qiime Quantitative Insights Into Microbial Ecology

Tutorial:

Analyzing microbial communities using high-throughput 16S rRNA sequencing data.

J. Gregory Caporaso[1]*, Justin Kuczynski[2]*, Jesse Stombaugh[1]*, Kyle Bittinger[3], Frederic D. Bushman[3], Elizabeth K. Costello[1], Noah Fierer[4], Antonio Gonzalez Peña[5], Julia K. Goodrich[5], Jeff I. Gordon[6], Gavin Huttley[7], Scott T. Kelley[8], Dan Knights[5], Jeremy E. Koenig[9], Ruth E. Ley[9], Cathy A. Lozupone[1],Daniel McDonald[1], Brian D. Muegge[6], Megan Pirrung[1], Jens Reeder[1], Joel R. Sevinsky[10], Peter J. Turnbaugh[6], Will Van Treuren[1], William A. Walters[2], Jeremy Widmann[1], Tanya Yatsunenko[6], Jesse Zaneveld[2] and Rob Knight[1,11]**

1. Department of Chemistry and Biochemistry, UCB 215, University of Colorado, Boulder, CO 80309
2. Department of Molecular, Cellular and Developmental Biology, UCB 347, University of Colorado, Boulder, CO 80309
3. Department of Microbiology, Johnson Pavilion 425, University of Pennsylvania, Philadelphia, PA 19104
4. Cooperative Institute for Research in Environmental Sciences, University of Colorado, Boulder, CO 80309, USA.; Department of Ecology and Evolutionary Biology, University of Colorado, Boulder, CO 80309, USA.
5. Department of Computer Science, University of Colorado, Boulder, Colorado, USA.
6. Center for Genome Sciences, Washington University School of Medicine, St. Louis, MO 63108
7. Computational Genomics Laboratory, John Curtin School of Medical Research, The Australian National University, Canberra, Australian Capital Territory, Australia.
8. Department of Biology, San Diego State University, San Diego CA 92182
9. Department of Microbiology, Cornell University, Ithaca NY 14853
10. Luca Technologies, 500 Corporate Circle, Suite C, Golden, Colorado 80401
11. Howard Hughes Medical Institute

Table of Contents

1. Installing QIIME

To use this tutorial the user should install the following programs and set the appropriate environment variables, before downloading and installing QIIME.

1.1 Tutorial Software Dependencies

The following programs and datasets were used to generate this tutorial and it is recommended to use the same versions as shown below.

- Python 2.6
- PyCogent (from svn) with ReportLab
- PyNAST
- Numpy 1.3
- MatPlotLib 0.98.5.2
- jre1.6.0_16
- rdp_classifier-2.0
- Green_genes_core_set
- blast-2.2.21
- fasttree 2.0
- cd-hit 3.1

1.2 Installing QIIME

To install QIIME, the user must download the source code from sourceforge:

svn co https://qiime.svn.sourceforge.net/svnroot/qiime /path/to/download/qiime

1.3 Environment Variables

Make sure the following environment variable are set:

1.3.1 PATH Environment Variable

Your PATH variable should contain the following:

- /path/to/python/bin/
- /path/to/cd-hit/
- /path/to/FastTree/executable
- /path/to/blast-2.2.21/bin/
- /path/to/PyNAST/scripts/
- /path/to/jre1.6_0_16/

In the bash shell, you can use the following command (example only shows the path to python, so other softwares can be added to the PATH, using a similar approach):

- export PATH=/path/to/python/bin/:$PATH

1.3.2 PYTHONPATH Environment Variable

Your PYTHONPATH should contain the following:

- /path/to/PyCogent
- /path/to/QIIME
- /path/to/PyNAST

In the bash shell, you can use the following command (example only shows the path to QIIME, so other softwares can be added to the PYTHONPATH, using a similar approach):

- export PYTHONPATH=/path/to/QIIME/:$PYTHONPATH

1.3.3 RDP_JAR_PATH Environment Variable

The user should also define an RDP_JAR_PATH variable, since this tutorial uses the RDP Classifier:

- /path/to/rdp_classifier-2.0.jar

In the bash shell, you can use the following command:

- export RDP_JAR_PATH=/path/to/rdp_classifier-2.0.jar

1.4 Testing QIIME Install

Once the source code is downloaded, the user should test QIIME to be sure all essential software is properly installed and the correct environment variables are set.

In a terminal window the user, should cd to their qiime/test directory using the following command:

```
cd /path/to/QIIME/tests/
```

Then run the following test command:

```
python all_tests.py -v
```

If all tests run properly, then QIIME was properly installed. Three of the test scripts may fail, but for this tutorial we are not using all the supported 3rd party applications which are supported, so we can disregard these errors. The following scripts may generate errors:

1. **test_align_seqs.py** - Make sure that the cd-hit test ran properly by parsing the traceback generated in . For this tutorial we are only using PyNAST to align sequences, so we are not worrying about installing MOTHER, DOTUR or MUSCLE at this time.
2. **test_pick_otus.py** - Make sure the cd-hit tests ran properly by parsing the traceback generated in your terminal window. For this tutorial we are only using cd-hit to pick OTU's, so we are not worrying about installing BLAST at this time.
3. **test_pyronoise.py** - For this tutorial, we are not denoising the sequences, so disregard the errors generated for this script.

2. Introduction

This tutorial explains how to use the QIIME (Quantitative Insights Into Microbial Ecology) Pipeline to process data from high-throughput 16S rRNA sequencing studies. The purpose of this pipeline is to provide a start-to-finish workflow, beginning with multiplexed sequence reads and finishing with taxonomic and phylogenetic profiles and comparisons of the samples in the study. With this information in hand, it is possible to determine biological and environmental factors that alter microbial community ecology in your experiment.

As an example, we will use data from a study of the response of mouse gut microbial communities to fasting (Crawford et al., 2009). To make this tutorial run quickly on a personal computer, we will use a subset of the data generated from 5 animals kept on the control ad libitum fed diet, and 4 animals fasted for 24 hours before sacrifice. At the end of our tutorial, we will be able to compare the community structure of control vs. fasted animals. In particular, we will be able to compare taxonomic profiles for each sample type, differences in diversity metrics within the samples and between the groups, and perform comparative clustering analysis to look for overall differences in the samples.

To process our data, we will perform the following steps, each of which is described in more detail in **Section 3** below:

A) Filter the sequence reads for quality and assign multiplexed reads to starting samples by nucleotide barcode.

B) Pick Operational Taxonomic Units (OTU's) based on sequence similarity within the reads, and pick a representative sequence from each OTU.
C) Assign the OTU to a taxonomic identity using reference databases.
D) Align the OTU sequences and create a phylogenetic tree.
E) Calculate diversity metrics for each sample and compare the types of communities, using the taxonomic and phylogenetic assignments.
F) Generate UPGMA and PCoA plots to visually depict the differences between the samples, and dynamically work with these graphs to generate publication quality figures.

3. Essential Files

All the files you will need for this tutorial are here (http://bit.ly/3zGJKI). Descriptions of these files are below.

3.1 Sequences (.fna)

This is the 454-machine generated FASTA file. Using the Amplicon processing software on the 454 FLX standard, each region of the PTP plate will yield a fasta file of form "1.TCA.454Reads.fna", where "1" is replaced with the appropriate region number.

For the purposes of this tutorial, we will use the fasta file "Fasting_Example.fna".

3.2 Quality Scores (.qual)

This is the 454-machine generated quality score file, which contains a score for each base in each sequence included in the FASTA file. Like the fasta file mentioned above, the Amplicon processing software will generate one of these files for each region of the PTP plate, named "1.TCA.454Reads.qual", etc.

For the purposes of this tutorial, we will use the quality scores file "Fasting_Example.qual".

3.3 Mapping File (Tab-delimited .txt)

The mapping file is generated by the user. This file contains all of the information about the samples necessary to perform the data analysis. At a minimum, the mapping file should contain the name of each sample, the barcode sequence used for each sample and a Description column. In general, you should also include in the mapping file any metadata that relates to the samples (for instance, health status or sampling site) and any additional information relating to specific samples that may be useful to have at hand when considering outliers (for example, what medications a patient was taking at time of sampling). Full format specifications can be found in the **Documentation**.

You are highly encouraged to validate your mapping file using ***check_id_map.py*** before attempting to analyze your data. This tool will check for errors, and make suggestions for other aspects of the file to be edited (errors and warnings are output to STDERR).

For the purposes of this tutorial, we will use the mapping file "Fasting_Map.txt". The contents of the mapping file are shown here - as you can see, a nucleotide barcode sequence is provided for each of the 9 samples, as well as metadata related to treatment group and date of birth, and general run descriptions about the project.

Fasting_Map.txt file contents:

```
#SampleID      BarcodeSequence       Treatment      DOB            Description
#Example mapping file for the QIIME analysis package.  These 9 samples are from a study of the effects of
exercise and diet on mouse cardiac physiology (Crawford, et al, PNAS, 2009).
PC.354         AGCACGAGCCTA          Control        20061218       Control mouse, I.D. 354
PC.355         AACTCGTCGATG          Control        20061218       Control mouse, I.D. 355
PC.356         ACAGACCACTCA          Control        20061126       Control mouse, I.D. 356
PC.481         ACCAGCGACTAG          Control        20070314       Control mouse, I.D. 481
PC.593         AGCAGCACTTGT          Control        20071210       Control mouse, I.D. 593
PC.607         AACTGTGCGTAC          Fast           20071112       Fasting mouse, I.D. 607
PC.634         ACAGAGTCGGCT          Fast           20080116       Fasting mouse, I.D. 634
PC.635        ACCGCAGAGTCA          Fast          20080116      Fasting mouse, I.D. 635
PC.636        ACGGTGAGTGTC          Fast          20080116      Fasting mouse, I.D. 636
```

3.4 Flowgram File (.sff) - (Optional)

This is the 454-machine generated file which stores the sequencing trace data. This is the largest file returned from a 454 run. The sffinfo command in the 454 software package can be used to generate sequence and quality files from sff file(s) as follows:

To generate a fasta file:

```
$ sffinfo -s NAME_OF_SFF_FILES > OUTPUT_NAME.fna
```

To generate a quality score file:

```
$ sffinfo -s NAME_OF_SFF_FILES >OUTPUT_NAME.qual
```

4. Data Analysis Steps

In this walkthrough, invocation of scripts will be noted with indentation and the $ sign signifying the start of a command line. You can find full usage information for each script by passing the -h option (help) and/or by reading the full description in the **Documentation**.

First, assemble the sequences (.fna), quality scores (.qual), and metadata mapping file into a directory. Execute all tutorial commands from within this directory. To access QIIME python scripts, it may be useful to set an environment variable to the location of the innermost QIIME directory (the one containing ***check_id_map.py***, for example):

```
$ qdir=/path/to/QIIME
```

then further commands of the form "python QIIMEscript.py -o option" can be invoked as "python $qdir/QIIMEscript.py -o option".

4.1 Pre-processing 454 Data

Filter the reads based on quality, and assign multiplexed reads to starting sample by nucleotide barcode.

4.1.1 Check Mapping File

Before beginning the pipeline, you should ensure that your mapping file is formatted correctly with the ***check_map_id.py*** script.

```
$ python $qdir/check_id_map.py -b 1 -m Fasting_Map.txt
```

This utility will print to STDERR any "Errors" or "Warnings" found in your file. Errors will cause fatal problems with subsequent scripts and must be corrected before moving forward. Warnings will not cause fatal problems, but it is encouraged that you fix these problems as they are often indicative of typos in your mapping file or other unintended errors that will impact downstream analysis.

The following is the output from check_map.py, which contains no "Errors" or "Warnings":

```
Reading id map:Fasting_Map.txt
#SampleID     BarcodeSequence       Treatment     DOB           Description
#Example mapping file for the QIIME analysis package.  These 9 samples are from a study of the effects of
exercise and diet on mouse cardiac physiology (Crawford, et al, PNAS, 2009).
PC.354        AGCACGAGCCTA          Control       20061218      Control mouse, I.D. 354
PC.355        AACTCGTCGATG          Control       20061218      Control mouse, I.D. 355
PC.356        ACAGACCACTCA          Control       20061126      Control mouse, I.D. 356
PC.481        ACCAGCGACTAG          Control        20070314       Control mouse, I.D. 481
PC.593        AGCAGCACTTGT          Control        20071210       Control mouse, I.D. 593
PC.607        AACTGTGCGTAC          Fast           20071112       Fasting mouse, I.D. 607
PC.634        ACAGAGTCGGCT          Fast           20080116       Fasting mouse, I.D. 634
PC.635        ACCGCAGAGTCA          Fast           20080116       Fasting mouse, I.D. 635
PC.636        ACGGTGAGTGTC          Fast           20080116       Fasting mouse, I.D. 636
```

4.1.2 Assign Samples to Mulitplex Reads

The next task is to assign the mulitplex reads to samples based on their nucleotide barcode. Also, this step performs quality filtering based on the characteristics of each sequence, removing any low quality or ambiguous reads. The script for this step is ***split_libraries.py***. A full description of parameters for this script are described in the **Documentation**. For this tutorial, we will use default parameters (minimum quality score = 25, minimum/maximum length = 200/1000, no ambiguous bases allowed and no mismatches allowed in the primer sequence).

```
$ python $qdir/split_libraries.py -m Fasting_Map.txt -f Fasting_Example.fna -q Fasting_Example.qual -d
    Fasting_Split_Lib_Out
```

This invocation will create three files in the new directory "Fasting_Split_Lib_Out":

- split_library_log.txt : This file contains the summary of splitting, including the number of reads detected for each sample and a brief summary of any reads that were removed due to quality considerations.
- histograms.txt: This tab delimited file shows the number of reads at regular size intervals before and after splitting the library.
- seqs.fna : This is a fasta formatted file where each sequence is renamed according to the sample it came from. The header line also contains the name of the read in the input fasta file and information on any barcode errors that were corrected.

A few lines from the "seqs.fna" file are shown below:

```
>PC.634_1 FLP3FBN01ELBSX orig_bc=ACAGAGTCGGCT new_bc=ACAGAGTCGGCT bc_diffs=0
CTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTTACCCTCTCAGGCCGGCTACGCATCATCGCC....
>PC.634_2 FLP3FBN01EG8AX orig_bc=ACAGAGTCGGCT new_bc=ACAGAGTCGGCT bc_diffs=0
TTGGACCGTGTCTCAGTTCCAATGTGGGGGCCTTCCTCTCAGAACCCCTATCCATCGAAGGCTT....
>PC.354_3 FLP3FBN01EEWKD orig_bc=AGCACGAGCCTA new_bc=AGCACGAGCCTA bc_diffs=0
TTGGGCCGTGTCTCAGTCCCAATGTGGCCGATCAGTCTCTTAACTCGGCTATGCATCATTGCCTT....
>PC.481_4 FLP3FBN01DEHK3 orig_bc=ACCAGCGACTAG new_bc=ACCAGCGACTAG bc_diffs=0
CTGGGCCGTGTCTCAGTCCCAATGTGGCCGTTCAACCTCTCAGTCCGGCTACTGATCGTCGACT....
```

4.2 Pick Operational Taxonomic Units (OTU's) and Representative Sequences

4.2.1 Pick OTU's based on Sequence Similarity within the Reads

At this step, all of the sequences from all of the samples will be clustered into Operational Taxonomic Units (OTU's) based on their sequence similarity. OTUs in QIIME are clusters of sequences, frequently intended to represent some degree of taxonomic relatedness. For example, when sequences are clustered at 97% sequence similarity with cd-hit, each resulting cluster is typically thought of as representing a genus. This model and the current techniques for picking OTUs are known to be flawed, and determining exactly how OTUs should be defined, and what they represent, is an active area of research. Thus, OTU-picking will identify highly similar sequences across the samples and provide a platform for comparisons of community structure. The script ***pick_otus.py*** takes as input the fasta file output from **step 3.A2** above, and returns a list of OTU's detected and the fasta header for sequences that belong in that OTU. To invoke the script using cd-hit to cluster and the default setting of 96% similarity determining an OTU, use:

```
$ python $qdir/pick_otus.py -i Fasting_Split_Lib_Out/seqs.fna -m cdhit -o Fasting_picked_otus
```

In the newly created directory "Fasting_picked_otus", there will be two files. One is "seqs.log", which contains information about the invocation of the script. The OTU's will be recorded in the tab-delimited file "seqs_otus.txt". The OTU's are arbitrarily named by a number, which is recorded in the first column. The subsequent columns in each line identify the sequence or sequences that belong in that OTU.

4.2.2 Pick Representative Sequences for each OTU

Since each OTU may be made up of many sequences, we will pick a representative sequence for that OTU for downstream analysis. This representative sequence will be used for taxonomic identification of the OTU and phylogenetic alignment. The script ***pick_rep_set.py*** uses the otu file created above and extracts a representative sequence from the fasta file by one of several methods. To use the default method, where the most abundant sequence in the OTU is used as the representative sequence, invoke the script as follows:

```
$ python $qdir/pick_rep_set.py -i Fasting_picked_otus/seqs_otus.txt -f Fasting_Split_Lib_Out/seqs.fna -l
    Fasting_picked_otus/rep_set.log -o Fasting_picked_otus/rep_set.fa
```

In the directory "Fasting_picked_otus", the script has created two new files - the log file "rep_set.log" and the fasta file "rep_set.fa" containing one representative sequence for each OTU. In this fasta file, the sequence has been renamed by the OTU, and the additional information on the header line reflects the sequence used as the representative:

```
>0 PC.636_424
CTGGGCCGTATCTCAGTCCCAATGTGGCCGGTCGACCTCTC....
>1 PC.481_321
TTGGGCCGTGTCTCAGTCCCAATGTGGCCGTCCGCCCTCTC....
```

4.3 Assign Taxonomy and Perform OTU Table Analyses

In this section, we will assign the representative sequences for each OTU to a taxonomic identity using reference databases, and make an OTU table. Once the OTU table is generated, we show some ways to visualize the table.

4.3.1 Assign Taxonomy

A primary goal of the QIIME pipeline is to assign high-throughput sequencing reads to taxonomic identities using established databases. This will give you information on the microbial lineages found in your samples. Using ***assign_taxonomy.py***, you can compare your OTU's against a reference database of your choosing. For our example, we will use the RDP classification system.

```
$ python $qdir/assign_taxonomy.py -i Fasting_picked_otus/rep_set.fa -m rdp -o Fasting_rdp_taxonomy
```

In the directory "Fasting_rdp_taxonomy", there will be a log file and a text file. The text file contains a line for each OTU considered, with the RDP taxonomy assignment and a numerical confidence of that assignment (1 is the highest possible confidence). For some OTU's, the assignment will be as specific as a bacterial species, while others may be assignable to nothing more specific than the bacterial domain. Below are the first few lines of the text file and the user should note that the taxonomic assignment and confidence numbers from their run may not coincide with the output shown below, due to the RDP classification algorithm:

```
41      PC.356_347     Root;Bacteria                                                            0.980
63      PC.635_130     Root;Bacteria;Firmicutes;"Clostridia";Clostridiales;"Lachnospiraceae"    0.960
353     PC.634_150     Root;Bacteria;Proteobacteria;Deltaproteobacteria                         0.880
18      PC.355_1011    Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;Rikenellaceae;Alistipes   0.990
```

4.3.2 Make OTU Table

Using these assignments and the OTU file created in **step B**, we can make a readable matrix of OTU by Sample with meaningful taxonomic identifiers for each OTU. The script is ***make_otu_table.py***.

```
$ python $qdir/make_otu_table.py -i Fasting_picked_otus/seqs_otus.txt -t
    Fasting_rdp_taxonomy/rep_set_tax_assignments.txt -o Fasting_otu_table.txt
```

The first few lines of "Fasting_otu_table.txt" are shown below (OTUs 2-9), where the first column contains the OTU number, the last column contains the taxonomic assignment for the OTU, and 9 columns between are for each of our 9 samples. The value of each *ij* entry in the matrix is the number of times OTU *i* was found in the sequences for sample *j*.

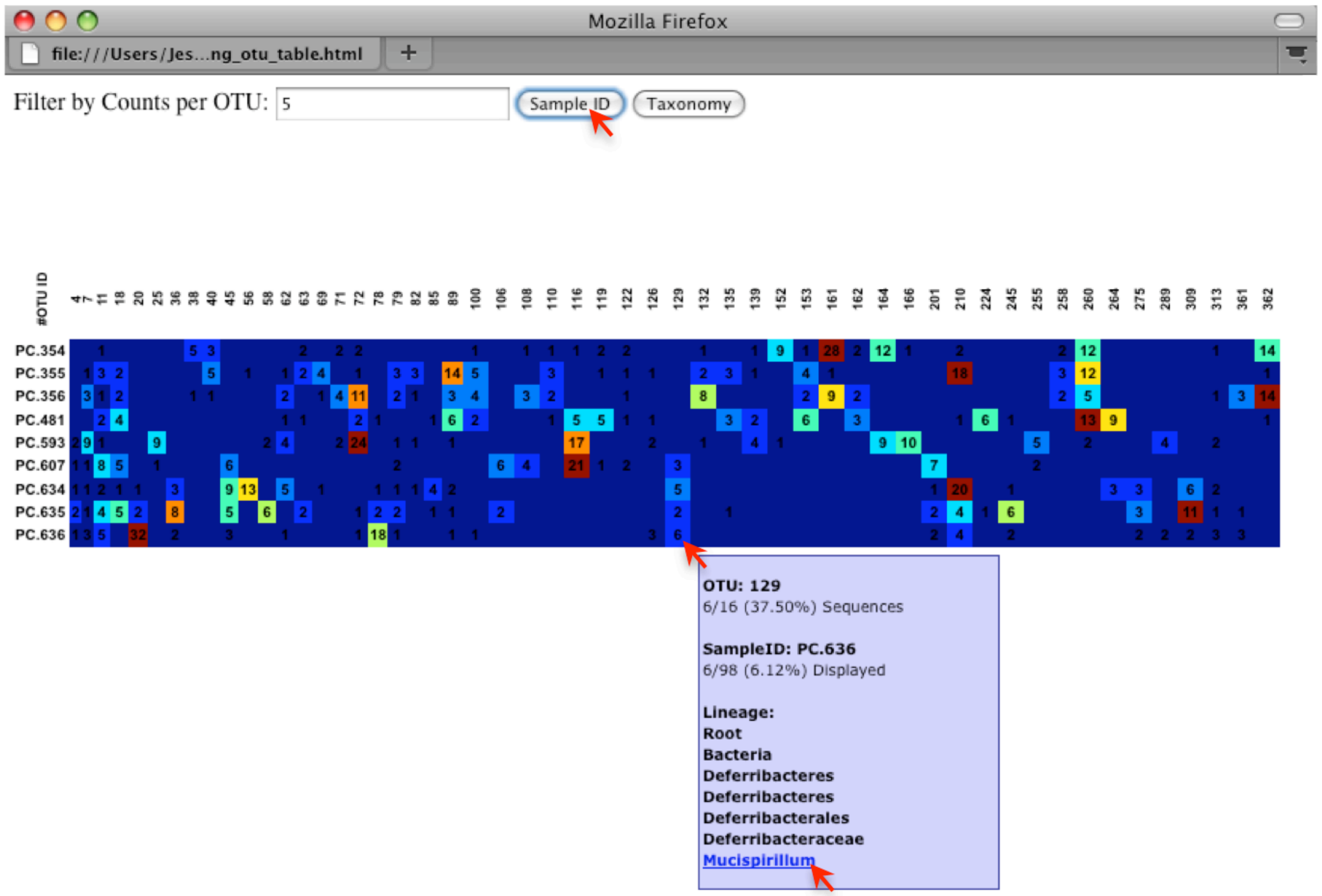
```
#Full OTU Counts
#OTU ID PC.354  PC.355  PC.356  PC.481  PC.593  PC.607  PC.634  PC.635  PC.636  Consensus Lineage
2       0       0       0       0       0       1       0       3       1       Root;Bacteria;Bacteroidetes
3       0       0       0       0       0       0       0       0       1       Root;Bacteria
4       0       0       0       0       2       1       1       2       1       Root;Bacteria
5       0       0       2       0       0       0       0       1       0       Root
6       0       0       0       0       0       0       0       1       0       Root;Bacteria
7       0       1       3       0       9       1       1       1       3       Root;Bacteria;Bacteroidetes
8       0       0       0       0       0       1       0       0       0       Root;Bacteria;Bacteroidetes
9       0       0       0       0       0       0       0       0       1       Root;Bacteria;Bacteroidetes
```

4.3.3 Make OTU Heatmap

The QIIME pipeline includes a very useful utility to generate images of the OTU table. The script is ***make_otu_heatmap_html.py***:

```
$ python $qdir/make_otu_heatmap_html.py -i Fasting_otu_table.txt -o Fasting_OTU_Heatmap/
```

An html file is created in the directory "Fasting_OTU_Heatmap". You can open this file with any web browser, and will be prompted to enter a value for "Filter by Counts per OTU". Only OTU's with total OTU counts at or above this threshold will be displayed. Click the "Sample ID" button, and a graphic will be generated like the figure below. For each sample, you will see in a heatmap the number of times each OTU was found in that sample. You can mouse over any individual count to get more information on the OTU (including taxonomic assignment). Within the mouseover, there is a link for the terminal lineage assignment, so you can easily search Google for more information about that assignment.

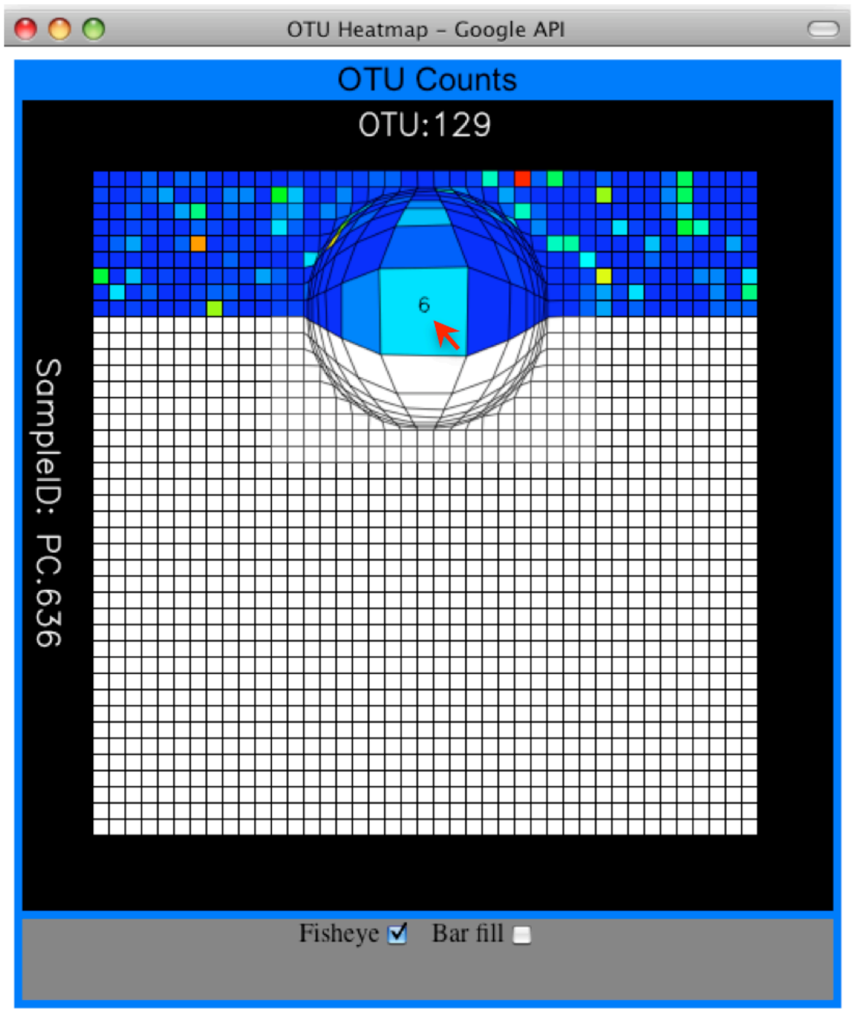


OTU: 129
6/16 (37.50%) Sequences

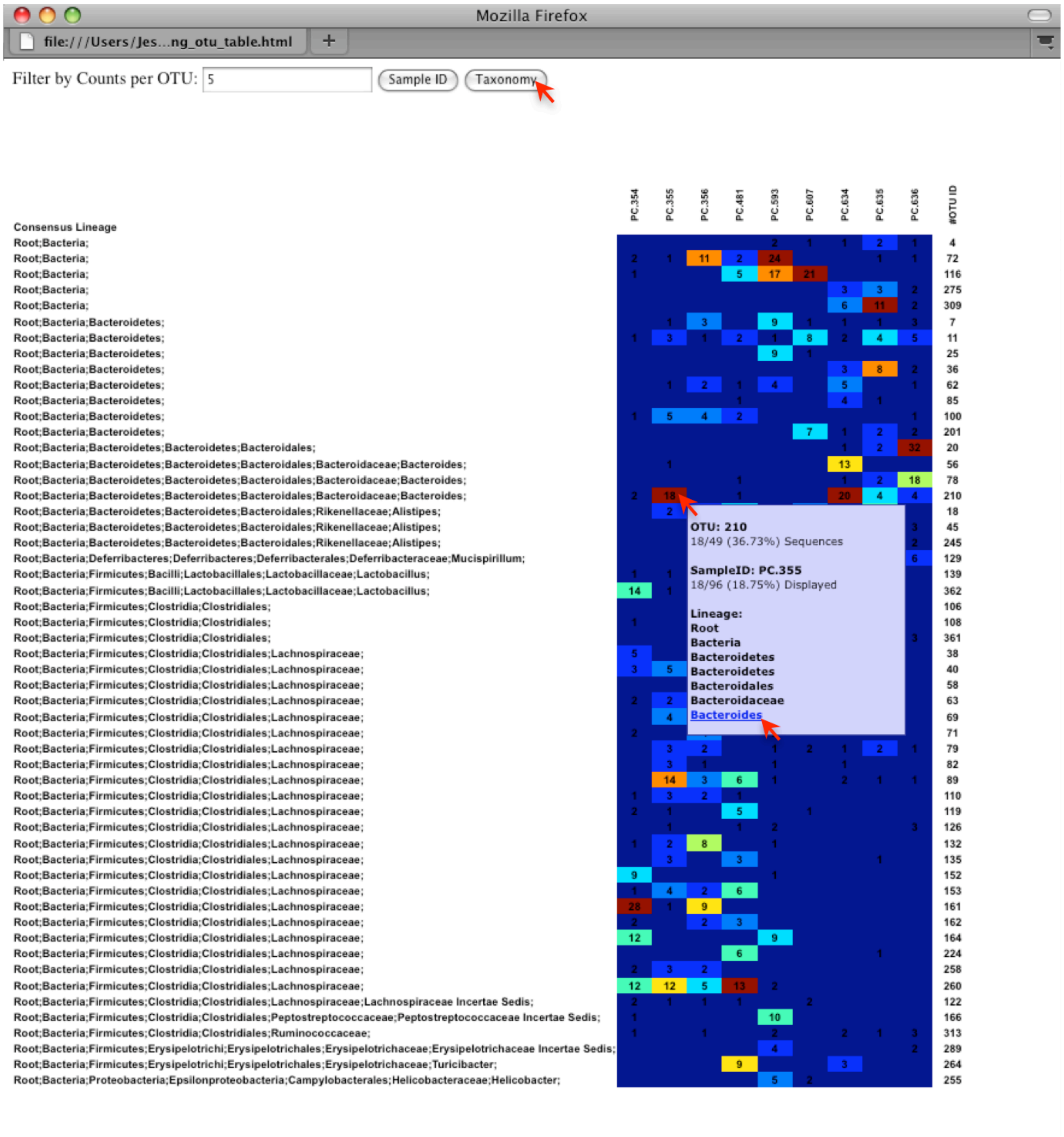
SampleID: PC.636
6/98 (6.12%) Displayed

Lineage:
Root
Bacteria
Deferribacteres
Deferribacteres
Deferribacterales
Deferribacteraceae
Mucispirillum

Alternatively, you can click on one of the counts in the heatmap and a new pop-up window will appear. The pop-up window uses a Google Visualization API called Magic-Table. Depending on which table count you clicked on, the pop-up window will put the clicked-on count in the middle of the pop-up heatmap as shown below. For the following example, the table count with the red arrow mouseover is the same one being focused on using the Magic-Table.



On the original heatmap webpage, if you select the "Taxonomy" button instead, you will generate a heatmap keyed by taxon assignment, which allows you to conveniently look for organisms and lineages of interest in your study. Again, mousing over an individual count will show additional information for that OTU and sample.



Filter by Counts per OTU: 5

Consensus Lineage	PC.354	PC.355	PC.356	PC.481	PC.593	PC.607	PC.634	PC.635	PC.636	#OTU ID
Root;Bacteria;					2	1	1	2	1	4
Root;Bacteria;	2	1	11	2	24			1	1	72
Root;Bacteria;	1			5	17	21				116
Root;Bacteria;							3	3	2	275
Root;Bacteria;							6	11	2	309
Root;Bacteria;Bacteroidetes;		1	3		9	1	1	1	3	7
Root;Bacteria;Bacteroidetes;	1	3	1	2	1	8	2	4	5	11
Root;Bacteria;Bacteroidetes;					9	1				25
Root;Bacteria;Bacteroidetes;							3	8	2	36
Root;Bacteria;Bacteroidetes;		1	2	1	4		5		1	62
Root;Bacteria;Bacteroidetes;				1			4	1		85
Root;Bacteria;Bacteroidetes;	1	5	4	2					1	100
Root;Bacteria;Bacteroidetes;						7	1	2	2	201
Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;							1	2	32	20
Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;Bacteroidaceae;Bacteroides;		1					13			56
Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;Bacteroidaceae;Bacteroides;				1			1	2	18	78
Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;Bacteroidaceae;Bacteroides;	2	18		1			20	4	4	210
Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;Rikenellaceae;Alistipes;		2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		18
Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;Rikenellaceae;Alistipes;			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	3	45
Root;Bacteria;Bacteroidetes;Bacteroidetes;Bacteroidales;Rikenellaceae;Alistipes;			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2	245
Root;Bacteria;Deferribacteres;Deferribacteres;Deferribacterales;Deferribacteraceae;Mucispirillum;			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	6	129
Root;Bacteria;Firmicutes;Bacilli;Lactobacillales;Lactobacillaceae;Lactobacillus;	1	1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		139
Root;Bacteria;Firmicutes;Bacilli;Lactobacillales;Lactobacillaceae;Lactobacillus;	14	1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		362
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		106
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;	1		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		108
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	3	361
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	5		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		38
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	3	5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		40
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;			[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		58
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	2	2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		63
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;		4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		69
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	2		[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		71
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;		3	2		1	2	1	2	1	79
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;		3	1		1		1			82
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;		14	3	6	1		2	1	1	89
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	1	3	2	1						110
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	2	1		5		1				119
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;		1		1	2				3	126
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	1	2	8		1					132
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;		3		3				1		135
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	9				1					152
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	1	4	2	6						153
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	28	1	9							161
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	2		2	3						162
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	12				9					164
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;				6				1		224
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	2	3	2							258
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;	12	12	5	13	2					260
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Lachnospiraceae;Lachnospiraceae Incertae Sedis;	2	1	1	1		2				122
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Peptostreptococcaceae;Peptostreptococcaceae Incertae Sedis;	1				10					166
Root;Bacteria;Firmicutes;Clostridia;Clostridiales;Ruminococcaceae;	1		1		2		2	1	3	313
Root;Bacteria;Firmicutes;Erysipelotrichi;Erysipelotrichales;Erysipelotrichaceae;Erysipelotrichaceae Incertae Sedis;					4				2	289
Root;Bacteria;Firmicutes;Erysipelotrichi;Erysipelotrichales;Erysipelotrichaceae;Turicibacter;				9			3			264
Root;Bacteria;Proteobacteria;Epsilonproteobacteria;Campylobacterales;Helicobacteraceae;Helicobacter;					5	2				255

OTU: 210
18/49 (36.73%) Sequences

SampleID: PC.355
18/96 (18.75%) Displayed

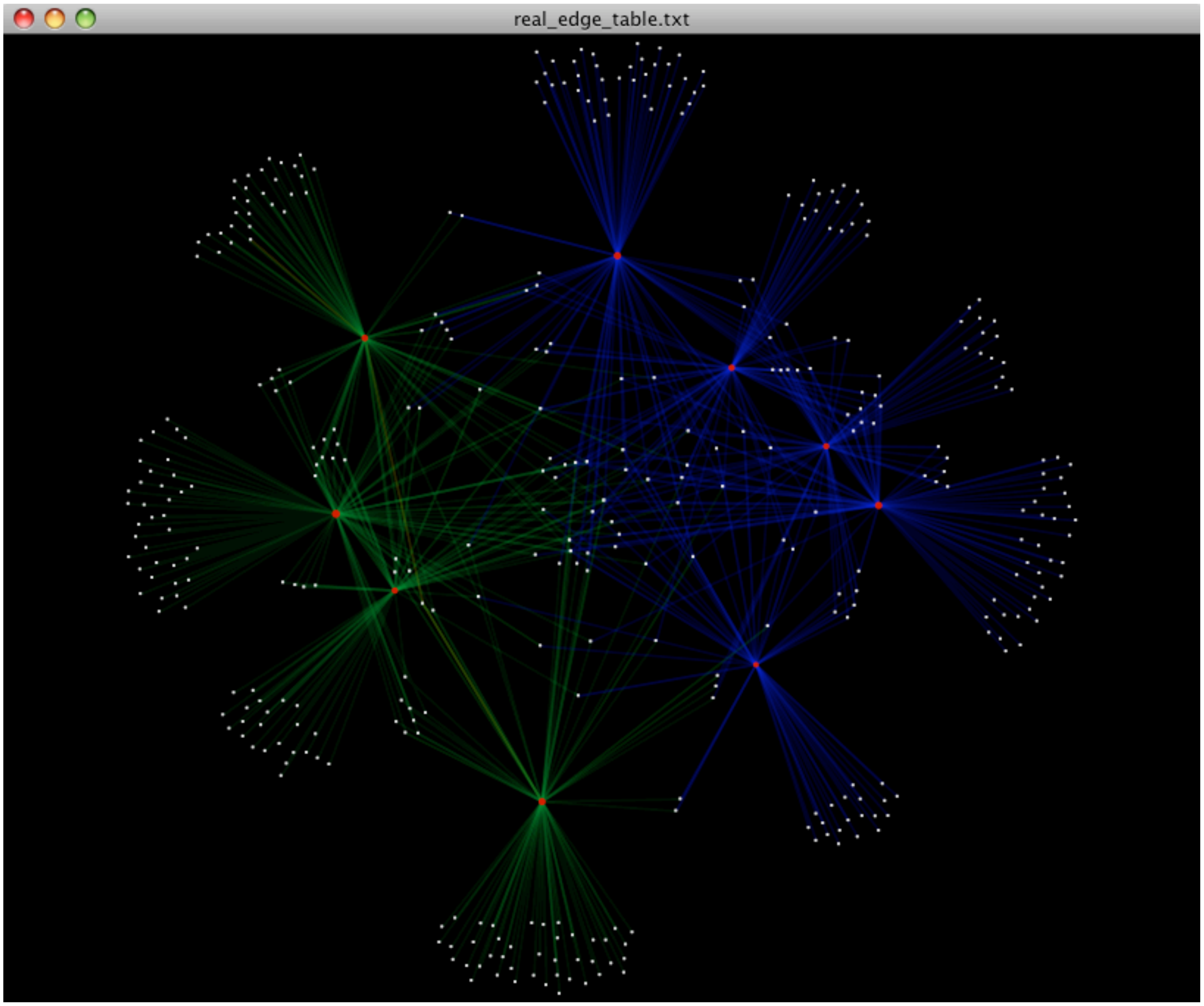
Lineage:
Root
Bacteria
Bacteroidetes
Bacteroidetes
Bacteroidales
Bacteroidaceae
Bacteroides

4.3.4 Make OTU Network

An alternative to viewing the OTU table as a heatmap, is to create an OTU network, using the following command.

```
$ python $qdir/make_otu_network.py -i Fasting_Map.txt -c Fasting_otu_table.txt -o OTU_Network
```

To visualize the network, we use the Cytoscape program, where each red circle represents a sample and each white square represents an otu. The lines represent the OTU's present in a particular sample (blue for controls and green for fasting). For more information about opening the files in Cytoscape please refer to **Documentation.**



You can group OTUs by different taxonomic levels (division, class, family) with the script ***summarize_taxa.py***. The input is the OTU table created above and the taxonomic level you need to group the OTUs. For the RDP taxonomy, the following taxonomic levels correspond to: 2 = Domain (Bacteria), 3 = Phylum (Actinobacteria), 4 = Class, and so on.

```
$ python $qdir/summarize_taxa.py -O Fasting_otu_table.txt -o Fasting_otu_table_Level3.txt -L 3 -r 0
```

The script will generate a new OTU table "Fasting_otu_table_Level3.txt", where the value of each *ij* entry in the matrix is the count of the number of times all OTUs belonging to the taxon *i* (for example, Phylum Actinobacteria) were found in the sequences for sample *j*.

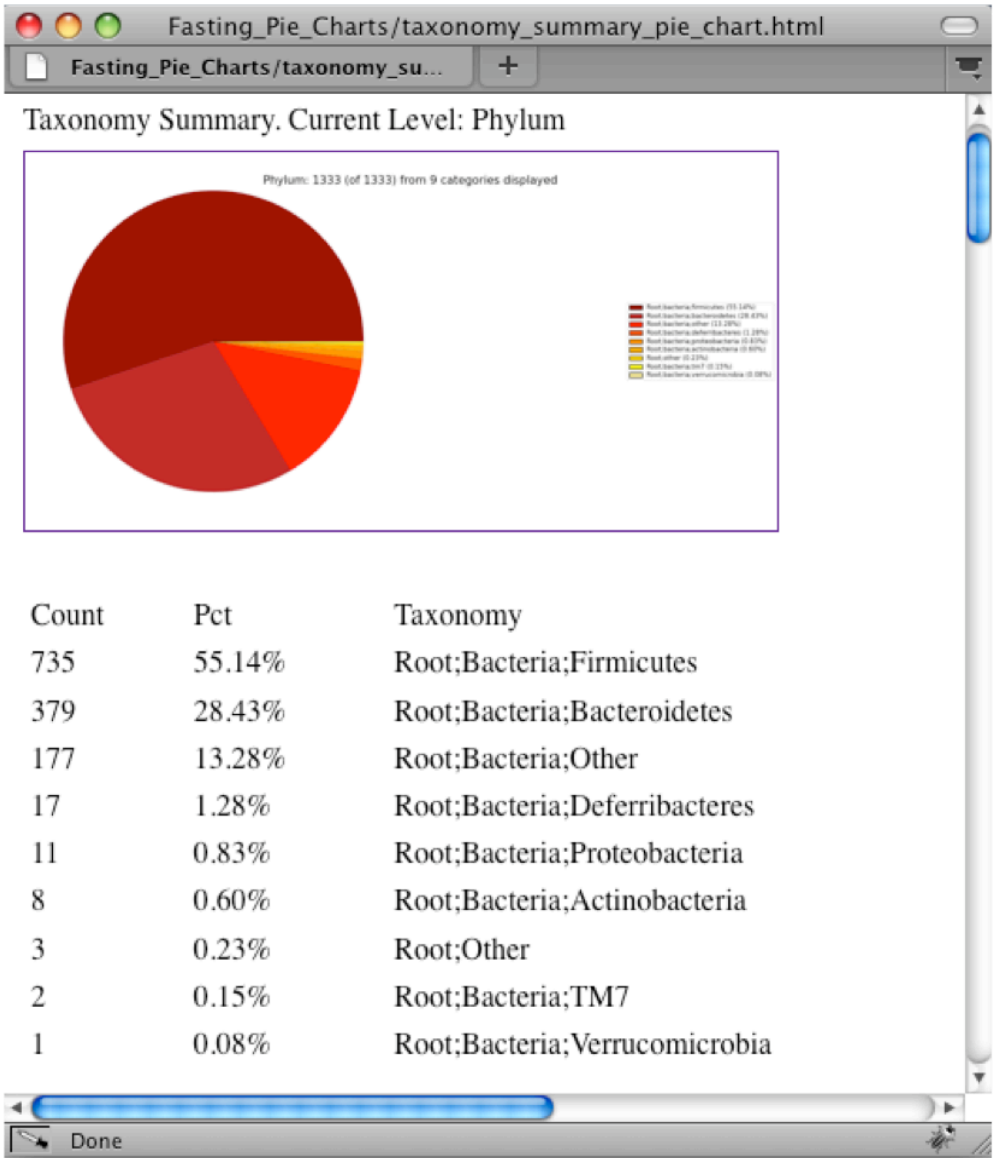
```
#Full OTU Counts
Taxon                          PC.354 PC.355  PC.356  PC.481  PC.593  PC.607  PC.634  PC.635  PC.636
Root;Bacteria;Actinobacteria   0.0    0.0     0.0     1.0     0.0     2.0     3.0     1.0     1.0
Root;Bacteria;Bacteroidetes    7.0    38.0    15.0    19.0    30.0    40.0    86.0    54.0    90.0
Root;Bacteria;Deferribacteres  0.0    0.0     0.0     0.0     0.0     3.0     5.0     2.0     7.0
Root;Bacteria;Firmicutes       136.0  102.0   115.0   117.0   65.0    66.0    37.0    63.0    34.0
Root;Bacteria;Other            5.0    6.0     18.0    9.0     49.0    35.0    14.0    27.0    14.0
Root;Bacteria;Proteobacteria   0.0    0.0     0.0     0.0     5.0     3.0     2.0     0.0     1.0
Root;Bacteria;TM7              0.0    0.0     0.0     0.0     0.0     0.0     2.0     0.0     0.0
Root;Bacteria;Verrucomicrobia  0.0    0.0     0.0     0.0     0.0     0.0     1.0     0.0     0.0
Root;Other                     0.0    0.0     2.0     0.0     0.0     0.0     0.0     1.0     0.0
```

4.3.5 Make Pie Charts

To visualize the summarized taxa, you can use the make_pie_charts.py script, which shows which taxons are present in all samples or within each sample (-s).

```
$ python $qdir/make_pie_charts.py -i Fasting_otu_table_Level3.txt -l Phylum -o Fasting_Pie_Charts –s
```

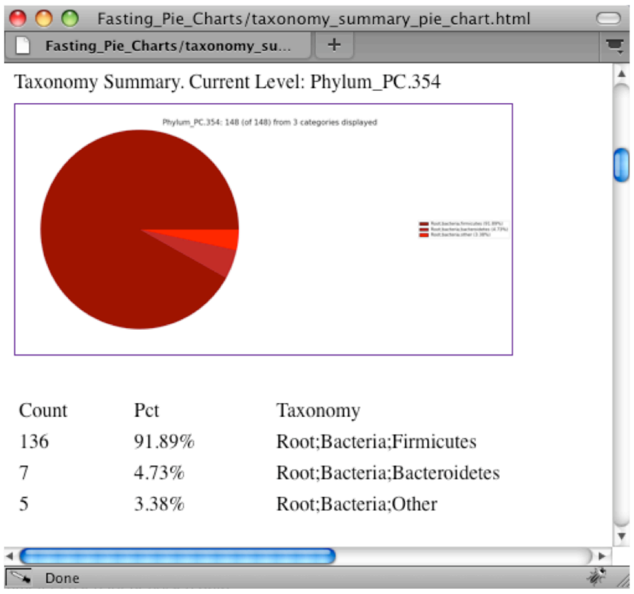
To view the resulting pie charts, open the html file located in the "Fasting_Pie_Charts" folder. The following pie chart shows the taxa assignments for all samples.


Fasting_Pie_Charts/taxonomy_summary_pie_chart.html
Fasting_Pie_Charts/taxonomy_su...
Phylum: 1333 (of 1333) from 9 categories displayed
Root;bacteria;firmicutes (55.14%)
Root;bacteria;bacteroidetes (28.43%)
Root;bacteria;other (13.28%)
Root;bacteria;deferribacteres (1.28%)
Root;bacteria;proteobacteria (0.83%)
Root;bacteria;actinobacteria (0.60%)
Root;other (0.23%)
Root;bacteria;tm7 (0.15%)
Root;bacteria;verrucomicrobia (0.08%)
Done

Taxonomy Summary. Current Level: Phylum

Count	Pct	Taxonomy
735	55.14%	Root;Bacteria;Firmicutes
379	28.43%	Root;Bacteria;Bacteroidetes
177	13.28%	Root;Bacteria;Other
17	1.28%	Root;Bacteria;Deferribacteres
11	0.83%	Root;Bacteria;Proteobacteria
8	0.60%	Root;Bacteria;Actinobacteria
3	0.23%	Root;Other
2	0.15%	Root;Bacteria;TM7
1	0.08%	Root;Bacteria;Verrucomicrobia

The following pie chart shows the taxa assignments for one of the samples (PC.354).


Fasting_Pie_Charts/taxonomy_summary_pie_chart.html
Fasting_Pie_Charts/taxonomy_su...
Phylum_PC.354: 148 (of 148) from 3 categories displayed
Root;bacteria;firmicutes (91.89%)
Root;bacteria;bacteroidetes (4.73%)
Root;bacteria;other (3.38%)
Done

Taxonomy Summary. Current Level: Phylum_PC.354

Count	Pct	Taxonomy
136	91.89%	Root;Bacteria;Firmicutes
7	4.73%	Root;Bacteria;Bacteroidetes
5	3.38%	Root;Bacteria;Other

4.4 Align OTU Sequences, Filter the Alignment and Generate a Phylogenetic Tree

4.4.1 Align OTU Sequences

Alignment of the sequences and phylogeny inference is necessary only if phylogenetic tools such as Unifrac will be subsequently invoked. Alignments can either be generated de novo using programs such as MUSCLE, or through assignment to an existing alignment with tools like PyNAST. For small studies such as this tutorial, either method is possible. However, for studies involving many sequences (roughly, more than 1000), the de novo aligners are very slow and assignment with PyNAST is preferred. Either alignment approach is accomplished with the script ***align_seqs.py***. Since this is one of the most computationally intensive bottlenecks in the pipeline, large studies would benefit greatly from parallelization of this task (described in detail in the **Documentation**)

```
$ python $qdir/align_seqs.py -i Fasting_picked_otus/rep_set.fa -o Default_Aln -t
    /path/to/Green_genes_core_set/core_set_aligned.fasta.imputed -e 150
```

A log file and an alignment file are created in the directory "Default_Aln".

4.4.2 Filter Alignment

Before building the tree, one must filter the alignment to removed columns comprised of only gaps.

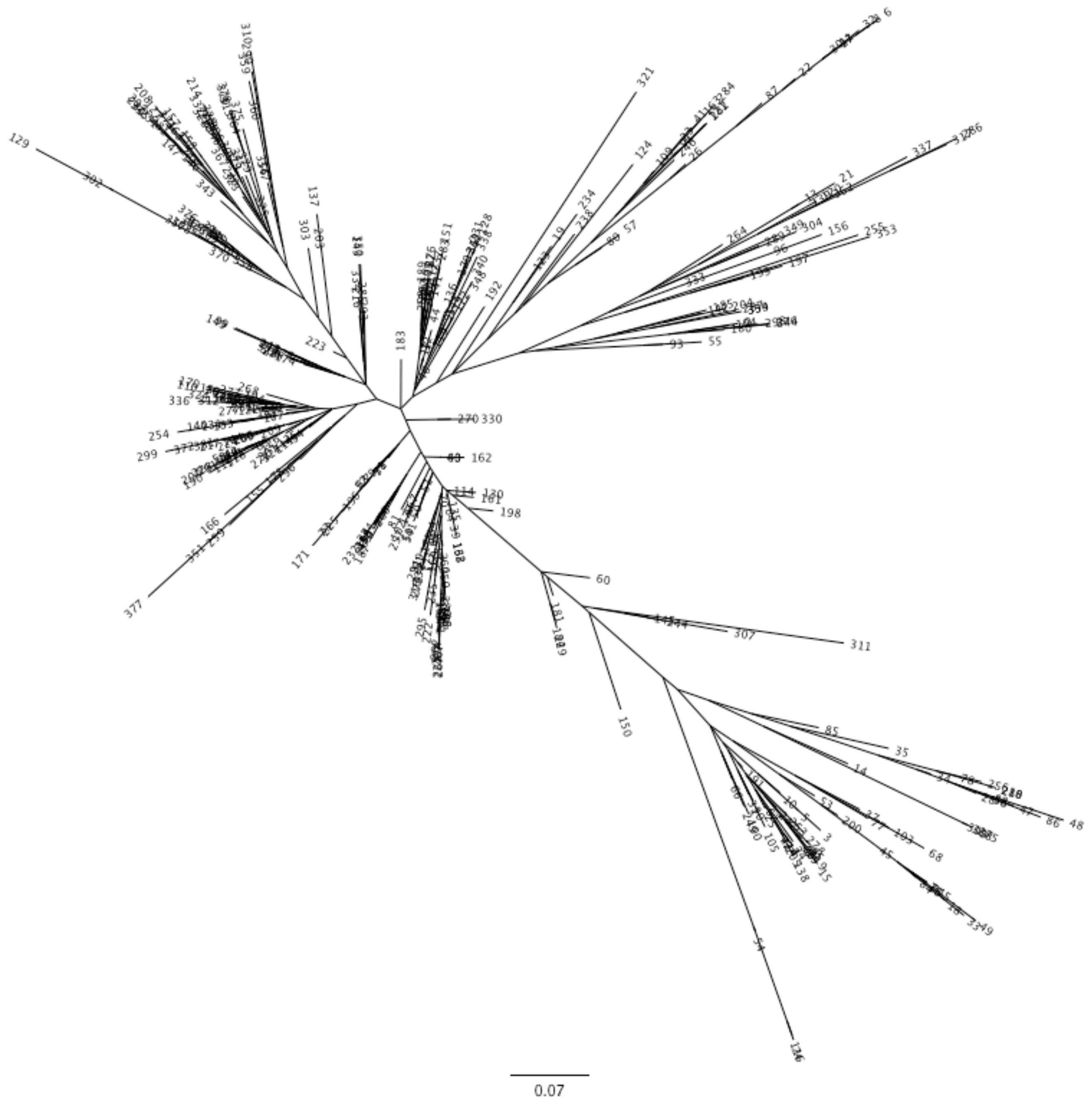
```
$ python $qdir/filter_alignment.py -m /path/to/Green_genes_core_set/lanemask_in_1s_and_0s.txt -i
    Default_Aln/rep_set_aligned.fa -o Filtered_Aln
```

4.4.3 Make Phylogenetic Tree

The filtered alignment file produced in the directory "Filtered_Aln" can be used to build a phylogenetic tree using a tree-building program. Here is an example of how to create a tree with the program FastTree and the script ***make_phylogeny.py*** :

```
$ python $qdir/make_phylogeny.py -t fasttree -o Filtered_Aln/rep_set_aligned.tre -l Filtered_Aln/rep_set_tree.log -
    i Filtered_Aln/rep_set_aligned_pfiltered.fasta
```

The Newick format tree file is written to " Filtered_Aln/rep_set_aligned.tre". This file can be used in tree visualization software, and is necessary for UniFrac diversity measurements (described below). For the following example the FigTree program was used to visualize the phylogenetic tree obtained from "rep_set_aligned.tre".



4.5 Compute Alpha Diversity within the Samples, and Generate Rarefaction Curves

Community ecologists typically describe the microbial diversity within their study. This diversity can be assessed within a sample (alpha diversity) or between a collection of samples (beta diversity). Here, we will determine the level of alpha diversity in our samples using a series of scripts from the QIIME pipeline.

4.5.1 Rarify OTU Table

The first task is to perform rarefaction on our OTU table. Rarefaction is an ecological approach that allows users to standardize the data obtained from samples with different sequencing efforts, and to compare the OTU richness of the samples using this standardized platform. For instance, if one of your samples yielded 10,000 sequence counts, and another yielded only 1,000 counts, the species diversity within those samples may be much more influenced by sequencing effort than underlying biology. The approach of rarefaction is to randomly sample the same number of OTU's from each sample, and use this data to compare the communities at a given level of sampling effort.

To perform rarefaction, you need to set the boundaries for sampling and the step size between sampling intervals. You can find the number of sequences associated with each sample by looking at the last line of the split_library_log.txt file generated in **Step 4.1.2** above. The line from our tutorial is pasted here:

```
Sample ct min/max/mean: 146 / 150 / 148.11
```

For this highly artificial example, all of the samples had sequence counts between 146 and 150. In real datasets, the range will generally be much larger. In practice, rarefaction is most useful when most samples have the specified number of sequences, so your upper bound of rarefaction should be close to the minimum number of sequences found in a sample. This is a judgment call, and the rarefaction can be repeated until satisfactory information is attained. We will use an upper bound (rarefaction maximum, option -x) of 141 sequences, a lower bound (minimum, option -m) of 1, and a step size of 10 (option -s). We want enough steps to create smooth curves, but not so many as to create a computational bottleneck. Finally, we will choose to do 5 iterations (option -n); higher numbers of iterations will give you greater confidence that the average result of the random sampling is representative, but will also increase the compute time.

```
$ python $qdir/rarefaction.py -i Fasting_otu_table.txt -m 1 -x 141 -s 10 -n 5 -o Fasting_Rarefaction/
```

The newly created directory "Fasting_Rarefaction" will contain many text files named rarefaction_##_#.txt ; the first set of numbers represents the number of sequences sampled, and the last number represents the iteration number. If you opened one of these files, you would find an otu table where for each sample the sum of the counts equals the number of samples taken.

4.5.2 Compute Alpha Diversity

The rarefaction tables are the basis for calculating diversity metrics, which reflect the diversity within the sample based on taxon counts of phylogeny. The QIIME pipeline allows users to conveniently caculate more than two dozen different diversity metrics. The full list of available metrics is available by passing the option -s to the script ***alpha_diversity.py***. Every metric has different strengths and limitations - technical discussion of each metric is readily available online and in ecology textbooks, but it is beyond the scope of this document. Here, we will calculate three metrics: 1) The **Simpson Index** is a very common measure of species abundance based on OTU counts: 2) The **Observed Species** metric is simply the count of unique OTU's found in the sample 3) **Phylogenetic Distance** *(*PD_whole_tree) is the only phylogenetic metric used in this script and requires a phylogenetic tree as an input.

```
$ python $qdir/alpha_diversity.py -i Fasting_Rarefaction/ -o Fasting_Alpha_Metrics/ -m
    simpson,PD_whole_tree,observed_species -t Default_Aln/rep_set_aligned.tre
```

4.5.3 Collate Rarified OTU Tables

The output directory "Fasting_Alpha_Metric/" will contain one text file alpha_rarefaction_##_# for every file input from ″Fasting_Rarefaction/″, where the numbers repesent the number of samples and iterations as before. The content of this tab delimited file is the calculated metrics for each sample. To collapse the individual files into a single combined table, use the script ***collate_alpha.py:***

```
$ python $qdir/collate_alpha.py -i Fasting_Alpha_Metrics/ -o Fasting_Alpha_Collated/
```

In the newly created directory "Fasting_Alpha_Collated", there will be one matrix for every diversity metric used in the ***alpha_diversity.py*** script. This matrix will contain the metric for every sample, arranged in ascending order from lowest number of sequences per sample to highest.

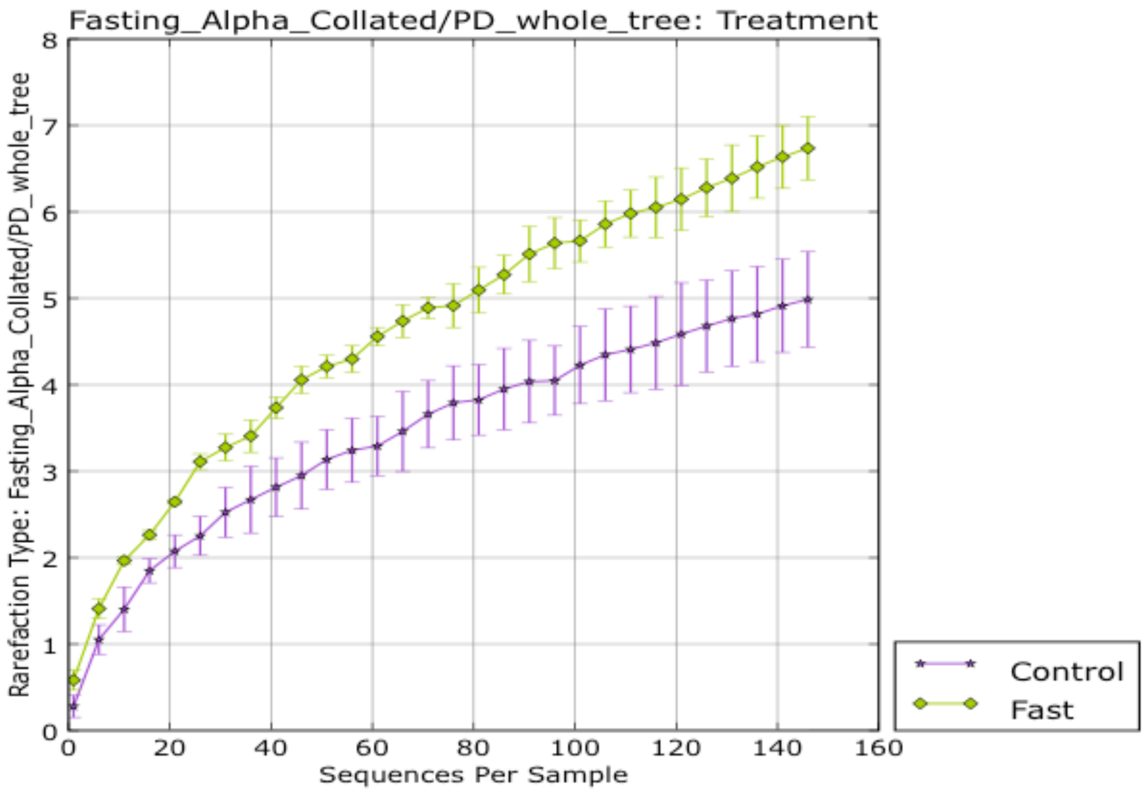
```
                          Sequences per sample  iteration     PC.354  PC.355  PC.356  PC.481  PC.593  PC.607
alpha_rarefaction_20_0.txt    20                0             2.87516 2.569   3.11047 3.12394 2.90972 4.28121
alpha_rarefaction_20_1.txt    20                1             2.74983 2.98409 3.80113 3.35763 3.88769 3.44925
alpha_rarefaction_20_2.txt    20                2             1.697   2.73752 3.34506 4.39953 3.01961 4.79355
alpha_rarefaction_20_3.txt    20                3             2.88929 2.54616 2.70628 3.45759 3.05061 4.62315
alpha_rarefaction_20_4.txt    20                4             2.57934 3.5986  3.27282 3.08616 3.30288 4.63737
alpha_rarefaction_20_5.txt    20                5             2.72065 2.59231 3.57125 3.49123 3.3541  4.06589
……
```

4.5.4 Generate Rarefaction Curves

The file ***make_rarefaction_plots.py*** takes a mapping file and any number of rarefaction files generated by ***collate_alpha.py*** and uses matplotlib to create rarefaction curves. Each curve represents a sample and can be colored by the sample metadata supplied in the mapping file. For this study, we will use the PD_whole_tree results and color by the Treatment Category.

```
$ python $qdir/make_rarefaction_plots.py -m Fasting_Map.txt -r Fasting_Alpha_Collated/PD_whole_tree.txt -o
    Rarefaction_Plots -p Treatment -i png
```

To view the rarefaction plot navigate to the bottom-level file within the "rarefaction_graph##" folder, where you will find a png file titled "Treatment.png". This plot shows the averages of each value for in the Treatment category with their corresponding error bars.



4.6 Compute Beta Diversity and Generate 2D/3D Principal Coordinate Analysis (PCoA) Plots

4.6.1 Compute Beta Diversity

Beta-diversity metrics assess the differences between microbial communities. In general, these metrics are calculated to study diversity along an environmental gradient (pH or temperature) or different disease states (lean vs. obese). The basic output of this comparison is a square matrix where a "distance" is calculated between every pair of samples reflecting the similarity between the samples. The data in this distance matrix can be visualized with clustering analyses, namely Principal Coordinate Analysis (PCoA) and hierarchical clustering (UPGMA). Like alpha diversity, there are many possible metrics which can be calculated with the QIIME pipeline - the full list of options can be seen by passing option -s to the script ***beta_diversity.py***. For our example, we will calculate the unweighted unifrac metric, which is a phylogenetic measure used extensively in recent microbial community sequencing projects.

```
$ python $qdir/beta_diversity.py -i Fasting_otu_table.txt -m dist_unweighted_unifrac -o
    Fasting_Unw_Unifrac_dist.txt -t Default_Aln/rep_set_aligned.tre
```

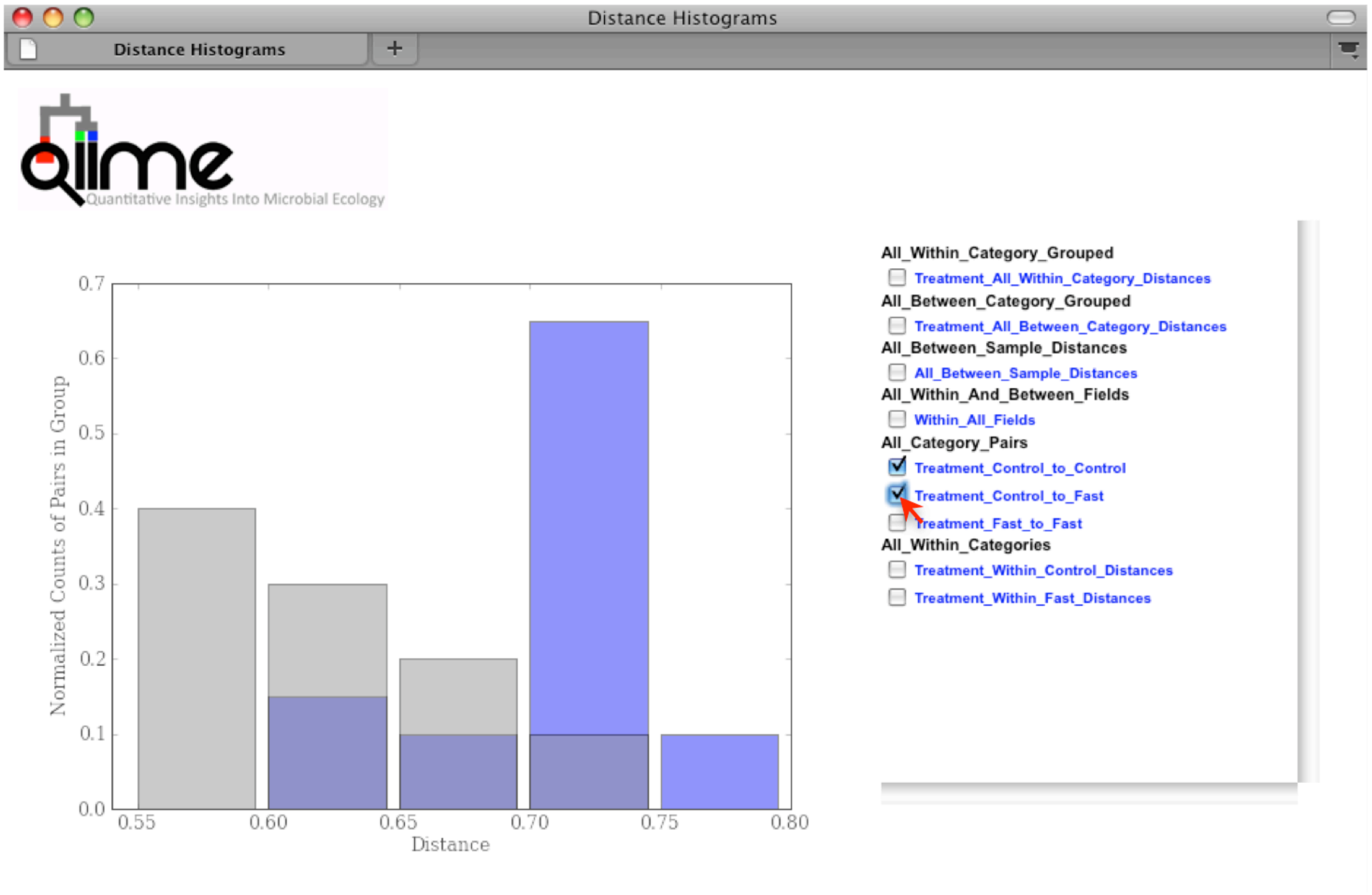
The resulting distance matrix (Fasting_Unw_Unifrac_dist.txt) is the basis for two methods of visualization and sample comparison: PCoA and UPGMA.

4.6.2 Generate Distance Histograms

Distance Histograms are a way to compare different categories and see which tend to have larger/smaller distances than others. For example, in the hand study, you may want to compare the distances between hands to the distances between individuals.

```
$ python $qdir/make_distance_histograms.py -d Fasting_Unw_Unifrac_dist.txt -m Fasting_Map.txt --fields
    Treatment -o Distance_Histograms --html_output True
```

For each of these groups of distances a histogram is made. The output is a HTML file ("QIIME_Distance_Histograms.html") where you can look at all the distance histograms individually, and compare them between each other. Within the webpage, the user can mouseover and/or select the checkboxes in the right panel to turn on/off the different distances within/between categories. For this example, we are comparing the distances between the samples in the Control versus themselves, along with samples from Fasting versus the Control.



All_Within_Category_Grouped
- [ ] Treatment_All_Within_Category_Distances

All_Between_Category_Grouped
- [ ] Treatment_All_Between_Category_Distances

All_Between_Sample_Distances
- [ ] All_Between_Sample_Distances

All_Within_And_Between_Fields
- [ ] Within_All_Fields

All_Category_Pairs
- [✓] Treatment_Control_to_Control
- [✓] Treatment_Control_to_Fast
- [ ] Treatment_Fast_to_Fast

All_Within_Categories
- [ ] Treatment_Within_Control_Distances
- [ ] Treatment_Within_Fast_Distances

4.6.3 Generate Principal Coordinates

Principal Coordinate Analysis (PCoA) is a technique that helps to extract and visualize a few highly informative gradients of variation from complex, multidimensional data. This is a complex transformation that maps the distance matrix to a new set of orthoganol axes such that a maximum amount of variation is explained by the first principal coordinate, the second largest amount of variation is explained by the second principal coordinate, etc. The principal coordinates can be plotted in two or three dimensions to provide an intuitive visualization of the data structure and look at differences between the samples, and look for similiarities by sample category. The transformation is accomplished with the script ***principal_coordinates.py***:

```
$ python $qdir/principal_coordinates.py -i Fasting_Unw_Unifrac_dist.txt -o Fasting_Unw_Unifrac_coords.txt
```

The file "Fasting_Unw_Unifrac_coords.txt" lists every sample in the first column, and the subsequent columns contain the value for the sample against the noted principal coordinate. At the bottom of each Principal Coordinate column, you will find the eigenvalue and percent of variation explained by the coordinate. To determine which axes are useful for your project, you can generate a "scree plot" by plotting the eigenvalues of each principal component in descending order.

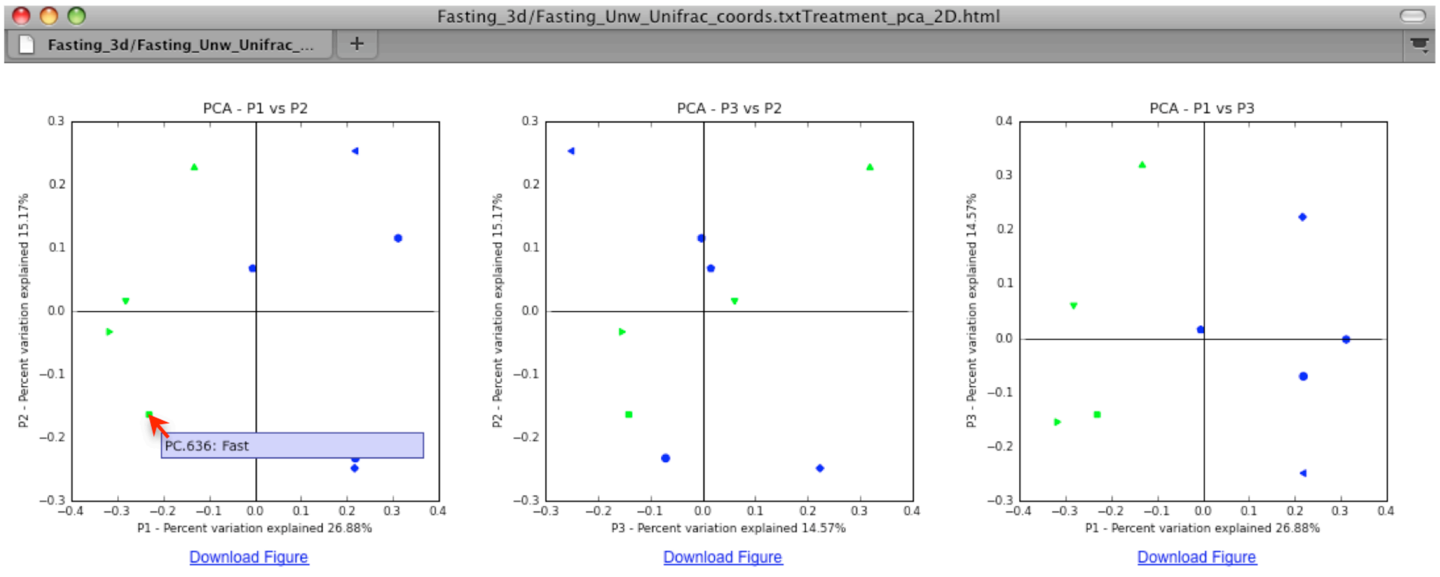
4.6.4 Generate 2D and 3D PCoA Plots

To plot the coordinates, you can use the QIIME scripts ***make_2d_plots.py*** and ***make_3d_plots.py***. The two dimensional plot will be rendered as a html file which can be opened with a standard web browser, while the three dimensional plot will be a kinemage file which requires additional software to render and manipulate. The usage for both scripts use the same convention, detailed in the **Documentation**.

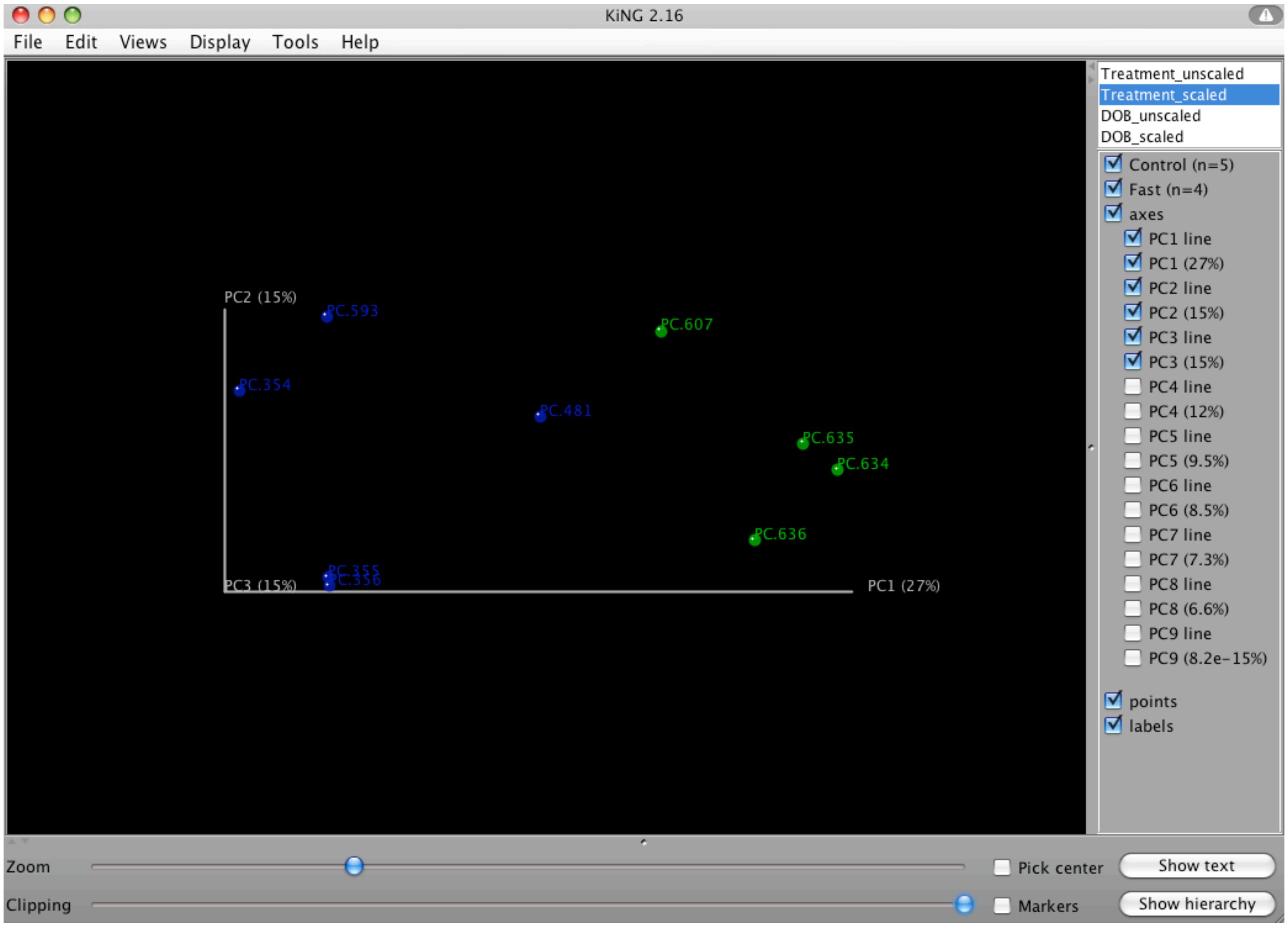
```
$ python $qdir/make_2d_plots.py -i Fasting_Unw_Unifrac_coords.txt -m Fasting_Map.txt -o Fasting_2d/ -b
        Treatment

$ python $qdir/make_3d_plots.py -i Fasting_Unw_unifrac_coords.txt -m Fasting_Map.txt  -o Fasting_3d/ -b
    "DOB,Treatment"
```

The html file created in directory "Fasting_2d/" shows a plot for each combination of the first three principal coorindates. Since we specified Treatment to use for coloring the samples, each sample colored according to the category it corresponds. You can get the name for each sample by holding your mouse over the data point.


Fasting_3d/Fasting_Unw_Unifrac_coords.txtTreatment_pca_2D.html
Fasting_3d/Fasting_Unw_Unifrac_...
PCA - P1 vs P2
P2 - Percent variation explained 15.17%
P1 - Percent variation explained 26.88%
PC.636: Fast
Download Figure
PCA - P3 vs P2
P2 - Percent variation explained 15.17%
P3 - Percent variation explained 14.57%
Download Figure
PCA - P1 vs P3
P3 - Percent variation explained 14.57%
P1 - Percent variation explained 26.88%
Download Figure

In the invocation of ***make_3d_plot.py***, we specified that the samples should be colored by the value of the "Treament" and "DOB" columns in the mapping file. For the "Treatment" column, all samples with the same "Treatment" will get the same color. For our tutorial, the five control samples are all blue and the four control samples are all green. This lets you easily visualize "clustering" by metadata category. The 3d visualization software allows you to rotate the axes to see the data from different perspectives. By default, the script will plot the first three dimensions in your file. Other combinations can be viewed using the "Views:Choose viewing axes" option in the KiNG viewer (may require the installation of kinemage software). The first 10 components can be viewed using "Views:Paralled coordinates" option or typing "/".

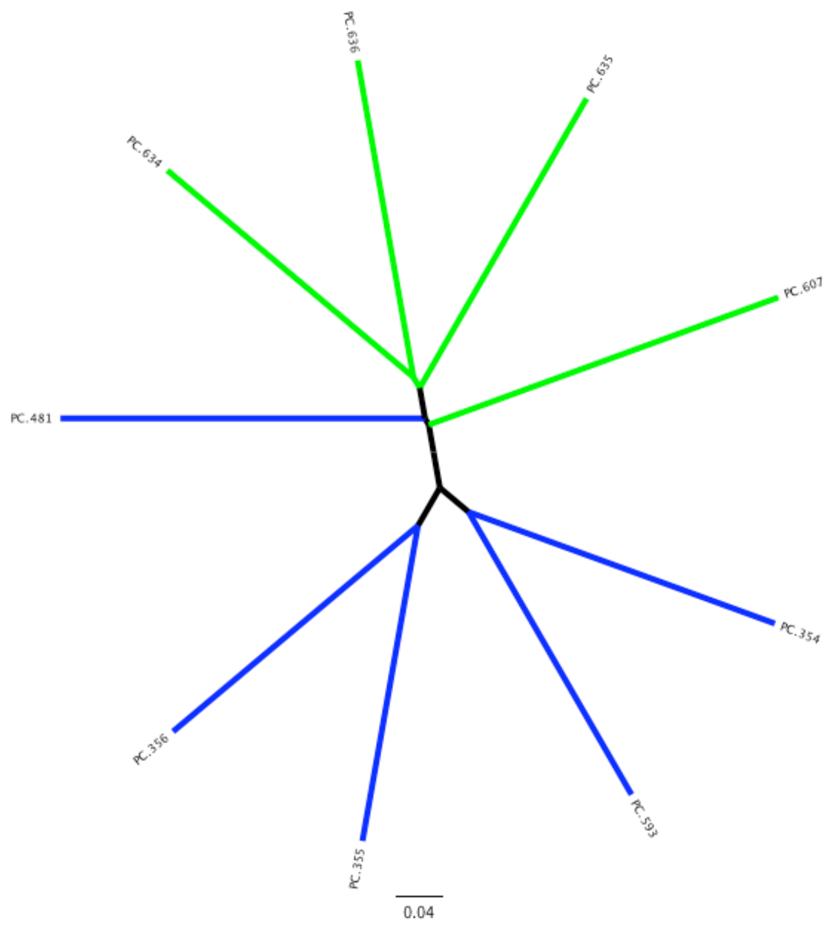


4.7 Hierarchical Clustering and Jackknifing Support

4.7.1 Hierarchical Clustering

Unweighted Pair Group Method with Arithmetic mean (UPGMA) is type of hierarchical clustering method using average linkage and can be used to visualize the distance matrix produced by ***beta_diversity.py***. The output is a file that can be opened with tree viewing software, such as FigTree.

```
$ python $qdir/hierarchical_cluster.py -i Fasting_Unw_Unifrac_dist.txt -o Fasting_Unw_Unifrac_upgma.tre
```



This tree shows the relationship among the 9 samples, and reveals that the 4 samples from the guts of fasting mice cluster together (PC.6xx, fasting data is in "Fasting_Map.txt").

4.7.2 Perform Jackknifing Support

To measure the robustness of this result to sequencing effort, we perform a jackknifing analysis, wherein a smaller number of sequences are chosen at random from each sample, and the resulting UPGMA tree from this subset of data is compared with the tree representing the entire available data set. This process is repeated with many random subsets of data, and the tree nodes which prove more consistent across jackknifed datasets are deemed more robust.

First the jackknifed otu tables must be generated, by subsampling the full available data set. In this tutorial, each sample contains between 146 and 150 sequences, as shown in the split_library_log.txt file:

```
Sample ct min/max/mean: 146 / 150 / 148.11
```

To ensure that a random subset of sequences is selected from each sample, we chose to select 110 sequences from each sample (75% of the smallest sample, though this value is only a guideline). We use ***rarefaction.py***, which has already been described:

```
$ python $qdir/rarefaction.py -i Fasting_otu_table.txt -m 110 -x 110 -s 1 -n 20 -o jackknife_otu/ --small_included
```

This generates 20 subsets of the available data by random sampling, simulating a smaller sampling effort (110 sequences in each sample). Because our goal is to compare these jackknifed trees to the tree already generated, we use --small_included to ensure that every sample present in the original tree is present in the jackknifed ones. (here --small_included isn't actually necessary, as all samples have > 110 sequences).

We then calculate the distance matrix for each jackknifed dataset, using ***beta_diversity.py*** as before, but now in batch mode:

```
$ python $qdir/beta_diversity.py -i jackknife_otu -m dist_unweighted_unifrac -t Default_Aln/rep_set_aligned.tre -o
    jackknife_dist
```

which results in 20 distance matrix files written to the jackknife_dist directory. Each of those is then used as the basis for hierarchical clustering, using ***hierarchical_cluster.py*** in batch mode:

```
$ python $qdir/hierarchical_cluster.py -i jackknife_dist -o jackknife_upgma
```

4.7.3 Compare Jackknifed Trees to Cluster Tree

Hierarchical clustering of the 20 distance matrix files results in 20 UPGMA samples clusters, each based on a random sub-sample of the available sequence data. These are then compared to the UPGMA result using all available data:

```
$ python $qdir/tree_compare.py -m Fasting_Unw_Unifrac_upgma.tre -s jackknife_upgma -o
    Fasting_jackknife_support
```

This compares the UPGMA clustering based on all available data (-m) with the jackknifed UPGMA results (-s). Two files are written to "Fasting_jackknife_support":

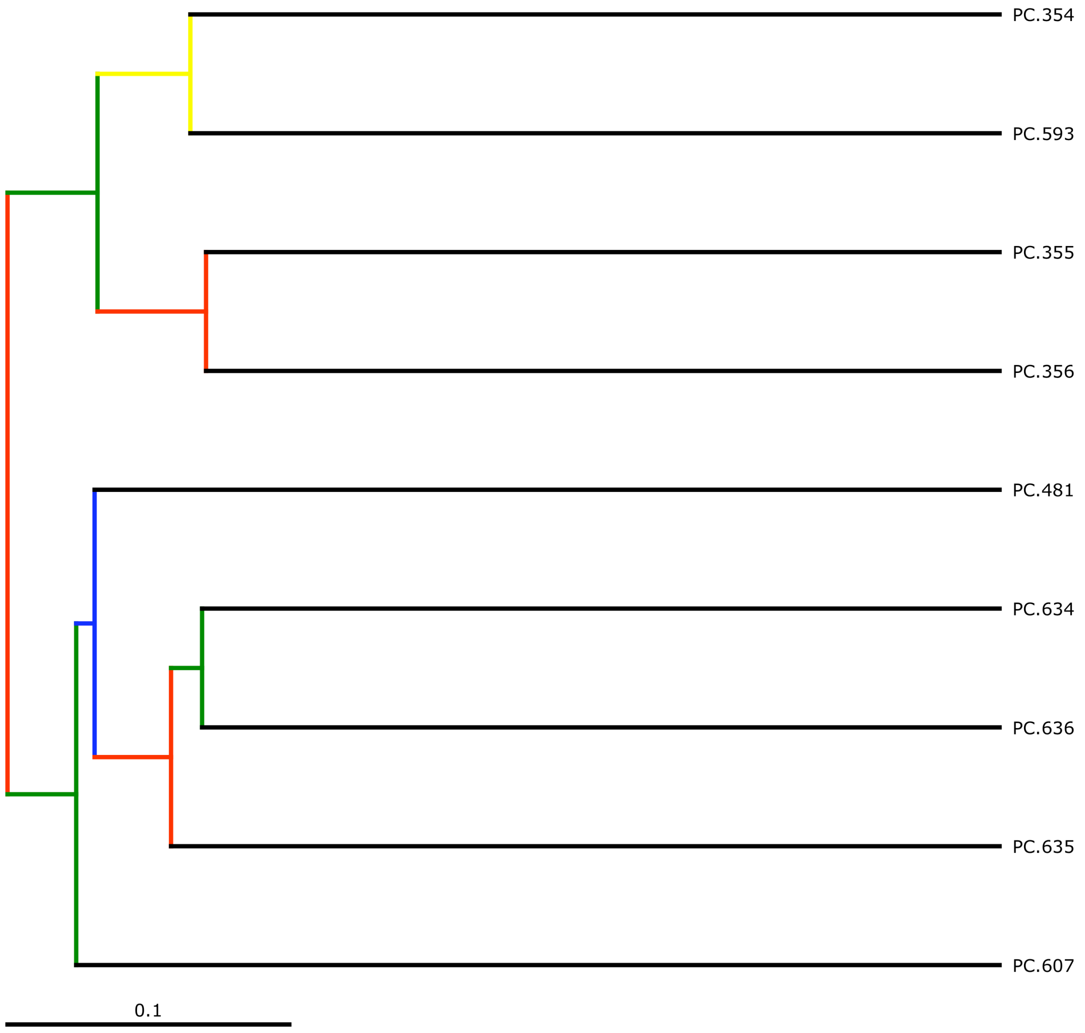
- master_tree.tre is virtually identical to "Fasting_Unw_Unifrac_upgma.tre", but each internal node of the UPGMA clustering is assigned a unique name
- jackknife_support.txt explains how frequently a given internal node had the same set of descendant samples in the jackknifed UPGMA clusters as it does in the UPGMA cluster using the full available data. A value of 0.5 indicates that half of the jackknifed data sets support that node, while 1.0 indicates perfect support.

4.7.4 Generate Bootstrapped Tree

This data can be visualized using ***make_bootstrapped_tree.py***:

```
$ python $qdir/make_bootstrapped_tree.py -m Fasting_jackknife_support/master_tree.tre -s
    Fasting_jackknife_support/jackknife_support.txt -o Fasting_jackknife_support/Unw_unifrac_support.pdf
```

The resulting pdf shows the tree with internal nodes colored, red for 75-100% support, yellow for 50-75%, green for 25-50%, and blue for < 25% support. Although UPGMA shows that PC.354 and PC.593 cluster together and PC.481 with PC.6xx cluster together, we can not have high confidence in that result. However, there is excellent jackknife support for all fasted samples (PC.6xx) which are clustering together, separate from the non-fasted (PC.35x) samples.



References

Crawford, P. A., Crowley, J. R., Sambandam, N., Muegge, B. D., Costello, E. K., Hamady, M., et al. (2009). Regulation of myocardial ketone body metabolism by the gut microbiota during nutrient deprivation. *Proc Natl Acad Sci U S A, 106*(27), 11276-11281.