Microbial Community Comparison Pipeline

https://github.com/ndeimler99/microbial_community_comparison

Nathaniel Deimler, Cameron Watson, Ryan Decourcy

Contents

Introduction

Background This is a pipeline for the consistent and unbiased comparison of two high-throughput amplicon sequenced microbial community datasets. Specifically, this pipeline has been designed for datasets that are produced from the amplicon sequencing of the V4 hypervariable region; however, it could be used with any hypervariable region or the entire 16S gene. This pipeline is necessary because comparing microbial datasets that have undergone different processing protocols introduces bias in the comparison. For example, comparing the taxonomies of two different datasets that were assigned from different reference databases introduces artifactual diversity into the comparison that may have originated due to the differences between the reference databases. There are many areas in amplicon sequence processing for microbial community analyses in which these biases could accumulate, so this pipeline aims to eliminate confounding variables so that comparisons may be as representative of true biological variation as possible.

Limitations The comparisons generated by this pipeline can only be as consistent as the input data allows. A major constraint is that the datasets being compared must be of the same amplicon, or hypervariable region. For example, it would introduce bias to have one dataset that was generated from the V3 hypervariable region and the other from the V4 hypervariable region. This is because different amplicons, and even different 16S hypervariable regions will resolve taxonomic classifications to different levels of granularity (Johnson et al., 2019). This is one example of how the input data can introduce bias into a microbial community comparison. The user of this pipeline must be careful to make sure their input data was collected in a way that mitigates biases as much as possible.

Installation and Set-up

Download from GitHub Clone the microbial_comparison_pipeline repository to the computer that you are working on. A directory with this name should now be present in your directory hierarchy, and all scripts needed to run the pipeline will be available.

The Conda Environment This pipeline has been developed to run inside of a Conda environment. This means that the user must have Anaconda (Anaconda Software Distribution, version 2-2.40) installed on the computer they are using. To set up the Conda environment needed for this pipeline to function, run the following in the command-line from the R_Environment subdirectory:

```
conda env create --file r_environment.yml
```

Running the Install Script install.R can be run either through the command-line from inside the R_Environment subdirectory:

```
./install.R
```

or via R-Studio; however, ensure that the newly created Conda environment is activated prior to running this script with:

```
conda activate R_Environment
```

install.R takes exactly one input parameter and must immediately follow the script call. This variable is the cran mirror in which the R packages will be downloaded from. If using INSTALL_SCRIPT.srun, no changes need to be made unless you want to respecify the cran mirror. If running the install script by the command-line, one will have to include either a variable containing the cran mirror or the hard url:

```
Cran="https://ftp.osuosl.org/pub/cran/"
install.R $cran
```

Or

```
install.R https://ftp.osuosl.org/pub/cran/
```

For a list of the cran mirrors available please visit https://cran.r-project.org/mirrors.html.

The installation script installs specific versions of all the software necessary to run the pipeline. Additionally, it also downloads SILVA reference databases to the working directory, which are used for taxonomic assignment during the DADA2 portion of the pipeline (Quast et al., 2013). These can easily be replaced with a different taxonomic reference database if desired; however, the user must assign all communities being compared from the same database.

Package Versions

R Package	Version	Documentation
phangorn	2.5.5	https://cran.r-project.org/web/packages/phangorn/phangorn.pdf
MASS	7.3-53	https://cran.r-project.org/web/packages/MASS/MASS.pdf
ape	5.4-1	https://cran.r-project.org/web/packages/ape/ape.pdf
ashr	2.2-47	https://cran.r-project.org/web/packages/ashr/index.html
tidyverse	1.3.0	https://www.tidyverse.org/
ggplot2	3.3.2	https://ggplot2.tidyverse.org/
Fossil	0.4.0	https://cran.r-project.org/web/packages/fossil/fossil.pdf
sys	3.4	https://cran.r-project.org/web/packages/sys/index.html
vegan	2.5-7	https://cran.r-project.org/web/packages/vegan/vegan.pdf
BiocManager	3.11	https://cran.r-project.org/web/packages/BiocManager/index.html
dada2	N/A*	https://benjjneb.github.io/dada2/
DESeq2	N/A*	https://www.rdocumentation.org/packages/DESeq2/versions/1.12.3
biostrings	N/A*	https://bioconductor.org/packages/release/bioc/html/Biostrings.html
phyloseq	N/A*	https://www.rdocumentation.org/packages/phyloseq/versions/1.16.2
edgeR	N/A*	https://bioconductor.org/packages/release/bioc/html/edgeR.html
msa	N/A*	https://bioconductor.org/packages/release/bioc/vignettes/msa/inst/doc/msa.pdf
microbiome	N/A*	https://microbiome.github.io/tutorials/

* These package versions are dependent on the BiocManager version.

Pipeline Overview

Running the Scripts Everything to run this pipeline can be done from the command-line with only a few commands. Before running the pipeline script, make sure that R_Environment is activated (conda activate

R_Environment). Then the pipeline.sh script can be called with only three user input arguments:

Argument	input
-m	path to metadata file
-d	path to input directory
-o	path to output directory

The actual command would look like this:

```
./pipeline.sh -m ./metadata.txt -d ./datasets -o ./output_dir
```

An example of this being called inside of a SLURM script for running on an HPC can be found in the GitHub repository in the *example* subdirectory.

For information on the metadata file and other input specifications, see the *Input Data and Pre-Processing* section.

Changing Parameters Users are free to change any parameter that they feel is necessary in any of the individual analysis scripts; however, for convenience, hyperparameters that were deemed important enough to be changed on a run-by-run basis have been compiled in the bash script pipeline.sh. This allows the user to change important variables without digging into many different analysis scripts. The next section contains a summary table with all the hyperparameters that can be changed from the Bash script and their default settings.

Hyperparameter Summary *ASV Filtering*

Variable Name	Description	Default
minimum_sum_asv	Minimum number of ASVs across all samples required for the ASV to be kept	100
minimum_sample_count	Minimum number of samples an ASV must be present in for the ASV to be kept	1
min_sample_save	If an ASV fails to exceed minimum_sample_count it must exceed this value across all samples to still be included	1500

ASV Normalization

Variable Name	Description	Default
normalization_technique	desired method of normalization	DESeq2
rarefaction_lib_size	If rarefaction normalization is chosen, how large is the desired library size	10000

ANOSIM

Variable Name	Description	Default
min_percentage	The minimum percent contribution to the difference observed between two communities for the ASV to be included	0.01
high_cumulative	Once the cumulative percentage is surpassed, ignore all other ASVs	0.8

Differential Abundance

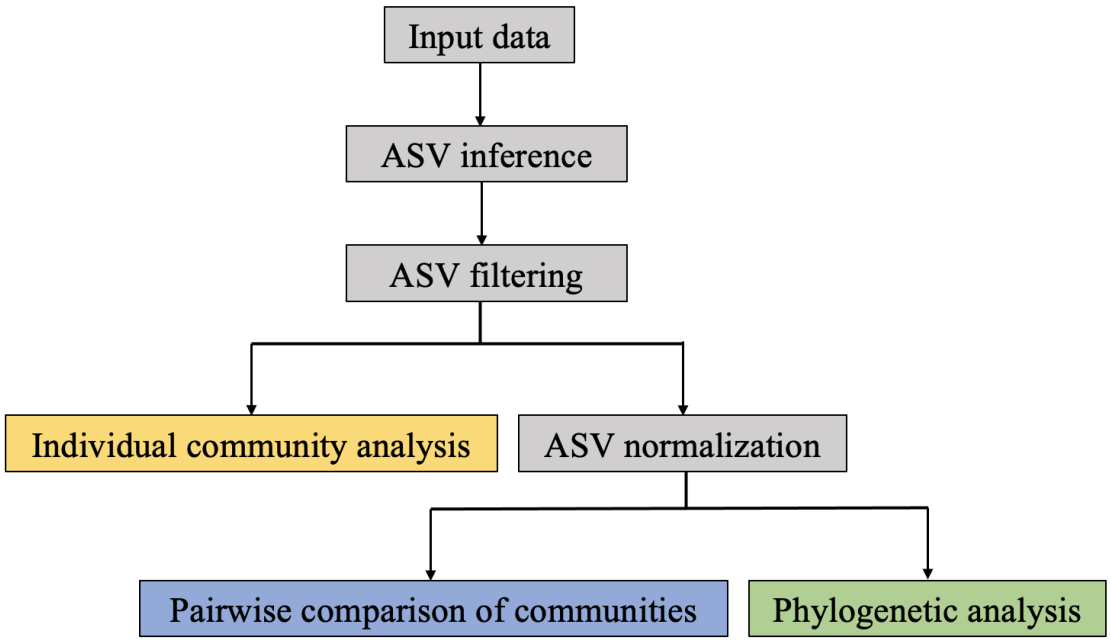
Variable Name	Description	Default
shrinkage_method	DESeq2 shrinkage method to account for ASVs with large variances	ashr
transform	DESeq2 method of trransformation for the creation of PCA plot	rld

Directories Created

Directory	Description
/output_dir/	This is the output directory specified by the user in the initial pipeline call. This directory will be found wherever pipeline.sh is called from
/output_dir/quality_plots	Contains sequencing quality control plots
/output_dir/normalization_results	Contains normalization results and normalized count data
/output_dir/stacked_bar_plots/	Contains all stacked bar plots created
/output_dir/heat_maps/	Contains all heat maps created
/output_dir/alpha_diversity/	Contains graphs, tables, and statistics related to alpha diversity measures
/output_dir/bray_nmds/	Contains bray-curtis dissimilarity matrices and resulting NMDS plots
/output_dir/anosim_simper/	Contains ANOSIM and SIMPER results for both communities
/output_dir/anosim_simper/community_one/	Contains SIMPER results when comparing samples within community one
/output_dir/anosim_simper/community_two/	Contains SIMPER results when comparing samples within community two
/output_dir/adonis/	Contains the results of the adonis statistical test
/output_dir/differential_expression/	Contains the results of differential abundance analysis, log2fold change plots, and PCA plots

Directory	Description
/output_dir/phylogenetics/	contains all phylogenetic analysis output
/output_dir/phylogenetics/community_one/	phylogenetic outputs for community one
/output_dir/phylogenetics/community_two/	phylogenetic outputs for community two
/output_dir/phylogenetics/community_merged/	phylogenetic outputs for both communities combined

Pipeline Schematic Below is a broad outline of every step in this pipeline. Detailed explanations of each step can be found in this manual.



ASV Search tool Along with the pipeline, the GitHub repository contains a tool for searching for specific ASVs after the pipeline is complete. This can be useful, for example, if a given ASV was identified to be of particular interest due to extraordinary differential abundance in a community. The *asv_search.R* script found withing the R_scripts directory takes the parameter of the output directory specified in the pipeline call as well as the unique identifier of an ASV. For example, a user could search for ASV320 with the following command:

```
./R_scripts/asv_search.R $output_dir ASV320
```

This script will then return a file titled by the ASV specified in the script call followed by "_summary.txt" containing the complete taxonomic summary of the ASV as well as the normalized count data of the ASV.

Input Data and Pre-Processing

Input Files Input files should be unzipped, demultiplexed, paired-end sequenced fastq files. Both communities input files should be stored in the same input directory. The naming conventions will be different

depending on whether the data is from the Sequence Read Archive or are files directly from the sequencer; however, both naming conventions require the sample name and forward/reverse strand identifiers to be in the file name.

In addition to the sample files, a metadata file is also required input. This is a tab-separated file with sample names in the first column, the community name in the second column, and a shorthand sample name in the third column. For example, below is part of a metadata file showing which communities (pyrosome or seawater) a given sample name belongs to. The shorthand sample name in column 3 will be used as labels on plots that show individual samples.

```
SRR12541962     wat     wat_1
SRR12541963     wat     wat_2
SRR12541975     pyr     pyr_1
SRR12541976     pyr     pyr_2
```

Using SRA Data Data from the Sequence Read Archive (SRA) should have filenames in the following format: *samplename_strand.fastq*. Examples for both forward strand (1) and reverse strand (2) files from the same sample can be seen below (Rasko et al., 2011).

```
SRR12541962_1.fastq
SRR12541962_2.fastq
```

This format can easily be achieved by downloading SRA data to the input directory using the fastq-dump command from the SRA Toolkit. To get the desired naming convention use the following command:

```
fastq-dump -I --split-files <INSERT_SRR_#>
```

Raw Sequencing Data Make sure all of the files are in the following format:

```
Samplename_R1_001.fastq for forward reads

Samplename_R2_001.fastq for reverse reads
```

This format should be the default for files coming from Illumina sequencers. If file names differ from either the SRA format or the raw sequencer format, the user will have to either change the file names to adhere to these specifications or change the file-recognition code in dada.R. Currently dada.R is lacking the functionality to parse mixed-named files, so both communities should follow the same naming convention. Mixed-file naming accommodation will be added to the pipeline in the future.

ASV Inference with DADA2

Overview and Resources This pipeline uses DADA2 for sample inference and ASV (Amplicon Sequence Variant) clustering. This portion of the pipeline is the first R script (dada.R) to be called by the master Bash script, and the inputs are automatically passed from the Bash script to the Dada2 workflow script. This script mostly follows the documentation and tutorials given here:

https://benjjneb.github.io/dada2/index.html

However, there are some changes made to the workflow and important notes to point out:

- This script is designed to automatically parse input files with different naming conventions as seen in Section 4 of this manual.

- Forward and reverse reads are filtered and trimmed based on the fastq quality score line using the Dada2 function filterAndTrim. There are many parameters in this step, and despite having defaults here, it is strongly suggested that users change these parameters based on the needs of their data. Additionally, this could be an area to troubleshoot if reads are being trimmed to the point of not being able to merge read pairs.
- Dada2 uses reference databases for taxonomic assignment. This pipeline defaults to using Silva reference training sets, which are automatically downloaded to the user's working directory in the install script (see Section 2). These can easily be changed; however, users should be cautious not to compare data that has been assigned from different databases.

Output Quality plots of the input reads are automatically generated, named, and saved as .png files in an output subdirectory /quality_plots. These plots can be useful to examine when troubleshooting or for simply exploring the data.

Compressed (gzipped) filtered and trimmed files can be found in the output subdirectory /filtered.

After all of the Dada2 processing is complete, ASV sequences are given a shorthand name (ASV1, ASV2...ASVn). These names and the full sequences are outputted in both plain-text and fasta format (*asv_sequences.txt*, *asv_sequences.fasta*).

The Dada2 sequence table and taxonomy table are outputted as *seqcounts.txt* and *taxmat_samples.txt*. All plain-text output files are tab-separated.

These outputs are automatically piped by the Bash script from DADA2 processing to the downstream analysis scripts.

Interpreting the Summary File As a part of the default output, the dada.R script produces a *dada_summary.txt* file containing summary information that allows the user to track what is happening to the input reads as they are being processed.

Throughout this file, there are step markers and time-stamps, for example:

```
[1] "Filtering"
[1] "2020-11-04 22:32:33 PST"
```

This denotes that the filtering process began at this specified time. This is useful for estimating future run times, gauging rate-limiting steps, and tracking where bugs may have occurred.

There is a lot of useful information in the .out file, but of particular importance is the read tracking table:

```
[1] "Tracking reads through Dada2 processing"
             input  filtered denoised.fw denoised.rv merged nonchimera
SRR12541962 954658  860252    809217    808079    256040    83084
SRR12541965 941861  854424    803462    732739    397988    131250
```

This table allows the user to see where reads are being lost throughout processing. This can help inform future runs or may alert the user of problems with the data. For example, seeing a large proportion of reads lost between merging and chimera removal may suggest adapter or primer contamination.

ASV Pre-Processing

Input As described earlier, the Dada2 portion of this pipeline outputs two important files; one of these files contains ASV names and abundance counts for each sample, while the second file contains the ASV names and their taxonomic assignment. The *data_manipulation_and_filtering.R* script directly inputs the counts table created by Dada2 from the directory specified by the user in the original pipeline command.

ASV Filtration ASV filtration is the process of removing ASV's with a low general abundance, the number of times an ASV occurs across all samples, or low frequency, the number of samples in which an ASV occurs. ASV's with a low abundance or frequency are not making a large impact on the community composition and therefore can and should be removed. In addition, these ASV's may be the result of sequencing error that remained uncorrected throughout the dada2 pipeline. Lastly, the removal of these ASV's will allow for the reduction of statistical noise and will increase the clarity of the resulting community comparison.

Two approaches are taken in order to filter the raw count data by abundance and frequency. The first method is by analyzing the total number of times an ASV occurs across all samples, defined as the abundance. If the abundance falls below a certain threshold, the ASV will be removed from the dataset and will not be included in subsequent analyses. This minimum abundance can be set by altering the hyperparamter, *minimum_sum_ASV*, default 100; therefore, each ASV must appear a minimum of 100 times across all the samples to be included. In order to filter by frequency, a minimum sample number threshold must also be set and met for each ASV to be included. The hyperparameter, *minimum_sample_count*, allows the user to specify how many samples an ASV must appear in for it to be considered in down-stream analysis, default 1. However, there is the chance that an ASV is in fewer samples than this specified threshold but may occur in a high enough abundance it is still significantly impacts that sample and represents a different version of the overall community and should not be discarded. *min_sample_save*, default 1500, allows the user to specify a value in which an ASV would have to occur for it to be included in down-stream analysis if it failed to exceed the *minimum_sample_count*.

The resulting, filtered count data is outputted to a new file in the specified output directory entitled *asv_counts_filtered.txt*. The filtered count data is also split by sample_group as specified in the metadata table from the original pipeline call. Now, there exists three dataframes; one containing the filtered count data of both communities, and one containing the filtered count data for each community. Any ASV containing a row sum of 0, meaning it does not appear in the community, is removed from that communities filtered count data frame. The count data for each community is returned in files with prefix of sample_group and the suffix "_asv_counts_filtered.txt."

ASV Normalization Data normalization is a process that allows for the adjustment of raw count data to accommodate for bias that may have arisen during sampling or next-generation sequencing due to amplification. However, data normalization in the field of microbial ecology is a highly debated topic. Certain studies argue the benefits of not normalizing your data, while others state it is a necessary step to take (Weiss et al., 2017). To further complicate things studies seem to heavily disagree which form of normalization to use and suggest there are various scenarios in which different types of normalization should be utilized (McMurdie & Holmes, 2014; Weiss et al., 2017).

Since the user of this pipeline will have their own goals in mind and understand the process used to obtain their sequencing data the best, this pipeline provides a variety of normalization options including no normalization, RPM normalization, DESeq2 normalization, and rarefaction normalization. The user can specify which form of normalization they would like to use through the hyperparameter, normalization_method. The first, and simplest, option is "none". This would result in the filtered count data to bypass normalization and proceed directly to downstream community analysis. However, the user can specify three other types of normalization: "RPM", "DESeq2", "Rarefaction". RPM, reads per million, simply scales each sample library to a set size of one million reads and does not account for library composition, only library size. DESeq2 normalization adjusts the filtered counts based on library size and composition making it a more comprehensive form of normalization. Lastly, rarefaction normalization is one of the most highly debated forms of normalization. Rarefying data is the process of randomly sampling the filter count data without replacement to a set library size. Therefore, it can be assumed ASV's with a higher abundance in the true data, will have a higher abundance in the rarefied data. The benefit of rarefaction, like DESeq2, is that this form of normalization does account for library composition; however, there are studies that say rarefaction is an inadmissable normalization technique since you are readily discarding available data. (Abrams et al., 2019; M. I. Love et al., 2014; Oksanen et al., 2009)

For additional information on DESeq2, see the following vignette: http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html

After normalization, three new files are created are in the subdirectory "/normalization_results/" within the specified pipeline output directory. "Total_asv_counts_normalized.txt" contains the filtered, normalized count data for both communities, while the filtered, normalized count data for each community individually is written into files containing the suffix "_asv_counts_normalized.txt". In addition, two bar plots which show the total number of ASV's for each sample are created: "non_normalized_seq_counts_plot.png" and "normalized_seq_counts_plot.png".

Phyloseq Object Creation The last step of data preprocessing, prior to analysis, is the creation of phyloseq objects which connect the count data to the taxonomic assignments for each ASV. This is done using the R package phyloseq in the script phyloseq_creation.R. This script creates a total of 9 phyloseq objects composed of count data and taxonomic assignments; both communities combined, community one, and community two, each have their own phyloseq objects for filtered count data, filtered and normalized count data, and lastly the filtered and normalized count data that is percent abundance transformed. These phyloseq objects are stored in the R workspace and are not written to files (McMurdie & Holmes, 2013).

In addition to the creation of the phyloseq objects two additional tables are written to files containing the percentage of ASV's identified to each taxonomic level. The first file, total_asv_taxonomic_assignments_summary.txts, contains the taxonomic summary of all ASV's prefiltration and normalization. The second file, "used_asv_taxonomic_assignments_summary.txt", contains the taxonomic summary of all the ASV's following the filtration conducted in the data_manipulation_and_filtering.R script. Both of these files can be found in the output directory specified by the user during the initial pipeline call.

Taxonomic Summary The script *identification_summary.R* will return a new subdirectory "taxonomic_summary". Within this directory will be two files for each level of taxonomic classification including Kingdom, Phylum, Class, Order, Family, Genus, and Species. The first file, entitled with the specific taxonomic level followed by "_summary_stats.txt" will contain a list of all unique clades at that taxonomic level, the sum occurrences of all ASV's within that clade, the mean occurrence of ASVs in that clade, as well as the standard deviation of the ASVs in that clade. This is done separately for each community. It will also contain the significance of a standard student's t-test comparing the two communities for that specific clade. In addition, a second file for the same taxonomic level followed by "_counts_normalized.txt" will contain a dataframe with rows being each clade at that taxonomic level and the columns referencing individual samples and the counts of ASVs in each sample belonging to that clade. All the data in both files is derived from the normalized count data. These files are created using the conglomeration functions in the R package phyloseq, tax_glom() (McMurdie & Holmes, 2013).

Individual Sample Analysis

Descriptive Statistics Prior to large scale community comparisons, each sample must be analyzed independently of the others. One such way has already been demonstrated by plotting the sequence counts per sample before and after normalization. The sequence counts per sample before and after normalization are also recorded and returned in a new metadata file created in the output directory specified in the initial pipeline command. Other such methods of low-level sample analysis include Alpha Diversity and Rarefaction. All the individual sample analyses are conducted on the filtered, but not normalized, count data. Non-normalized count data can be used since we are not directly comparing samples or proportions to each other and using normalized count data may add bias into alpha diversity statistics.

Alpha Diversity Metrics Alpha Diversity is measure of biodiversity in each habitat, or in this case, in each sample. Most alpha diversity measure account for species richness, the number of species present in an environment, as well as abundance, the number of individuals present, and evenness. Evenness can be described as the percent abundance of each species present in an environment; a more even community will have more similar percent abundances and a higher alpha diversity statistic. This pipeline provides three

widely accepted methods for measuring alpha diversity: the Shannon Diversity Index, the Simpson Diversity Index, and the Chao1 Diversity Index (Morris et al., 2014).

The Shannon Diversity index of each sample is calculated using the diversity function in the R package vegan with the index argument set to "shannon". The shannon diversity index accounts for species abundance and evenness through the following equation.

$H = -\sum_{i=1}^{k} p_i * ln(p_i)$

Where H is the Shannon-Weiner index, k is the number of ASV's per sample, and p_i the proportion of each ASV in the sample. In this case, a larger Shannon Diversity index, indicates a more biodiverse community. In addition, a Shannon's equitability index (EH) is calculated; this index will range from 0 to 1 and can be derived by dividing the true H by H¬max. Hmax is the maximum Shannon index possible assuming k number of species; therefore $H_{max} = \ln(k)$. If EH is closer to one the community is very even, while a low equitability index indicates a majority of the ASV's belong to a few of the species (Oksanen et al., 2009; Shannon & Weaver, 1949).

The Simpson Diversity index can also be calculated using the diversity function in the R package vegan by setting the index argument to "simpson". The simpson's diversity index, like the Shannon index, accounts for both species richness and evenness and can be calculated through the following equation.

$D = \frac{\sum n(n-1)}{N(N-1)}$

Where D is the Simpsons diversity index, n is the number of individuals belonging to each ASV, while N is the total number of individuals across all ASVs. The Simpson diversity index represents an inverse correlation with biodiversity; a lower Simpson's index correlates to a larger degree of biodiversity. To simplify things, the Simpson index can be subtracted from one to positively correlate biodiversity with the index (Oksanen et al., 2009; Simpson, 1949).

The last form of alpha diversity measured in this pipeline is the chao1 index. This index is more favorable to use when it is thought that sampling is incomplete or when there are numerous low abundance species present. The chao1 index extrapolates the number of species present in an environment based on the present data assuming the data is partially incomplete, as it often is. The chao1 estimator is as follows.

$S = S_{obs} + \frac{F_1^2}{2F_2}$

Where S_{obs} is the number of observed species from the data, F_1 is the number of species with a single occurrence in the sample, F_2 is the number of species with two occurrences, and S is the chao1 estimator, or the estimated number of species actually present in that sample. While this tells nothing about the evenness of the data like the Shannon and Simpson indices, the chao1 estimator helps demonstrate the completeness of the data as well as the species richness of each sample (Chao, 2006, p. 1; Vavrek, 2011)

Each of the five alpha diversity metrics described above are added into the metadata table for each sample and plotted by community resulting in the formation of notched and unnotched barplots comparing the indices between the two communities. These barplots can be found in the alpha_diversity subdirectory within the output directory specified in the initial call of this pipeline. The files are named based on the index used followed by "_diveristy_index_notched.png" if notched and "_diversity_index.png" if not notched. In addition, a file entitled "alpha_diversity_significance.txt" is created which contains the significance values of a student's t-test comparing each alpha diversity index between the two communities allowing for a low-level comparison. Additionally, plots for each unique community are created containing the different diversity index results and are named by the diversity index and community they represent.

Rarefaction Rarefaction, similar to the chao1 index, is a method of determining species richness through sampling. The rarecurve function in the vegan package can be used to construct rarefaction curve plots; many argue that rarefaction is unnecessary in the field of microbial ecology due to the advances of next-generation sequencing, but they can show basic, low level comparisons between the two microbial communities. The rarecurve function randomly samples the filtered count data prior to normalization without replacement and checks if it is a new species, or if the species has been found before: the more new species, the larger

the slope. However, when sampling provides fewer new species, if it all, the slope begins to flatten and approach an asymptote. If this is observed in the plot, one can be confident their sampling methodology was sufficient in accurately representing the community present. This portion of the pipeline will output a rarefaction plot for every sample in both communities to allow for the comparison of species richness across the communities (Oksanen et al., 2009; Simberloff, 1978). This plot can be found in the specified output directory, "rarefaction_curve.png".

Pairwise Comparison of Datasets

Bray Curtis Dissimilarity Matrices The Bray-Curtis statistic is used to compare two samples based on their composition using the following formula.

$BC_{ij} = 1 - \frac{2C_{ij}}{S_i + S_j}$

Where BC_{ij} is the Bray Curtis statistic, C_{ij} is the sum of the lesser values for species in common between sample I and J, and S_i and S_j are the total number of ASV's counted at each site. Using the vegdist() function, method= "bray", from the vegan R package a Bray Curtis dissimilarity matrix can be produced in which each sample is compared to every other sample resulting in the production of a matrix containing Bray Curtis statistic values. The bray_curtis_nmds.R script creates a total of nine matrices and writes them to appropriately named files. (Oksanen et al., 2009) A BC matrix for each community normalized, non-normalized, and presence-absence transformed data is created. In addition, a BC matrix containing both communities is created for normalized and non-normalized count data, as well as presence-absence transformed data.

NMDS Plot Creation Using the bray Curtis dissimilarity matrices created earlier, the metaMDS function is used to analyze the matrix and create a non-metric multidimensional scaling plots in two dimensions. The results of the metaMDS function are plotted and saved in the "/bray_nmds/" subdirectory with appropriate names. A total of six NMDS plots are created including both communities normalized and non-normalized, and each community separately normalized and non-normalized. NMDS plots allow for the visualization of variation between datasets as well as withing datasets. Points closer together represent samples with a higher degree of similarity then points further apart (Oksanen et al., 2009; RPubs - Running NMDS Using MetaMDS in Vegan, n.d.)

See the following NMDS vignette for more detail: https://rpubs.com/CPEL/NMDS

ANOSIM Analsyis of Similarity, or ANOSIM, is a non-parametric anova like tests that operates on bray curtis dissimilarity matrices in order to compare two communities. The anosim function in the vegan R package requires a BC matrix and method of grouping in order to calculate a R statistic and significance level. The R statistic ranges from 0 to 1 and is a measure of dissimilarity; the closer this value is to one, the larger the difference between the two communities is. A significance value is calculated by conducting permutations and randomly reassigning the data to different groups to determine the likelihood of experiencing an R statistic just as extreme as the one achieved in the true data. The summary data containing the R statistic, number of permutations, and significance value can be found in the output directory specified in the subdirectory "/anosim_simper/anosim_results.txt". In addition, a second file is created which contains a plot of the anosim results, "/anosim_simper/anosim_between_communities.png." This plot compares the difference observed between communities with the differences observed within each community (Clarke, 1993; Oksanen et al., 2009).

SIMPER While ANOSIM informs the user about if a difference between two communities exists, SIMPER, or similarity percentages can help determine what is causing that difference to appear. The simper vegan function takes a Bray Curtis dissimilarity matrix and a grouping factor as arguments to compare the two communities. It then returns a table containing the columns below.

Rownames	ASV number
Average	Average ASV contribution to between-group dissimilarity
Sd	Standard deviation of ASV contribution to between-group dissimilarity
Ratio	Average / standard deviation
Ava	Average abundance per group A
Avb	Average abundance per group B
Cumsum	Cumulative sum of differences
Significance	Kruskal Wallis test significance value
Adjusted_Significance	Benjamini-Hochberg corrected p-values

It is important to note the cumulative sum will always add up to 1.00, or all of the difference observed between the communities. However, if two communities are very similar there may be certain ASVs that only contribute slightly to the difference observed and may not be of interest. Therefore, two hyperparameters can be set to help filter the results of the simper data. The first, min_percentage, indicates the minimum percent contribution an ASV must exceed for it be returned to the user. The second, high_cumulative, indicates the value at which the cumulative sum is reached. For example, all SIMPER tables are organized with an increasing cumulative sum, with the largest contribution at the top of the table. As the table progresses the cumulative sum contribution will decrease, but the cumulative sum will continue to increase. The user can set a value that will discard any ASV's that fall above a certain cumulative sum. For example, if this value is set at 80, all ASV's that contribute to 80% of the difference observed will be included in the analysis. (Clarke, 1993; Oksanen et al., 2009)

The major downfall of the SIMPER test is that it does not provide information on the statistical significance of the difference between each ASV in the simper table between the two communities. To account for this a kruskal wallis t-test with a Benjamini Hochberg correction is performed on each ASV in the simper table after filtration (Kruskal & Wallis, 1952).

Two major files are returned, also in the anosim_simper subdirectory, that contain the filtered simper results of both communities normalized data as well as presence absence transformed data. These files are named

```
"simper_summary_filtered_ASVs_both_communities.txt"

and

"simper_summary_filtered_ASVs_presence_absence_both_communities.txt"
```

respectively. In addition to the two main files, the intercommunity samples are compared among themselves. The resulting simper tables are filtered and returned in appropriately named files containing the names of both samples being compared. These intercommunity comparisons can be found in the "anosim_simper" subdirectory, in the directory "/community_one/" or "/community_two/." Since we are no longer comparing two communities with multiple biological replicates, ava and avb, the average abundances are not included as they are simply the count values.

ADONIS Adonis is another permutational anova-like test measuring analysis of variance between populations through the vegan R package. The adonis function requires a bray-curtis dissimilarity matrix and grouping factor to compare communities. It returns an adonis object containing a model like matrix containing the significance values of each comparison. (Anderson, 2001; Oksanen et al., 2009). In the adonis subdirectory are six files as follows.

File Name	BC Matrix
Community_merge.txt	Community_merge_norm
Community_merge_presence_Absence.txt	Community_merge_presence_absence

File Name	BC Matrix
Community_one.txt	Community_one_norm
Community_one_presence_Absence.txt	Community_one_norm_presence_absence
Community_two.txt	Community_two_norm
Community_two_presence_Absence.txt	Community_two_norm_presence_absence

Note: the first two rows are sample groups, the subsequent four files are individual samples

---

An example output is provided below. The column, Pr(>F) contains the significance value determining if our data, when separated by the groups specified, is significantly different. In this case we can see we are comparing the normalized communities to each other with a very significant difference.

```
Call:
adonis(formula = bray_curtis_community_merge_norm ~ metadata$sample_group)

Permutation: free
Number of permutations: 999

Terms added sequentially (first to last)

                     Df SumsOfSqs MeanSqs F.Model     R2 Pr(>F)
metadata$sample_group  1    1.6559 1.65586  5.6837 0.3042  0.001 ***
Residuals             13    3.7874 0.29134         0.6958
Total                 14    5.4432                 1.0000
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Differential Abundance Analysis Differential abundance allows for the comparison of count data between different communities. This process is identical to differential expression. Using the R pakcage DESeq2, differential abundance between communities is determined for each ASV as well as each classification present at various taxonomic levels.

The first step of differential abundance using DESeq2 is to convert the phyloseq objects previously created can be converted into DESeq2 object types which are required for differential abundance analysis. This is easily done with the use of the phyloseq_to_deseq2 function. Immediately following, differential abundance analysis can be conducted on the ASV's through the DESeq() function which outputs a table containing the following components.

Rownames	ASV number or taxonomic clade
baseMean	mean of normalized counts across all samples
Log2FoldChange	Observed Difference between groupings quantified
lfcSE	Log fold change standard error
Pvalue	Significance Value of Differntial Abundance
Padj	Benjamini-Hochberg adjusted p-value

However, the base function of DESeq() fails to account for communities with a large variance between samples. Shrinkage of the log2fold change and significance values results in the adjustment of data based on

variance. For example, if there is initially a very large log2 fold change between two communities, but one communities samples have a remarkably large variance, the log2 fold change would be reduced to account for this variation and uncertainty in the data (Badri et al., 2018). Three different forms of shrinkage are supplied in this pipeline specified by the hyperparameter, "normal" (M. I. Love et al., 2014), "ashr" (Stephens, 2016), and "apeglm"(Zhu et al., 2019). For more information on the differences between the various shrinkage methods please visit the resources below.

apeglm Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distributions for sequence count data: removing the noise and preserving large differences. Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895

ashr Stephens, M. (2016) False discovery rates: a new deal. Biostatistics, 18:2. https://doi.org/10.1093/biostatistics/kxw041

Normal Love, M.I., Huber, W., Anders, S. (2014) Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15:550. https://doi.org/10.1186/s13059-014-0550-8

The shrunken differential expression results are saved in the subdirectory "differential_expression" in the file "results_table.txt". This table is ordered in increasing adjusted p-value. In addition, a file named differentially_expressed.txt" is also included in the same subdirectory. This file contains the results of each ASV that have an adjusted p-value of less than 0.01. A summary file containing the number of asv's with log fold changes greater than 0 (more abundant in community two) and the number of asv's with log fold changes less than 0 (more abundant in community one) which have an adjust p-value of less than 0.1. Lastly, an MA plot is created. For more information on what an MA plot is the following URL can be used as a valuable resource.

https://www.bioinformatics.babraham.ac.uk/projects/seqmonk/Help/3%20Visualisation/3.2%20Figures%20and%20Graphs/3.2.13%20The%20MA%20Plot.html.

However, in our context the MA plot plots log2 fold change on the y axis and the mean of the normalized counts for the ASV on the x-axis. ASV's that are differentially expressed are plotted in blue, while all other ASV's are plotted in gray. It is important to note for all differential expression analysis a positive log2fold change indicates the asv is more abundant in community two, while a negative log2fold change means an asv is more abundant in community one. If you are unsure of which of your sample_groups is community one and community two, you can reference summary.txt in the main output directory. Four more files are outputted into the differential_expression subdirectory. These four files are entitle with a community name followed by differential_0.01.txt or _differential_0.00001.txt. In each of these files is the differential abundance analysis results containing ASVs that are only significantly or very significantly overly abundant in that community. In addition to the normal results, an extra column is included which contains either a Yes or a No. If this value is Yes, then that ASV is also present in the other community. The plot entitled differential_expression_plot.png contains all ASV's on the x-axis and the log 2 fold change on the y-axis. plotted in blue are the ASVs overly abundant in community two (padj <0.00001) while the red is ASV's overly abundant in community one (padj < 0.0001).

The last step of differential abundance analysis is the creation of a PCA plot which shows clusters of samples based on their similarities, very similar to an NMDS plot. However, before the differential expression results can be plotted in this method it must first be transformed to allow clustering. There are three methods supplied of doing so which can be altered through the hyperparameter "transform". "rld" will conduct rlog transformation, "vsd" - variance stabilizing transformation, and "vsd.fast" - also a variance stabilizing transformation, but rather than using the entire dataset it subsets a few genes from each sample and extrapolates the results across the entire sample. "rld", rlog transformation, should be used when each sample contains aproximately the same number of reads; however, when size factors vary it is recommended to use vsd. It is not recommended to use vsd.fast unless time or computer resources are of utmost importance (most datasets will not require the use of vsd.fast). From the results of the transformation a PCA plot can be created using the plotPCA function and is saved in "all_asvs_pca.png". This PCA plot can be useful in identifying outlying samples.

However, the previously mentioned results are according to each ASV. As a result of dada2, every ASV is

classified to a different taxonomic level. Therefore, the differntial abundant anslysis is repeated on taxonomically conglomerated ASVs for each taxonomic level of Kingdom, Phylum, Class, Order, Family, and Genus. Returned are files containing the differential abundance results (taxonomic level "_differential.txt"), a simple differential abundance plot containing unqiue clades on the x-axis and log2fold change on the y-axis (level "_differential_expression.png", a summary file containing the number of clades over abundant in community one (underexpressed) or clades over abundant in community two (overexpressed) (level "_differential_expression_summary.txt"), an MA plot (level "_MA.png"), and a PCA plot entitled (level "_pca.png"). The same form of shrinkage and transformation specified in the hyperparameters will be used during this stage of analysis. All of these files can be found in the "differential_expression" subdirectory (M. Love et al., 2021).

Phylogenetic Analysis

Phylogenetic analysis shows, in an easily digestible and graphic manner, the relatedness of microbiomes both within and between communities with phylogenetic trees built on ASV reads from samples. This is done with three main sets of trees: two individual trees for each separate community and one tree built on a union of reads from both individual communities. Within each set of trees, there are slight modifications to allow for easier interpretation and readability. This is done with the Biostrings, MSA, ape, phangorn and phyloseq R packages (*see Installation and Set-up: Package Versions*).

Further Filtration The first step in the phylogenetic analysis is further filtration of the ASV reads. The input for this portion is the *asv_counts_filtered.txt* output from the *ASV Pre-processing* portion of the pipeline. Prior to this in the pipeline, ASVs were filtered collectively, and not with respect to individual communities. Because of this, the phylogenetic analysis portion of the pipeline requires further filtration to be useful.

ASVs are filtered on an individual community basis. For an ASV to be passed through this portion's filtration, they must meet two criteria: they must be present an average of 100 times in each of that community's samples, and they must be present in 25% or more of that community's samples. For an ASV to be passed through in a community with 5 samples, there must be at least 500 occurrences of that ASV within the entire community, and that ASV must appear in at least 2 of the community's samples.

This filtration is done on both individual communities separately. Afterward, to make the collective base of ASVs for the merged (shared) tree, those two sets of filtered ASVs are combined, so both are included in one group, without replicates. ASVs that are present in the 3 pools post-filtration are then matched with their respective sequences for further use (instead of the abundance count data used for selection), and outputted as 3 FASTA files for the following portions of the phylogenetic analysis.

Alignment For the ASV sequences to be used for tree construction, they need to be aligned with each other. This is necessary for the building of distance matrices, from which the phylogenetic trees are constructed. Not all the sequences are the same length, due to trimming of sequences, or poor fidelity. Orthologous portions of sequences are aligned with each other. The R packages Biostrings and MSA are used for the alignment portion. The input for this portion of the phylogenetic analysis are the 3 FASTA files outputted from the *Further Filtration* portion.

The alignment algorithm *ClustalW* is used by default, although 2 other algorithms are included in the package - *ClustalOmega* and *Muscle*. An example of how to select one of these alternative alignment algorithms can be seen here:

```
Align_c_one <- msa(orf_comm_one, "ClustalOmega")
```

Finally, a graphic display of the alignments can be produced with the *msaPrettyPrint()* function.

Tree Construction For construction of phylogenetic trees for output, distance matrices are built from the alignments, they are made into neighbor-joining trees and outgroup rooted. This is done with the R Packages Phyloseq, Phangorn and Ape. MSA alignment objects are converted into phyDat objects so that they can be used by the phangorn/phyloseq/ape package set. These objects are used to construct a maximum likelihood distance matrix upon which the tree is built with the neighbor-joining method.

The tree is then outgroup rooted – a single tip (ASV) that is the most distantly related to the others in the pool (the outlier) is chosen, and the rest of the ASVs in the community are plotted on the tree relative to it. Another common rooting method (not used in this script) is midpoint rooting, wherein the longest distance between 2 tips on a tree is selected, and the point in the exact middle of those 2 tips is selected as the root for the tree.

An outgroup is chosen by selecting the last tip in the vector of tips built from the distance matrix. The tips are ASV sequences which are ordered by the distance matrix function dist.ml(), and it is presumed that the least related sequence is last in the vector. This ASV is used to re-root the tree, with an outgroup rooting method, using the root() function. This can be commented out if re-rooting is not desired; the output of the nj() function, is by default unrooted. Trees can be midpoint rooted, if so desired, using the midpoint() function (although this is not recommended).

Parsimony and Agglomeration Trees are first plotted using all the ASVs in each of the 3 pools (both communities and the set of merged communities). The resulting trees can be quite difficult to process – there are too many tips, and the tree is too crowded to view and make sense of.

For each community, one "full" tree is plotted with all the ASVs used and labeled. The same tree is then again plotted without ASV labels, in order to show the underlying structure of the "full" tree. Two trees with progressive levels of tip agglomeration are then plotted, with less important tips removed. ASV labels remain on these trees, as they are easier to read when less tips are plotted. When trees are tip agglomerated, tips are removed and the complexity of the tree is reduced. Parsimony scores are taken to quantify the progressive drops in complexity, by measuring parsimony both before and after agglomeration is carried out. Parsimony scores are added to the bottom of the tree plots, as a measure of complexity of the trees. The default method (*Fitch*) to calculate parsimony is used in this script, and it can be changed to the *Sankoff* method, for example, by adding `method="sankoff"` in any of the *parsimony()* command lines. The default is consistent throughout the script.

At the final level of agglomeration, 2 trees are output. The first is the "completely" agglomerated tree with tip names of the ASVs that are used to construct it. In the second tree, the ASV names are replaced with the most specific level of taxonomic classification known for that particular ASV – explained more in depth in the *Annotation* section below. In addition, the robustness of the trees are optimized and quantified via a bootstrapping procedure. The trees are rearranged for optimization into a pml (parsimony max likelihood) object with the TBR (Tree Bisection and Reconnection) algorithm, and although this is hard-coded in the script, it can be changed, without loss of function, to several other algorithms, including Subtree Pruning and Regrafting (SPR) or Nearest Neighbor Interchange (NNI). This is done by simply changing `rearrangement='tbr'` to `rearrangement='spr'`. This can be found in the script under *OPTIMIZING WITH SELECTED METHOD*.

Trees are then bootstrapped with this rearrangment method – a subset, but not all, of the aligned sequences (ASVs) are used to re-construct a tree, which is then rearranged. A number of iterations are chosen. In the script, this default is set at 100*, although more (up to 1000 per tree) are often recommended. The number of times a junction (node at which the tree's branches split) is created, over all the iterations, is displayed as a percentage – the bootstrap support value – which is a measure of robustness of the optimized tree object that is created. If, for instance, there are 100 bootstrapping iterations, and a junction appears 90 out of 100 times, it will display a 90 next to it. The tree with bootstrap support values is plotted, without ASV labels, for assessment of the quality of the tree construction process. Bootstrapping is done before trees are tip agglomerated. Only values of bootstrapping percentages above 70% are shown on this tree, although this can be changed in the script with the bs_support variable.

The number of bootstrapping iterations can be changed in the *Bootstrapping* section of the script: `bs=100` can be changed to `bs=1000` in the *bootstrap.pml()* function.

Annotation Taxonomic classifications are pulled from a taxonomic table showing the speciation by ASV. Only the ASVs present in the "merged" pool are selected, as they encompass ASVs from both communities. The table includes columns of classifications for each ASV of Domain, Kingdom, Phylum, Class, Order, Family, Genus and Species where possible, or an "NA" if that classification is not known.

A final column ("Final") is added to the table, with the most specific classification of that ASV available. For example:

ASV	Kingdom	Phylum	Class	Order
ASV80	Bacteria	Proteobacteria	Alpha-proteobacteria	Rhodobacterales

Family	Genus	Species	Final
Rhodo-bacteraceae	Cognatiyoonia	NA	Cognatiyoonia

The "Final" column is what is used to annotate the trees with taxonomies. The trees annotated with taxonomies will have the same parsimony score as the "complete" agglomeration tree for the same community – they are the same trees, except they are labeled differently. This process is done in the script section denoted by the following comment: *PLOTTING TREE WITH TAXON LABELS INSTEAD OF ASVs, FOR FINAL AGGLOMERATION (h = 0.30)*

The general idea behind this portion of the script is to make a vector of the tip labels from the agglomerated tree, and then convert it to its corresponding taxonomic label, using the *tax_selected* object. This can be modified to display taxonomies on other trees, and different agglomeration levels. For instance, to show taxonomies on the partial agglomeration tree (with the parameter h = 0.15), simply change the variable *pyr_glom_30* to *pyr_glom_15*. The same can be done for the full trees, or the bootstrap support value trees, with little effort.

Citations

Abrams, Z. B., Johnson, T. S., Huang, K., Payne, P. R. O., & Coombes, K. (2019). A protocol to evaluate RNA sequencing normalization methods. BMC Bioinformatics, 20(Suppl 24). https://doi.org/10.1186/s12859-019-3247-x

Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. Web. https://anaconda.com.

Anderson, M. J. (2001). A new method for non-parametric multivariate analysis of variance. Austral Ecology, 26(1), 32–46. https://doi.org/10.1111/j.1442-9993.2001.01070.pp.x

Badri, M., Kurtz, Z. D., Bonneau, R., & Müller, C. L. (2018). Shrinkage improves estimation of microbial associations under different normalization methods [Preprint]. Bioinformatics. https://doi.org/10.1101/406264

Callahan, B. J., McMurdie, P. J., Rosen, M. J., Han, A. W., Johnson, A. J. A., & Holmes, S. P. (2016). DADA2: High-resolution sample inference from Illumina amplicon data. Nature Methods, 13(7), 581–583. https://doi.org/10.1038/nmeth.3869

Chao, A. (2006). Species Estimation and Applications. https://doi.org/10.1002/0471667196.ess5051

Clarke, K. R. (1993). Non-parametric multivariate analyses of changes in community structure. Australian Journal of Ecology, 18(1), 117–143. https://doi.org/10.1111/j.1442-9993.1993.tb00438.x

Johnson, J. S., Spakowicz, D. J., Hong, B.-Y., Petersen, L. M., Demkowicz, P., Chen, L., Leopold, S. R., Hanson, B. M., Agresta, H. O., Gerstein, M., Sodergren, E., & Weinstock, G. M. (2019). Evaluation of 16S rRNA gene sequencing for species and strain-level microbiome analysis. Nature Communications, 10(1), 5029. https://doi.org/10.1038/s41467-019-13036-1

Kruskal, W. H., & Wallis, W. A. (1952). Use of Ranks in One-Criterion Variance Analysis. Journal of the American Statistical Association, 47(260), 583–621. https://doi.org/10.1080/01621459.1952.10483441

Leinonen, Rasko et al. "The sequence read archive." Nucleic acids research vol. 39,Database issue (2011): D19-21. doi:10.1093/nar/gkq1019

Love, M., Ahlmann-Eltze, C., Forbes, K., Anders, S., & Huber, W. (2021). DESeq2: Differential gene expression analysis based on the negative binomial distribution (1.30.0) [Computer software]. Bioconductor version: Release (3.12). https://doi.org/10.18129/B9.bioc.DESeq2

Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8

McMurdie, P. J., & Holmes, S. (2013). phyloseq: An R Package for Reproducible Interactive Analysis and Graphics of Microbiome Census Data. PLOS ONE, 8(4), e61217. https://doi.org/10.1371/journal.pone.0061217

McMurdie, P. J., & Holmes, S. (2014). Waste not, want not: Why rarefying microbiome data is inadmissible. PLoS Computational Biology, 10(4), e1003531. https://doi.org/10.1371/journal.pcbi.1003531

Morris, E. K., Caruso, T., Buscot, F., Fischer, M., Hancock, C., Maier, T. S., Meiners, T., Müller, C., Obermaier, E., Prati, D., Socher, S. A., Sonnemann, I., Wäschke, N., Wubet, T., Wurst, S., & Rillig, M. C. (2014). Choosing and using diversity indices: Insights for ecological applications from the German Biodiversity Exploratories. Ecology and Evolution, 4(18), 3514–3524. https://doi.org/10.1002/ece3.1155

Oksanen, J., Kindt, R., Legendre, P., Hara, B., Simpson, G., Solymos, P., Henry, M., Stevens, H., Maintainer, H., & Oksanen@oulu, jari. (2009). The vegan Package. RPubs—Running NMDS using metaMDS in vegan. (n.d.). Retrieved January 3, 2021, from https://rpubs.com/CPEL/NMDS

Quast C, Pruesse E, Yilmaz P, Gerken J, Schweer T, Yarza P, Peplies J, Glöckner FO (2013) The SILVA ribosomal RNA gene database project: improved data processing and web-based tools.

Shannon, C. E., & Weaver, W. (1949). A Mathematical Theory of Communication. The University of Illinois Press, Urbana, 55.

Simberloff, D. (1978). Use of Rarefaction and Related Methods in Ecology. Biological Data in Water Pollution Assessment: Quantitative and Statistical Analyses. https://doi.org/10.1520/STP35663S

Simpson, E. H. (1949). Measurement of Diversity. Nature, 163(4148), 688–688. https://doi.org/10.1038/163688a0

Stephens, M. (2016). False discovery rates: A new deal. Biostatistics, kxw041. https://doi.org/10.1093/biostatistics/kxw041

Vavrek, M. J. (2011). fossil: Palaeoecological and palaeogeographical analysis tools. Palaeontologia Electronica, 14(1).

Weiss, S., Xu, Z. Z., Peddada, S., Amir, A., Bittinger, K., Gonzalez, A., Lozupone, C., Zaneveld, J. R., Vázquez-Baeza, Y., Birmingham, A., Hyde, E. R., & Knight, R. (2017). Normalization and microbial

differential abundance strategies depend upon data characteristics. Microbiome, 5(1), 27. https://doi.org/10.1186/s40168-017-0237-y

Zhu, A., Ibrahim, J. G., & Love, M. I. (2019). Heavy-tailed prior distributions for sequence count data: Removing the noise and preserving large differences. Bioinformatics, 35(12), 2084–2092. https://doi.org/10.1093/bioinformatics/bty895

Acknowledgements

Thank you to:

Dr. Kelly Sutherland, University of Oregon

Dr. Anne Thompson, Portland State University

The instructors and the 2020 Cohort of the Knight Campus Graduate Internship Program Bioinformatics Track