 26-27). It is based on the Amplicon SOP v2 available on the Microbiome Helper repository and previous workshops designed by Monica Alvaro Fuss, Diana Haider and Robert Beiko.

Author: Robyn Wright

3.2.1 Introduction

Modules 2 and 3 provide a walkthrough of an end-to-end pipeline using the command line interface for the analysis of high-throughput marker gene data. Commonly used marker genes for microbiome analysis include the 16S ribosomal RNA (rRNA) for prokaryotes, 18S rRNA for eukaryotes, and the internal transcribed spacer (ITS) for fungi.

BEFORE YOU START!

In this tutorial, you can choose between using:
16S dataset from wild blueberry, *Vaccinium angustifolium* (soil microbiome)
- Variation in Bacterial and Eukaryotic Communities Associated with Natural

and Managed Wild Blueberry Habitats
- Metagenomic Functional Shifts to Plant Induced Environmental Changes
18S dataset from plastics incubated in a coastal marine environment (plastisphere)
- Microbial pioneers of plastic colonisation in coastal seawaters
ITS2 dataset from stool samples from pregnant women (gut microbiome)
- Landscape of the gut mycobiome dynamics during pregnancy and its relationship with host metabolism and pregnancy health

You can jump to one of the below sections for the commands needed for processing each of these amplicons. We recommend choosing one of them - 16S is the most widely used and also allows you to easily generate a phylogenetic tree. If you have never done any amplicon analysis before then we recommend choosing the 16S dataset.

In Module 2 we will cover the basics of marker gene analysis from raw reads to filtered feature table. The pipeline described is embedded in the latest version of QIIME2 (Quantitative Insights into Microbial Ecology version 2025.4), which is a popular microbiome bioinformatics platform for microbial ecology built on user-made software packages called plugins that work on QIIME2 artifact or QZA files. Documentation for these plugins can be found in the QIIME 2 user documentation, along with tutorials and other useful information. QIIME2 also provides interpretable visualizations that can be accessed by opening any generated QZV files within QIIME2 View.

3.2.2 1. 16S

Create a directory for Module 2 inside `workspace` and create a symlink to the raw FASTQ files and the metadata file.

```
cd ~/workspace
mkdir bmb_module2/ bmb_module2/16S_analysis
cd bmb_module2/16S_analysis
ln -s ~/CourseData/MIC_data/bmb_raw_data/16S_raw_data raw_data
ln -s ~/CourseData/MIC_data/bmb_raw_data/Blueberry_16S_metadata.tsv .
```

You will have installed the latest version of QIIME2 in Module 1, so you can activate that environment with the command below:

```
conda activate qiime2-amplicon-2025.4
```

If you didn't get to that part, you can use this environment instead:

```
conda activate qiime2-amplicon-backup
```

Throughout this module, there are some questions aimed to help your understanding of some of the key concepts. You'll find the answers at the bottom of this page, but no one will be marking them.

3.2.3 1.1. 16S First steps

3.2.3.1 1.1.1. Inspect raw data

First, let's take a look at the directory containing our raw reads as well as our metadata file.

```
ls raw_data
```

```
head Blueberry_16S_metadata.tsv
```

QUESTION!

- **Question 1**: How many samples are there?
- **Question 2**: Into what group(s) are the samples classified?

3.2.3.2 1.1.2. Import FASTQs as QIIME2 artifact

To standardize QIIME 2 analyses and to keep track of provenance (i.e. a list of what commands were previously run to produce a file) a special format is used for all QIIME 2 input and output files called an "artifact" (with the extension QZA). The first step is to import the raw reads as a QZA file. We will first create a new directory.

```
mkdir reads_qza
```

```
qiime tools import \
  --type SampleData[PairedEndSequencesWithQuality] \
  --input-path raw_data/ \
  --output-path reads_qza/reads.qza \
  --input-format CasavaOneEightSingleLanePerSampleDirFmt
```

This might take a minute! If it hasn't come back up with the command prompt that looks something like `(qiime2-amplicon-2025.4) ubuntu@ip-10-0-1-248:~/workspace/bmb_module2/16S_analysis$` yet, then it hasn't finished running yet and you'll need to be patient :)

All of the FASTQs are now in the single artifact file reads_qza/reads.qza. This file format can be a little confusing at first, but it is actually just a zipped folder. You can manipulate and explore these files better with the `qiime tools utilities` (e.g. `peek` and `view`).

3.2.3.3 1.1.3. Trim primers with cutadapt

Screen out reads that do not begin with primer sequence and remove primer sequence from reads using the cutadapt QIIME 2 plugin. The below primers correspond to the 16S V6-V8 region (bacteria-specific primer set). You can see more about different primers and the taxa that they target here.

```
qiime cutadapt trim-paired \
  --i-demultiplexed-sequences reads_qza/reads.qza \
  --p-cores 4 \
  --p-front-f ACGCGHNRAACCTTACC \
  --p-front-r ACGGGCRGTGWGTRCAA \
  --p-discard-untrimmed \
  --p-no-indels \
  --o-trimmed-sequences reads_qza/reads_trimmed.qza
```

Visualizing your output data is a good idea after any step to make sure nothing unexpected occurred. The following command generates a "visualization" file with the extension QZV.

We still have a few preprocessing requirements to check off our list before denoising, so we can wait until these steps are complete to visualize our data. However, if you would like to see what paired-end reads look like before joining, run the following command and open the QZV file in QIIME2 View.

```
qiime demux summarize \
  --i-data reads_qza/reads_trimmed.qza \
  --o-visualization reads_qza/reads_trimmed_summary.qzv
```

Remember that you can see and download these files on the webpage: http://#.uhn-hpc.ca/ (replace ## with your number!)

QUESTION!

- **Question 3**: What would happen if you ran this exact command on V4/V5-amplified sequences?

3.2.4 1.2. 16S Denoising the reads into amplicon sequence variants

Different denoising tools require different levels of preprocessing before the actual denoising happens. For example, DADA2 performs read joining and quality filtering as part of the denoising step itself, and can be run directly after trimming the primers. Due to speed considerations, we'll be using Deblur instead, which requires that these steps be carried out separately. Guidelines for running DADA2 can be found here.

3.2.4.1 1.2.1. Join paired-end reads

Forward and reverse reads can be joined with VSEARCH as shown below. This will generate QZA files for both the joined/merged sequences and unmerged sequences.

```
qiime vsearch merge-pairs \
  --i-demultiplexed-seqs reads_qza/reads_trimmed.qza \
  --output-dir reads_qza/reads_joined
```

3.2.4.2 1.2.2. Filter out low-quality reads

This command will filter out low-quality reads based on the default options.

```
qiime quality-filter q-score \
  --i-demux reads_qza/reads_joined/merged_sequences.qza \
  --o-filter-stats filt_stats.qza \
  --o-filtered-sequences reads_qza/reads_trimmed_joined_filt.qza
```

3.2.4.3 1.2.3. Summarize joined and filtered reads

It is a good idea at this point just to verify that there haven't been any substantial losses of reads, before going through the whole ASV process, at either the joining or quality-filtering steps above. You will also need to select a length to trim back to that maintains the largest/acceptable quantity of reads during denoising.

```
qiime demux summarize \
  --i-data reads_qza/reads_trimmed_joined_filt.qza \
  --o-visualization reads_qza/reads_trimmed_joined_filt_summary.qzv
```

Now open the file in QIIME2 View and look at the Overview and Interactive Quality Plot tabs to explore your data and answer the following questions.

QUESTION!

- **Question 4**: How long are our forward reads? Why are there no reverse reads in our file?
- **Question 5**: What would be a good trim length for our reads? Remember that there are answers at the bottom of the page if you would like to check this.

3.2.4.4 1.2.4. Running Deblur

Running the Deblur workflow will correct the raw reads into amplicon sequence variants (ASVs). This denoising tool filters out reads that either do match to known noise or that do not match with low similarity to the expected amplicon region. Note that the below command will retain singletons, which would have been filtered out unless we set –p-min-reads 1, and is for 16S sequences only. For other amplicon regions, you can either use the denoise-other option in the command and specify a reference database of sequences to use for positive filtering (as in the below versions for 18S and ITS) or use DADA2.

You'll need to replace X here with the trim length that was decided in Question 5.

```
qiime deblur denoise-16S \
  --i-demultiplexed-seqs reads_qza/reads_trimmed_joined_filt.qza \
  --p-trim-length X \
  --p-sample-stats \
  --p-jobs-to-start 4 \
  --p-min-reads 1 \
  --output-dir deblur_output
```

NOTE!

This command may take a few minutes to run.

3.2.4.5 1.2.5. Summarizing Deblur output

Once a denoising pipeline has been run you can summarize the output table with the below command, which will create a visualization artifact for you to view. We will use this visualization later to determine the the cut-offs for filtering the table below, but for now you should mainly take a look at the visualization to ensure that sufficient reads have been retained after running deblur. This denoising tool filters out reads that either do match to known noise or that do not match with low similarity to the expected amplicon region. If your samples have very low depth after running deblur (compared to the input read depth) this could be a red flag that either you ran the tool incorrectly, you have a lot of noise in your data, or that deblur is inappropriate for your dataset.

```
qiime feature-table summarize \
  --i-table deblur_output/table.qza \
  --o-visualization deblur_output/deblur_table_summary.qzv
```

QUESTION!

- **Question 6**: What is the mean sequencing depth per sample after denoising?
- **Question 7**: Which sample has the least reads?

3.2.5 1.3. 16S Assign taxonomy to ASVs

```
You can assign taxonomy to your ASVs using a Naive-Bayes approach implemented in the [scikit lear

-   Full ITS - fungi only (classifier_sh_refs_qiime_ver9_99_s_27.10.2022_ITS.qza)
-   Full ITS - all eukaryotes (classifier_sh_refs_qiime_ver9_99_s_all_27.10.2022_ITS.qza)
```

We're going to use the Greengenes2 classifier. We can download that from the FTP site here - click on the folder `sklearn-1.4.2-compatible-nb-classifiers/`, and then copy the link for the `2022.10.backbone.full-length.nb.sklearn-1.4.2.qza` classifier. (Right-click and select "Copy Link Address". Then we'll use the `wget` command like we did in Module 1. That will look something like this:

```
wget link-address
```

3.2.5.1 1.3.1. Run taxonomic classification

You can run the taxonomic classification with this command, which is one of the longest running and most memory-intensive command of the tutorial. If you receive an error related to insufficient memory (and if you cannot increase your memory usage) then you can look into the `--p-reads-per-batch` option and set this to be lower than the default (which is dynamic depending on sample depth and the number of threads) and also try running the command with fewer jobs (e.g. set`--p-n-jobs 1`).

Because of the memory constraints of this workshop, you can copy the output into your folder:

But here is the code you would use to run the taxonomic classification:

```
qiime feature-classifier classify-sklearn \
  --i-reads deblur_output/representative_sequences.qza \
  --i-classifier 2022.10.backbone.full-length.nb.sklearn-1.4.2.qza \
  --p-n-jobs 4 \
  --output-dir taxa
```

NOTE!

Unfortunately we don't actually have enough memory to run this so you'll see a "Killed" output for this. Instead, we'll copy across the output that we would have got.

```
mkdir taxa
cp ~/CourseData/MIC_data/bmb_module2/16S/taxa/classification.qza taxa
```

As with all QZA files, you can export the output file to take a look at the classifications and confidence scores:

```
qiime tools export \
  --input-path taxa/classification.qza \
  --output-path taxa
```

3.2.5.2 1.3.2 Assess subset of taxonomic assignments with BLAST

The performance of the taxonomic classification is difficult to assess without a gold-standard reference, but nonetheless one basic sanity check is to compare the taxonomic assignments with the top BLASTn hits for certain ASVs. First, generate a QZV file for the denoised representative sequences in QIIME 2 by running:

```
qiime feature-table tabulate-seqs \
  --i-data deblur_output/representative_sequences.qza \
  --o-visualization deblur_output/representative_sequences.qzv
```

This QZV file tabulates the denoised sequences. Clicking on the nucleotide sequence links to a BLASTn search for that sequence. By comparing these BLAST hits with the taxonomic assignment of ASVs generated above you can reassure yourself that the taxonomic assignments overall worked correctly. It's a good idea to select ~5 ASVs to BLAST for this validation, which should be from taxonomically different groups, such as different phyla, according to the taxonomic classifier.

You can then check the taxonomy that was assigned to them using your classifier - you can either open up the `taxonomy.tsv` file and search for the Feature ID/ASV names there, or you can use the `grep` command like so:

```
grep "02fda7604175053e87dc11e2598a89e0" taxa/taxonomy.tsv
```

3.2.6 1.4. 16S Filtering resultant table

Filtering the denoised table is an important step of microbiome data analysis. You can see more details on this process in the QIIME 2 filtering tutorial.

3.2.6.1 1.4.1. Filter out rare ASVs

Based on the summary visualization created in step 2.5 above you can choose a cut-off for how frequent a variant needs to be (and optionally how many samples need to have the variant) for it to be retained. Here we will remove all ASVs that have a frequency of less than 0.1% of the mean sample depth. This cut-off excludes ASVs that are likely due to MiSeq bleed-through between runs (reported by Illumina to be 0.1% of reads). To calculate this cut-off you would identify the mean sample depth in the visualization created in step 2.5 (`deblur_table_summary.qzv`), multiply it by 0.001, and round to the nearest integer.

Once you've determined how you would like to filter your table you can do so with this command (X is a placeholder for your choice):

```
qiime feature-table filter-features \
  --i-table deblur_output/table.qza \
  --p-min-frequency X \
  --p-min-samples 1 \
  --o-filtered-table deblur_output/deblur_table_filt.qza
```

3.2.6.2 1.4.2. Filter out contaminant and unclassified ASVs

Once we have assigned taxonomy to our ASVs we can use that information to remove ASVs which are likely contaminants or noise based on the taxonomic labels. Two common contaminants in 16S sequencing data are mitochondrial and chloroplast 16S sequences, which can be removed by excluding any ASV which contains those terms in its taxonomic label. It can also be **sometimes** useful to exclude any ASV that is unclassified at the phylum level since these sequences could be noise (e.g. possible chimeric sequences). Note that if your data has not been classified against the default database you may need to change `p\_\_` to be a string that enables phylum-level assignments to be identified or simply omit that line.

In general though, it can be very informative if your sequencing reads are coming back with significant amounts of unclassified ASVs as it can indicate upstream analysis problems or indicate you are studying a poorly characterized environment where you have a good chance of identifying a lot of novel phyla. Therefore, our recommendation is to not filter out the unclassified sequences by default, but we will do so here.

```
qiime taxa filter-table \
  --i-table deblur_output/deblur_table_filt.qza \
  --i-taxonomy taxa/classification.qza \
  --p-include p__ \
  --p-exclude mitochondria,chloroplast \
  --o-filtered-table deblur_output/deblur_table_filt_contam.qza
```

3.2.6.3 *1.4.3. (Optional) Exclude low-depth samples*

Often certain samples will have quite low depth after these filtering steps, which can be excluded from downstream analyses since they will largely add noise. There is no single cut-off that works best for all datasets, but researchers often use minimum cut-offs within the range of 1000 to 4000 reads. You can also use a cut-off much lower than this if you want to retain all samples except those that failed entirely (e.g. depth < 50 reads).

Ideally you would choose this cut-off after visualizing rarefaction curves to determine at what read depth the richness of your samples plateaus and choose a cut-off as close to this plateau as possible while retaining sufficient sample size for your analyses. We learn more about rarefaction curves in Module 3.

3.2.6.4 1.4.4. Subset and summarize filtered table

Check output after filtering.

```
qiime feature-table summarize \
  --i-table deblur_output/deblur_table_filt_contam.qza \
  --o-visualization deblur_output/deblur_table_filt_contam_summary.qzv
```

QUESTION!

- **Question 8**: What is the minimum and maximum sequencing depth across all samples?

Happy? Copy a final table.

```
cp deblur_output/deblur_table_filt_contam.qza deblur_output/deblur_table_final.qza
```

Once we have our final filtered table we will need to subset the QZA file containing the ASV sequences to the same set. You can exclude any removed ASVs from the sequence file with this command:

```
qiime feature-table filter-seqs \
  --i-data deblur_output/representative_sequences.qza \
  --i-table deblur_output/deblur_table_final.qza  \
  --o-filtered-data deblur_output/rep_seqs_final.qza
```

Finally, you can make a new summary of the final filtered abundance table:

```
qiime feature-table summarize \
  --i-table deblur_output/deblur_table_final.qza \
  --o-visualization deblur_output/deblur_table_final_summary.qzv
```

3.2.7 16S Answers

Question 1: How many samples are there?

We have 10 samples. The raw data folder contains one fastq.gz file for each set of the forward and reverse sequences (labelled R1 and R2, respectively) for each of the samples (B-Rtxxx).

Question 2: Into what group(s) are the samples classified?

The samples we are using are classified into two different groups: Forest or Managed sites. The study we're looking at does also look at bulk or rhizosphere samples, but we're just using a small subset of the samples for this workshop.

Question 3: What would happen if you ran this exact command on V4/V5-amplified sequences?

`Cutadapt` will only trim reads that match the specified primer sequence. Therefore, most reads would be discarded because we are including the `--p-discard-untrimmed` option.

Question 4: How long are our forward reads? Why are there no reverse reads in our file?

The forward read median length is 405 nucleotides. There are no reverse reads because forward and reverse reads were merged into one sequence during read joining.

Question 5: What would be a good trim length for our reads?

There is no one right answer for this question, but a trim length of 390 nucleotides will maintain most of our sequences apart from any that are really short. You could choose a different length, but this will give different answers further on. Typically, this choice is a trade-off between maintaining a longer sequence length where we're more likely to get finer-resolution taxonomic classification, or trimming more so that we retain more sequences.

Question 6: What is the mean sequencing depth per sample after denoising?

The mean sequencing depth (frequency) across all denoised samples is 6,669.8 reads.

Question 7: Which sample has the least reads?

Sample B-Rt151 has the lowest sequencing depth (4,332 reads).

Question 8: What is the minimum and maximum sequencing depth across all samples?

The final minimum sequencing depth is 3,432 and the maximum sequencing depth is 8,650 reads.

3.2.8 2. 18S

Create for Module 2 inside `workspace` and create a symlink to the raw FASTQ files and the metadata file.

```
cd ~/workspace
mkdir bmb_module2/ bmb_module2/18S_analysis
```

```
cd bmb_module2/18S_analysis
ln -s ~/CourseData/MIC_data/bmb_raw_data/18S_raw_data raw_data
ln -s ~/CourseData/MIC_data/bmb_raw_data/Plastisphere_18S_metadata.tsv .
```

You will have installed the latest version of QIIME2 in Module 1, so you can activate that environment with the command below:

```
conda activate qiime2-amplicon-2025.4
```

If you didn't get to that part, you can use this environment instead:

```
conda activate qiime2-amplicon-backup
```

Throughout this module, there are some questions aimed to help your understanding of some of the key concepts. You'll find the answers at the bottom of this page, but no one will be marking them.

3.2.9 2.1. 18S First steps

3.2.9.1 2.1.1. Inspect raw data

First, let's take a look at the directory containing our raw reads as well as our metadata file.

```
ls raw_data
```

```
head Plastisphere_18S_metadata.tsv
```

QUESTION!

- **Question 1**: How many samples are there?
- **Question 2**: Into what group(s) are the samples classified?

3.2.9.2 2.1.2. Import FASTQs as QIIME2 artifact

To standardize QIIME 2 analyses and to keep track of provenance (i.e. a list of what commands were previously run to produce a file) a special format is used for all QIIME 2 input and output files called an "artifact" (with the extension QZA). The first step is to import the raw reads as a QZA file. We will first create a new directory.

```
mkdir reads_qza
```

```
qiime tools import \
  --type SampleData[PairedEndSequencesWithQuality] \
  --input-path raw_data/ \
  --output-path reads_qza/reads.qza \
  --input-format CasavaOneEightSingleLanePerSampleDirFmt
```

This might take a minute! If it hasn't come back up with the command prompt that looks something like `(qiime2-amplicon-2025.4) ubuntu@ip-10-0-1-248:~/workspace/bmb_module2/18S_analysis$` yet, then it hasn't finished running yet and you'll need to be patient :)

All of the FASTQs are now in the single artifact file reads_qza/reads.qza. This file format can be a little confusing at first, but it is actually just a zipped folder. You can manipulate and explore these files better with the `qiime tools utilities` (e.g. `peek` and `view`).

3.2.9.3 2.1.3. Trim primers with cutadapt

Screen out reads that do not begin with primer sequence and remove primer sequence from reads using the cutadapt QIIME 2 plugin. The below primers correspond to the 18S V4 region (eukaryote-specific primer set). You can see more about different primers and the taxa that they target here.

```
qiime cutadapt trim-paired \
  --i-demultiplexed-sequences reads_qza/reads.qza \
  --p-cores 4 \
  --p-front-f ATAACAGGTCTGTGATGCCCT \
  --p-front-r CCTTCYGCAGGTTCACCTAC \
  --p-discard-untrimmed \
  --p-no-indels \
  --o-trimmed-sequences reads_qza/reads_trimmed.qza
```

Visualizing your output data is a good idea after any step to make sure nothing unexpected occurred. The following command generates a "visualization" file with the extension QZV.

We still have a few preprocessing requirements to check off our list before denoising, so we can wait until these steps are complete to visualize our data. However, if you would like to see what paired-end reads look like before joining, run the following command and open the QZV file in QIIME2 View.

```
qiime demux summarize \
  --i-data reads_qza/reads_trimmed.qza \
  --o-visualization reads_qza/reads_trimmed_summary.qzv
```

Remember that you can see and download these files on the webpage: http://#.uhn-hpc.ca/ (replace ## with your number!)

QUESTION!

- **Question 3**: What would happen if you ran this exact command on 16S V4/V5-amplified sequences?

3.2.10 2.2. 18S Denoising the reads into amplicon sequence variants

Different denoising tools require different levels of preprocessing before the actual denoising happens. For example, DADA2 performs read joining and quality filtering as part of the denoising step itself, and can be run directly after trimming the primers. Due to speed considerations, we'll be using Deblur instead, which requires that these steps be carried out separately. Guidelines for running DADA2 can be found here.

3.2.10.1 2.2.1. Join paired-end reads

Forward and reverse reads can be joined with VSEARCH as shown below. This will generate QZA files for both the joined/merged sequences and unmerged sequences.

```
qiime vsearch merge-pairs \
  --i-demultiplexed-seqs reads_qza/reads_trimmed.qza \
  --output-dir reads_qza/reads_joined
```

3.2.10.2 2.2.2. Filter out low-quality reads

This command will filter out low-quality reads based on the default options.

```
qiime quality-filter q-score \
  --i-demux reads_qza/reads_joined/merged_sequences.qza \
  --o-filter-stats filt_stats.qza \
  --o-filtered-sequences reads_qza/reads_trimmed_joined_filt.qza
```

3.2.10.3 2.2.3. Summarize joined and filtered reads

It is a good idea at this point just to verify that there haven't been any substantial losses of reads, before going through the whole ASV process, at either the joining or quality-filtering steps above. You will also need to select a length to trim back to that maintains the largest/acceptable quantity of reads during denoising.

```
qiime demux summarize \
  --i-data reads_qza/reads_trimmed_joined_filt.qza \
  --o-visualization reads_qza/reads_trimmed_joined_filt_summary.qzv
```

Now open the file in QIIME2 View and look at the Overview and Interactive Quality Plot tabs to explore your data and answer the following questions.

QUESTION!

- **Question 4**: How long are our forward reads? Why are there no reverse reads in our file?
- **Question 5**: What would be a good trim length for our reads? Remember that there are answers at the bottom of the page if you would like to check this.

3.2.10.4 2.2.4. Running Deblur

Running the Deblur workflow will correct the raw reads into amplicon sequence variants (ASVs). This denoising tool filters out reads that either do match to known noise or that do not match with low similarity to the expected amplicon region. Note that the below command will retain singletons, which would have been filtered out unless we set `--p-min-reads 1`, and is for 16S sequences

only. While for 16S there is a denoise-16S option that doesn't require any reference sequences, for other amplicon regions (like 18S and ITS), you can use the denoise-other option and specify a reference database of sequences to use for positive filtering. We keep these files available for others on our lab server here.

We'll go ahead and download the 18S sequences. You can do that by right-clicking on the `gb203_pr2_all_10_28_99p_clean_prob-rm.fasta` file, and clicking "Copy Link Address". You can then use the `wget` command, like we did in Module 1. So your command should look something like:

```
wget link-address
```

Then you can import this into QIIME:

```
qiime tools import \
   --input-path gb203_pr2_all_10_28_99p_clean_prob-rm.fasta \
   --output-path gb203_pr2_all_10_28_99p_clean_prob-rm.qza \
   --type 'FeatureData[Sequence]'
```

You'll need to replace X here with the trim length that was decided in Question 5.

```
qiime deblur denoise-other \
   --i-demultiplexed-seqs reads_qza/reads_trimmed_joined_filt.qza \
   --i-reference-seqs gb203_pr2_all_10_28_99p_clean_prob-rm.qza \
   --p-trim-length X \
   --p-sample-stats \
   --p-jobs-to-start 4 \
   --p-min-reads 1 \
   --output-dir deblur_output
```

NOTE!

This command may take around ten minutes to run.

3.2.10.5 2.2.5. Summarizing Deblur output

Once a denoising pipeline has been run you can summarize the output table with the below command, which will create a visualization artifact for you to view. We will use this visualization later to determine the the cut-offs for filtering

the table below, but for now you should mainly take a look at the visualization to ensure that sufficient reads have been retained after running deblur. This denoising tool filters out reads that either do match to known noise or that do not match with low similarity to the expected amplicon region. If your samples have very low depth after running deblur (compared to the input read depth) this could be a red flag that either you ran the tool incorrectly, you have a lot of noise in your data, or that deblur is inappropriate for your dataset.

```
qiime feature-table summarize \
  --i-table deblur_output/table.qza \
  --o-visualization deblur_output/deblur_table_summary.qzv
```

QUESTION!

- **Question 6: What is the mean sequencing depth per sample after denoising?**
- **Question 7: Which sample has the least reads?**

3.2.11 2.3. 18S Assign taxonomy to ASVs

You can assign taxonomy to your ASVs using a Naive-Bayes approach implemented in the scikit learn Python library and the SILVA or UNITE databases. This approach requires that a classifier be trained in advance on a reference database. We recommend users use a widely used classifier to help ensure there are no unexpected issues with the Naive-Bayes model. We previously maintained primer-specific classifiers, which theoretically can provide more accurate classifications, but we no longer do this due to concerns regarding issues with the trained models that are difficult to catch if only a couple people are running them. The full-length 16S/18S classifier can be downloaded from the QIIME 2 website (silva-138-99-nb-classifier.qza for the latest classifier). Sometimes there are custom classifiers or other specific databases that are good for specific purposes. For 18S, we're going to be using the PR2 (Protist Ribosomal Reference) database. A custom classifier that we've generated is available (see downloads and commands used to create classifiers).

We're going to use the PR2 classifier. As you did above, click on "Copy Link Address" for the file `pr2_version_5.1.0_SSU_18S.qza` and then use `wget` to download it. Check how you ran this command before if you're unsure!

3.2.11.1 2.3.1. Run taxonomic classification

You can run the taxonomic classification with this command, which is one of the longest running and most memory-intensive commands of the tutorial. If you receive an error related to insufficient memory (and if you cannot increase your memory usage) then you can look into the `--p-reads-per-batch` option and set this to be lower than the default (which is dynamic depending on sample depth and the number of threads) and also try running the command with fewer jobs (e.g. set `--p-n-jobs 1`).

```
qiime feature-classifier classify-sklearn \
  --i-reads deblur_output/representative_sequences.qza \
  --i-classifier pr2_version_5.1.0_SSU_18S.qza \
  --p-n-jobs 4 \
  --output-dir taxa
```

NOTE!

Unfortunately we don't actually have enough memory to run this so you'll see a "Killed" or "Terminated" output for this. Instead, we'll copy across the output that we would have got.

```
mkdir taxa
cp ~/CourseData/MIC_data/bmb_module2/18S/taxa/classification.qza taxa
```

As with all QZA files, you can export the output file to take a look at the classifications and confidence scores:

```
qiime tools export \
  --input-path taxa/classification.qza \
  --output-path taxa
```

3.2.11.2 2.3.2 Assess subset of taxonomic assignments with BLAST

The performance of the taxonomic classification is difficult to assess without a gold-standard reference, but nonetheless one basic sanity check is to compare the taxonomic assignments with the top BLASTn hits for certain ASVs. First, generate a QZV file for the denoised representative sequences in QIIME 2 by running:

```
qiime feature-table tabulate-seqs \
  --i-data deblur_output/representative_sequences.qza \
  --o-visualization deblur_output/representative_sequences.qzv
```

This QZV file tabulates the denoised sequences. Clicking on the nucleotide sequence links to a BLASTn search for that sequence. By comparing these BLAST hits with the taxonomic assignment of ASVs generated above you can reassure yourself that the taxonomic assignments overall worked correctly. It's a good idea to select ~5 ASVs to BLAST for this validation, which should be from taxonomically different groups, such as different phyla, according to the taxonomic classifier.

You can then check the taxonomy that was assigned to them using your classifier - you can either open up the `taxonomy.tsv` file and search for the Feature ID/ASV names there, or you can use the `grep` command like so:

```
grep "5ecf17cbc8af149eb7fbec898b841d72" taxa/taxonomy.tsv
```

You should see that this sequence has been classified as `d__Eukaryota; sg__Archaeplastida; di__Rhodophyta; sd__Eurhodophytina; c__Florideophyceae; o__Ceramiales; f__Rhodomelaceae` with confidence ~0.99. If we look at the taxonomy on NCBI BLAST, then we'll see that some of the top hits are *Osmundea pinnatifida* (identity 99.34%) and *Laurencia filiformis* (identity 98.01%). A quick google tells me that both of these are in the family Rhodomelaceae, so it seems pretty reasonable that - with two hits with relatively high identity that are from different genera - our classifier would determine that this ASV can be classified at the family level as Rhodomelaceae.

Try taking a look at a few more and seeing how they seem.

3.2.12 2.4. 18S Filtering resultant table

Filtering the denoised table is an important step of microbiome data analysis. You can see more details on this process in the QIIME 2 filtering tutorial.

3.2.12.1 2.4.1. Filter out rare ASVs

Based on the summary visualization created in step 2.5 above you can choose a cut-off for how frequent a variant needs to be (and optionally how many samples need to have the variant) for it to be retained. Here we will remove all ASVs that have a frequency of less than 0.1% of the mean sample depth. This cut-off excludes ASVs that are likely due to MiSeq bleed-through between runs (reported by Illumina to be 0.1% of reads). To calculate this cut-off you would identify the mean sample depth in the visualization created in step 2.5

(`deblur_table_summary.qzv`), multiply it by 0.001, and round to the nearest integer.

Once you've determined how you would like to filter your table you can do so with this command (X is a placeholder for your choice):

```
qiime feature-table filter-features \
  --i-table deblur_output/table.qza \
  --p-min-frequency X \
  --p-min-samples 1 \
  --o-filtered-table deblur_output/deblur_table_filt.qza
```

3.2.12.2 2.4.2. Filter out contaminant and unclassified ASVs

Once we have assigned taxonomy to our ASVs we can use that information to remove ASVs which are likely contaminants or noise based on the taxonomic labels. Two common contaminants in 16S sequencing data are mitochondrial and chloroplast 16S sequences, which can be removed by excluding any ASV which contains those terms in its taxonomic label. Obviously this is different for 18S, so we don't need to exclude those here. It can also be **sometimes** useful to exclude any ASV that is unclassified at the phylum level since these sequences could be noise (e.g. possible chimeric sequences). Note that here we actually don't have phylum-level classifications because the PR2 database doesn't use these - it uses levels of domain, supergroup, division, subdivision, class, order, family, genus, and species. So in our taxonomy file you'll see that these are d, sg, di, sd, c, o, f, g, and s, respectively. We'll therefore probably want to use the sg___ instead of p___ like we used for 16S data.

In general though, it can be very informative if your sequencing reads are coming back with significant amounts of unclassified ASVs as it can indicate upstream analysis problems or indicate you are studying a poorly characterized environment where you have a good chance of identifying a lot of novel phyla. Therefore, our recommendation is to not filter out the unclassified sequences by default, but we will do so here.

```
qiime taxa filter-table \
  --i-table deblur_output/deblur_table_filt.qza \
  --i-taxonomy taxa/classification.qza \
  --p-include sg__ \
  --o-filtered-table deblur_output/deblur_table_filt_contam.qza
```

3.2.12.3 *2.4.3. (Optional) Exclude low-depth samples*

Often certain samples will have quite low depth after these filtering steps, which can be excluded from downstream analyses since they will largely add noise.

There is no single cut-off that works best for all datasets, but researchers often use minimum cut-offs within the range of 1000 to 4000 reads. You can also use a cut-off much lower than this if you want to retain all samples except those that failed entirely (e.g. depth < 50 reads).

Ideally you would choose this cut-off after visualizing rarefaction curves to determine at what read depth the richness of your samples plateaus and choose a cut-off as close to this plateau as possible while retaining sufficient sample size for your analyses. We learn more about rarefaction curves in Module 3.

3.2.12.4 2.4.4. Subset and summarize filtered table

Check output after filtering.

```
qiime feature-table summarize \
  --i-table deblur_output/deblur_table_filt_contam.qza \
  --o-visualization deblur_output/deblur_table_filt_contam_summary.qzv
```

QUESTION!

- **Question 8**: What is the minimum and maximum sequencing depth across all samples?

Happy? Copy a final table.

```
cp deblur_output/deblur_table_filt_contam.qza deblur_output/deblur_table_final.qza
```

Once we have our final filtered table we will need to subset the QZA file containing the ASV sequences to the same set. You can exclude any removed ASVs from the sequence file with this command:

```
qiime feature-table filter-seqs \
  --i-data deblur_output/representative_sequences.qza \
  --i-table deblur_output/deblur_table_final.qza  \
  --o-filtered-data deblur_output/rep_seqs_final.qza
```

Finally, you can make a new summary of the final filtered abundance table:

```
qiime feature-table summarize \
  --i-table deblur_output/deblur_table_final.qza \
  --o-visualization deblur_output/deblur_table_final_summary.qzv
```

3.2.13 18S Answers

Question 1: How many samples are there?

We have 12 samples. The raw data folder contains one fastq.gz file for each set of the forward and reverse sequences (labelled R1 and R2, respectively) for each of the samples (MHxxxx18S).

Question 2: Into what group(s) are the samples classified?

The samples we are using are classified using two different variables: light and time. These are pieces of plastic that were either incubated in a sunny coastal environment (OpenCoast) or in a cave (Cave; the light variable), and they were incubated for 6h or 28h (time).

Question 3: What would happen if you ran this exact command on 16S sequences?

`Cutadapt` will only trim reads that match the specified primer sequence. Therefore, most reads would be discarded because we are including the `--p-discard-untrimmed` option.

Question 4: How long are our forward reads? Why are there no reverse reads in our file?

The forward read median length is 331 nucleotides. There are no reverse reads because forward and reverse reads were merged into one sequence during read joining.

Question 5: What would be a good trim length for our reads?

There is no one right answer for this question, but a trim length of 302 nucleotides will maintain most of our sequences apart from any that are really short. You could choose a different length, but this will give different answers further on. Typically, this choice is a trade-off between maintaining a longer sequence length where we're more likely to get finer-resolution taxonomic classification, or trimming more so that we retain more sequences.

Question 6: What is the mean sequencing depth per sample after denoising?

The mean sequencing depth (frequency) across all denoised samples is 6,588.8 reads.

Question 7: Which sample has the least reads?

Sample MH28C318S has the lowest sequencing depth (only 27 reads).

Question 8: What is the minimum and maximum sequencing depth across all samples?

The final minimum sequencing depth is 7 and the maximum sequencing depth is 12,890 reads.

3.2.14 3. ITS

Create for Module 2 inside `workspace` and create a symlink to the raw FASTQ files and the metadata file.

```
cd ~/workspace
mkdir bmb_module2/ bmb_module2/ITS_analysis
cd bmb_module2/ITS_analysis
ln -s ~/CourseData/MIC_data/bmb_raw_data/ITS_raw_data raw_data
ln -s ~/CourseData/MIC_data/bmb_raw_data/pregnancy_ITS_metadata.tsv .
```

You will have installed the latest version of QIIME2 in Module 1, so you can activate that environment with the command below:

```
conda activate qiime2-amplicon-2025.4
```

If you didn't get to that part, you can use this environment instead:

```
conda activate qiime2-amplicon-backup
```

Throughout this module, there are some questions aimed to help your understanding of some of the key concepts. You'll find the answers at the bottom of this page, but no one will be marking them.

3.2.15 3.1. ITS First steps

3.2.15.1 3.1.1. Inspect raw data

First, let's take a look at the directory containing our raw reads as well as our metadata file.

```
ls raw_data
```

```
head pregnancy_ITS_metadata.tsv
```

QUESTION!

- **Question 1**: How many samples are there?
- **Question 2**: Into what group(s) are the samples classified?

3.2.15.2 3.1.2. Import FASTQs as QIIME2 artifact

To standardize QIIME 2 analyses and to keep track of provenance (i.e. a list of what commands were previously run to produce a file) a special format is used for all QIIME 2 input and output files called an "artifact" (with the extension QZA). The first step is to import the raw reads as a QZA file. We will first create a new directory.

```
mkdir reads_qza
```

```
qiime tools import \
  --type SampleData[PairedEndSequencesWithQuality] \
  --input-path raw_data/ \
  --output-path reads_qza/reads.qza \
  --input-format CasavaOneEightSingleLanePerSampleDirFmt
```

This might take a minute! If it hasn't come back up with the command prompt that looks something like `(qiime2-amplicon-2025.4) ubuntu@ip-10-0-1-248:~/workspace/bmb_module2/ITS_analysis$` yet, then it hasn't finished running yet and you'll need to be patient :)

All of the FASTQs are now in the single artifact file `reads_qza/reads.qza`. This file format can be a little confusing at first, but it is actually just a zipped folder. You can manipulate and explore these files better with the `qiime tools utilities` (e.g. `peek` and `view`).

3.2.15.3 3.1.3. Trim primers with cutadapt

Screen out reads that do not begin with primer sequence and remove primer sequence from reads using the cutadapt QIIME 2 plugin. The below primers correspond to the 18S V8-V9 region (eukaryote-specific primer set). You can see more about some other primers and the taxa that they target here.

```
qiime cutadapt trim-paired \
  --i-demultiplexed-sequences reads_qza/reads.qza \
  --p-cores 4 \
  --p-front-f GCATCGATGAAGAACGCAGC \
  --p-front-r TCCTCCGCTTATTGATATGC \
  --p-discard-untrimmed \
  --p-no-indels \
  --o-trimmed-sequences reads_qza/reads_trimmed.qza
```

Visualizing your output data is a good idea after any step to make sure nothing unexpected occurred. The following command generates a "visualization" file with the extension QZV.

We still have a few preprocessing requirements to check off our list before denoising, so we can wait until these steps are complete to visualize our data. However, if you would like to see what paired-end reads look like before joining, run the following command and open the QZV file in QIIME2 View.

```
qiime demux summarize \
  --i-data reads_qza/reads_trimmed.qza \
  --o-visualization reads_qza/reads_trimmed_summary.qzv
```

Remember that you can see and download these files on the webpage: http://#.uhn-hpc.ca/ (replace ## with your number!)

QUESTION!

- **Question 3**: What would happen if you ran this exact command on 16S V4/V5-amplified sequences?

3.2.16 3.2. ITS Denoising the reads into amplicon sequence variants

Different denoising tools require different levels of preprocessing before the actual denoising happens. For example, DADA2 performs read joining and quality filtering as part of the denoising step itself, and can be run directly after trimming the primers. Due to speed considerations, we'll be using Deblur instead, which requires that these steps be carried out separately. Guidelines for running DADA2 can be found here.

3.2.16.1 2.1. Join paired-end reads

Forward and reverse reads can be joined with VSEARCH as shown below. This will generate QZA files for both the joined/merged sequences and unmerged sequences.

```
qiime vsearch merge-pairs \
  --i-demultiplexed-seqs reads_qza/reads_trimmed.qza \
  --output-dir reads_qza/reads_joined
```

3.2.16.2 3.2.2. Filter out low-quality reads

This command will filter out low-quality reads based on the default options.

```
qiime quality-filter q-score \
  --i-demux reads_qza/reads_joined/merged_sequences.qza \
  --o-filter-stats filt_stats.qza \
  --o-filtered-sequences reads_qza/reads_trimmed_joined_filt.qza
```

3.2.16.3 3.2.3. Summarize joined and filtered reads

It is a good idea at this point just to verify that there haven't been any substantial losses of reads, before going through the whole ASV process, at either the joining or quality-filtering steps above. You will also need to select a length to trim back to that maintains the largest/acceptable quantity of reads during denoising.

```
qiime demux summarize \
  --i-data reads_qza/reads_trimmed_joined_filt.qza \
  --o-visualization reads_qza/reads_trimmed_joined_filt_summary.qzv
```

Now open the file in QIIME2 View and look at the Overview and Interactive Quality Plot tabs to explore your data and answer the following questions.

QUESTION!

- **Question 4**: How long are our forward reads? Why are there no reverse reads in our file?
- **Question 5**: What would be a good trim length for our reads? Remember that there are answers at the bottom of the page if you would like to check this.

3.2.16.4 3.2.4. Running Deblur

Running the Deblur workflow will correct the raw reads into amplicon sequence variants (ASVs). This denoising tool filters out reads that either do match to known noise or that do not match with low similarity to the expected amplicon region. Note that the below command will retain singletons, which would have been filtered out unless we set `--p-min-reads 1`, and is for 16S sequences

only. While for 16S there is a denoise-16S option that doesn't require any reference sequences, for other amplicon regions (like 18S and ITS), you can use the denoise-other option and specify a reference database of sequences to use for positive filtering. We keep these files available for others on our lab server here.

We'll go ahead and download the 18S sequences. You can do that by right-clicking on the `UNITE_sh_refs_qiime_ver8_99_s_all_02.02.2019.fasta` file, and clicking "Copy Link Address". You can then use the `wget` command, like we did in Module 1. So your command should look something like:

```
wget link-address
```

Then you can import this into QIIME:

```
qiime tools import \
   --input-path UNITE_sh_refs_qiime_ver8_99_s_all_02.02.2019.fasta \
   --output-path UNITE_sh_refs_qiime_ver8_99_s_all_02.02.2019.qza \
   --type 'FeatureData[Sequence]'
```

You'll need to replace X here with the trim length that was decided in Question 5.

```
qiime deblur denoise-other \
   --i-demultiplexed-seqs reads_qza/reads_trimmed_joined_filt.qza \
   --i-reference-seqs UNITE_sh_refs_qiime_ver8_99_s_all_02.02.2019.qza \
   --p-trim-length X \
   --p-sample-stats \
   --p-jobs-to-start 4 \
   --p-min-reads 1 \
   --output-dir deblur_output
```

NOTE!

This command may take around ten minutes to run

3.2.16.5 3.2.5. Summarizing Deblur output

Once a denoising pipeline has been run you can summarize the output table with the below command, which will create a visualization artifact for you to view. We will use this visualization later to determine the the cut-offs for filtering

the table below, but for now you should mainly take a look at the visualization to ensure that sufficient reads have been retained after running deblur. This denoising tool filters out reads that either do match to known noise or that do not match with low similarity to the expected amplicon region. If your samples have very low depth after running deblur (compared to the input read depth) this could be a red flag that either you ran the tool incorrectly, you have a lot of noise in your data, or that deblur is inappropriate for your dataset.

```
qiime feature-table summarize \
  --i-table deblur_output/table.qza \
  --o-visualization deblur_output/deblur_table_summary.qzv
```

QUESTION!

- **Question 6**: What is the mean sequencing depth per sample after denoising?
- **Question 7**: Which sample has the least reads?

3.2.17 3.3. ITS Assign taxonomy to ASVs

You can assign taxonomy to your ASVs using a Naive-Bayes approach implemented in the scikit learn Python library and the SILVA or UNITE databases. This approach requires that a classifier be trained in advance on a reference database. We recommend users use a widely used classifier to help ensure there are no unexpected issues with the Naive-Bayes model. We previously maintained primer-specific classifiers, which theoretically can provide more accurate classifications, but we no longer do this due to concerns regarding issues with the trained models that are difficult to catch if only a couple people are running them. The full-length 16S/18S classifier can be downloaded from the QIIME 2 website (silva-138-99-nb-classifier.qza for the latest classifier). Custom classifiers for the ITS region that we have generated from the UNITE database are available as well (see downloads and commands used to create these files).

We're going to use one that contains fungal ITS sequences from the UNITE database. As you did above, click on "Copy Link Address" for the file `classifier_fungi_ITS_sh_taxonomy_sh_taxonomy_qiime_ver10_99_04.04.2024_dev.qza` and then use `wget` to download it. Check how you ran this command before if you're unsure!

3.2.17.1 3.3.1. Run taxonomic classification

You can run the taxonomic classification with this command, which is one of the longest running and most memory-intensive command of the tutorial. If you receive an error related to insufficient memory (and if you cannot increase your memory usage) then you can look into the `--p-reads-per-batch` option and set this to be lower than the default (which is dynamic depending on sample depth and the number of threads) and also try running the command with fewer jobs (e.g. set `--p-n-jobs 1`).

Because of the memory constraints of this workshop, you can copy the output into your folder:

But here is the code you would use to run the taxonomic classification:

```
qiime feature-classifier classify-sklearn \
  --i-reads deblur_output/representative_sequences.qza \
  --i-classifier classifier_fungi_ITS_sh_taxonomy_sh_taxonomy_qiime_ver10_99_04.04.202
  --p-n-jobs 4 \
  --output-dir taxa
```

Unfortunately we don't actually have enough memory to run this so you'll see a `Killed` output for this. Instead, we'll copy across the output that we would have got.

```
mkdir taxa
cp ~/CourseData/MIC_data/bmb_module2/ITS/taxa/classification.qza taxa
```

As with all QZA files, you can export the output file to take a look at the classifications and confidence scores:

```
qiime tools export \
  --input-path taxa/classification.qza \
  --output-path taxa
```

3.2.17.2 3.3.2. Assess subset of taxonomic assignments with BLAST

The performance of the taxonomic classification is difficult to assess without a gold-standard reference, but nonetheless one basic sanity check is to compare the taxonomic assignments with the top BLASTn hits for certain ASVs. First, generate a QZV file for the denoised representative sequences in QIIME 2 by running:

```
qiime feature-table tabulate-seqs \
  --i-data deblur_output/representative_sequences.qza \
  --o-visualization deblur_output/representative_sequences.qzv
```

This QZV file tabulates the denoised sequences. Clicking on the nucleotide sequence links to a BLASTn search for that sequence. By comparing these BLAST hits with the taxonomic assignment of ASVs generated above you can reassure yourself that the taxonomic assignments overall worked correctly. It's a good idea to select ~5 ASVs to BLAST for this validation, which should be from taxonomically different groups, such as different phyla, according to the taxonomic classifier.

You can then check the taxonomy that was assigned to them using your classifier - you can either open up the `taxonomy.tsv` file and search for the Feature ID/ASV names there, or you can use the `grep` command like so:

```
grep "af38114da4fce5a55693cb937b8991ad" taxa/taxonomy.tsv
```

You should see that this sequence has been classified as `k__Fungi;p__Ascomycota;c__Eurotiomycetes;o__Eurotial` with confidence ~0.99. If we look at the taxonomy on NCBI BLAST, then we'll see that the top hits are *Aspergillus cristatus* and *Aspergillus chevalieri*, both with 99.67% identity. If it's just as good a match to two different species within the same genus, it makes sense why we just have the genus-level classification *Aspergillus* here, and seems reasonable.

Try taking a look at a few more and seeing how they seem.

3.2.18 3.4. ITS Filtering resultant table

Filtering the denoised table is an important step of microbiome data analysis. You can see more details on this process in the QIIME 2 filtering tutorial.

3.2.18.1 3.4.1. Filter out rare ASVs

Based on the summary visualization created in step 2.5 above you can choose a cut-off for how frequent a variant needs to be (and optionally how many samples need to have the variant) for it to be retained. Here we will remove all ASVs that have a frequency of less than 0.1% of the mean sample depth. This cut-off excludes ASVs that are likely due to MiSeq bleed-through between runs (reported by Illumina to be 0.1% of reads). To calculate this cut-off you would identify the mean sample depth in the visualization created in step 2.5 (`deblur_table_summary.qzv`), multiply it by 0.001, and round to the nearest integer.

Once you've determined how you would like to filter your table you can do so with this command (X is a placeholder for your choice):

```
qiime feature-table filter-features \
  --i-table deblur_output/table.qza \
```

```
  --p-min-frequency X \
  --p-min-samples 1 \
  --o-filtered-table deblur_output/deblur_table_filt.qza
```

3.2.18.2 3.4.2. Filter out contaminant and unclassified ASVs

Once we have assigned taxonomy to our ASVs we can use that information to remove ASVs which are likely contaminants or noise based on the taxonomic labels. Two common contaminants in 16S sequencing data are mitochondrial and chloroplast 16S sequences, which can be removed by excluding any ASV which contains those terms in its taxonomic label. We don't really have that issue with ITS data, however. It can also be **sometimes** useful to exclude any ASV that is unclassified at the phylum level since these sequences could be noise (e.g. possible chimeric sequences). Note that if your data has not been classified against the default database you may need to change p___ to be a string that enables phylum-level assignments to be identified or simply omit that line.

In general though, it can be very informative if your sequencing reads are coming back with significant amounts of unclassified ASVs as it can indicate upstream analysis problems or indicate you are studying a poorly characterized environment where you have a good chance of identifying a lot of novel phyla. Therefore, our recommendation is to not filter out the unclassified sequences by default, but we will do so here.

```
qiime taxa filter-table \
  --i-table deblur_output/deblur_table_filt.qza \
  --i-taxonomy taxa/classification.qza \
  --p-include p__ \
  --o-filtered-table deblur_output/deblur_table_filt_contam.qza
```

3.2.18.3 *3.4.3. (Optional) Exclude low-depth samples*

Often certain samples will have quite low depth after these filtering steps, which can be excluded from downstream analyses since they will largely add noise. There is no single cut-off that works best for all datasets, but researchers often use minimum cut-offs within the range of 1000 to 4000 reads. You can also use a cut-off much lower than this if you want to retain all samples except those that failed entirely (e.g. depth < 50 reads).

Ideally you would choose this cut-off after visualizing rarefaction curves to determine at what read depth the richness of your samples plateaus and choose a cut-off as close to this plateau as possible while retaining sufficient sample size for your analyses. We learn more about rarefaction curves in Module 3.

3.2.18.4 3.4.4. Subset and summarize filtered table

Check output after filtering.

```
qiime feature-table summarize \
  --i-table deblur_output/deblur_table_filt_contam.qza \
  --o-visualization deblur_output/deblur_table_filt_contam_summary.qzv
```

QUESTION!

- **Question 8**: What is the minimum and maximum sequencing depth across all samples?

Copy a final table.

```
cp deblur_output/deblur_table_filt.qza deblur_output/deblur_table_final.qza
```

Once we have our final filtered table we will need to subset the QZA file containing the ASV sequences to the same set. You can exclude any removed ASVs from the sequence file with this command:

```
qiime feature-table filter-seqs \
  --i-data deblur_output/representative_sequences.qza \
  --i-table deblur_output/deblur_table_final.qza  \
  --o-filtered-data deblur_output/rep_seqs_final.qza
```

Finally, you can make a new summary of the final filtered abundance table:

```
qiime feature-table summarize \
  --i-table deblur_output/deblur_table_final.qza \
  --o-visualization deblur_output/deblur_table_final_summary.qzv
```

3.2.19 ITS Answers

Question 1: How many samples are there?

We have 20 samples. The raw data folder contains one fastq.gz file for each set of the forward and reverse sequences (labelled R1 and R2, respectively) for each of the samples (CRRxxxxxxx).

Question 2: Into what group(s) are the samples classified?

The samples we are using are classified into two different groups: below_20 or above_35. The study we're looking at had information on the age of the pregnant women, so that's what we're looking at here. We also have their exact age in these samples.

Question 3: What would happen if you ran this exact command on V4/V5-amplified sequences?

`Cutadapt` will only trim reads that match the specified primer sequence. Therefore, most reads would be discarded because we are including the `--p-discard-untrimmed` option.

Question 4: How long are our forward reads? Why are there no reverse reads in our file?

The forward read median length is 364 nucleotides. There are no reverse reads because forward and reverse reads were merged into one sequence during read joining.

Question 5: What would be a good trim length for our reads?

There is no one right answer for this question, but a trim length of 300 nucleotides will maintain most of our sequences apart from any that are really short. You could choose a different length, but this will give different answers further on. Typically, this choice is a trade-off between maintaining a longer sequence length where we're more likely to get finer-resolution taxonomic classification, or trimming more so that we retain more sequences.

Question 6: What is the mean sequencing depth per sample after denoising?

The mean sequencing depth (frequency) across all denoised samples is 62,566.6 reads.

Question 7: Which sample has the least reads?

Sample CRR1039764 has the lowest sequencing depth (21,453 reads).

Question 8: What is the minimum and maximum sequencing depth across all samples?

Question 8 answer: The final minimum sequencing depth is 192 and the maximum sequencing depth is 80,762 reads. This huge difference from the previous step (where it was actually a different sample that had the lowest depth!) would definitely indicate some problems if this were our own project. This was not a project that I was involved with, and this could just be because the ITS region is more difficult to accurately taxonomically classify than the 16S region, but I would definitely want to do some further investigating as to what is going on. We only removed ASVs that didn't have phylum-level classifications, so the first thing that I would do is look at the taxonomy.tsv file. If we open this in Excel and sort based on Column B, we can see that 114/961 total ASVs

within our samples have only been classified as Fungi (these are the ones that we would have filtered out). If we just have a quick search of a few of these ASVs in the deblur_table_summary.qzv Feature Detail tab, then we'll see that e.g. `87121182b8f0d093e937d915b2b58b22` is present in 13 samples at a total frequency of 15,889 reads. Another one, `1086009b09b17b9364cb1ae984a552e1` is similar; it is present in 18 samples at a total frequency of 35,799 reads. Based on this, I don't think I'd want to filter these out. So unlike with the other amplicons, I think I'd probably use the unfiltered table here.

Chapter 4

Module 3: Microbiome statistics and visualizations

4.1 Lecture

4.2 Lab

Material by Juan Santana

This tutorial is part of the 2025 Canadian Bioinformatics Workshops Beginner Microbiome Analysis (Vancouver, BC, May 26-27). It is based on the Amplicon SOP v2 available on the Microbiome Helper repository and previous workshops designed by Diana Haider, Robert Beiko and Monica Alvaro Fuss.

4.2.1 Table of Contents

7. Heatmaps
8. Exporting data from QIIME 2 for use in other software
9. Considerations for 18S and ITS data

4.2.2 Introduction

Modules 2 and 3 offer a step-by-step walkthrough of an end-to-end pipeline for analyzing high-throughput marker gene data using the command-line interface. Commonly used marker genes in microbiome research include 16S ribosomal RNA (rRNA) for prokaryotes, 18S rRNA for eukaryotes, and the internal transcribed spacer (ITS) region for fungi.

In this tutorial, we primarily focus on the same 16S rRNA dataset derived from wild blueberry (*Vaccinium angustifolium*) soil communities from natural and managed habitats we processed in Module 2. At the end of the tutorial, we also provide guidance on processing the 18S rRNA dataset from plastics incubated in a coastal marine environment (plastisphere), as well as the ITS dataset from the gut mycobiome (i.e., fungi) during pregnancy.

You can learn more about these studies in the following publications:

- Variation in Bacterial and Eukaryotic Communities Associated with Natural and Managed Wild Blueberry Habitats
- Metagenomic Functional Shifts to Plant Induced Environmental Changes -Microbial pioneers of plastic colonisation in coastal seawaters
- Landscape of the gut mycobiome dynamics during pregnancy and its relationship with host metabolism and pregnancy health

In Module 2, we covered the fundamentals of marker gene analysis, from processing raw reads to generating a filtered feature table. In Module 3, we will explore examples of downstream analyses used to draw biological insights from these data. The workflow described is integrated into the latest release of QIIME 2 (Quantitative Insights into Microbial Ecology, version 2025.4). As introduced in Module 2, this widely used microbiome bioinformatics platform is built around user-developed software packages called plugins, which operate on QIIME 2 artifact files (with the .qza extension). Documentation for these plugins—including tutorials and additional resources—can be found in the QIIME 2 user documentation. QIIME 2 also offers interpretable visualizations, which can be viewed by opening .qzv files in QIIME2 View

Let's set up our Module 3 directory inside `workspace` and create a symlink to the data we will be using (from Module 2).

```
cd ~/workspace
mkdir bmb_module3 bmb_module3/16S_analysis
cd bmb_module3/16S_analysis
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/deblur_output .
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/taxa .
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/Blueberry_16S_metadata.tsv .
```

If you deactivated your QIIME2 environment, reactivate it with the command below.

```
conda activate qiime2-amplicon-2025.4
```

When you are finished this tutorial you can deactivate the conda environment using:

```
conda deactivate
```

Throughout this module, there are some questions aimed to help your understanding of some of the key concepts. You'll find the answers at the bottom of this page, but no one will be marking them.

4.2.3 1. Build tree with SEPP QIIME 2 plugin

SEPP (SATé-enabled Phylogenetic Placement) is a tool used to place short DNA sequences—such as 16S rRNA amplicon sequence variants (ASVs)—into an existing, high-quality reference phylogenetic tree. This is particularly helpful when you are working with microbiome data and want to infer evolutionary relationships more accurately. We will use QIIME 2's `q2-fragment-insertion` plugin to place ASVs derived from our **16S data** into a reference phylogenetic tree using the command below.

```
qiime fragment-insertion sepp
  --i-representative-sequences deblur_output/rep_seqs_final.qza \
  --i-reference-database ~/CourseData/MIC_data/bmb_module3/16S_analysis/sepp-refs-gg-13-8.qza \
  --o-tree asvs-tree.qza \
  --o-placements insertion-placements.qza \
  --p-threads 4
```

Due to memory constraints, you can instead copy the output into your folder with the following command:

```
ln -s ~/CourseData/MIC_data/bmb_module3/output/16S_analysis/asvs-tree.qza .
ln -s ~/CourseData/MIC_data/bmb_module3/output/16S_analysis/insertion-placements.qza .
```

High-quality reference phylogenetic trees can be downloaded from QIIME2's data resources. However, it is important to ensure that the reference tree used for sequence placement matches the same reference database used for taxonomic classification (e.g., Greengenes or SILVA). In our workflow, we use `sepp-refs-gg-13-8.qza`, as our sequences were classified using the Greengenes database. Alternatively, custom reference files can be specified for placing other types of amplicons. However, for marker genes such as **18S and ITS**, the recommended approach is to construct a *de novo* phylogenetic tree, as outlined in section 9. Considerations for 18S and ITS data and and further detailed in the Microbiome Helper repository.

4.2.4 2. Generate rarefaction curves

After inferring ASVs from our sequence reads, we can generate **rarefaction curves** to evaluate whether our sequencing depth was sufficient to capture most of the microbial diversity in each sample. The command below will generate these plots. The `--p-max-depth` parameter sets the maximum sequencing depth to sample across. Inspect the `deblur_table_final_summary.qzv` file (created in Module 2) using QIIME2 View to determine this number. For our 16S dataset, the sample with the highest number of reads contains 8,650 sequences, so we'll use that as the max depth.

```
qiime diversity alpha-rarefaction \
  --i-table deblur_output/deblur_table_final.qza \
  --p-max-depth 8650 \
  --p-steps 20 \
  --i-phylogeny asvs-tree.qza \
  --o-visualization rarefaction_curves.qzv
```

This will produce a `.qzv` file that can be opened in QIIME 2 View to explore the number of observed features (as well as two alpha diversity metrics: Shannon index and Faith's phylogenetic diversity) across rarefied sequencing depths for each sample.

Question 1: What is a good rarefaction depth for diversity analysis?

4.2.5 3. Calculating diversity metrics and generating ordination plots

QIIME 2 can calculate commonly used alpha and beta diversity metrics—including Faith's Phylogenetic Diversity, Shannon diversity, and UniFrac distances—using a single command. Ordination plots (e.g., PCoA plots based on weighted UniFrac distances) are also generated automatically. Before calculating these metrics, QIIME 2 rarefies all samples to the same sequencing depth to ensure fair comparisons. The `--p-sampling-depth` parameter defines this cutoff. Any samples with fewer reads than this threshold will be excluded from the analysis. Based on the rarefaction curves we generated earlier, we determined that the lowest reasonable sequencing depth across all samples is **3,432** reads. Using this value ensures we retain all samples while still capturing most of the microbial diversity.

```
qiime diversity core-metrics-phylogenetic \
  --i-table deblur_output/deblur_table_final.qza \
  --i-phylogeny asvs-tree.qza \
  --p-sampling-depth 3432  \
  --m-metadata-file Blueberry_16S_metadata.tsv \
  --p-n-jobs-or-threads 4 \
  --output-dir diversity
```

This command will output a `diversity/` folder containing:

Alpha diversity vectors (e.g., Shannon, Faith's PD) Beta diversity distance matrices (e.g., UniFrac, Bray-Curtis) *Rarefied feature table* PCoA ordination plots (`emperor.qzv` files)

You can explore these ordination plots to visually assess whether microbial communities cluster based on sample groupings (e.g., natural vs. managed habitats).However, to statistically and visually compare alpha diversity across sample categories (e.g., habitat type), you have to generate boxplots and perform statistical tests using the following command:

```
qiime diversity alpha-group-significance \
  --i-alpha-diversity diversity/shannon_vector.qza \
  --m-metadata-file Blueberry_16S_metadata.tsv \
  --o-visualization diversity/shannon_compare_groups.qzv
```

Hint:To analyze other alpha diversity metrics, simply replace ***`shannon_vector.qza`*** *with another file from the* ***`diversity/`*** *directory. To view available metrics, run:*

```
ls diversity/*_vector.qza
```

Note that you can also export this or any other diversity metric file (ending in *.qza*) using `qiime tools export` and analyze them with a different program (e.g., R, Python, Excel).

```
qiime tools export \
  --input-path diversity/shannon_vector.qza \
  --output-path exported_shannon
```

This will produce a file called `alpha-diversity.tsv` inside the `exported_shannon/` folder.

Question 2: are there any significant differences in alpha diversity between any of our metadata categories?

Question 3: which metadata category appears to provide more separation in the beta diversity PCoA plots?

If you want to run a PERMANOVA test to assess whether beta diversity differs significantly between groups, you can use the following command. Be sure to update the command with the specific beta diversity metric you're analyzing (e.g., Bray-Curtis, UniFrac) and replace category with the appropriate column name from your metadata file that defines your groups of interest.

```
qiime diversity beta-group-significance \
  --i-distance-matrix diversity/bray_curtis_distance_matrix.qza \
  --m-metadata-file Blueberry_16S_metadata.tsv \
  --m-metadata-column Category \
  --o-visualization diversity/bray_curtis_compare_groups_category.qzv
```

This will produce a `.qzv` file containing PERMANOVA results and boxplots for visualizing distances within and between groups.

Question 4: what do you mean I'm getting an error message? It should be working fine, it worked last time! I can't see what I'm doing wro... oooooh there we go, I see now :)

4.2.6 4. Generate stacked bar chart of taxa relative abundances

A useful way to visualize the composition of microbial communities across samples is through interactive stacked bar charts of taxonomic relative abundances.

This type of plot allows you to explore differences in microbial composition at various taxonomic levels (e.g., phylum, genus) across experimental groups. You can generate the visualization using the command below.

```
qiime taxa barplot \
  --i-table deblur_output/deblur_table_final.qza \
  --i-taxonomy taxa/classification.qza \
  --m-metadata-file Blueberry_16S_metadata.tsv \
  --o-visualization taxa_barplot.qzv
```

The interactive `.qzv` file generated with this command can be inspected with the QIIME2 viewer. In the plot you can explore dominant taxa across samples or groups, navigate between taxonomic levels (kingdom to species) and compare relative abundances between metadata-defined sample groups.

Hint: your metadata file includes any grouping columns you want to use to color or group the samples in the visualization (e.g., treatment, site, time point).

Question 5: can you identify any patterns between the metadata groups?

4.2.7 5. Identifying differentially abundant features with ANCOM

ANCOM (Analysis of Composition of Microbiomes) is a statistical method used to identify taxa (or features) that differ significantly in relative abundance between sample groups. ANCOM is robust to the compositional nature of microbiome data and does not assume a particular distribution for the features. However, ANCOM requires that all features in the table be non-zero, so a pseudocount (commonly set to 1) must first be added to avoid issues with zeros in the count matrix. The command below creates a new feature table where a count of 1 is added to all zero entries.

```
qiime composition add-pseudocount \
  --i-table deblur_output/deblur_table_final.qza \
  --p-pseudocount 1 \
  --o-composition-table deblur_output/deblur_table_final_pseudocount.qza
```

We then run ANCOM using the command below. `--m-metadata-column` defines the groups you'd like to compare (e.g., Treatment, Site, Timepoint, etc.) according to the columns in your metadata file.

```
qiime composition ancom \
  --i-table deblur_output/deblur_table_final_pseudocount.qza \
  --m-metadata-file Blueberry_16S_metadata.tsv \
  --m-metadata-column category \
  --output-dir ancom_output
```

The results from ANCOM will be saved in the `ancom_output/` directory.The key file is ancom.qzv, which can be viewed interactively at QIIME 2 View. The output highlights features (e.g., ASVs, OTUs, or taxa) that are differentially abundant between the specified groups based on their W statistic (number of significant pairwise comparisons).

Hint: ANCOM is sensitive to sparsity in the data. Consider filtering low-abundance or low-frequency features before running ANCOM to improve detection power and reduce false positives. Alternatively, we can use ANCOM-BC (see below)

ANCOM-BC addresses limitations in the original ANCOM method by incorporating bias correction and estimating absolute abundances from compositional data. It performs better under sparsity and unequal library sizes. Note that the `--p-formula` parameter lets you specify covariates or design models (e.g., Treatment + Timepoint), in our workflow we only test for significant abundances based on the "category" variable. In addition, we supply the original raw counts instead of the compositionally transformed table.

```
qiime composition ancombc \
  --i-table deblur_output/deblur_table_final.qza \
  --m-metadata-file Blueberry_16S_metadata.tsv \
  --p-formula category \
  --output-dir ancombc_output/
```

While `qiime composition ancombc` produces a `.qza` file of differentials (log-fold changes and statistics), QIIME 2 does not currently provide built-in commands to visualize this artifact as a `.qzv`. Therefore, we use another command to generate visualizations.

```
qiime composition da-barplot \
  --i-data ancombc_output/differentials.qza \
  --p-significance-threshold 0.05 \
  --p-effect-size-threshold 1.0 \
  --o-visualization ancombc_output/ancombc_results.qzv
```

The resultant `.qzv` barplot will show you features significantly enriched in one or another condition.

Question 6: Does ANCOM identify any differentially abundant taxa between any of the metadata groups? If so, which one(s)? Is the output from ANCOM-BC any different?

4.2.8 6. Identifying shared features across groups of samples

The core microbiome refers to the set of microbial taxa consistently found across a group of samples or within specific treatment groups. Identifying core taxa helps understand the stable and potentially important microbes in your system. In QIIME2 we can filter features by prevalence across samples using the command below. `-p-min-fraction 0.8` means features must be present in 80% of samples to be considered "core".

```
qiime feature-table core-features \
  --i-table deblur_output/deblur_table_final.qza \
  --p-min-fraction 0.8 \
  --o-visualization core_features.qzv
```

This will output a feature table summary with the fraction of the ASVs shared across all samples. The core microbiome plugin `qiime feature-table core-features` doesn't have a an option to group samples by treatment, so, if you want to identify the shared ASVs across samples from a group (e.g. managed) you need to subset your feature table first by treatment group using `qiime feature-table filter-samples`.

```
qiime feature-table filter-samples \
  --i-table deblur_output/deblur_table_final.qza \
  --m-metadata-file Blueberry_16S_metadata.tsv \
  --p-where "[category]='Managed'" \
  --o-filtered-table table_managed.qza
```

We can then run `qiime feature-table core-features` on the filtered table `table_managed.qza` to identify shared ASVs across "Managed" samples.

4.2.9 7. Heatmaps

Heatmaps visualize abundance patterns of taxa across samples or groups, allowing easy spotting of taxa that vary in abundance or co-occur across conditions. We can generate these heatmaps using `qiime feature-table heatmap`. However, raw feature tables at the ASV level often contain thousands of features,

which can make heatmaps visually overwhelming and difficult to interpret. To create a clearer and more interpretable heatmap, you can collapse your feature table to a higher taxonomic rank, such as genus or family, before visualization, using the command below. `--p-level` sets the taxonomic rank, here we will collapse at the Class level.

```
qiime taxa collapse \
  --i-table deblur_output/deblur_table_final.qza \
  --i-taxonomy taxa/classification.qza \
  --p-level 3 \
  --o-collapsed-table deblur_output/table_class.qza
```

Now we generate the heatmap using the collapsed class-level table.

```
qiime feature-table heatmap \
  --i-table deblur_output/table_class.qza \
  --m-sample-metadata-file Blueberry_16S_metadata.tsv \
  --m-sample-metadata-column category \
  --p-metric braycurtis \
  --p-color-scheme RdYlBu \
  --o-visualization heatmap_class.qzv
```

Open the resulting `heatmap_class.qzv` in QIIME 2 View to interactively explore taxa abundance patterns across samples.

4.2.10 8. Exporting data from QIIME 2 for use in other software

While QIIME 2 offers a wide range of tools for microbial community analysis, you may want to perform additional custom analyses in software like R, Python, or MATLAB. To do so, you'll need to export your QIIME 2 artifacts into formats that are compatible with these platforms.

Representative sequences (i.e., ASVs) are stored in a `.qza` artifact that contains the DNA sequences used in downstream analysis. To export them into a standard FASTA format, use the following command:

```
qiime tools export \
   --input-path deblur_output/rep_seqs_final.qza \
   --output-path deblur_output_exported
```

Your sequences will be saved as `dna-sequences.fasta` inside the `deblur_output_16S_exported` folder. This file can be read by any downstream tool that accepts FASTA files.

BIOM (Biological Observation Matrix) is a standardized format for representing feature tables, typically containing: -Rows = features (e.g., ASVs, OTUs, taxa) -Columns = samples -Cells = abundance values (counts, relative abundances, etc.) -Optional metadata (taxonomy, sample info)

BIOM files are widely used in microbiome analysis and supported by R packages like phyloseq, microbiome, and tools in Python such as scikit-bio and biom-format.To export a BIOM table (with taxonomy added as metadata) you can use the commands below.

```
#First we fix taxonomy header with sed (required for biom add-metadata)
sed -i -e '1 s/Feature/#Feature/' -e '1 s/Taxon/taxonomy/' taxa/taxonomy.tsv

#Second we export the raw feature table and create the biom table
qiime tools export \
   --input-path deblur_output/deblur_table_final.qza \
   --output-path deblur_output_exported

#Third we add taxonomy metadata to the BIOM file
biom add-metadata \
   -i deblur_output_exported/feature-table.biom \
   -o deblur_output_exported/feature-table_w_tax.biom \
   --observation-metadata-fp taxa/taxonomy.tsv \
   --sc-separated taxonomy
#Last we convert the BIOM file to TSV format (tab-separated values)
biom convert \
   -i deblur_output_exported/feature-table_w_tax.biom \
   -o deblur_output_exported/feature-table_w_tax.txt \
   --to-tsv \
   --header-key taxonomy
```

This will give you a plain-text feature table (feature-table_w_tax.txt) with taxonomy annotations in the header row, which is especially useful for tools like R (phyloseq), Excel, or even manual inspection.

To export the tree of your ASVs in a `.nwk` format, use the command below.

```
qiime tools export \
  --input-path asvs-tree.qza \
  --output-path deblur_output_exported
```

4.2.11 9. Considerations for 18S and ITS data

Most of the analyses described in this tutorial apply equally to 18S and ITS datasets, **with one key exception**: building a phylogenetic tree. Unlike 16S data, 18S and ITS amplicons generally lack a universally accepted reference phylogeny. Therefore, we must generate *de novo* phylogenetic trees.

To start, create symbolic links to the relevant data folders for your 18S and ITS datasets:

```
cd ~/workspace/bmb_module3/
mkdir 18S_analysis
cd 18S_analysis
ln -s ~/CourseData/MIC_data/bmb_module2/output/18S_analysis/deblur_output .
ln -s ~/CourseData/MIC_data/bmb_module2/output/18S_analysis/taxa .
ln -s ~/CourseData/MIC_data/bmb_module2/output/18S_analysis/Plastisphere_18S_metadata.t

cd ~/workspace/bmb_module3/
mkdir ITS_analysis
cd ITS_analysis
ln -s ~/CourseData/MIC_data/bmb_module2/output/ITS_analysis/deblur_output .
ln -s ~/CourseData/MIC_data/bmb_module2/output/ITS_analysis/taxa .
ln -s ~/CourseData/MIC_data/bmb_module2/output/ITS_analysis/pregnancy_ITS_metadata.tsv
```

4.2.11.1 Building a de novo tree for 18S

First, we'll need to make a *de novo* multiple-sequence alignment of the ASVs using MAFFT.

```
cd ~/workspace/bmb_module3/18S_analysis
mkdir tree_out

qiime alignment mafft --i-sequences deblur_output/rep_seqs_final.qza \
                      --p-n-threads 4 \
                      --o-alignment tree_out/rep_seqs_final_aligned.qza
```

Variable positions in the alignment need to be masked before FastTree is run, which can be done with the command below:

```
qiime alignment mask --i-alignment  tree_out/rep_seqs_final_aligned.qza \
                     --o-masked-alignment  tree_out/rep_seqs_final_aligned_masked.qza
```

We finally run FastTree on this masked multiple-sequence alignment to make our tree with the command below:

```
qiime phylogeny fasttree --i-alignment tree_out/rep_seqs_final_aligned_masked.qza \
                         --p-n-threads 4 \
                         --o-tree tree_out/rep_seqs_final_aligned_masked_tree
```

FastTree returns an unrooted tree. One basic way to add a root to a tree is to add it at the midpoint of the largest tip-to-tip distance in the tree, which is done with this command:

```
qiime phylogeny midpoint-root --i-tree tree_out/rep_seqs_final_aligned_masked_tree.qza \
                              --o-rooted-tree tree_out/rep_seqs_final_aligned_masked_tree_rooted.
```

Let's rename the tree and place it in our analysis folder.

```
cp tree_out/rep_seqs_final_aligned_masked_tree_rooted.qza asvs-tree.qza
```

Due to memory or time constraints, you may opt to use a tree that has already been computed. Use the following command to create symbolic links to precomputed trees

```
cd ~/workspace/bmb_module3/18S_analysis
ln -s ~/CourseData/MIC_data/bmb_module3/output/18S_analysis/asvs-tree.qza .

cd ~/workspace/bmb_module3/ITS_analysis
ln -s ~/CourseData/MIC_data/bmb_module3/output/ITS_analysis/asvs-tree.qza .
```

4.2.11.2 Building the ITS tree

Repeat the same commands shown above, but make sure you use the `deblur_output` from the ITS analysis. This will generate a *de novo* tree for your ITS dataset.

Once you have generated or linked the *de novo* trees for your 18S and ITS data, you can proceed with all downstream analyses — including diversity metrics, ordinations, and statistical testing — just as you would with 16S data.

4.2.12 Answers

Question 1: What is a good rarefaction depth for diversity analysis?

A cut-off of 4,000 reads will be sufficient: the curve plateaus around this depth and we won't exclude any samples.

Question 2: are there any significant differences in alpha diversity between any of our metadata categories?

There are significant differences in richness and phylogenetic diversity between forest and managed environment samples. There are no significant differences between bulk and rhizosphere soil.

Question 3: which metadata category appears to provide more separation in the beta diversity PCoA plots?

This is hard to say just by looking at the Emperor plots provided by QIIME2, but the forest/managed category appears to exhibit more distinct separation. The PERMANOVA test from the `beta-group-significance` command shows this as well.

Question 4: what do you mean I'm getting an error message? It should be working fine, it worked last time! I can't see what I'm doing wro... oooooh there we go, I see now :)

There is a typo in `--m-metadata-column Category`: The "C" should be lowercase, matching the column header in your metadata file. ;)

Question 5: can you identify any patterns between the metadata groups?

Because stacked barcharts are limited in their analytical capabilities, it is hard to discern anything except very obvious patterns.

Question 6: Does ANCOM identify any differentially abundant taxa between forest and managed environments?

One ASVs is identified as differentially abundant between forest and managed environments using ANCOM. You can look up the identity of each ASV in the taxonomy.tsv file. With ANCOM-BC many more ASVs are identified as diferentially abundant.

Chapter 5

Module 4 Lab: Functional prediction and additional analyses

This tutorial is part of the 2025 CBW Beginner Microbiome Analysis (held in Vancouver, BC, May 26-27).

Authors: Robyn Wright and Ryan McLaughlin

5.1 Lecture

5.2 Lab

5.2.1 Introduction

In this module, we'll be taking a look at three of the major ways in which we use amplicon sequencing data to learn about microbial communities; (1) identifying taxa that are significantly differentially abundant between different sample groups, (2) predicting the functional capabilities of these communities based on the taxa that are present, and (3) constructing and visualising a co-occurrence network.

There is likely more in this module than you can get through in the time provided, so we suggest that you choose which of these is of most interest to you, and start there. You can always come back and do some of the others if you are finished in time. PICRUSt2 is really only appropriate for 16S data, so if you used one of the other datasets then you can either copy

across the 16S output or do one of the other sections of this lab. Ask one of us for help if you need it! You can read some information on each of these sections below before deciding which to run.

Throughout this module, there are some questions aimed to help your understanding of some of the key concepts. As with the previous modules, you'll find the answers at the bottom of this page, but no one will be marking them.

5.2.1.1 Differential abundance testing with MaAsLin3 and ALDEx2

Tens (if not more) of tools have been used for differential abundance testing of microbiome samples - identifying taxa that differ in abundance between sample groups (for example, treatment versus control) or with variables of interest (for example, correlations with pH or blood metabolite measurements). While these tools/methods have often been used interchangeably in the literature, there are large differences in the performance of different tools even on the same sample group. There have now been several large scale comparisons of different tools (*e.g.,* Calgaro *et al.*, Thorsen *et al.*), and our lab also carried one out (published here). While these studies have typically found that different methods lead to different biases in the results, as there is no one single *best* method for determining differentially abundant taxa, the best practice that we have for overcoming these issues currently is to use multiple differential abundance tests, report these clearly, and focus on the taxa that are identified by multiple tests.

For these purposes, we have chosen to use two tools today, MaAsLin3 (preprint/wiki) and ALDEx2 (paper/vignette). In our comparison of different tools, we found that ALDEx2 was one of the tools that controlled the false discovery rate well, but this came at the cost of reduced sensitivity, while MaAsLin2 had much better sensitivity than ALDEx2, but this came at the cost of reduced specificity. It is worth noting, however, that MaAsLin3 is a new version and Jacob Nearing - a previous PhD student in the Langille lab who was one of the first authors on our DA tool comparison - was heavily involved in its development. In some of the recent studies from our lab, we have then chosen to focus on only the taxa that are identified by at least two of these tools (*e.g.,* this paper or this paper).

In this tutorial, we'll be running both of these tools on the output from Module 2, and then we'll combine the results and look at the taxa that are identified. We can then apply this to the PICRUSt2 output, too.

5.2.1.2 Functional prediction with PICRUSt2

PICRUSt2 is a tool that predicts the functional capacity of a microbial community based on the taxa that are present in amplicon sequencing data. The first iteration of PICRUSt (**P**hylogenetic **I**nvestigation of **C**ommunities by

Reconstruction of **U**nobserved **St**ates) was developed by Morgan (and many other collaborators) during his postdoc. This was expanded and improved upon to make PICRUSt2 by a previous PhD student in the lab, Gavin Douglas (currently a postdoc at North Carolina State University), and Morgan and I (Robyn) have recently updated the database used by PICRUSt2 to improve the functional predictions.

There are several other tools that have also been developed for this purpose, *e.g.*, Tax4Fun2, Piphillin and PanFP. I don't personally have experience including these, so we're going to be focusing on PICRUSt2, however, it's always important to understand the strengths and weaknesses of different bioinformatic tools before choosing which one to use for your own research.

You can see full information on PICRUSt2 here, but it includes several key steps (the new database changes these slightly!): 1. Placement of sequences into reference phylogenies for bacteria and archaea 2. Hidden-state prediction to get 16S copy numbers and Nearest Sequenced Taxon Indices (NSTI) per genome 3. Determine the best domain for each sequence (whether they fit best in the bacterial or archaeal reference phylogeny) 4. Hidden-state prediction to get trait abundances (usually KOs or EC numbers) per predicted genome 5. Combine bacterial and archaeal files from hidden-state prediction 6. Predict metagenomes for each sample for each trait 7. Infer MetaCyc pathway abundances and coverages (using EC numbers)

These steps can be run separately, but are all wrapped together into a single command for what we'll be running today.

5.2.1.3 Constructing and visualising a co-occurrence network using `SPIEC-EASI` (Bonus: we run an indicator species analysis too!!)

Understanding how microbial taxa interact with one another in an environment is a fundamental question in microbial ecology. Co-occurrence network analysis attempts to capture these interactions by estimating associations between taxa across samples. However, it's important to be cautious here: naive approaches like correlation matrices (e.g., Pearson or Spearman) are prone to false positives due to the compositional nature of microbiome data—where only relative abundances are measured, not absolute counts. This introduces spurious correlations that don't reflect real biological interactions.

The Sparse InversE Covariance Estimation for Ecological Association Inference (SPIEC-EASI) method was developed to help with these limitations. Rather than relying on correlation, SPIEC-EASI estimates the underlying network by inferring the inverse covariance matrix under the assumption of sparsity—meaning most taxa are not directly interacting. It supports two main models: the neighborhood selection method (mb, or Meinshausen-Bühlmann) and the graphical lasso (glasso). Both use regularization to identify a stable and sparse

set of direct associations among taxa. (*NOTE: don't use glasso for today, you need more samples than taxa for glasso to have power.*)

In our analysis today, we'll use the R implementation of SPIEC-EASI to construct a microbial association network based on a filtered and prevalence-adjusted version of the ASV table from Module 2. Importantly, SPIEC-EASI models are not directional and do not imply causation, but they offer a more robust way to explore ecological structure and *hypothesize* about microbial interactions.

We'll begin by identifying indicator taxa using the `indicspecies` package and the `multipatt()` function, which detects taxa that are significantly associated with particular treatment groups. These indicator taxa are labeled accordingly and carried through the rest of the analysis to visualize their role in the microbial network.

Next, we filter the ASV table to remove rare or low-abundance taxa based on minimum prevalence (30% of samples) and relative abundance (0.05%), ensuring we focus on taxa with sufficient statistical power for robust network inference. `SpiecEasi` is then run using the neighborhood selection (`mb`) method with cross-validation (`pulsar`) to determine the optimal sparsity level. The resulting adjacency matrix is converted into an `igraph` object, and we replace default node names with the original ASV IDs to maintain interpretability. The `phyloseq` object is also pruned to match the taxa used in the network, and we annotate the `tax_table` with indicator group assignments where available.

Finally, the network is visualized using two separate layouts: one colored by indicator status (e.g., Forest, Managed, or Not Significant), and the other by a taxonomic rank (ta3). Both are saved as standalone PDF plots and also combined into a side-by-side panel figure for direct comparison. These visualizations use the Fruchterman-Reingold layout for reproducible positioning and are saved using `pdf()` output with customized legends for clarity. This approach allows us to not only explore the overall structure of microbial associations but also see how treatment-associated or taxonomically distinct taxa are embedded within the broader ecological network.

5.2.2 4.1 Differential abundance analysis using MaAsLin3 and ALDEx2

You can use whichever amplicon you like for this! We've given instructions for 16S, but you can hopefully work out how to modify these commands for 18S or ITS :).

5.2.2.1 4.1.1 Prepare data from the end of Module 2

First, we'll create a new directory for whichever dataset we want to use and we'll copy the output from the end of Module 2 into this.

16S:

```
cd ~/workspace
mkdir bmb_module4 bmb_module4/16S_analysis
cd bmb_module4/16S_analysis
ln -s ~/CourseData/MIC_data/bmb_module3/output/16S_analysis/deblur_output_exported/ .
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/Blueberry_16S_metadata.tsv .
```

18S:

```
cd ~/workspace
mkdir bmb_module4 bmb_module4/18S_analysis
cd bmb_module4/18S_analysis
ln -s ~/CourseData/MIC_data/bmb_module3/output/18S_analysis/deblur_output_exported/ .
ln -s ~/CourseData/MIC_data/bmb_module2/output/18S_analysis/Plastisphere_18S_metadata.tsv .
```

ITS:

```
cd ~/workspace
mkdir bmb_module4 bmb_module4/ITS_analysis
cd bmb_module4/ITS_analysis
ln -s ~/CourseData/MIC_data/bmb_module3/output/ITS_analysis/deblur_output_exported/ .
ln -s ~/CourseData/MIC_data/bmb_module2/output/ITS_analysis/pregnancy_ITS_metadata.tsv .
```

5.2.3 4.1.2 Read output into R/Phyloseq

Next, we'll go to RStudio on the server. Go to your browser and navigate to:

```
http://##.uhn-hpc.ca:8080/
```

(where ## is the student number that you have been given).

Enter "ubuntu" as your username and the password that have been given. It takes a little while to load up the first time but should be quicker on subsequent times!

Everything that you're entering in here should be entered into the "Console" part of the page. This is like the Terminal that we've been using on the Server, but it uses the R programming language. Most statistical analyses in bioinformatics will use either R or Python - many bioinformaticians have a preference for one or the other, but most use both. In my opinion, each has different strengths and weaknesses - R has many built in programs and functions, while Python is much more customisable but perhaps has a steeper learning curve. The syntax varies slightly for both (and both are different from the Terminal

that we have been using, which is the bash programming language), but there are also many similarities. RStudio is an Integrated Developing Environment (IDE) for R and Python - it is an application that provides further functionality than the R programming language itself.

5.2.3.1 Importing packages

First of all, we'll import the packages that we'll be using today:

```
library(phyloseq)
library(maaslin3)
library(microbiome)
library(ALDEx2)
library(ggplot2)
library(tidyr)
library(stringr)
library(vegan)
library(stats)
library(SpiecEasi)
library(indicspecies)
library(igraph)
```

You'll notice that in each of these, we use the `library()` command, and inside the brackets we give the name of the package. We've already installed these packages for you, but in most cases these could be installed by *e.g.,* `install.packages("tidyr")`. In some cases, they might be installed using another package, BiocManager, *e.g.,* `BiocManager::install("ALDEx2")`. If you're unsure, you can always google how to install a package and you can usually find the code to copy and paste, *e.g.,* "r install aldex2". You could also try typing `??aldex2` into the console and see what comes up. It should give you some information about that package.

If you've used the 18S or the ITS datasets, I'm assuming that you'd like to challenge yourself a little more and I've made this code and explanation for 16S, but you'll need to figure out some of the changes to make for 18S/ITS. If you're struggling, you can find the full working code in the Answers, but without any of the explanation that we have up here.

5.2.3.1.1 Setting variables Next, we'll set the name of the folder where our results are saved:

```
folder = '/home/ubuntu/workspace/bmb_module4/16S_analysis/'
```

Note that if you used 18S or ITS then you'll need to modify this!

Here, we have defined a variable called "folder" - we always need to be careful that we don't save over these variables by calling something else the same thing, but you can always check these in the "Environment" area in the top right where you should see that "folder" now exists. We could have called this anything, but it's always helpful to name our variables something that is simple and explains what they are, incase someone else is using our code. This is a string, as it is inside the punctuation marks `' '` or `" "`, and this is used in programming to show that we are representing text rather than numbers. These variables can be really useful to avoid having to type in the same information multiple times, which always increases the chances that we'll make a mistake!

You can see how these would be different by running the following:

```
random_string = "12345"
random_number = 12345
```

You should see how these are different by how they show in the "Environment" area in the top right.

Now that we've explained the basics, we can read in some of our data.

5.2.3.1.2 Read in the feature table First, we'll read in the feature table. Note that we're calling it `asv_table`:

```
asv_table <- read.csv(paste(folder, "deblur_output_exported/feature-table_w_tax.txt", sep=""), se
```

Here we're combining a few different things. We're using the `read.csv` function - functions are modules of code that accomplish a specific task. We can write them ourselves, they are what is contained in the packages that we imported at the start, or there are several that are built in to R. If you want more information about what a function does as well as the input that it expects and the output that it gives, you can type in `?read.csv`.

We're also using the `paste` function to add together the full file path of the feature table. We're giving it the `folder` variable that we already defined, as well as the additional folder and file name (both as positional arguments), and finally the `sep=""` named argument - this tells the `paste` function that there should be no spaces or other punctuation between the two parts that are being pasted together. Try running just this part and see what the result is: `paste(folder, "exports_filtered/feature-table.txt", sep="")` - note that as we haven't given this a name, it is not saved as a variable.

Next, we've given the `sep='\t'` variable - this time, we're telling the `read.csv` function that the file that we're importing is tab-delimited, then we've told it

to skip the first line of the file `skip=1`, and that there is a header `header=T` (where T is the same as TRUE). We've skipped the first line because this just reads "# Constructed from biom file" and is not actually data.

Question 1: Type in `asv_table` or look at it in the "Environment" area in the top right. How many rows are there? Does this make sense to you?

5.2.3.1.3 Manipulate the feature table We often find in programming that things are not quite in the format that is expected by the packages that we're using, so we'll make a few modifications to `asv_table`:

```
asv_table_num = data.matrix(asv_table[,2:11]) #convert the ASV table to a numeric matri
rownames(asv_table_num) = asv_table[,1] #give the matrix row names
```

You can see here that I've added `#` after the code and have written what each line does. You can use `#` to make comments through your code documents, to explain what each line is doing. Here you should also notice that we didn't save over the previous `asv_table`, but created a new one called `asv_table_num`. Now, we've just told R that it should be expecting numberic data within this table, and then we gave it the same row names as our previous `asv_table` object.

Next, take a quick look at your sample names within asv_table_num.

Question 2: Are they the same as you remember - any different punctuation?

This is kind of annoying because it means that the sample names won't match the metadata table, so we'll change those now.

```
colnames(asv_table_num) = gsub("B.", "B-", colnames(asv_table_num))
```

5.2.3.1.4 Get the taxonomy information You may have noticed that the last column in the initial `asv_table` was our taxonomy information. We'll get that first:

```
taxonomy <- data.frame(asv_table[,"taxonomy"])
rownames(taxonomy) <- asv_table[,1]
colnames(taxonomy) = c("taxonomy")
```

If you print this out by typing `taxonomy` or by clicking on this in the "Environment" area, you'll see that we currently only have a single column that contains all of the taxonomy information.

We want to split this so that we have columns for each of Domain, Phylum, Class, Order, Family, Genus and Species. Luckily, there is already a function that we can use for this:

```
taxonomy_split <- separate(data = taxonomy, col = taxonomy, into = c("Domain", "Phylum", "Class",
```

You'll see that we tell the `separate` function that the data it should use is in the `taxonomy` table, the `col` (column) is called "taxonomy". The "into" named argument takes a list as input, which is defined in R with the `c()`, and the `taxonomy` column should be split into a new column each time the `;` symbol is found. You'll see a warning after running this line, but it still ran so that's fine - it's just telling us that there weren't values for all of Domain/Phylum/Class/Order/Family/Genus/Species for every ASV, which makes sense as not all of them will have a species-level classification.

Again, you can take a look at the results in the "Environment" area.

5.2.3.1.5 Read in the metadata Now we'll read in the metadata. I won't go through each stage step-by-step as hopefully you're getting the idea by now, but I've still added comments to each of the rows:

```
metadata <- read.csv(paste(folder, "Blueberry_16S_metadata.tsv", sep=""), sep='\t')
samples = data.frame(metadata[,2], stringsAsFactors = FALSE) #convert this to a data frame - agai
rownames(samples) = metadata[,1] #add the sample names as row names
colnames(samples) = c('treatment') #and add a column name that makes sense. If you have extra var
```

5.2.3.1.6 Combine these into a phyloseq object Phyloseq is a really useful package that contains many useful functions for analysing microbiome data. While it can be a bit fiddly to get our data into the format that it is expecting (most good packages should give you examples of how data should be formatted to go into them, although it does take a lot of practice to get good at quickly working out how this is different from what you have!), once that it is in this format, the analyses are then very easy to perform.

```
ASV = otu_table(asv_table_num, taxa_are_rows = TRUE) #convert asv_table_num to an otu_table
TAX = tax_table(taxonomy_split) #convert taxonomy_split to a tax_table
taxa_names(TAX) <- rownames(taxonomy_split) #add names to TAX/the tax_table
SAMPLE = sample_data(samples) #convert samples to a sample_data
physeq = phyloseq(ASV, TAX, SAMPLE) #combine these all to make a phyloseq object
physeq #print this out to see what a phyloseq object looks like
```

Now, we have combined all of these different parts into one phyloseq object. Note that you can also import a phylogenetic tree to phyloseq, but we won't be using that for this part of the workshop. You can also get each of the individual objects back after performing manipulations, *e.g.,* `otu_table(physeq)`.

As we typically find that ASVs are not always shared across many samples (and neither would we expect them to be, if they potentially represent species or

strain level differences between taxa), we'll collapse the phyloseq object to the genus level. If you're not sure what the different taxonomy levels are called within the phyloseq object, you can always look at `tax_table(physeq)`.

So we're collapsing at the genus level, which is "ta6":

```
physeq_genus = tax_glom(physeq, taxrank="ta6")
```

If you take a look at this taxonomy table (`tax_table(physeq_genus)`), you'll notice a lot of missing information (because many environmental ASVs don't have similar taxa within reference databases!), so we can add the full taxonomy information in to the ASV names, so that this is hopefully a little more informative:

```
all_tax = paste(tax_table(physeq_genus)[,2], tax_table(physeq_genus)[,3], tax_table(phy
taxa_names(physeq_genus) = all_tax
otu_table(physeq_genus)
```

Take a look at each of these if you want to see what the differences were in each step!

Question 3: How does the number of genera compare with the number of ASVs?

5.2.4 4.1.3 Run MaAsLin3

Now that we've prepared the phyloseq objects, it's time to run our first differential abundance test. Each of the differential abundance tools expects data to be normalised in a different way (or some of them normalise for you), and we're going to run MaAsLin with rarefied data.

So to do that, we'll need to rarefy it:

```
set.seed(345)
physeq_rare = rarefy_even_depth(physeq_genus, sample.size = min(sample_sums(physeq_genu
```

Note that if you aren't using the 16S data, you might want to take a look at `min(sample_sums(physeq_genus))` before you run the rarefy command. Does that seem like a sensible number of sequences to rarefy to? Take a look at all of the sample sums using `sample_sums(physeq_genus)` to see what might be sensible. Setting this to be higher than some samples will remove any samples with below this number of reads.

Remember that you can always use `?rarefy_even_depth` if you want to see what a function is doing! Here we've rarefied all of the samples to the lowest

number of reads, ASVs are replaced after sampling (`replace = TRUE`), those that are no longer present are trimmed (`trimOTUs = TRUE`) and the function will tell us about what it is doing (`verbose = TRUE`).

Next, we'll get the feature table and metadata that we're using:

```
feat_table = data.frame(t(otu_table(physeq_rare)), check.rows=T, check.names=T, stringsAsFactors=
metadata = data.frame(sample_data(physeq_rare), stringsAsFactors = F)
```

You'll notice that these will be formatted as tables again - we could have just read in some tables to be used with MaAsLin2, but it's useful to perform manipulations in Phyloseq first, and then to take these tables out again to ensure that all samples match up!

Now we'll run MaAsLin3:

```
fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_treatment', sep=""),
                    formula = '~ treatment',
                    normalization = 'TSS',
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)
```

If you have used another amplicon, you can run this step more than once with each of the variables! Or you can run both together by including them in the formula as `~ variable1 + variable2`

Take a look at `?maaslin3` to see what each of the inputs to this function are! Note that we'd usually set the max significance to 0.1, but for the sake of this workshop we're using a higher level.

MaAsLin will save the results into a folder, so you can take a look in these folders and look at the `all_results.tsv` as well as `significant_results.tsv` files. You'll see that `significant_results.tsv` is just a subsection of `all_results.tsv` - have a look at one of these files. It also makes some nice plots for you of the significant results - have a look through these and see if you can work out what they're showing you.

There are a lot of columns in the output; you'll see some of the ones we're interested in are: (1) the genus ("feature"), (2) the metadata variable being

tested ("metadata"), (3) for categorical features, the specific feature level for which the coefficient and significance of association is being reported ("value"), (4) the model effect size/coefficient value ("coef"), (5) the standard error from the model ("stderr"), (6) pval_individual (7) qval_individual (8) pval_joint (9) qval_joint (10) error (11) model (12) N (13) N_not_zero

Question 4: How many taxa does MaAsLin3 say are significantly differentially abundant?

5.2.5 4.1.4 Run ALDEx2

Next, we'll run ALDEx2. Note that here the first step is to normalise the samples using a CLR normalisation, so you can see how both of the methods require similar steps - normalisation and then testing - but this is done in a slightly different way for each. We then use a Kruskal-Wallis test and a GLM ANOVA on the data, and we can do this test with multiple cores (`useMC=2`). You can see that we're again running this separately for each metadata variable, and then saving the outputs as .csv files:

```
x <- aldex.clr(otu_table(physeq_genus), sample_data(physeq_genus)$treatment, mc.samples
kw.test <- aldex.kw(x, useMC=2, verbose=FALSE)
write.csv(kw.test, paste(folder, "ALDEx2_taxa.csv", sep=""))
```

If you take a look at these results files then you'll see that we have 4 columns: kw.ep, kw.eBH, glm.ep, glm.eBH - the p-values and Benjamini-Hochberg (BH)-corrected p-values for the Kruskal-Wallis and GLM ANOVA tests between the sample groups. There are several other options for testing within the ALDEx2 R package (including aldex.ttest and aldex.glm), but in this case, we can use the GLM ANOVA results for parametric tests and the Kruskal-Wallis results for non-parametric tests. We could also use aldex.effect to see the magnitude of the differences between groups.

Note: parametric tests make assumptions about the distribution of the population that the samples are taken from (typically that it is normally distributed), while non-parametric tests are distribution free and can be used for non-normal variables. Microbiome data are not typically normally distributed, but we can test this. You don't need to run this next part, but can do if you like.

```
shapiro.test(as.data.frame(otu_table(physeq_genus))$"B-Rt96")
physeq_genus_clr = microbiome::transform(physeq_genus, "clr")
shapiro.test(as.data.frame(otu_table(physeq_genus_clr))$"B-Rt96")
```

Transformations/normalisations of microbiome data are often used to make the usually non-normal microbiome data normal, however, in this case, it is not normal (significant p-value so the null hypothesis that the data are normal can

be rejected - they are not normal). This means that we should use the Kruskal-Wallis test results.

Question 5: How many taxa does ALDEx2 find to be significantly differentially abundant?

5.2.6 4.1.5 Combine and plot differential abundance results from MaAsLin3 and ALDEx2

In order to compare the results that we've got from the three differential abundance tools, we'll want to first import the results and do some manipulations on these tables.

For MaAsLin2, we'll read in the table and then we'll rename some of the taxa names because some R formats don't like punctuation other than "_" or ".", so we'll convert these back to how they started (so they match up with the other tables):

```
maaslin = read.csv(paste(folder, "maaslin3_treatment/significant_results.tsv", sep=""), sep="\t")
maaslin$feature = gsub("\\g__Palsa.187", "g__Palsa-187", maaslin$feature) #replace anything match
maaslin$feature = gsub("\\.", ";", maaslin$feature) #replace any remaining . with ;
```

Maaslin3 compares both abundance and prevalence, but as it is the only one to do prevalence, we will just look at abundance for now so we'll filter the table:

```
maaslin = maaslin[maaslin$model == 'abundance', ]
```

Question 6: Are there any taxa left? Is that what you expected?

Now we'll do the same for ALDEx2 (although here we don't need to do the renaming):

```
aldex = read.csv(paste(folder, "ALDEx2_taxa.csv", sep=""))
aldex = aldex[aldex$kw.eBH <= 0.2, ]
```

You should see that there are a few taxa that ALDEx2 found to be significantly differentially abundant.

I mentioned previously that often we will consider those taxa found by >=2 tests to be significantly differentially abundant as actually being differentially abundant.

To do that, we'll create a list of the taxa that are found to be significantly differentially abundant by both tests:

```
overlap = intersect(maaslin$feature, aldex$X)
```

If we take a look at this, we should see that there are 7 taxa that were found by both tests, although a few of them are unclassified at the genus level.

And now we'll convert the physeq_genus phyloseq object to relative abundance using the `transform_sample_counts` function:

```
physeq_relabun  = transform_sample_counts(physeq_genus, function(x) (x / sum(x))*100 )
```

5.2.6.0.1 Plot DA single taxon Next we can look at plotting the abundance of a single taxon. Have a look at the list of taxa (`overlap`). Replace the empty quotation marks below with one of these:

```
single_taxon = ""
```

And now we'll make a boxplot of it's abundance based on the categories in treatment:

```
microbiome::boxplot_abundance(physeq_relabun, x='treatment', y=single_taxon, line = NU
```

Note that if you didn't replace the quotation marks with the name of a taxon above, then you will get an error message! Otherwise, you should see the plot pop up in the "Plots" section on the bottom right of the window.

To explain what we have just done here a little, we're using a function from the microbiome R package - sometimes there can be functions with the same name from multiple R packages, so adding `microbiome::` at the start tells R which one we are wanting to use. The first positional argument is the phyloseq object that we are using, and then everything else consists of named arguments: The metadata variable that we'd like to use for the x axis (and making the boxplots), what we'd like on the y axis (this needs to be a taxon that is in the phyloseq object), whether we'd like to map a variable onto the lines, whether we'd like this to be a violin plot rather than a boxplot, whether NAs should be removed, and whether points should be shown in addition to the boxes.

This `boxplot_abundance` function builds upon the R package ggplot2, and we can therefore use additional arguments as we would with ggplot2. These are added with the + and are fairly self explanatory; we have added a title (ggtitle) and a y label (ylab). For the title, we've replaced part of the text with the `"\n"` which means that a line break has been added in to make this a little more readable.

Question 7: Did it look like there are differences between the two treatments for the taxon that you plotted?

5.2.6.0.2 Plot DA all taxa We can also make the same plots within a for loop:

```
for(i in 1:length(overlap)) {
  print(microbiome::boxplot_abundance(physeq_relabun, x='treatment', y=overlap[i]) + ggtitle(str_
}
```

Now, if you press the forward and back buttons in the "Plots" pane, you can scroll through all of the taxa that we've identified as differentially abundant by one of the tests. You'll probably be able to see that a lot of these taxa are really only found in one of the treatments and not the other, so it's fairly clear why they're coming up as being significantly different.

5.2.6.0.2.1 For loops For loops are an incredibly useful thing that we do in programming - to my knowledge, they exist in every programming language, and they allow us to repeat a section of code many times, with a different input each time. To explain what we are doing above, we can go through this step-by-step. If you already know about for loops, feel free to skip ahead to section 2.8.

It is easiest to explain by showing you, so try running this:

```
print(overlap)
print(length(overlap))
```

You'll see that overlap is a list containing all of the taxa that we found to be significantly differentially abundant, and `length(overlap)` tells you how long it is, or how many taxa are in it.

Now try running this:

```
for(i in 1:length(overlap)) {
  print(i)
  print(overlap[i])
}
```

You should see that now, a number is printed out, and then that number item from the list is also printed out after it. The for loop is telling us that for every value of i between 1 and the length(overlap), print out i and then print out the i'th item in the list. You can use this for accessing particular items in lists, too, *e.g.*, `overlap[2]`.

This is a really simple example, but you can see how in the loop above, we just replaced `single_taxon`, from when we just plotted one taxon, with `overlap[i]` within the loop, so that it would change each time.

For loops can get incredibly complicated, can do lots of things, and can be nested inside other for loops, but we won't be doing anything more on them for now!

5.2.6.0.2.2 Back to the results Question 8: Are there any taxa that you're surprised to see are significantly differentially abundant? Why?

5.2.7 4.2 Functional prediction with PICRUSt2

5.2.7.1 4.2.1 Filter input for PICRUSt2

As PICRUSt2 can take quite a long time to run, we want to do some filtering on the files so that we'll have fewer ASVs. Usually we might investigate several different cutoffs for the minimum number of times an ASV should occur to be included, or the minimum prevalence of ASVs, but for the purposes of this workshop, we'll just use 50 for the minimum frequency of an ASV and 2 for the minimum number of samples that it must occur in.

First, we'll copy over the output:

```
cd ~/workspace
mkdir bmb_module4 bmb_module4/16S_analysis
cd bmb_module4/16S_analysis
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/deblur_output/ .
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/taxa/ .
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/Blueberry_16S_metadata.tsv
```

First, we need to reactivate the QIIME2 environment:

```
conda activate qiime2-amplicon-2025.4
```

Then we'll filter the ASVs based on the parameters I mentioned above:

```
qiime feature-table filter-features \
  --i-table deblur_output/deblur_table_final.qza \
  --p-min-frequency 50 \
  --p-min-samples 2 \
  --o-filtered-table final_table_filtered.qza
```

Now we'll filter the sequences to only include those that are in our new filtered table (`final_table_filtered.qza`):

```
qiime feature-table filter-seqs \
  --i-data deblur_output/representative_sequences.qza \
  --i-table final_table_filtered.qza \
  --o-filtered-data  representative_sequences_final_filtered.qza
```

Then we can export these files - first the feature table, which we'll export and then convert from .biom format to .txt format:

```
qiime tools export \
  --input-path final_table_filtered.qza \
  --output-path exports_filtered

biom convert \
  -i exports_filtered/feature-table.biom \
  -o exports_filtered/feature-table.txt \
  --to-tsv
```

Then we can export the sequences file:

```
qiime tools export \
  --input-path representative_sequences_final_filtered.qza \
  --output-path exports_filtered
```

This will be exported as a .fasta file, which you can look at in the `exports_filtered` folder, if you like.

Finally, we'll deactivate the conda environment:

```
conda deactivate
```

5.2.7.2 4.2.2 Start running PICRUSt2

PICRUSt2 can take quite a long time to run - for PICRUSt2, as well as other programs that may take a while, there are several tools that are pre-installed on most Linux systems that we can use to make sure that our program carries on running even if we get disconnected from the server. One of the most frequently used ones is called `tmux`. To activate it, just type in `tmux` and press enter. It should take a second to start up, and then load up with a similar looking command prompt to previously, but with a coloured bar at the bottom of the screen.

To get out of this window again, press `ctrl`+`b` at the same time, and then d. You should see your original command prompt and something like

```
[detached (from session 0)]
```

We can actually use tmux to have multiple sessions, so to see a list of the active sessions, use:

```
tmux ls
```

We can rename the tmux session that we just created with this:

```
tmux rename-session -t 0 picrust
```

Note that we know it was session 0 because it said that we detached from session 0 when we exited it.

If we want to re-enter this window, we use:

```
tmux attach-session -t picrust
```

Now, we can run PICRUSt2 inside this tmux session. First, we'll activate the conda environment:

```
conda activate picrust2-v2.6.2
```

Now, as we mentioned above, we'll just set PICRUSt2 to run for now, and we'll come back to see the output later:

```
picrust2_pipeline.py -s exports_filtered/dna-sequences.fasta -i exports_filtered/featu
```

You can see that here the options we're setting are: - `-s`: the fasta file of DNA sequences - `-i`: the feature table in .biom format - `-o`: the folder to save the PICRUSt2 output to - `-p`: the number of threads to use - `-t`: the method to use for placement of ASVs into the phylogenetic tree - note that the default is to use EPA-NG, but this takes much more memory, and as we are limited on these servers we are using SEPP - `--in_traits EC`: we don't need to add this option as the default is to run both EC and KO, but as this can take quite a long time and we don't have very much memory, we'll just run EC numbers as this is what we can get pathway predictions from - `--verbose`: this tells PICRUSt2 to print out each of the steps that it's running so that we know what's going on.

You might want to move on to something else while PICRUSt2 runs, as it's likely to take a little while (~20 mins).

5.2.7.3 4.2.3 Read PICRUSt2 output into R/Phyloseq

Once it's run, we can come back to the output.

There are a few files that we'll take a look at now, so to prepare for that, we'll just unzip them:

```
gunzip picrust2_out_pipeline_filtered/*.gz
```

There is a key file that we can take a look at to see how well PICRUSt2 is likely to work for our data, and that is the `combined_marker_predicted_and_nsti.tsv` file. Download this and open it with Excel (or similar). You'll see that it has five columns - the first contains the ASV name, the second the number of 16S rRNA gene copies that this ASV is predicted to have, the third the NSTI, the fourth contains the domain that this ASV matched best with (our 16S dataset used the bacteria-specific primers, so it shouldn't be surprising that we only have bacteria here!), and the fifth, the closest reference genome within the database used for predictions. The NSTI is the Nearest Sequenced Taxon Index and refers to the distance within the phylogenetic tree of that ASV to the closest relative that it has in the reference database. A value of 0 would indicate that the database that PICRUSt2 uses contains a genome with an identical sequence. By default, PICRUSt2 excludes all ASVs with a value of above 2, but the lower these NSTI values are, the better the predictions are likely to be. In our case, the median NSTI is ~0.09. This is not 0, but it is at least a long way off 2. If you look in the EC_metagenome_out folder then you'll also see a `weighted_nsti.tsv.gz` - in this one, the NSTI has been calculated on a per-sample basis but it's been weighted by the abundance of the ASVs within each sample. You'll see that the weighted NSTI's are actually quite similar to our median, although they are typically slightly higher, indicating that the abundant ASVs within our dataset are less similar to the reference genomes. You can see in our recent paper that these values vary - these samples are actually a subset of the Blueberry ones shown, so it makes sense that we get a similar median NSTI value - with the NSTI of less well characterised environments (e.g. ocean samples) typically being higher than in better characterised environments (like HMP samples).

In this folder, you'll also see `EC_predicted.tsv` - this shows the number of copies of Enzyme Commission (EC) numbers predicted to be within each ASV. You'll also see `out.tre` files which contains a tree of the reference as well as study 16S sequences.

The `intermediate` folder shows us some of the intermediate files that were produced/used, but we don't really need those for now.

Finally, the files that we are likely most interested in are `picrust2_out_pipeline_filtered/EC_metagenome_out/pr` and `picrust2_out_pipeline_filtered/pathways_out/path_abun_unstrat.tsv.gz`.

We're going to focus on the pathways file today, because it groups the other functions into functional categories, but this could be repeated in the same way with the EC results.

Note that if we were interested in seeing which ASVs contribute to the functions within each sample then we would include the `--stratified` option when running PICRUSt2, although this increases the time taken to run. We also often predict the KOs (KEGG orthologs), but we left these for now for the sake of time.

So now try to do the same for the PICRUSt2 results as you did for the taxa results above in RStudio. I've again copied that code below here to avoid you needing to scroll up and down, and the code needed is in the answers if you are struggling.

If you didn't do 4.1, just go through the first step of importing packages. If you already did 4.1 and would like to challenge yourself, try just modifying the commands that we gave above for reading in the files and performing differential abundance testing with the PICRUSt2 pathways file!

Read in the pathways file as a feature table:

```
folder = '/home/ubuntu/workspace/bmb_module4/16S_analysis/'
pwy_table <- read.csv(paste(folder, "picrust2_out_pipeline_filtered/pathways_out/path_a
```

Hint: You'll need to unzip the `picrust2_out_pipeline_filtered/pathways_out/path_abun_unstrat` file before you can start.

Take a look at pwy_table. Does that look right? Compare it with `path_abun_unstrat.tsv`. We just copied this code from when we were reading in our taxa file above.

Question 9: What do you think might be wrong with it?

OK, so modify this line of code to fix it and let's try that again:

```
pwy_table <- read.csv(paste(folder, "picrust2_out_pipeline_filtered/pathways_out/path_a
```

Manipulate the feature table:

```
pwy_table_num = data.matrix(pwy_table[,2:11]) #convert the ASV table to a numeric matri
rownames(pwy_table_num) = pwy_table[,1] #give the matrix row names
```

Change the sample names:

```
colnames(pwy_table_num) = gsub("B.", "B-", colnames(pwy_table_num))
```

We don't have taxonomy information this time, so that part isn't necessary, but next we'll want to read in the metadata. This will actually be exactly the same as before, but it won't hurt if you do it again.

```
metadata <- read.csv(paste(folder, "Blueberry_16S_metadata.tsv", sep=""), sep='\t')
samples = data.frame(metadata[,2], stringsAsFactors = FALSE) #convert this to a data frame
rownames(samples) = metadata[,1] #add the sample names as row names
colnames(samples) = c('treatment') #and add a column name that makes sense
```

Now combine everything into the phyloseq object:

```
PWY = otu_table(pwy_table_num, taxa_are_rows = TRUE) #convert asv_table_num to an otu_table
SAMPLE = sample_data(samples) #convert samples to a sample_data
physeq_pwy = phyloseq(PWY, SAMPLE) #combine these all to make a phyloseq object
physeq_pwy #print this out to see what a phyloseq object looks like
```

We also aren't collapsing this at the genus level because there is no taxonomy information/higher levels to collapse on.

Now, we can convert the pathways to relative abundance, take the top 100 pathways for differential abundance testing, and convert the numbers into integers (whole numbers) for some of the alpha diversity calculations.

```
physeq_pwy_relabun = transform_sample_counts(physeq_pwy, function(x) (x / sum(x))*100 )
top_100_abun <- names(sort(taxa_sums(physeq_pwy_relabun), TRUE)[1:100]) #get most abundant pathwa
physeq_pwy_top_100 = prune_taxa(top_100_abun, physeq_pwy) #now filter the table to have only the
otu_table(physeq_pwy_top_100) = round(otu_table(physeq_pwy_top_100)) #round the abundance table s
mode(pwy_table_num) = "integer" #convert the table to integers (whole numbers)
PWY_int = otu_table(pwy_table_num, taxa_are_rows = TRUE)
physeq_pwy_int = phyloseq(PWY_int, SAMPLE) #make a new phyloseq object with this
```

5.2.7.4 4.2.3 Look at PICRUSt2 alpha and beta diversity

Now we will just take a quick look at these to see how they compare with the analyses that we did in Module 3 based on the taxonomy data.

First we will plot the Alpha diversity:

```
plot_richness(physeq_pwy_int, x="treatment", measures=c("Observed", "Simpson", "Shannon")) + geom
```

Luckily, phyloseq has some built-in functions for these that make it very easy. You can see more information about these functions with `?plot_richness`.

Then have a look at the Beta diversity:

```
ps.ord <- ordinate(physeq_pwy_relabun, "PCoA", "bray")
plot_ordination(physeq_pwy_relabun, ps.ord, type="samples", color="treatment", shape="treatment")
```

Are you getting an error here? With most errors that we get, a quick Google search will give us some clues on what might be going wrong. When I google the error message "No available covariate data to map on the points for this plot `type`", one of the first things that comes up is this page, and one of the users on there suggests that this is due to only having one column in our metadata table. Sometimes there are weird bugs in things and this seems to be one of those times! We'll just add a fake second metadata column to get around this.

```
sample_data(physeq_pwy_relabun)$second_column = 'fake_data'
```

Now try running the above `plot_ordination` function again. Hopefully you'll see that it works now! It's annoying when we have to find work-arounds like this, but it happens reasonably often.

You should notice that we've chosen to colour and shape the samples by treatment. If you had more than one sample variable here, you could choose colour for one and shape for the other.

Question 10: Can you see differences between the samples based on the metadata variables?
Question 11: What are these differences like compared with those seen in the taxonomy data?

5.2.7.5 4.2.4 Run PICRUSt2 differential abundance

Now we'll try running the differential abundance tests.

First, rarefy the pathway data:

```
set.seed(345)
physeq_rare_pwy = rarefy_even_depth(physeq_pwy, sample.size = min(sample_sums(physeq_pw
physeq_rare_pwy_top_100 = prune_taxa(top_100_abun, physeq_rare_pwy)
```

Get the feature tables and metadata:

```
feat_table = data.frame(t(otu_table(physeq_rare_pwy)), check.rows=T, check.names=T, str
metadata = data.frame(sample_data(physeq_rare_pwy), stringsAsFactors = F)
```

Now we'll run MaAsLin3, first with our treatment grouping:

```
fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_pathway_treatment', sep=""),
                    formula = '~ treatment',
                    normalization = 'TSS',
```

```
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)
```

Run ALDEx2:

```
x <- aldex.clr(otu_table(physeq_pwy_top_100), sample_data(physeq_pwy_top_100)$treatment, mc.sampl
kw.test <- aldex.kw(x, useMC=2, verbose=FALSE)
write.csv(kw.test, paste(folder, "ALDEx2_pathway.csv", sep=""))
```

5.2.7.6 4.2.5 Combine and plot PICRUSt2 differential abundance results from MaAsLin3 and ALDEx2

Import the MaAsLin results:

```
maaslin = read.csv(paste(folder, "maaslin3_pathway_treatment/significant_results.tsv", sep=""), s
maaslin$feature = gsub("\\.", "-", maaslin$feature) #replace any . in the pathway names with -
```

Again, we're just going to be looking at the abundance models, although MaAsLin3 does carry out differential abundance testing based on both prevalence and abundance.

```
maaslin = maaslin[maaslin$model == 'abundance', ]
```

Now we'll do the same for ALDEx2:

```
aldex = read.csv(paste(folder, "ALDEx2_pathway.csv", sep=""))
aldex = aldex[aldex$kw.eBH <= 0.2, ]
```

ALDEx2 doesn't find anything to be differentially abundant. This means that there isn't anything that we would consider to be significantly differentially abundant (DA) with 2 tests, but we can still plot out some of the pathways that were identified by MaAsLin3 as DA.

```
pathways = maaslin$feature
```

5.2.7.7 Plot DA single pathway

Next we can look at plotting the abundance of a single pathways. Have a look at the list of pathways (`pathways`). Replace the empty quotation marks below with one of these:

```
single_pathway = ""
```

Now make the boxplot:

```
microbiome::boxplot_abundance(physeq_pwy_relabun, x='treatment', y=single_pathway, line
```

5.2.7.8 Plot DA all pathways

Now make all of the boxplots:

```
for(i in 1:length(pathways)) {
  print(microbiome::boxplot_abundance(physeq_pwy_relabun, x='treatment', y=pathways[i])
}
```

You'll see that these are all quite low abundance (<1% abundance), but they do all seem to have clear differences between the two treatments.

5.2.8 4.3 Constructing and visualising a co-occurrence network

5.2.8.1 Introduction

Microbes don't live in isolation—they form complex ecological communities where interactions between taxa can be cooperative, competitive, or neutral. One way to explore these interactions is through co-occurrence networks, where nodes represent microbial taxa and edges represent inferred associations. To make these networks more biologically meaningful, we need to account for the compositional nature of microbiome data and avoid common pitfalls such as spurious correlations.

In this section, we'll use the `SpiecEasi` package to construct a sparse inverse covariance network—designed specifically for compositional data—and visualize it using igraph. Before we do that, however, we'll identify which ASVs are significantly associated with treatment groups using indicator species analysis, and then filter our dataset to include only sufficiently prevalent and abundant taxa. Let's start by identifying those indicator species.

We would HIGHLY suggest you use 16S for this section, but if you're feeling adventurous you can see if you can get it to run for 18S or ITS.

5.2.8.2 4.3.1 Prepare data from the end of Module 2

First, we'll create a new directory for whichever dataset we want to use and we'll copy the output from the end of Module 2 into this.

16S:

```
cd ~/workspace
mkdir bmb_module4 bmb_module4/16S_analysis
cd bmb_module4/16S_analysis
ln -s ~/CourseData/MIC_data/bmb_module3/output/16S_analysis/deblur_output_exported/ .
ln -s ~/CourseData/MIC_data/bmb_module2/output/16S_analysis/Blueberry_16S_metadata.tsv .
```

5.2.8.2.1 Read output into R/Phyloseq Next, we'll go to RStudio on the server. Go to your browser and navigate to:

```
http://##.uhn-hpc.ca:8080/
```

(where ## is the student number that you have been given).

Enter "ubuntu" as your username and the password that have been given. It takes a little while to load up the first time but should be quicker on subsequent times!

Everything that you're entering in here should be entered into the "Console" part of the page. This is like the Terminal that we've been using on the Server, but it uses the R programming language. Most statistical analyses in bioinformatics will use either R or Python - many bioinformaticians have a preference for one or the other, but most use both. In my opinion, each has different strengths and weaknesses - R has many built in programs and functions, while Python is much more customisable but perhaps has a steeper learning curve. The syntax varies slightly for both (and both are different from the Terminal that we have been using, which is the bash programming language), but there are also many similarities. RStudio is an Integrated Developing Environment (IDE) for R and Python - it is an application that provides further functionality than the R programming language itself.

5.2.8.2.2 Importing packages First of all, we'll import the packages that we'll be using today:

```
library(phyloseq)
library(SpiecEasi)
library(indicspecies)
library(igraph)
```

You'll notice that in each of these, we use the `library()` command, and inside the brackets we give the name of the package. We've already installed these packages for you, but in most cases these could be installed by *e.g.,* `install.packages("igraph")`. In some cases, they might be installed using another package, BiocManager, *e.g.,* `BiocManager::install("phyloseq")`. If you're unsure, you can always google how to install a package and you can usually find the code to copy and paste, *e.g.,* "r install phyloseq". You could also try typing `??phyloseq` into the console and see what comes up. It should give you some information about that package.

5.2.8.2.3 Setting variables Next, we'll set the name of the folder where our results are saved:

```
folder = '/home/ubuntu/workspace/bmb_module4/16S_analysis/'
```

Here, we have defined a variable called "folder" - we always need to be careful that we don't save over these variables by calling something else the same thing, but you can always check these in the "Environment" area in the top right where you should see that "folder" now exists. We could have called this anything, but it's always helpful to name our variables something that is simple and explains what they are, incase someone else is using our code. This is a string, as it is inside the punctuation marks `' '` or `" "`, and this is used in programming to show that we are representing text rather than numbers. These variables can be really useful to avoid having to type in the same information multiple times, which always increases the chances that we'll make a mistake!

5.2.8.2.4 Read in the feature table First, we'll read in the feature table. Note that we're calling it `asv_table`:

```
asv_table <- read.csv(paste(folder, "deblur_output_exported/feature-table_w_tax.txt", s
```

Here we're combining a few different things. We're using the `read.csv` function - functions are modules of code that accomplish a specific task. We can write them ourselves, they are what is contained in the packages that we imported at the start, or there are several that are built in to R. If you want more information about what a function does as well as the input that it expects and the output that it gives, you can type in `?read.csv`.

We're also using the `paste` function to add together the full file path of the feature table. We're giving it the `folder` variable that we already defined, as well as the additional folder and file name (both as positional arguments), and finally the `sep=""` named argument - this tells the `paste` function that there should be no spaces or other punctuation between the two parts that are being pasted together. Try running just this part and see what the result is: `paste(folder,`

`"exports_filtered/feature-table.txt", sep="")` - note that as we haven't given this a name, it is not saved as a variable.

Next, we've given the `sep='\t'` variable - this time, we're telling the `read.csv` function that the file that we're importing is tab-delimited, then we've told it to skip the first line of the file `skip=1`, and that there is a header `header=T` (where T is the same as TRUE). We've skipped the first line because this just reads "# Constructed from biom file" and is not actually data.

5.2.8.2.5 Manipulate the feature table We often find in programming that things are not quite in the format that is expected by the packages that we're using, so we'll make a few modifications to `asv_table`:

```
asv_table_num = data.matrix(asv_table[,2:11]) #convert the ASV table to a numeric matrix. Note th
rownames(asv_table_num) = asv_table[,1] #give the matrix row names
```

You can see here that I've added `#` after the code and have written what each line does. You can use `#` to make comments through your code documents, to explain what each line is doing. Here you should also notice that we didn't save over the previous `asv_table`, but created a new one called `asv_table_num`. Now, we've just told R that it should be expecting numeric data within this table, and then we gave it the same row names as our previous `asv_table` object.

Next, take a quick look at your sample names within asv_table_num.

This is kind of annoying because it means that the sample names won't match the metadata table, so we'll change those now.

```
colnames(asv_table_num) = gsub("B.", "B-", colnames(asv_table_num))
```

5.2.8.2.6 Get the taxonomy information You may have noticed that the last column in the initial `asv_table` was our taxonomy information. We'll get that first:

```
taxonomy <- data.frame(asv_table[,"taxonomy"])
rownames(taxonomy) <- asv_table[,1]
colnames(taxonomy) = c("taxonomy")
```

If you print this out by typing `taxonomy` or by clicking on this in the "Environment" area, you'll see that we currently only have a single column that contains all of the taxonomy information.

We want to split this so that we have columns for each of Domain, Phylum, Class, Order, Family, Genus and Species. Luckily, there is already a function that we can use for this:

```
taxonomy_split <- separate(data = taxonomy, col = taxonomy, into = c("Domain", "Phylum"
```

You'll see that we tell the `separate` function that the data it should use is in the `taxonomy` table, the `col` (column) is called "taxonomy". The "into" named argument takes a list as input, which is defined in R with the `c()`, and the `taxonomy` column should be split into a new column each time the `;` symbol is found. You'll see a warning after running this line, but it still ran so that's fine - it's just telling us that there weren't values for all of Domain/Phylum/Class/Order/Family/Genus/Species for every ASV, which makes sense as not all of them will have a species-level classification.

Again, you can take a look at the results in the "Environment" area.

5.2.8.2.7 Read in the metadata Now we'll read in the metadata. I won't go through each stage step-by-step as hopefully you're getting the idea by now, but I've still added comments to each of the rows:

```
metadata <- read.csv(paste(folder, "Blueberry_16S_metadata.tsv", sep=""), sep='\t')
samples = data.frame(metadata[,2], stringsAsFactors = FALSE) #convert this to a data fr
rownames(samples) = metadata[,1] #add the sample names as row names
colnames(samples) = c('treatment') #and add a column name that makes sense. If you have
```

5.2.8.2.8 Combine these into a phyloseq object Phyloseq is a really useful package that contains many useful functions for analysing microbiome data. While it can be a bit fiddly to get our data into the format that it is expecting (most good packages should give you examples of how data should be formatted to go into them, although it does take a lot of practice to get good at quickly working out how this is different from what you have!), once that it is in this format, the analyses are then very easy to perform.

```
ASV = otu_table(asv_table_num, taxa_are_rows = TRUE) #convert asv_table_num to an otu_t
TAX = tax_table(taxonomy_split) #convert taxonomy_split to a tax_table
taxa_names(TAX) <- rownames(taxonomy_split) #add names to TAX/the tax_table
SAMPLE = sample_data(samples) #convert samples to a sample_data
physeq = phyloseq(ASV, TAX, SAMPLE) #combine these all to make a phyloseq object
physeq #print this out to see what a phyloseq object looks like
```

Now, we have combined all of these different parts into one phyloseq object. Note that you can also import a phylogenetic tree to phyloseq, but we won't be using that for this part of the workshop. You can also get each of the individual objects back after performing manipulations, *e.g.,* `otu_table(physeq)`.

As we typically find that ASVs are not always shared across many samples (and neither would we expect them to be, if they potentially represent species or

strain level differences between taxa), we'll collapse the phyloseq object to the genus level. If you're not sure what the different taxonomy levels are called within the phyloseq object, you can always look at `tax_table(physeq)`.

So we're collapsing at the genus level, which is "ta6":

```
physeq_genus = tax_glom(physeq, taxrank="ta6")
```

If you take a look at this taxonomy table (`tax_table(physeq_genus)`), you'll notice a lot of missing information (because many environmental ASVs don't have similar taxa within reference databases!), so we can add the full taxonomy information in to the ASV names, so that this is hopefully a little more informative:

```
all_tax = paste(tax_table(physeq_genus)[,2], tax_table(physeq_genus)[,3], tax_table(physeq_genus)
taxa_names(physeq_genus) = all_tax
otu_table(physeq_genus)
```

Take a look at each of these if you want to see what the differences were in each step!

5.2.8.3 4.3.2 Indicator Species Analysis (ISA)

First, let's use `indicspecies` to run an Indicator Species Analysis (ISA) to find some ASVs whose abundance and prevalence suggests they are important for one sample type or another.

```
cat("Running indicator species analysis...\n")

# Transpose OTU table to samples x taxa
otu_mat <- as.data.frame(t(otu_table(physeq)))
# Extract treatment vector
group_vector <- sample_data(physeq)$treatment
# Run multipatt
set.seed(123)
indval_res <- multipatt(otu_mat, group_vector, func = "IndVal.g", duleg = TRUE, control = how(npe
cat("Indicator species analysis complete.\n")
print(summary(indval_res))
```

Once we have run the ISA, we need to extract the significant (pval <= 0.05) ASVs

```
sig_hits <- indval_res$sign
sig_asvs <- rownames(sig_hits)[which(sig_hits$p.value <= 0.05)]
```

Assign indicator group based on specificity columns

```
asv_indicator_group <- apply(sig_hits[sig_asvs, c("s.Forest", "s.Managed")], 1, functi
    if (x["s.Forest"] == 1 && x["s.Managed"] == 0) return("Forest")
    if (x["s.Forest"] == 0 && x["s.Managed"] == 1) return("Managed")
    return("Both")
})
# Named vector of ASV → group
indicator_vector <- setNames(asv_indicator_group, sig_asvs)
```

5.2.8.4 4.3.3 Abundance and Prevalence filtering

Now that we've identified which ASVs are associated with each treatment group, we'll filter the dataset to remove taxa that are too rare or too low in abundance. This step ensures the network we infer focuses on ecologically relevant players. (*NOTE: the ISA should have dealt with this, but it's good to double-check*)

Filter by prevalence and abundance

```
cat("Filtering phyloseq object by prevalence and abundance...\n")
min_prev <- 0.30
min_abun <- 0.0005
# Prevalence
prev_df <- apply(otu_table(physeq), 1, function(x) sum(x > 0) / length(x))
# Relative abundance
rel_abun <- transform_sample_counts(physeq, function(x) x / sum(x))
mean_abun <- taxa_sums(rel_abun) / nsamples(physeq)
# Logical filtering
keep_taxa <- (prev_df >= min_prev) & (mean_abun >= min_abun)
```

Now subset the `phyloseq` object we created earlier

```
# Subset
physeq_filtered <- prune_taxa(keep_taxa, physeq)
cat("Remaining taxa after filtering:", ntaxa(physeq_filtered), "\n")
```

Question 12: We say the ISA should filter out ASVs with low prevalence and abundance, but why? What does an ISA do?

5.2.8.5 4.3.4 Contructing the Co-occurrence Network

Now that we've selected our ASVs of focus, we can infer the co-occurrence network using `SpiecEasi`. We're using the MB (neighborhood selection) method

here, and we'll allow the algorithm to determine the best level of sparsity through cross-validation (via `pulsar`). This will give us a network where each edge represents a direct association between two taxa.

This part takes some time, be patient!!

```
cat("Running SpiecEasi...\n")
se.mb.16s <- spiec.easi(physeq_filtered, method = 'mb',
                        lambda.min.ratio = 1e-2, nlambda = 20,
                        pulsar.params = list(rep.num = 50, ncores = 3))

cat("SpiecEasi complete.\n")
```

Question 13: What other `method` algorithms are available for `SpiecEasi` other than `mb`? Can you find them in the manual? in and online search?

Next, we'll convert the adjacency matrix from Spiec-Easi into an igraph object and relabel the nodes with the original ASV identifiers. We'll also match the filtered phyloseq object to the ASVs used in the network and annotate each node with its indicator status.

```
ig2.mb <- adj2igraph(getRefit(se.mb.16s))
# Replace numeric node names with actual ASV IDs
correct_taxa_names <- taxa_names(physeq_filtered)[as.integer(igraph::V(ig2.mb)$name)]
igraph::V(ig2.mb)$name <- correct_taxa_names
# Prune phyloseq object to match taxa in network
physeq_net <- prune_taxa(correct_taxa_names, physeq_filtered)
# Add indicator status as a tax_table column (NA if not an indicator)
indicator_status <- indicator_vector[taxa_names(physeq_net)]
tax_table(physeq_net) <- cbind(tax_table(physeq_net), Indicator = indicator_status)
```

5.2.8.6 4.3.5 Visualizing Networks

Finally, we'll visualize the network using `igraph`. First, we'll color nodes by their indicator status (e.g., Forest, Managed, Not significant), and then generate a second plot colored by taxonomic assignment at the ta3 rank. These two plots will also be combined into a panel figure for comparison.

```
cat("Plotting network...\n")

set.seed(42)
layout_fr <- layout_with_fr(ig2.mb)
```

The ISA plot:

```
# --- Indicator plot ---
cat("Saving Indicator network...\n")
pdf(paste(folder, "network_indicator.pdf", sep=""), width = 7, height = 6)

indicator_vals <- tax_table(physeq_net)[igraph::V(ig2.mb)$name, "Indicator"]
indicator_vals[is.na(indicator_vals)] <- "Not significant"
indicator_factor <- factor(indicator_vals)
colors_ind <- scales::hue_pal()(length(levels(indicator_factor)))
node_colors_ind <- colors_ind[as.integer(indicator_factor)]

par(mar = c(4, 4, 4, 8))
par(xpd = TRUE)
plot(ig2.mb,
     layout = layout_fr,
     vertex.label = NA,
     vertex.color = node_colors_ind,
     vertex.size = 5,
     edge.color = "gray80",
     main = "Indicator Status")
legend("topright", inset = c(-0.35, 0), legend = levels(indicator_factor),
       col = colors_ind, pch = 19, pt.cex = 1.5, bty = "n", title = "Indicator")

dev.off()
```

The Taxonomy Plot:

```
# --- ta3 plot ---
cat("Saving ta3 network...\n")
pdf(paste(folder, "network_ta3.pdf", sep=""), width = 9, height = 6)

ta3_vals <- tax_table(physeq_net)[igraph::V(ig2.mb)$name, "ta3"]
ta3_vals[is.na(ta3_vals)] <- "Unclassified"
ta3_factor <- factor(ta3_vals)
colors_ta3 <- scales::hue_pal()(length(levels(ta3_factor)))
node_colors_ta3 <- colors_ta3[as.integer(ta3_factor)]

par(mar = c(4, 4, 4, 8))
par(xpd = TRUE)
plot(ig2.mb,
     layout = layout_fr,
     vertex.label = NA,
     vertex.color = node_colors_ta3,
     vertex.size = 5,
     edge.color = "gray80",
     main = "ta3 Taxonomy Rank")
```

```
legend("topright", inset = c(-0.18, 0), legend = levels(ta3_factor),
       col = colors_ta3, pch = 19, pt.cex = 1.5, bty = "n", title = "ta3")

dev.off()
```

Question 14: Can you visualize the `ta2` rank? `ta4` rank?

The Comparative Panel:

```
# --- Panel figure ---
cat("Saving combined panel figure...\n")
pdf(paste(folder, "network_panel.pdf", sep=""), width = 18, height = 6)
par(mfrow = c(1, 2), mar = c(4, 4, 4, 8), xpd = TRUE)

# Left panel: Indicator
plot(ig2.mb,
     layout = layout_fr,
     vertex.label = NA,
     vertex.color = node_colors_ind,
     vertex.size = 5,
     edge.color = "gray80",
     main = "Indicator Status")
legend("topright", inset = c(-0.05, 0), legend = levels(indicator_factor),
       col = colors_ind, pch = 19, pt.cex = 1.2, bty = "n", title = "Indicator")

# Right panel: ta3
plot(ig2.mb,
     layout = layout_fr,
     vertex.label = NA,
     vertex.color = node_colors_ta3,
     vertex.size = 5,
     edge.color = "gray80",
     main = "ta3 Taxonomy Rank")
legend("topright", inset = c(-0.18, 0), legend = levels(ta3_factor),
       col = colors_ta3, pch = 19, pt.cex = 1.2, bty = "n", title = "ta3")

dev.off()
```

5.2.8.7 Bonus: Size by network statistics

More advanced plotting can include adding network statistics as the sizes of the nodes.

Metric	Ecological Interpretation
Degree	Taxa with many direct associations (potential generalists or hubs)
Betweenness	Taxa that act as bridges between sub-communities (network bottlenecks)
Closeness	Taxa that can quickly influence others (central in information flow)
Eigenvector	Taxa connected to other highly connected taxa (influential nodes)
PageRank	Like eigenvector but more robust—considers quality and quantity of links

Thought Question: This above table shows how nodes could be statistically linked to ecological principles. If we see an ASV with high/low values in any of these statistics, what does it say about their influence in their community?

Here is `Degree`:

```
# Panel figure with degree-based sizing
cat("Saving panel figure with degree-based node sizing...\n")
pdf(paste(folder, "network_panel_degree.pdf", sep=""), width = 18, height = 6)
par(mfrow = c(1, 2), mar = c(4, 4, 4, 8), xpd = TRUE)

# Compute node degrees
node_degree <- degree(ig2.mb, mode = "all")
scaled_degree <- scales::rescale(node_degree, to = c(3, 10))  # size between 3 and 10
```

And plot it now:

```
# Left panel: Indicator (with degree-based size)
plot(ig2.mb,
     layout = layout_fr,
     vertex.label = NA,
     vertex.color = node_colors_ind,
     vertex.size = scaled_degree,
     edge.color = "gray80",
     main = "Indicator Status (by degree)")
legend("topright", inset = c(-0.05, 0), legend = levels(indicator_factor),
       col = colors_ind, pch = 19, pt.cex = 1.2, bty = "n", title = "Indicator")
# Add manual legend for node size (degree)
legend("bottomright", inset = c(-0.05, 0),
       legend = c("1", "3", "5"),
       pt.cex = c(1, 3, 5) / 2,  # Adjust to match your scaling
       pch = 19, col = "gray40",
```

```
       title = "Node Degree",
       bty = "n")

# Right panel: ta3 (with degree-based size)
plot(ig2.mb,
     layout = layout_fr,
     vertex.label = NA,
     vertex.color = node_colors_ta3,
     vertex.size = scaled_degree,
     edge.color = "gray80",
     main = "ta3 Taxonomy Rank (by degree)")
legend("topright", inset = c(-0.18, 0), legend = levels(ta3_factor),
       col = colors_ta3, pch = 19, pt.cex = 1.2, bty = "n", title = "ta3")
# Add manual legend for node size (degree)
legend("bottomright", inset = c(-0.05, 0),
       legend = c("1", "3", "5"),
       pt.cex = c(1, 3, 5) / 2,  # Adjust to match your scaling
       pch = 19, col = "gray40",
       title = "Node Degree",
       bty = "n")

dev.off()
cat("Degree-scaled panel figure saved.\n")
```

5.2.9 Answers

5.2.9.1 Answers 16S differential abundance and PICRUSt2

Question 1: Type in `asv_table` or look at it in the "Environment" area in the top right. How many rows are there? Does this make sense to you? There should be the same number of rows as there were in the `feature_table.txt`, but without the `# Constructed from biom file` line.

Question 2: Are they the same as you remember - any different punctuation? You should notice that where there were hyphens (`-`) before, these have now been replaced with `.` - this is an annoying thing that R does when importing data. There are options to turn this behaviour off, but we'll just rename the columns for now.

Question 3: How does the number of genera compare with the number of ASVs? There are a lot fewer genera than ASVs.

Question 4: How many taxa does MaAsLin3 say are significantly differentially abundant? These numbers might vary slightly depending on how you have rarefied your data and whether you have kept the random seed the same, but there are 16 rows in the `significant_results.tsv` file, eight of which are

unique (they are each shown twice - once for the abundance DA model and once for the prevalence DA model).

Question 5: How many taxa does ALDEx2 find to be significantly differentially abundant? Looking at the Kruskal-Wallis false discovery rate corrected p-value (`kw.eBH`), there are 26 taxa with p <= 0.2.

Question 6: Are there any taxa left? Is that what you expected? There are seven genera that are found to be DA by both MaAsLin3 and ALDEx2! This means that almost all of the genera identified by MaAsLin3 are also DA with ALDEx2, which is relatively surprising given the different models that they each use.

Question 7: Did it look like there are differences between the two treatments for the taxon that you plotted? Because you chose the taxon, I can't give the answer here!! Feel free to talk to one of us if you aren't sure on how to interpret these results.

Question 8: Are there any taxa that you're surprised to see are significantly differentially abundant? Why? Again, chat to one of us if you aren't sure on how to interpret these results!

Question 9: What do you think might be wrong with it? For the ASVs, we were needing to skip the first line of the file that didn't contain a useful header. Now we have the column names in our header line, so the `skip=1` part of this is making the second line of the file be read in as the header, and this contains pathway abundance information. We should remove the `skip=1` part before running this again.

Question 10: Can you see differences between the samples based on the metadata variables? Yes - Shannon and Simpson diversity seems to be slightly higher in Managed than Forest samples, although Observed richness is about the same in both. The samples don't group entirely separately on the PCoA plot, although there does seem to be some more variation among the Managed than the Forest samples.

Question 11: What are these differences like compared with those seen in the taxonomy data?

5.2.9.2 Answers 18S differential abundance

See below for the answers to questions!

5.2.9.2.1 Code needed

Importing packages:

```
library(phyloseq)
library(maaslin3)
library(microbiome)
```

```
library(ALDEx2)
library(ggplot2)
library(tidyr)
library(stringr)
library(vegan)
library(stats)
library(SpiecEasi)
```

Setting variables:

```
folder = '/home/ubuntu/workspace/bmb_module4/18S_analysis/'
```

Read in ASV table:

```
asv_table <- read.csv(paste(folder, "deblur_output_exported/feature-table_w_tax.txt", sep=""), se
```

Manipulate the feature table:

```
asv_table_num = data.matrix(asv_table[,2:13])
rownames(asv_table_num) = asv_table[,1] #give the matrix row names
```

Get the taxonomy information:

```
taxonomy <- data.frame(asv_table[,"taxonomy"])
rownames(taxonomy) <- asv_table[,1]
colnames(taxonomy) = c("taxonomy")
```

We want to split this so that we have columns for each of Domain, Supergroup, Division, Subdivision, Class, Order, Family, Genus and Species:

```
taxonomy_split <- separate(data = taxonomy, col = taxonomy, into = c("Domain", "Supergroup", "Div
```

Read in the metadata:

```
metadata <- read.csv(paste(folder, "Plastisphere_18S_metadata.tsv", sep=""), sep='\t')
samples = data.frame(metadata[,2:3], stringsAsFactors = FALSE)
rownames(samples) = metadata[,1] #add the sample names as row names
```

Combine these into a phyloseq object:

```
ASV = otu_table(asv_table_num, taxa_are_rows = TRUE) #convert asv_table_num to an otu_t
TAX = tax_table(taxonomy_split) #convert taxonomy_split to a tax_table
taxa_names(TAX) <- rownames(taxonomy_split) #add names to TAX/the tax_table
SAMPLE = sample_data(samples) #convert samples to a sample_data
physeq = phyloseq(ASV, TAX, SAMPLE) #combine these all to make a phyloseq object
physeq #print this out to see what a phyloseq object looks like
```

Collapse at the genus level, which is "ta8" for 18S:

```
physeq_genus = tax_glom(physeq, taxrank="ta8")
```

Add names:

```
all_tax = paste(tax_table(physeq_genus)[,4], tax_table(physeq_genus)[,5], tax_table(phy
taxa_names(physeq_genus) = all_tax
otu_table(physeq_genus)
```

Rarefy:

```
set.seed(345)
sample_sums(physeq_genus)
physeq_rare = rarefy_even_depth(physeq_genus, sample.size = 1000, replace = TRUE, trim(
```

Note that we've changed the minimum sample size here because we had some samples with *very* low numbers of reads. I've printed out those sums to see how many samples we'll lose. This isn't ideal, but we don't really want to go below 1000 sequences in our samples. You'll see that 4 samples get removed, almost all of which are cave samples.

If this were a "real" analysis, we might find this reduced sample size a bit problematic, but we'll just carry on for now.

Get the feature table and metadata that we're using:

```
feat_table = data.frame(t(otu_table(physeq_rare)), check.rows=T, check.names=T, strings
metadata = data.frame(sample_data(physeq_rare), stringsAsFactors = F)
```

Run MaAsLin3:

```
fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_treatment', sep=""),
                    formula = '~ light + time',
                    normalization = 'TSS',
```

```
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)
```

Probably unsurprisingly given the small numbers of samples in each group, none of these were significant :(

We can also see what happens if we run each of these separately:

```
fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_light', sep=""),
                    formula = '~ light',
                    normalization = 'TSS',
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)

fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_time', sep=""),
                    formula = '~ time',
                    normalization = 'TSS',
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)
```

But none of these are significant either :(

Run ALDEx2 separately for light and time:

```
x <- aldex.clr(otu_table(physeq_genus), sample_data(physeq_genus)$light, mc.samples = 1
kw.test <- aldex.kw(x, useMC=2, verbose=FALSE)
write.csv(kw.test, paste(folder, "ALDEx2_light.csv", sep=""))

x <- aldex.clr(otu_table(physeq_genus), sample_data(physeq_genus)$time, mc.samples = 12
kw.test <- aldex.kw(x, useMC=2, verbose=FALSE)
write.csv(kw.test, paste(folder, "ALDEx2_time.csv", sep=""))
```

Get ALDEx2 results:

```
aldex_light = read.csv(paste(folder, "ALDEx2_light.csv", sep=""))
aldex_light = aldex_light[aldex_light$kw.eBH <= 0.2, ]

aldex_time = read.csv(paste(folder, "ALDEx2_time.csv", sep=""))
aldex_time = aldex_time[aldex_time$kw.eBH <= 0.2, ]
```

None of these are significant either :(

5.2.9.2.2 18S Answers to questions **Question 1**: Type in `asv_table` or look at it in the "Environment" area in the top right. How many rows are there? Does this make sense to you? There should be the same number of rows as there were in the `feature_table.txt`, but without the `# Constructed from biom file` line.

Question 2: Are they the same as you remember - any different punctuation? Yes! Unlike with the 16S data, there was no punctuation in our sample names to begin with so these have remained correct.

Question 3: How does the number of genera compare with the number of ASVs? There are a lot fewer genera than ASVs.

Question 4: How many taxa does MaAsLin3 say are significantly differentially abundant? None :(

Question 5: How many taxa does ALDEx2 find to be significantly differentially abundant? None :(

Question 6: Are there any taxa left? Is that what you expected? There are none found with either MaAsLin3 or ALDEx2, and not with either (or both) metadata variables, so there is no overlap here. If you didn't run the 16S data already but want to see the plots, try running through the 16S data instead.

Question 7: Did it look like there are differences between the two treatments for the taxon that you plotted? Not applicable.

Question 8: Are there any taxa that you're surprised to see are significantly differentially abundant? Why? Not applicable.

5.2.9.3 Answers ITS differential abundance

5.2.9.3.1 Code needed Importing packages:

```
library(phyloseq)
library(maaslin3)
library(microbiome)
library(ALDEx2)
library(ggplot2)
library(tidyr)
library(stringr)
library(vegan)
library(stats)
library(SpiecEasi)
```

Setting variables:

```
folder = '/home/ubuntu/workspace/bmb_module4/ITS_analysis/'
```

Read in ASV table:

```
asv_table <- read.csv(paste(folder, "deblur_output_exported/feature-table_w_tax.txt", sep=""), se
```

Manipulate the feature table:

```
asv_table_num = data.matrix(asv_table[,2:21])
rownames(asv_table_num) = asv_table[,1] #give the matrix row names
```

Get the taxonomy information:

```
taxonomy <- data.frame(asv_table[,"taxonomy"])
rownames(taxonomy) <- asv_table[,1]
colnames(taxonomy) = c("taxonomy")
```

We want to split this so that we have columns for each of Kingdom, Phylum, Class, Order, Family, Genus, Species and Subspecies:

```
taxonomy_split <- separate(data = taxonomy, col = taxonomy, into = c("Kingdom", "Phylum", "Class"
```

Read in the metadata:

```
metadata <- read.csv(paste(folder, "pregnancy_ITS_metadata.tsv", sep=""), sep='\t')
samples = data.frame(metadata[,2:3], stringsAsFactors = FALSE)
rownames(samples) = metadata[,1] #add the sample names as row names
```

Combine these into a phyloseq object:

```
ASV = otu_table(asv_table_num, taxa_are_rows = TRUE) #convert asv_table_num to an otu_t
TAX = tax_table(taxonomy_split) #convert taxonomy_split to a tax_table
taxa_names(TAX) <- rownames(taxonomy_split) #add names to TAX/the tax_table
SAMPLE = sample_data(samples) #convert samples to a sample_data
physeq = phyloseq(ASV, TAX, SAMPLE) #combine these all to make a phyloseq object
physeq #print this out to see what a phyloseq object looks like
```

Collapse at the genus level, which is "ta6" for ITS:

```
physeq_genus = tax_glom(physeq, taxrank="ta6")
```

Add names:

```
all_tax = paste(tax_table(physeq_genus)[,2], tax_table(physeq_genus)[,3], tax_table(phy
taxa_names(physeq_genus) = all_tax
otu_table(physeq_genus)
```

Rarefy:

```
set.seed(345)
sample_sums(physeq_genus)
physeq_rare = rarefy_even_depth(physeq_genus, sample.size = 1000, replace = TRUE, trim(
```

Note that we've changed the minimum sample size here because we had one samples with a *very* low number of reads. I've printed out those sums to see how many samples we'll lose. This isn't ideal, but we don't really want to go below 1000 sequences in our samples. You'll see that 1 sample gets removed - if we wanted to be even more robust then we might set this to say 10,000 sequences, which would remove 3 samples.

Get the feature table and metadata that we're using:

```
feat_table = data.frame(t(otu_table(physeq_rare)), check.rows=T, check.names=T, strings
metadata = data.frame(sample_data(physeq_rare), stringsAsFactors = F)
```

Run MaAsLin3:

```
fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_treatment', sep=""),
                    formula = '~ exact_age + category',
```

```
                    normalization = 'TSS',
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)
```

None of these were significant :(

We can also see what happens if we run each of these separately:

```
fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_exact_age', sep=""),
                    formula = '~ exact_age',
                    normalization = 'TSS',
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)

fit_out <- maaslin3(input_data = feat_table,
                    input_metadata = metadata,
                    output = paste(folder, 'maaslin3_category', sep=""),
                    formula = '~ category',
                    normalization = 'TSS',
                    transform = 'LOG',
                    augment = TRUE,
                    standardize = TRUE,
                    max_significance = 0.2,
                    median_comparison_abundance = TRUE,
                    median_comparison_prevalence = FALSE,
                    max_pngs = 100,
                    cores = 1,
                    save_models = TRUE)
```

But none of these are significant either :(

Run ALDEx2 separately for exact_age and category:

```
x <- aldex.clr(otu_table(physeq_genus), sample_data(physeq_genus)$exact_age, mc.samples
kw.test <- aldex.kw(x, useMC=2, verbose=FALSE)
write.csv(kw.test, paste(folder, "ALDEx2_exact_age.csv", sep=""))

x <- aldex.clr(otu_table(physeq_genus), sample_data(physeq_genus)$category, mc.samples
kw.test <- aldex.kw(x, useMC=2, verbose=FALSE)
write.csv(kw.test, paste(folder, "ALDEx2_category.csv", sep=""))
```

Get ALDEx2 results:

```
aldex_exact_age = read.csv(paste(folder, "ALDEx2_exact_age.csv", sep=""))
aldex_exact_age = aldex_exact_age[aldex_exact_age$kw.eBH <= 0.2, ]

aldex_category = read.csv(paste(folder, "ALDEx2_category.csv", sep=""))
aldex_category = aldex_category[aldex_category$kw.eBH <= 0.2, ]
```

None of these are significant either :(

5.2.9.3.2 ITS Answers to questions **Question 1**: Type in `asv_table` or look at it in the "Environment" area in the top right. How many rows are there? Does this make sense to you? There should be the same number of rows as there were in the `feature_table.txt`, but without the `# Constructed from biom file` line.

Question 2: Are they the same as you remember - any different punctuation? Yes! Unlike with the 16S data, there was no punctuation in our sample names to begin with so these have remained correct.

Question 3: How does the number of genera compare with the number of ASVs? There are a lot fewer genera than ASVs.

Question 4: How many taxa does MaAsLin3 say are significantly differentially abundant? None :(

Question 5: How many taxa does ALDEx2 find to be significantly differentially abundant? None :(

Question 6: Are there any taxa left? Is that what you expected? There are none found with either MaAsLin3 or ALDEx2, and not with either (or both) metadata variables, so there is no overlap here. If you didn't run the 16S data already but want to see the plots, try running through the 16S data instead.

Question 7: Did it look like there are differences between the two treatments for the taxon that you plotted? Not applicable.

Question 8: Are there any taxa that you're surprised to see are significantly differentially abundant? Why? Not applicable.