Part 10_Using MSI Resources

Statistics in Human Genetics and Molecular biology Bioinformatics & RNAseq tutorial
Using MSI for bioinformatics analyses

Part 11_QIIME setup instructions_HMP_MACQIIME

Unix Guide
QIIME installation Guide from the HMP website
MacQIIME installation guide from Jeff Werner's website
QIIME 2 installation Guide from the HMP website
R guide for MACOSX

Part 12_QIIME Tutorial_MSI_MACQIIME_HMP

Using MSI for bioinformatics analyses
Abridged MACQIIME Microbiome analysis and MSI adaptation (MACQIIME dataset)
Unix Guide
Complete HMP QIIME 454 microbiome analysis tutorial (HMP dataset)
Complete MACQIIME microbiome analysis tutorial (MACQIIME dataset)

Part 13_QIIME and R Microbiome Analysis_MSI_GUERRERO_NEGRO

Using MSI for bioinformatics analyses
Abridged Discovering the Microbiome analysis tutorial and MSI adaptation (GUERRERO_NEGRO dataset)
Complete Discovering the Microbiome analysis tutorial and MSI adaptation (GUERRERO_NEGRO dataset)

Part 14_Microbiome Differential Analyses Summary_LeFse_PiCRUST_KEGG

Statistical Analysis Summary Review
Phyloseq Tutorial
DESeq2 tutorial
Picrust Tutorial
LEFse & Galaxy Network tutorial
Complete Phyloseq + DESeq2 + Picrust + LEFse Tutorial

Part 15_Intro to Microbiome Analysis_MSI_MACQIIME_GUERRERO_NEGRO

Discovering the Microbiome Study plan
R analysis basics and review
Using MSI for bioinformatics analyses (Part_10)
Discovering the Microbiome summary (based on MACQIIME tutorial)
Abridged MACQIIME Microbiome analysis (MACQIIME dataset) (Part_12)
Statistical Analyses in Microbiome studies (Based on Guerrero Negro tutorial)
Abridged Discovering the Microbiome analysis tutorial and MSI adaptation (GUERRERO_NEGRO dataset) (Part_13)
Statistical Analysis Summary Review / Phyloseq Tutorial + DESeq2 tutorial / Picrust Tutorial + LEFse & Galaxy Network tutorial (Part 14)

Discovering the microbiome

Academic Planning Goals
Minimum (2 projects/week) Regular (3 projects/week) Maximum (4 Projects/week)
Schedule V1: 1 grant (4 months) + Manuscript (4 Months) + Manuscript (4 Months) = 1 year
Schedule V2: 1 grant (2 months) + Manuscript (2 Months) + Manuscript (2 Months) = 6 months

Research Topics
Area of Interest 1: Inflammation + CRC (ABO Genetics > Inflammation > CRC / Periodontal disease > Inflammation > CRC / Immune cells > Inflammation > CRC)
Area of Interest 2: Microbiome + cancer (Oral Microbiome > Inflammation > Cancer / Gut Microbiome > Diet > Cancer / Gut Microbiome > Short Chain Fatty Acids > Cancer)
Area of Interest 3: Lifestyle factors + CRC (Diet exposures > Metabolism > CRC / Physical Activity > Metabolism > CRC / White & Brown Fat Metabolism> Metabolism > CRC)
Area of Interest 4: Environmental Exposure + cancer (Pesticides > Oxygen Radicals > Cancer / Immigrants > Diet change > Cancer / HIV and MALT Lymphoma > Immune reaction > Cancer)
Area of Interest 5: Cancer Survivors (Cancer > Provoking Agent > CVD / Cancer > Weakened immunity > Co-infections / Cancer > Weakened Immune System > Auto-immune disease)

Continuing Education Contents for Review (Version 1: PhD program notes on weekdays / Core + Advanced Epi + Advanced Biostats on Mondays + Tuesdays + Wednesdays weekday evenings / Reading on Sundays & Thursdays. Version 2: PhD program notes on weekdays / QIIME analysis + Differential Abundance + Metagenomic Analysis on Mondays + Tuesdays + Wednesdays weekday evenings / Microbiome Methods + Microbiome Contents on Sundays & Thursdays)

Core: Epi III + Epi III Code / Correlated Data + Correlated Data Code / GRT + GRT Code
Advanced Epi: 8341 + Code / 8342 + Code / 8343 + Code
Advanced Biostatistics: Biostatics R review + Molecular Genetics Review / MSI + MACQIIME + QIIME / Phyloseq + DESeq2)

Review (Weekdays on Mondays + Tuesdays + Wednesdays: 4 hour session)
Core: Epi 3 + Correlated Data Analysis + GRT
Advanced: Advanced Epi I + Advanced Epi 2 + Advanced Epi 3
Specialization: Statistics in Genetics & Molecular Biology + Intro to Microbiome Analysis

Bioinformatics Methods (Weekdays on Mondays + Tuesdays + Wednesdays: 4 hour session)

- QIIME Analysis (MSI + MACQIIME + GUERRERO NEGRO)
- Differential Abundance (EdgeR / Phyloseq + DESeq2)
- Metagenomic Analysis (PiCRUST + KEGG / LEFse + Galaxy Network)

Microbiome Contents (Weekends on Saturdays and Sundays: 4 hour session)

- 8 Content papers (2 hours)
- 8 Methods papers (2 hours)

Microbiome Analysis course Part 1 (2.5 hours)

- Introduction
- Data Generation
- 16S Sequencing
- Quiime Overview
- UNIX command Line Overview
- Picking OTUs
- Assigning Taxonomy

Microbiome Analysis course Part 2 (2 Hours)

- Alpha Diversity
- Beta Diversity
- UniFracs
- Statistical Testing Part 1
- Statistical Testing Part 2

Microbiome Analysis course Part 3 (3.5 Hours)

- Visualizing Diversity
- Detrending & Detecting Gradients
- Constrained Ordination
- Clustering
- Supervised Learning Background
- Supervised Learning Application
- Source Tracking
- Compositionality
- Picrust and Predicting functions
- Shotgun Taxonomy

Microbiome analysis review

1_QIIME / MACQIIME (Clean sequences > Phylogeny tree > OTU Table)
2_Alpha / Beta diversity / PCA Differences in Microbiome community composition
3_Differential abundance on pre-specified taxa (SAS / GEE)
4_Ranked differential abundance on rarefied OTU table (R / edgeR)
5_Ranked differential abundance on non-rarefied OTU table (Phyloseq / DESeq2)
6_Ranked differential abundance on functional pathways (Picrust / LEFse & LDA)

Microbiome Contents Papers

Set 1: Background

1. General Background
    - Epidemiologic studies of the human microbiome and cancer (Vogtman et al, 2015)
    - Cancer Promoting effects of microbial dysbiosis (Shelfin et al, 2014)
    - Oral microbiomes: more and more importance in oral cavity and whole body (Gao et al, 2018)
    - Metagenomic analysis of nitrate reducing bacteria in the oral cavity (Hyde et al, 2014)
    - Periodontal microbiota and phospholipases: The Oral infections and Vascular Disease Epidemiology Study (INVEST)
    - The Subgingival microbiome, systemic inflammation and insulin resistance: The oral infections, glucose intolerance and insulin resistance study (ORIGINS)

Set 2: Bacterial Genes and colorectal cancer

- Direct detection and quantification of bacterial genes associated with inflammation in DNA isolated form stool (Gomez-Moreno et al, 2014)
- Predictive functional profiling of microbial communities using 16S rRNA marker gene sequences (Langille et al, 2013)
- Virulence genes are a signature of the microbiome in the colorectal tumor microenvironment (Burns et al, 2015)

Set 3: Oral Microbiome in association with CRC

- Metabolic and community synergy of oral bacteria in colorectal cancer (Flynn et al, 2016)
- Co-Occurrence of anaerobic bacteria in colorectal carcinoma (Warren et al, 2013)
- Oral Microbiome and history of smoking and colorectal cancer (Kato et al, 2016)

Microbiome Methods Papers

Set 1: Differential association of gut microbiota with inflammation and CRC

- Leveraging sequence-based faecal microbial community survey data to identify a composite biomarker for colorectal cancer (Shah et al, 2017)
- The influence of non-steroidal anti-inflammatory drugs on the gut microbiome (Rodgers et al, 2017)
- Gut microbiota, Inflammation and colorectal cancer (Brennan et al, 2016)
- Dysbiosis of gut microbiota in promoting the development of colorectal cancer (Zou et al, 2018)
- Role of colonic microbiota in colorectal carcinogenesis: A systematic review (Borges Canha et al, 2015)

Set 2: Model improvement strategies with the gut microbiome

- The human gut microbiome as a screening tool for colorectal cancer (Zackular et al, 2014)

Set 3: Advanced ecologic and bioinformatics analyses

- Waste not, want not: Why rarefying microbiome data is inadmissible (McMurdie et al, 2014)
- Metagenomic biomarker discovery and explanation (Segata et al, 2011)

Part 10_Using MSI Resources

Statistics in Human Genetics and Molecular biology Bioinformatics & RNAseq tutorial
Using MSI for bioinformatics analyses

Using MSI for bioinformatics analyses

A Minnesota Supercomputing Institute (MSI) account was created for
Guillaume Onyeaghala in the group prizm001 with the username onyea005.

In addition, you have been granted access to the prizm001 group's Service Units.
This access will allow you to run jobs on MSI's High-Performance Computing resources.

Passwords

Your password at MSI is the same as the password for your University Internet ID.
If you ever need to change or reset your password, go to https://www.umn.edu/dirtools.

Getting Started

Quick Start Guides for new MSI users are available on our website
at https://www.msi.umn.edu/quick-start-guides

Frequently Asked Questions

You can find answers to a range of frequently asked questions
at https://www.msi.umn.edu/support/faq

Getting Help

Our help desk is staffed 8:30 to 5:00 Monday through Friday. Send an email
to help@msi.umn.edu, call (612) 626-0802, or visit 587 Walter Library.

Connecting to and using MSI resources (Refer to Part 10)

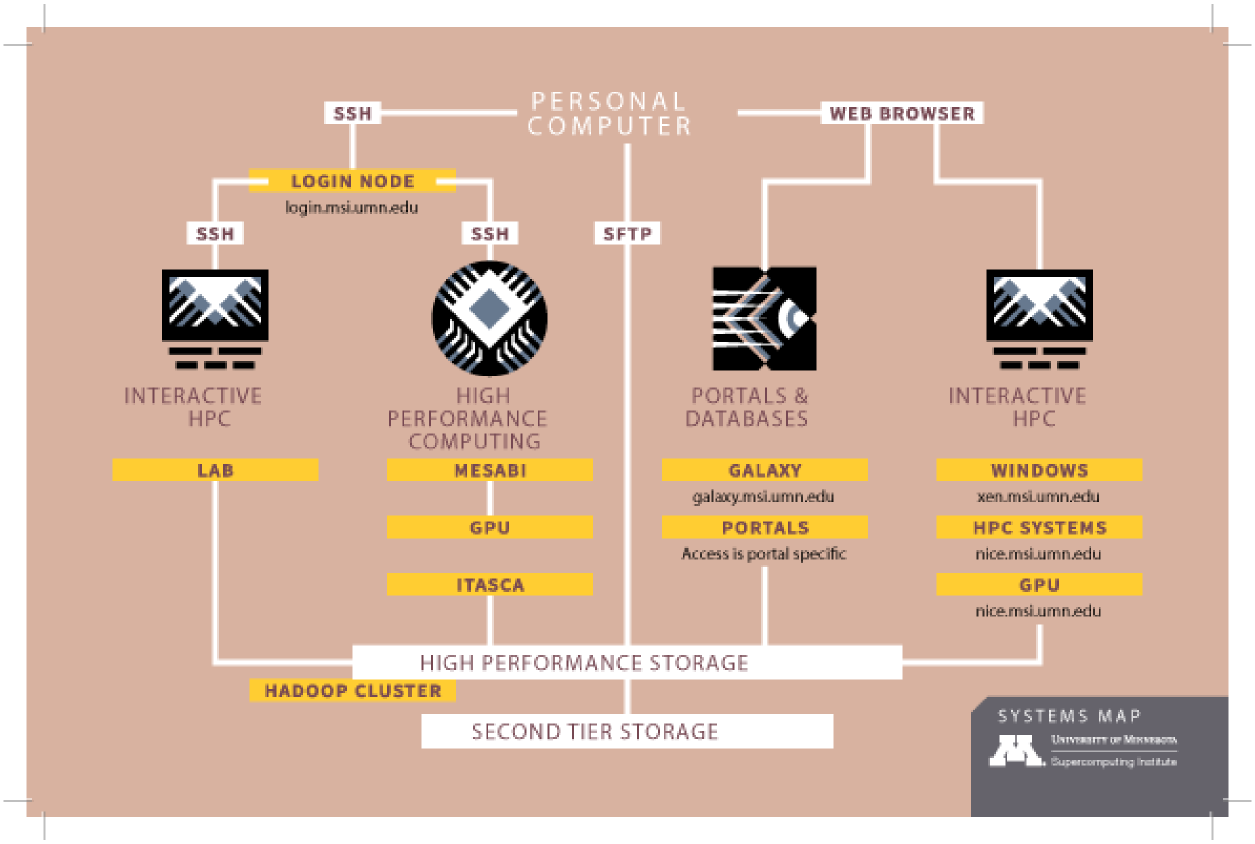
https://www.msi.umn.edu/content/connecting-hpc-resources

https://www.msi.umn.edu/support/faq

https://www.msi.umn.edu/getting-started

Summary

MSI provides access to a variety of High-Performance Computing (HPC) systems designed to tackle a wide range of computational needs. This document will show you how you can access all of MSI computational resources from your personal computer.



Skills/Software Needed

If you are connecting from a computer running Windows OS you will need to download and install PuTTY. OSX and Linux does not require additonal software to connect.

Connecting to MSI

SSH (Use Secure Shell)

SSH, also known as Secure Socket Shell, is a network protocol that provides administrators with a secure way to access a remote computer. SSH also refers to the suite of utilities that implement the protocol. Secure Shell provides strong authentication and secure encrypted data communications between two computers connecting over an insecure network such as the Internet. SSH is widely used by network administrators for managing systems and applications remotely, allowing them to log in to another computer over a network, execute commands and move files from one computer to another.

The main features of SSH include secure remote logins, terminal emulation, fully integrated secure file transfers, and secure tunneling of X11 traffic. Many University of Minnesota systems require secure telnet and FTP connections.

Install a SSH Client

Unix/Linux

- OpenSSH is freely usable and re-usable by everyone under a BSD license. OpenSSH is available from http://www.openssh.org/portable.html.

Windows

- Cygwin provides a Windows port of the most current OpenSSH tools.

- PuTTY is a full-featured, SSH compliant client for Windows. You can get Putty at http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html, as well as PSCP, a tool for encrypted remote file transfer

- WinSCP is a freeware Secure Copy (SCP) protocol client for Windows, which can be used to connect to an SSH server, mainly to UNIX machines, for file transfer using the SCP service. http://winscp.net/eng/index.php

- SecureCRT software requires you to purchase a license. http://www.vandyke.com/products/securecrt/

Mac OS X

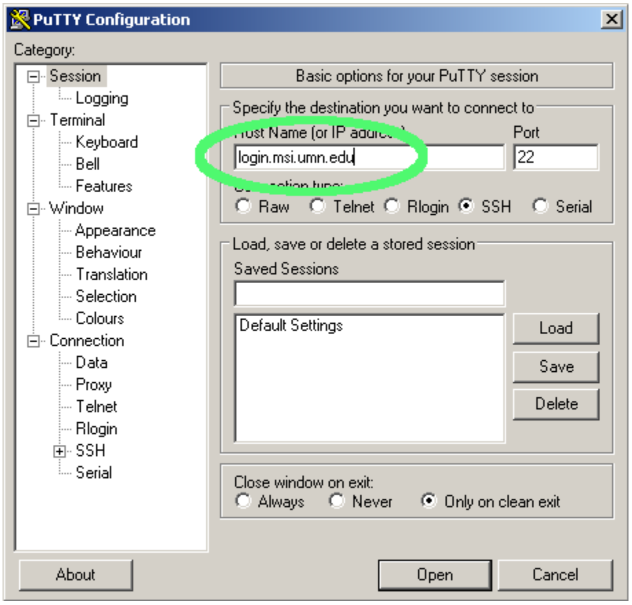
- Mac OS X comes with OpenSSH pre-installed. Open the Terminal application from inside the Utilities folder, and then type "ssh *user@server_name_or_IP_address*" to get started.

- Portable OpenSSH is available at http://www.openssh.com/portable.html at no charge. The portable OpenSSH follows development of the official version, but releases are not synchronized. Portable releases are marked with a 'p' (e.g. 2.3.0p1).

Connecting to MSI with Windows OS

Configure PuTTY to connect to MSI

- Double-click on the PuTTY icon and a configuration window will pop up.
- In the Host Name box type: **login.msi.umn.edu**



Specify the destination you want to connect to

Host Name (or IP address): login.msi.umn.edu

Port: 22

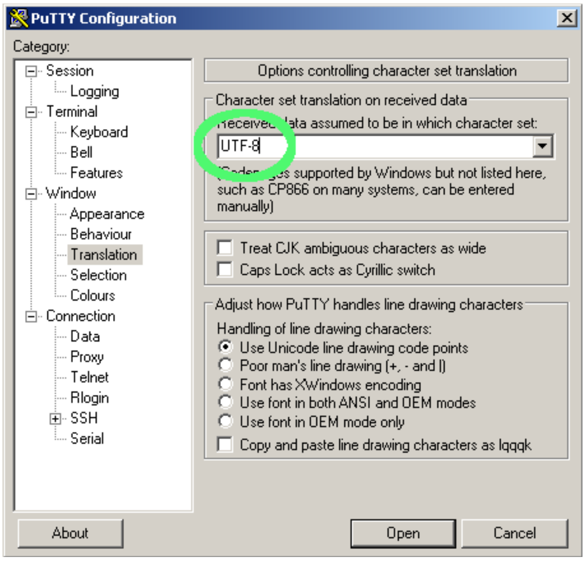
Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

Saved Sessions: Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

- In the left panel, under Window, select Translation
- Change Receive data assumed to bin in which character set to : UTF-8



Character set translation on received data

Received data assumed to be in which character set: UTF-8

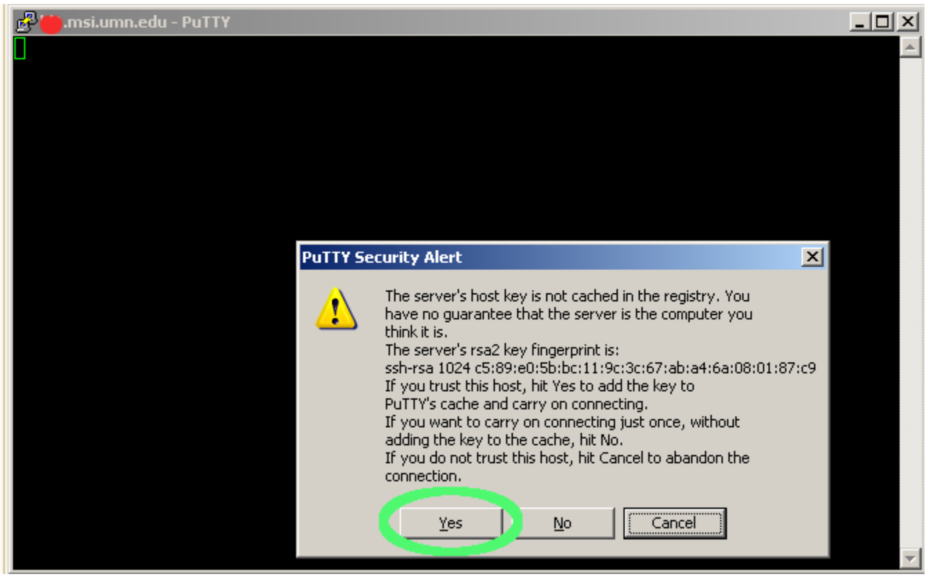
(Codepages supported by Windows but not listed here, such as CP866 on many systems, can be entered manually)

☐ Treat CJK ambiguous characters as wide
☐ Caps Lock acts as Cyrillic switch

Adjust how PuTTY handles line drawing characters

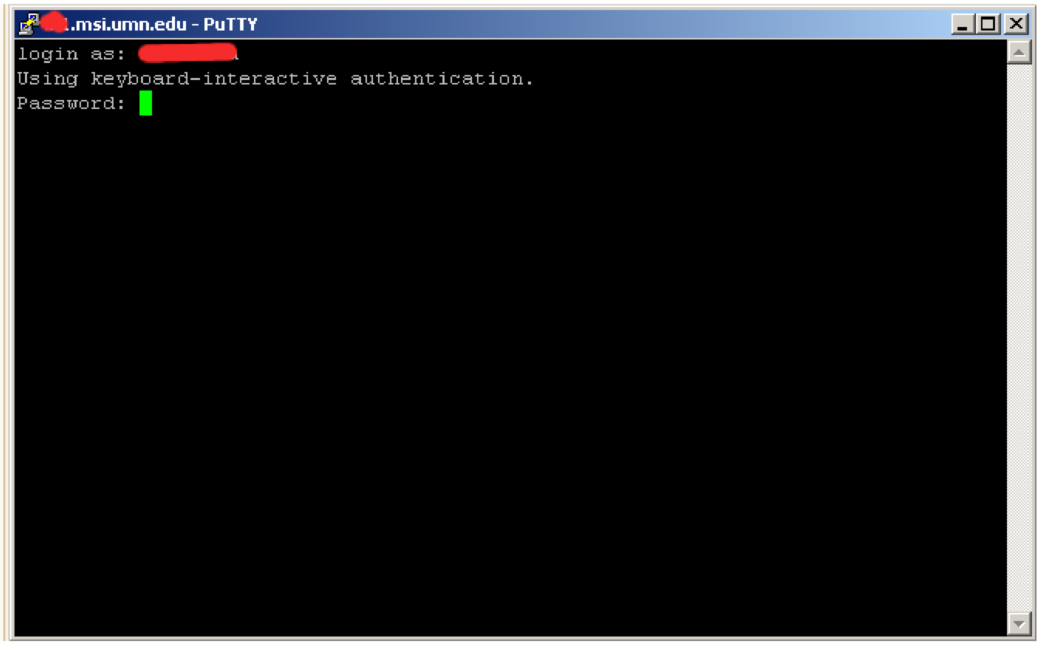
Handling of line drawing characters:
● Use Unicode line drawing code points
○ Poor man's line drawing (+, - and |)
○ Font has XWindows encoding
○ Use font in both ANSI and OEM modes
○ Use font in OEM mode only
☐ Copy and paste line drawing characters as lqqqk

- Select Session in the left panel and in the Saved Session box type the name you would like to use to refer to this session. We suggest MSI server, then select Save.
- The next time you open PuTTY you will be able to recall a saved profile by clicking on the name and clicking Load then Open to start the session.
- The first time you connect to each MSI system you may be asked to cache the server fingerprint, select Yes.



The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 1024 c5:89:e0:5b:bc:11:9c:3c:67:ab:a4:6a:08:01:87:c9
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

- When prompted enter your MSI username and password



```
login as: [redacted]
Using keyboard-interactive authentication.
Password:
```

- You are now connected to the MSI Login host (or bastion host). This host will let you view the directories but you can not submit jobs or run interactive computation on this host.

Connecting to MSI with OSX

Terminal is a preinstalled application which can be used to connect to MSI systems. Terminal can be found in the Utilities folder found in the Applications folder.

Connecting to MSI with Linux OS

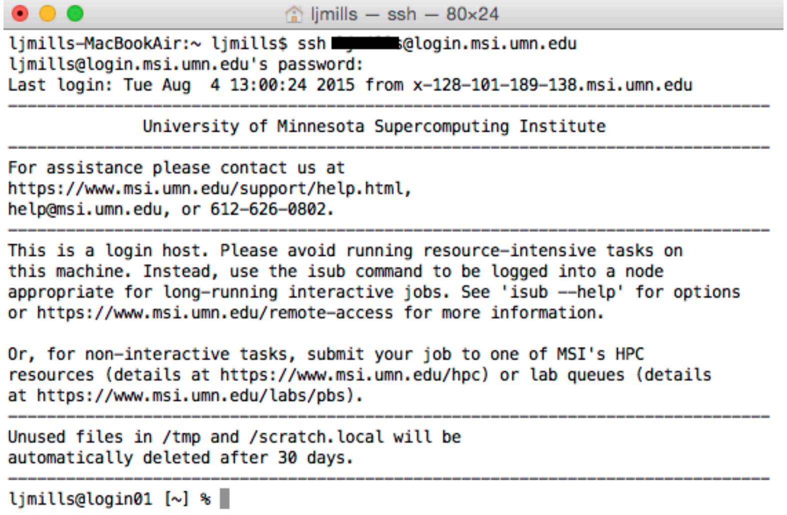
Terminal is a preinstalled application which can be used to connect to MSI systems.

For OSX and Linux OS Terminal Apps

Once you open the Terminal App type:

- ssh yourMSIusername@login.msi.umn.edu

- You will be prompted for your MSI password type it in then press enter. The terminal will not display your password as you type.
- The first time you log into MSI systems you will be asked if you would like to cache the server fingerprint, type yes and press return.



```
ljmills-MacBookAir:~ ljmills$ ssh [redacted]@login.msi.umn.edu
ljmills@login.msi.umn.edu's password:
Last login: Tue Aug  4 13:00:24 2015 from x-128-101-189-138.msi.umn.edu
-------------------------------------------------------------------------------
              University of Minnesota Supercomputing Institute
-------------------------------------------------------------------------------
For assistance please contact us at
https://www.msi.umn.edu/support/help.html,
help@msi.umn.edu, or 612-626-0802.
-------------------------------------------------------------------------------
This is a login host. Please avoid running resource-intensive tasks on
this machine. Instead, use the isub command to be logged into a node
appropriate for long-running interactive jobs. See 'isub --help' for options
or https://www.msi.umn.edu/remote-access for more information.

Or, for non-interactive tasks, submit your job to one of MSI's HPC
resources (details at https://www.msi.umn.edu/hpc) or lab queues (details
at https://www.msi.umn.edu/labs/pbs).
-------------------------------------------------------------------------------
Unused files in /tmp and /scratch.local will be
automatically deleted after 30 days.
-------------------------------------------------------------------------------
ljmills@login01 [~] %
```

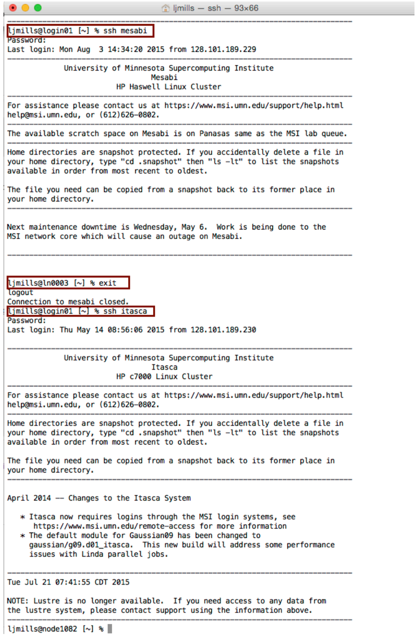
- You are now connected to the MSI Login host (or bastion host). This host will let you view the directories but you can not submit jobs or run interactive computation on this host.

Connecting to Mesabi and Itasca from Login Host

- Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh.

ssh mesabi or ssh itasca

- MSI prohibits moving directly from one system to another. If you are connected to Mesabi and would like to connect to Itasca you will first have to exit back to the Login Host, then ssh to the new system. Commands are highlighted with red boxes in the example below.



```
--------------------------------------------------------------------------------
ljmills@login01 [~] % ssh mesabi
Password:
Last login: Mon Aug  3 14:34:20 2015 from 128.101.189.229
--------------------------------------------------------------------------------
             University of Minnesota Supercomputing Institute
                                Mesabi
                         HP Haswell Linux Cluster
--------------------------------------------------------------------------------
For assistance please contact us at https://www.msi.umn.edu/support/help.html
help@msi.umn.edu, or (612)626-0802.
--------------------------------------------------------------------------------
The available scratch space on Mesabi is on Panasas same as the MSI lab queue.
--------------------------------------------------------------------------------
Home directories are snapshot protected. If you accidentally delete a file in
your home directory, type "cd .snapshot" then "ls -lt" to list the snapshots
available in order from most recent to oldest.

The file you need can be copied from a snapshot back to its former place in
your home directory.
--------------------------------------------------------------------------------

Next maintenance downtime is Wednesday, May 6.  Work is being done to the
MSI network core which will cause an outage on Mesabi.

--------------------------------------------------------------------------------


ljmills@ln0003 [~] % exit
logout
Connection to mesabi closed.
ljmills@login01 [~] % ssh itasca
Password:
Last login: Thu May 14 08:56:06 2015 from 128.101.189.230

--------------------------------------------------------------------------------
             University of Minnesota Supercomputing Institute
                                Itasca
                         HP c7000 Linux Cluster
--------------------------------------------------------------------------------
For assistance please contact us at https://www.msi.umn.edu/support/help.html
help@msi.umn.edu, or (612)626-0802.
--------------------------------------------------------------------------------
Home directories are snapshot protected. If you accidentally delete a file in
your home directory, type "cd .snapshot" then "ls -lt" to list the snapshots
available in order from most recent to oldest.

The file you need can be copied from a snapshot back to its former place in
your home directory.
--------------------------------------------------------------------------------

April 2014 -- Changes to the Itasca System

   * Itasca now requires logins through the MSI login systems, see
      https://www.msi.umn.edu/remote-access for more information
   * The default module for Gaussian09 has been changed to
     gaussian/g09.d01_itasca.  This new build will address some performance
     issues with Linda parallel jobs.

--------------------------------------------------------------------------------
Tue Jul 21 07:41:55 CDT 2015

NOTE: Lustre is no longer available.  If you need access to any data from
the lustre system, please contact support using the information above.
--------------------------------------------------------------------------------
ljmills@node1082 [~] % 
```

- Each time you move between systems you will see the welcome message associated with the system you are connecting to. These messages often contain useful information and should be read.
- **NOTE:** Like the login node Mesabi and Itasca have specific nomenclatures in the prompt. When connected to Mesabi your prompt will contain **ln** followed by some number when connected to Itasca your prompt will contain **node** followed by some numbers. This is a quick short cut to remind you which system you are connected to at any time.

Choosing a Job Queue

Summary

Most MSI systems use job queues to efficiently and fairly manage when computations are executed. A job queue is an automated waiting list for use of a particular set of computational hardware. When computational jobs are submitted to a job queue they wait in the queue in line until the appropriate resources become available. Different job queues have different resources and limitations. When submitting a job, it is very important to choose a job queue which has resources and limitations suitable to the particular calculation.

This document outlines factors to consider when choosing a job queue. These factors are important when choosing where to place a job. This document is best used on all MSI systems and in conjunction with the Queues page that outlines the resource limitations for each queue.

Please note that Mesabi's "widest" queue requires special permission to use. Please submit your code for review at: **help@msi.umn.edu**.

Guidelines

There are several important factors to consider when choosing a job queue for a specific program or custom script. In most cases, jobs are submitted via PBS scripts as described in Job Submission and Scheduling.

Overall System

Each MSI system contains job queues managing sets of hardware with different resource and policy limitations. MSI currently has three primary systems: the newest supercomputer Mesabi, the supercomputer Itasca, and the Lab compute cluster. Mesabi is MSI's newest supercomputer, with the highest performance hardware, and a wide variety of queues suitable for many different job types. **Mesabi should be your first choice when doing any computation at MSI**. Itasca is a supercomputer with queues most suitable for multi-node jobs which will complete within 1-2 days. The Lab cluster is primarily for interactive software that is graphical in nature, and testing. Which system to choose depends highly on which system has queues appropriate for your software/script. The variety of queues on Mesabi will be suitable for most users, but the Queue page should be examined.

Job Walltime (walltime=)

The job walltime is the time from the start to the finish of a job (as you would measure it using a clock on a wall), not including time spent waiting to run. This is in contrast to cputime, which measures the cumulative time all cores spent working on a job. Different job queues have different walltime limits, and it is important to choose a queue with a sufficiently high walltime that enables your job to complete. Jobs which exceed the requested walltime are killed by the system to make room for other jobs. Walltime limits are maximums only, and you can always request a shorter walltime, which will reduce the amount of time you wait in the queue for your job to start. If you are unsure how much walltime your job will need start with the queues with shorter walltime limits and only move to others if needed.

Job Nodes and Cores (nodes=X:ppn=Y)

Many calculations have the ability to use multiple cores (ppn), or (less often) multiple nodes, to improve calculation speed. Certain job queues have maximum or minimum values for the number nodes and cores a job may use. If **Node Sharing** is enabled for a queue you can request fewer cores (ppn) than exist on an entire node. If Node Sharing is not enabled then you must request resources equivalent to a multiple of an entire node. All Itasca queues, and Mesabi’s widest and large queues, **do not allow** Node Sharing**.**

Job Memory (mem=)

The memory which a job requires is an important factor when choosing a queue. The largest amount of memory (RAM) that can be requested for a job is limited by the memory on the hardware associated with that queue. Mesabi has two queues (ram256g and ram1t) with high memory hardware, the largest memory hardware is available through the ram1t queue. Itasca also has two queues with high memory hardware (sb128 and sb256).

User and Group Limitations

To efficiently share resources, many queues have limits on the number of jobs or cores a particular user or group may simultaneously use. If a workflow requires many jobs to complete, it can be helpful to choose queues which will allow many jobs to run simultaneously. Mesabi allows more simultaneous jobs to run than Itasca.

Special Hardware

Some queues contain nodes with special hardware, GPU accelerators and solid-state scratch drives being the most common. If a calculation needs to use special hardware, then it is important to choose a queue with the correct hardware available. Furthermore, those queues may require additional resources to be specified (e.g., GPU nodes require ":gpus=X").

Queue Congestion

At certain times particular queues may become overloaded with submitted jobs. In such a case, it can be helpful to send jobs to queues with lower utilization (node status). Sending jobs to lower utilization queues can decrease wait time and improve throughput. Care must be taken to make sure calculations will fit within queue limitations.

Accessing Software Resources

MSI provides access to approximately 400 software packages. To find out if a particular software package is installed, you can browse or search a list of the software packages that MSI provides under the Help and Documentation section of this website. The MSI webpage includes descriptions of software packages including example usage to help you get started.

Support Tiers

Software packages will be marked with a support tier -- either Primary, Secondary, or Minimal. Primary support is given to mature, popular scientific software important to a large fraction of the user base generally receive the highest level of support. These software are compiled, installed, benchmarked, and documented as a service to you. They are tested after major system upgrades, new versions are installed in a timely manner, and we expect to be able to offer advice to assist with the majority of inquiries. Conversely, Secondary or Minimally supported software are only useful to a very small number of users. These software may not be actively maintained, regularly updated, or tested after system updates. They may be removed without notice if usage is found to be too low to continue support.

How To Access Software

HPC and Interactive HPC Systems

To access software on Linux systems, use the module command to load the software into your environment. You can find the module name to load on the MSI software page for a package.

For example, if you want to use the 2014b version of MATLAB, you would type in your job script or at the command line:

```
% module load matlab/R2014b
% matlab
```

To find out what module versions are available, consult the software page or use the module avail command to search by name. For example to find out what versions of Python are available:

```
% module avail python
```

Windows Software

Windows-only software can be accessed and run via the Windows Virtual Environment (Citrix). Once a windows session is started you access software applications in the same manner as if you were using a physical Windows computer.

Need a Specific Package?

First try and install it yourself in your MSI home directory following the guidelines below:

Software Installation Guide

Installing software on any Unix platform can be challenging for inexperienced and veteran users alike.

While it is impossible to cover every issue you may experience, the following steps will allow you to compile and install in your own home directory many of the libraries and scientific applications that you will come across.

Introductory Compiling

Building a Python interpreter from its source code is straightforward, requiring only a C compiler, and will serve as an instructive example.

Installation materials are most commonly distributed as compressed tar files with the extension .tar.gz or .tar.bz.

These files can be untarred and unzipped with the tar -xzf and tar -xjf commands, respectively.

We have downloaded the file Python-2.7.2.tar.gz from www.python.org to our home directory, so the appropriate command is:

```
tar -xzf Python-2.7.2.tar.gz

cd Python-2.7.2
```

Your next step should always be to review the distributed files for a README file, which can be viewed in a text editor, or some other file that is named to indicate it contains installation instructions.

In many cases, and as is the case with Python, a configure script is included. In the simplest case, all that is necessary to compile the code is to run the configurescript, then make.

By default many codes will try to install to system directories you cannot access. The recommended installation method is to create a directory in your home directory for software installation.

Pwd (It will display a result such as /home/xe2/nlabello/Python-2.7.2)

mkdir /home/xe2/nlabello/software/python

./configure --prefix=/home/xe2/nlabello/software/python/

make install

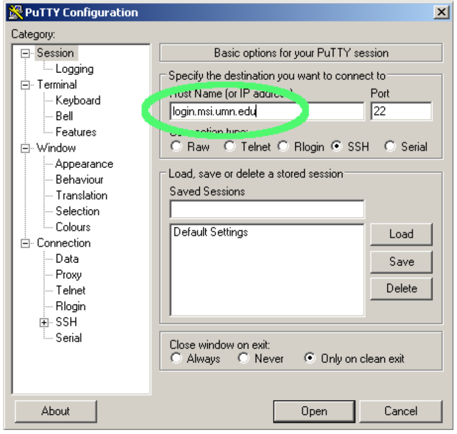
When the make install step completes you will have access to binary in /home/xe2/nlabello/software/python/bin

Installing QIIME

First Things First: Open the Terminal

Using QIIME on MSI

1. Download Putty on Windows (http://www.putty.org/) and configure it for MSI access



PuTTY Configuration

Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Close window on exit: ○ Always ○ Never ● Only on clean exit

1. Connecting to Mesabi and Itasca from Login Host. Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh (QIIME is currently loaded under Itasca so use ssh Itasca for QIIME analyses)

```
ssh mesabi or ssh itasca
```

2. Alternatively, connect to MSI using Terminal on Mac/Linux, located in Applications>Utilities>Terminal.app.

```
ssh yourusername@login.msi.umn.edu
```

3. Log in to an interactive node, making sure you have enough time and memory

```
isub -n nodes=1:ppn=4 -m 16GB -w 24:00:00
```

4. Load the QIIME module

```
module load qiime/1.9.1
```

MacQIIME Users Only:

If you are using MacQIIME, you will have to use your Mac OS X terminal, located in Applications>Utilities>Terminal.app.
Normally, when you run a command in the terminal, all the packages in MacQIIME will be invisible to your computer. In order to access macqiime, after starting a new terminal session you have to source the environment variables with the "macqiime" command. To source the QIIME environment variables and load up all the MacQIIME scripts in the PATH, you have to start the MacQIIME subshell:

macqiime

You should now be looking at a dollar sign ($) with a cursor after it. This is the command line interface for your computer. Here, you can type commands in order to "execute" programs. This Unix/Linux command line is a powerful place to analyze and process data. Everything you do in QIIME is executed through this command line interface. I'm going to assume you have some familiarity with the Unix command line, but I'll try to help you along the way with commands like **cp**, **cd**, **ls**, **pwd**, **less**, **nano**, etc.

***Note: Getting MacQIIME to work in El Capitan (OS 10.11)*

This section is required only if you are running OS 10.11 (El Capitan) or higher. Apple added a new security feature in El Capitan that makes it impossible to install software into /usr/bin/ folder (that was where I was putting the “macqiime” sourcing script. I'll fix this in the next release of MacQIIME. In the meantime, we can work around it. The quick solution is to use this command, instead of the “macqiime” script:

source /macqiime/configs/bash_profile.txt

How do I transfer data between a Mac or Windows computer and MSI Unix computers?

For transferring data from Mac / Windows computers to MSI Unix computers, any **SFTP** or **SCP** software can be used. Connect to login.msi.umn.edu using your MSI username and password and you will be able to transfer files to and from your home directory and group shared directory. For Windows, we recommend WinSCP. For Mac OS X, we recommend using FileZilla.

If you are on the Unix side and need to pull data from your Windows U: drive, you can use smbclient:

$ smbclient -U 'MSI\*username*' -D *username* //falcon2/Users

From the prompt:

smb: \username\>

You can issue ftp style commands ('ls', 'get', 'cd', etc.). To get an entire directory recursively, use:

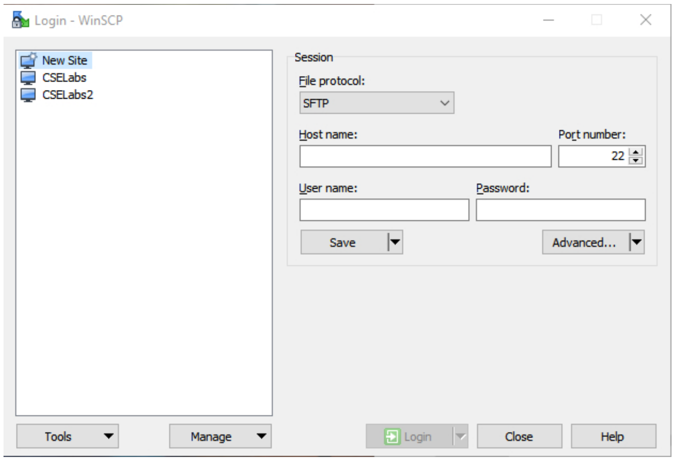
smb: \username\> prompt
smb: \username\> recurse
smb: \username\> mget *DirectoryName*

How do I upload files to my MSI home directory from Windows?

To upload files from your Windows machine to your home directory on a Unix machine, use WinSCP.

How do I use WinSCP to transfer data?

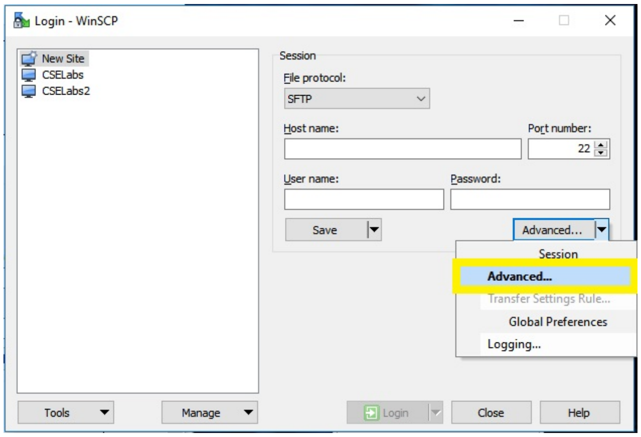
1. These directions explain how to transfer files back and forth between a Windows client and a Unix server. Please see the directions for remote access for an MSI Unix machine from a Windows client if you need to log in interactively.
2. Download and install WinSCP, a graphical SCP (secure copy protocol) file transfer client for Windows. These directions are based on version 5.7.5; upgrade if needed.
3. Double-click the WinSCP icon to reach this screen. If you just want to transfer files to or from your group home or lab scratch, you can skip steps 4 and 5 and proceed to step 6.



File protocol: SFTP

Port number: 22

4. (**Optional:** if you want to transfer files elsewhere other than login.msi) Click the "Advanced Options" box in the lower left-hand corner:

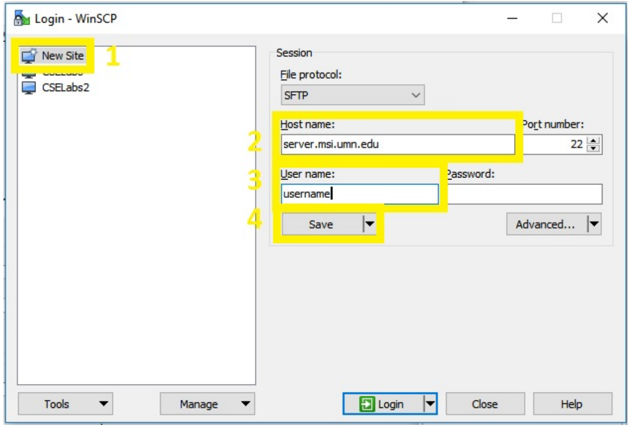


File protocol: SFTP

Port number: 22

5. (**Optional:** if you want to transfer files elsewhere other than login.msi) Under the Connection section, click on "Tunnel" to get to this screen and click "Connect through SSH tunnel". Fill in **login.msi.umn.edu** and your MSI username. Then click back in the left column on Session at the top.

6. (Mandatory for all cases) Enter the name of the MSI server you wish to use, such as itasca, cascade, hosting, etc, in this format: *server*.msi.umn.edu. (Do not use the actual name server.msi.umn.edu, which does not exist.) **If you skipped steps 4 and 5, the only option here is login.msi.umn.edu.** Enter your MSI username. Click "Save".



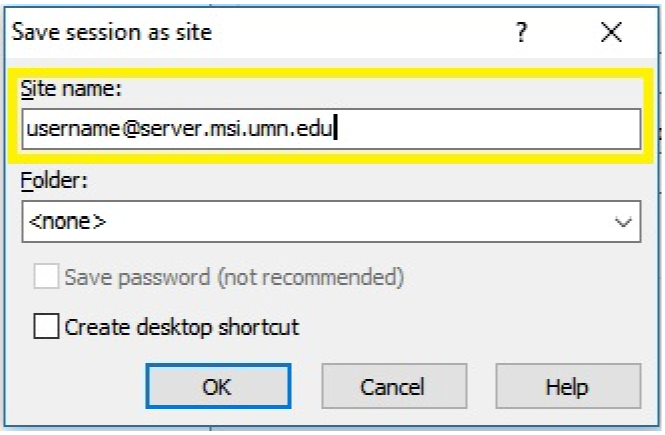
File protocol: SFTP

Host name: server.msi.umn.edu

Port number: 22

User name: username

7. Make sure the session is called something you want it to be called, then click "OK" on this screen:



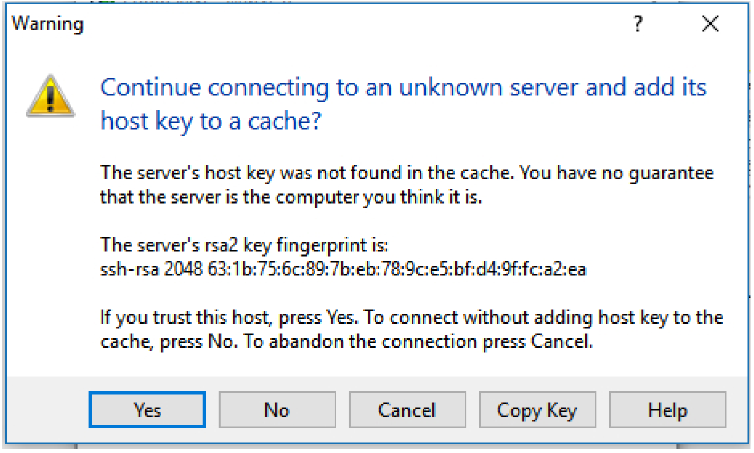
Site name: username@server.msi.umn.edu

Folder: <none>

□ Save password (not recommended)

□ Create desktop shortcut

8. The first time you connect you will get a "Warning" window concerning the server's key fingerprint. This is normal. Click "Yes". You may get this window twice.



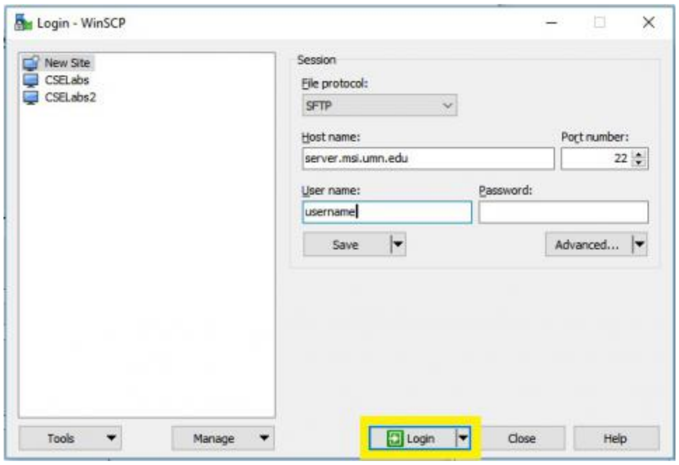
Continue connecting to an unknown server and add its host key to a cache?

The server's host key was not found in the cache. You have no guarantee that the server is the computer you think it is.

The server's rsa2 key fingerprint is:
ssh-rsa 2048 63:1b:75:6c:89:7b:eb:78:9c:e5:bf:d4:9f:fc:a2:ea

If you trust this host, press Yes. To connect without adding host key to the cache, press No. To abandon the connection press Cancel.

9. Login to the session



File protocol: SFTP

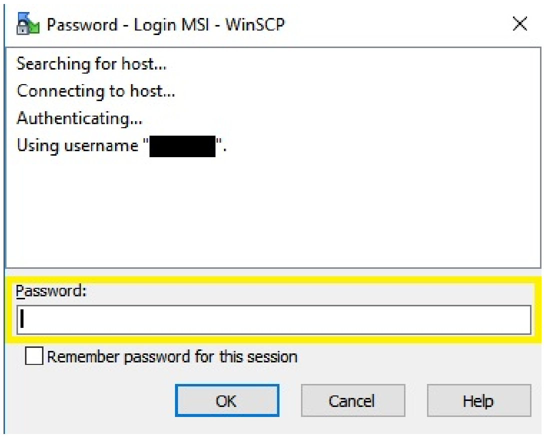
Host name: server.msi.umn.edu

Port number: 22

User name: username

Password:

10. You will now get a window asking for your password. After entering your password click "OK". Do not save your password! For alternatives to saving your password using ssh keys contact help@msi.umn.edu.



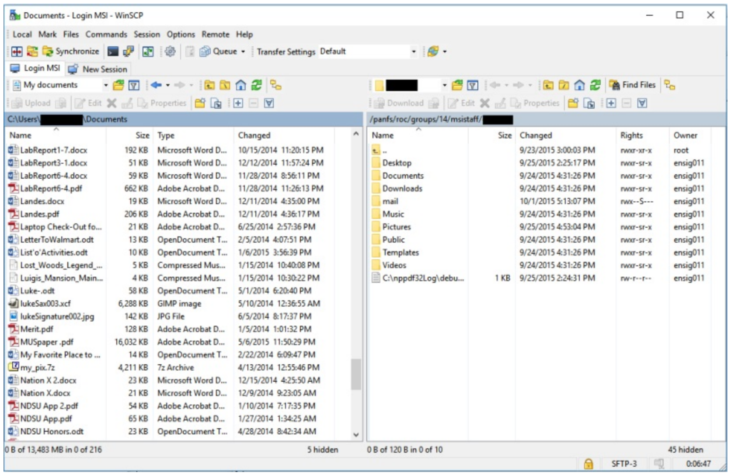
Password - Login MSI - WinSCP

```
Searching for host...
Connecting to host...
Authenticating...
Using username "[redacted]".
```

Password:

□ Remember password for this session

11. You may get the "Warning" message again at this point.
12. You may be prompted for the password a second time.
13. Once connected to the remote system, a window will appear showing the local and remote systems.



Documents - Login MSI - WinSCP

C:\Users\[redacted]\Documents

Name	Size	Type	Changed
LabReport1-7.docx	192 KB	Microsoft Word D...	10/15/2014 11:20:15 PM
LabReport3-1.docx	51 KB	Microsoft Word D...	12/12/2014 11:57:24 PM
LabReport6-4.docx	59 KB	Microsoft Word D...	11/28/2014 8:56:11 PM
LabReport6-4.pdf	662 KB	Adobe Acrobat D...	11/28/2014 11:26:13 PM
Landes.docx	19 KB	Microsoft Word D...	12/11/2014 4:35:00 PM
Landes.pdf	206 KB	Adobe Acrobat D...	12/11/2014 4:36:17 PM
Laptop Check-Out fo...	21 KB	Adobe Acrobat D...	6/25/2014 2:57:36 PM
LetterToWalmart.odt	13 KB	OpenDocument T...	2/5/2014 4:07:51 PM
List'o'Activities.odt	10 KB	OpenDocument T...	1/6/2015 3:56:39 PM
Lost_Woods_Legend_...	5 KB	Compressed Mus...	1/15/2014 10:40:08 PM
Luigis_Mansion_Main...	4 KB	Compressed Mus...	1/15/2014 10:30:22 PM
Luke-.odt	58 KB	OpenDocument T...	5/1/2014 6:20:40 PM
lukeSax003.xcf	6,288 KB	GIMP image	5/10/2014 12:36:55 AM
lukeSignature002.jpg	142 KB	JPG File	6/5/2014 8:17:37 PM
Merit.pdf	128 KB	Adobe Acrobat D...	1/5/2014 1:01:32 PM
MUSpaper .pdf	16,032 KB	Adobe Acrobat D...	5/6/2015 11:50:29 PM
My Favorite Place to ...	14 KB	OpenDocument T...	2/22/2014 6:09:47 PM
my_pix.7z	4,211 KB	7z Archive	4/13/2014 12:55:46 PM
Nation X 2.docx	23 KB	Microsoft Word D...	12/15/2014 4:25:50 AM
Nation X.docx	21 KB	Microsoft Word D...	12/9/2014 9:23:05 AM
NDSU App 2.pdf	54 KB	Adobe Acrobat D...	1/10/2014 7:17:35 PM
NDSU App.pdf	65 KB	Adobe Acrobat D...	1/27/2014 1:34:25 AM
NDSU Honors.odt	23 KB	OpenDocument T...	4/28/2014 8:42:34 AM

0 B of 13,483 MB in 0 of 216 — 5 hidden

/panfs/roc/groups/14/msistaff/[redacted]

Name	Size	Changed	Rights	Owner
..		9/23/2015 3:00:03 PM	rwxr-xr-x	root
Desktop		9/25/2015 2:25:17 PM	rwxr-sr-x	ensig011
Documents		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Downloads		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
mail		10/1/2015 5:13:07 PM	rwx--S---	ensig011
Music		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Pictures		9/25/2015 4:53:04 PM	rwxr-sr-x	ensig011
Public		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Templates		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Videos		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
C:\nppdf32Log\debu...	1 KB	9/25/2015 2:24:31 PM	rw-r--r--	ensig011

0 B of 120 B in 0 of 10 — 45 hidden

SFTP-3 0:06:47

14. To move to the desired directories, use the navigation bars at the top. The local system is on the left and the remote system on the right. The toolbar at the bottom of the window contains function keys for performing copies, deletes, etc.
15. To perform an action, highlight the desired files or directories and press the appropriate function key (F5 for a copy). It will then prompt you for confirmation. Click "OK". Depending on the size of the file(s) and the speed of your network connection, an upload or download could take anywhere from a few seconds to several minutes.
16. When you are ready to disconnect from the remote system, press F10 or just exit from the WinSCP program.

*****Note**
I saved the microbiome analysis files under the following paths

C:\Users\onyea005\Documents\mice_tutorial
C:\Users\onyea005\Documents\qiime_tutorial

Command Line Shortcuts

- Cd = change directory
- Ls = list files within a folder
- Mkdir = create a new folder
- Git + clone + pathfile + filename = copying a folder from another source
- Head = look at the first 10 lines of a file
- Clear = remove a command
- -I = input file
- -O = output file
- Mote.emperor = 3D plot package for ordination and clustering

Part 11_QIIME setup instructions_HMP_MACQIIME

Unix Guide
QIIME installation Guide from the HMP website
MacQIIME installation guide from Jeff Werner's website
QIIME 2 installation Guide from the HMP website
R guide for MACOSX

R - ***Installing R on your system and adding the following libraries has become much more important in the latest QIIME versions 1.9+.*** There are now many neat features in QIIME that require R. If you don't have R installed, you can get it from here. *Please note* that even if you installed R and these libraries previously for MacQIIME 1.8.0, you should still upgrade to/install the latest version of R, at least version **3.1.2**, and re-install all these R packages to get everything working. Make sure the R executable is in your PATH. You will have to open R and install additional packages: in the R command line, do the following, in the following order, one line at a time. **(*** Figure out how to pull out the script window in R for MAC: Open R > Open a document editor, type your code there, then copy and paste into the R code window as needed)**

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("DESeq2")
biocLite("biomformat")
biocLite("metagenomeSeq")

install.packages('randomForest')
install.packages('ape')
install.packages('vegan')
install.packages('optparse')
install.packages('gtools')
install.packages('klaR')
install.packages('RColorBrewer')
```

What it's used for in QIIME: There are now **many** statistical functions in QIIME that use these R libraries

How split code over multiple lines in an R script

```
setwd('~/a/very/long/path/here/that/goes/beyond/80/characters/and/then/some/more')
```

I want to split a line in an R script over multiple lines (because it is too long). How do I do that? Is it possible to split the long path over multiple lines? I tried

```
setwd('~/a/very/long/path/here/that/goes/beyond/80/characters/and/
then/some/more')
```

with `return` key at the end of the first line; but that does not work.

R doesn't need to be told the code starts at the next line. It will just continue to read the next line whenever it considers the statement as "not finished". Actually, in your case it also went to the next line, but R takes the return as a character when it is placed between "".

You are not breaking *code* over multiple lines, but rather *a single identifier*. There is a difference. So, when spreading code over multiple lines, you have to make sure that R knows something is coming, either by:

- leaving a bracket open, or
- ending the line with an operator

You'll have to make sure your code isn't finished. For example, compare

```
a <- 1 + 2
+ 3
```

with

```
a <- 1 + 2 +
3
```

For your issue, try

```
R> setwd(paste("~/a/very/long/path/here",
               "/and/then/some/more",
               "/and/then/some/more",
               "/and/then/some/more", sep=""))
```

which also illustrates that it is perfectly fine to break code across multiple lines. When we're talking strings, this still works but you need to be a bit careful. You can open the quotation marks and R will read on until you close it. But every character, including the newline, will be seen as part of the string :

```
x <- "This is a very
long string over two lines."
x
## [1] "This is a very\nlong string over two lines."
cat(x)
## This is a very
## long string over two lines.
```

That's the reason why in this case, your code didn't work: a path can't contain a newline character (`\n`). So that's also why you better use the solution with `paste()` or `paste0()` Dirk proposed.
Dirk's method above will absolutely work, but if you're looking for a way to bring in a long string where whitespace/structure is important to preserve (example: a SQL query using RODBC) there is a two step solution.

1) Bring the text string in across multiple lines

```
long_string <- "this
is
a
long
string
with
whitespace"
```

2) R will introduce a bunch of `\n` characters. Strip those out with `strwrap()`, which destroys whitespace, per the documentation:

```

strwrap(long_string, width=10000, simplify=TRUE)
```

By telling strwrap to wrap your text to a very, very long row, you get a single character vector with no whitespace/newline characters.

The preview version of RStudio is a lot smarter about statement execution. Ctrl + Enter or Cmd + Enter on OS X) will execute your whole statement no matter how many lines it's spread over. You can download the preview here:

https://www.rstudio.com/products/rstudio/download/preview/

IMPORTING DATA INTO R

R has many functions that allow you to import data from other applications. The following table lists some of the useful text import functions, what they do, and examples of how to use them.

Function	What It Does	Example
read.table()	Reads any tabular data where the columns are separated (for example by commas or tabs). You can specify the separator (for example, commas or tabs), as well as other arguments to precisely describe your data.	read.table(file=”myfile”, sep=”t”, header=TRUE)
read.csv()	A simplified version of read.table()with all the arguments preset to read CSV files, like Microsoft Excel spreadsheets.	read.csv(file=”myfile”)
read.csv2()	A version of read.csv() configured for data with a comma as the decimal point and a semicolon as the field separator.	read.csv2(file=”myfile”, header=TRUE)
read.delim()	Useful for reading delimited files, with tabs as the default separator.	read.delim(file=”myfile”, header=TRUE)
scan()	Allows you finer control over the read process when your data isn’t tabular.	scan(“myfile”, skip = 1, nmax=100)
readLines()	Reads text from a text file one line at a time.	readLines(“myfile”)
read.fwf	Read a file with dates in fixed-width format. In other words, each column in the data has a fixed number of characters.	read.fwf(“myfile”, widths=c(1,2,3)

In addition to these options to read text data, the package foreign allows you to read data from other popular statistical formats, such as SPSS. To use these functions, you first have to load the built-in foreign package, with the following command:

```
library("foreign")
```

The following table lists the functions to import data from SPSS, Stata, and SAS.

Function	**What It Does**	**Example**
read.spss	Reads SPSS data file	read.spss(“myfile”)
read.dta	Reads Stata binary file	read.dta(“myfile”)
read.xport	Reads SAS export file	read.export(“myfile”)

HOW TO STRUCTURE YOUR CODE IN R

Names aren’t the only things that can influence the readability of your R code. When you start nesting functions or perform complex calculations, your code can turn into a big mess of text and symbols rather quickly.

Luckily, you have some tricks to clear up your code so you can still decipher what you did three months down the road.

Nesting functions and doing complex calculations can lead to very long lines of code. If you want to make a vector with the names of your three most beloved song titles, for example, you’re already in for trouble. Luckily, R lets you break a line of code over multiple lines in your script, so you don’t have to scroll to the right the whole time.

You don’t even have to use a special notation or character. R will know that the line isn’t finished as long as you give it some hint. Generally speaking, you have to make sure the command is undoubtedly incomplete. There are several ways to do that:

- **You can use a quotation mark to start a string.** R will take all the following input — including the line breaks — as part of the string, until it meets the matching second quotation mark.
- **You can end the incomplete line with an operator (like +, /, <-, and so on).** R will know that something else must follow. This lets you create structure in longer calculations.
- **You can open a parenthesis for a function.** R will read all the input it gets as one line until it meets the matching parenthesis. This allows you to line up arguments below a function, for example.

The following little script shows all these techniques:

```
baskets.of.Geraldine <-
    c(5,3,2,2,12,9)
Intro <- "It is amazing! The All Star Grannies scored
a total of"
Outro <- "baskets in the last six games!"
Total.baskets <- baskets.of.Granny +
       baskets.of.Geraldine
Text <- paste(Intro,
      sum(Total.baskets),
      Outro)
cat(Text)
```

You can copy this code into a script file and run it in the console. If you run this little snippet of code, you see the following output in the console:

```
It is amazing! The All Star Grannies scored
a total of 71 baskets in the last six games!
```

This immediately shows what the cat() function does. It prints whatever you give it as an argument directly to the console. It also interprets special characters like line breaks and tabs. If you look at the vector Text, you would see this:

```
> Text
[1] "It is amazing! The All Star Grannies scored na total of 71 baskets in the last six games!"
```

The n represents the line break. Even though it's pasted to the a, R will recognize n as a separate character. All this also works at the command line. If you type an unfinished command, R will change the prompt to a + sign, indicating that you can continue to type your command:

```
> cat('If you doubt whether it works,
+ just try it out.')
If you doubt whether it works,
just try it out.
RStudio automatically adds a line break at the end of a cat() statement if there is none, but R doe
sn't do that. So, if you don't use RStudio, remember to add a line break (or the symbol n) at the e
nd of your string.
```

HOW TO COMBINE LOGICAL STATEMENTS IN R

Life would be boring in R if you couldn’t combine logical statements. If you want to test whether a number lies within a certain interval, for example, you want to check whether it’s greater than the lowest value and less than the top value.

To illustrate, let’s assume you have two vectors containing the number of baskets that Granny and her friend Geraldine scored in the six games of this basketball season:

```
> baskets.of.Granny <- c(12,4,4,6,9,3)

> baskets.of.Geraldine <- c(5,3,2,2,12,9)
```

Maybe you want to know the games in which Granny scored the fewest or the most baskets. For that purpose, R has a set of logical operators that — you guessed it — are nicely vectorized.

To illustrate, using the knowledge you have now, try to find out the games in which Granny scored the fewest baskets and the games in which she scored the most baskets:

1. Create two logical vectors, as follows:

   ```
   > min.baskets <- baskets.of.Granny == min(baskets.of.Granny)

   > max.baskets <- baskets.of.Granny == max(baskets.of.Granny)
   ```

   `min.baskets` tells you whether the value is equal to the minimum, and `max.baskets` tells you whether the value is equal to the maximum.

2. Combine both vectors with the `OR` operator (|), as follows:

   ```
   > min.baskets | max.baskets

   [1] TRUE FALSE FALSE FALSE FALSE TRUE
   ```

   This method actually isn’t the most efficient way to find those values. This example clearly shows you how vectorization works for logical operators.

   The `NOT` operator (`!`) is another example of the great power of vectorization. The `NA` values in the vector `x` have caused some trouble already, so you’d probably like to get rid of them. You know that you have to check whether a value is missing by using the `is.na()` function.

   But you need the values that are *not* missing values, so invert the logical vector by preceding it with the `!` operator. To drop the missing values in the vector `x`, for example, use the following code:

   ```
   > x[!is.na(x)]

   [1] 3 6 2 1
   ```

R Code Tutorial based on PUBH 7445 Homework Fall 2016

```
##############################################################################
##############################################################################

### PUBH 7445 assignment 1

### Problem 2.3
bill = c(46, 33, 39, 37, 46, 30, 48, 32, 49, 35, 30, 48)
sum (bill)
max (bill)
min (bill)
which (bill >= 40)
summary (bill >40)
Table (bill >=40)
5/12

### Problem 2.6

x = c(1, 8, 2, 6, 3, 8, 5, 5, 5, 5)
(sum (x))/10
y = log10 (x)
y
z = (x - 4.4)/2.875
z
max(x) - min(x)

### Problem 3.4
data(faithful)
dim(faithful)
names(faithful)
summary (faithful$eruption)
summary (faithful$waiting)

data(iris)
dim(iris)
names(iris)
summary (iris$Sepal.Length)
summary (iris$Sepal.Width)
summary (iris$Petal.Length)
summary (iris$Petal.Width)
summary (iris$Species)

### Problem 3.6
oring = c(0, 1, 0, NA, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 3, 0, 0, 0, 0, 0, 2, 0, 1)
table (oring)
```

```
mean(oring,na.rm=TRUE)

### Problem 4.2
library(MASS)
data(UScereal)
attach(UScereal)
names(UScereal)

table (UScereal$mfr)
table (UScereal$shelf)

# The relationship between manufacturer and shelf

barplot(table(mfr,shelf),
beside=TRUE # put beside not stacked

plot (mfr,shelf),

# The relationship between fat and vitamins
plot (fat,vitamins)
plot (vitamins,fat)

# The relationship between fat and shelf

barplot (shelf,fat)
boxplot (shelf,fat)
plot (shelf,fat)

# The relationship between carbohydrates and sugars

plot (carbo, sugars)

# The relationship between ?bre and manufacturer

plot (mfr, fibre)

# the relationship between sodium and sugars

plot (sodium,sugars)

# Custom Problem

data(DNase)
names(DNase)
table(DNase$Run)
plot (DNase$conc,DNase$density)
plot (DNase$Run,DNase$conc)
plot (DNase$Run,DNase$density)

# Multiple Linear Regression Example
fit <- lm(y ~ x1 + x2 + x3, data=mydata)
summary(fit) # show results

fit <- lm(DNase$density ~ DNase$Run + DNase$conc)
summary(fit)

#Custom Problem 2

data(npk)
names(npk)

#use the melt function from the reshape2 package to convert the wide dataset to the long one,
before making the boxplot)
#http://www.cookbook-
r.com/Manipulating_data/Converting_data_between_wide_and_long_format/

install.packages("reshape2")
library (reshape2)
npk

### Example of how to use the melt command
data_long <- melt(olddata_wide,
      # ID variables - all the variables to keep but not split apart on
   id.vars=c("subject", "sex"),
      # The source columns
   measure.vars=c("control", "cond1", "cond2" ),
      # Name of the destination column that will identify the original
      # column that the measurement came from
   variable.name="condition",
   value.name="measurement")
###

test_npk <- melt(npk, id.vars=c("block", "yield"), measure.vars=c("N", "P", "K"),
variable.name="condition", value.name="measurement")
test_npk

#need to make a boxplot of yield by block, with different values for N,P,K within each block
boxplot (test_npk$yield,test_npk$condition)
boxplot(yield ~ condition       , data=test_npk)
```

```
#Problem 6.3
#rnorm(n,mean=0,sd=1)
random <- rnorm (n=100, mean=100, sd=10)
table (random < 80)
table (random > 120)

#Problem 6.8
> pnorm(.7,1,1) # normal mean 1, std 1
> qnorm(.75) = 0.6744898 Notationally, this is ?nding z which solves 0.75 = P(Z = z)

qnorm(0.525,0,1)

#Problem 6.9
pnorm (1.5, 0, 2)

#Problem 7.3
#X = rnorm(100,mu,sigma) # generate random data
#rbinom(N,1,p)
#generates a sequence of N trials, each with probability p of success. Actually, R returns a
sequence of 1's and 0's, but we can identify 1's with successes and 0's with failures if we like.
#Remember this is different from a bernoulli distribution because there is only 1 trial (n=1)
#Binomial(n,p) This distribution arises from a sequence of n independent Bernoulli trials, each
with probability p of success. Since the number of successes in n trials can range from 0 to n, the
sample space is just the integers 0,1,2, ... n. For example, the number of heads that occur in 20
independent tosses of a fair coin has a Binomial(20,.5) distribution. In R:rbinom(N,n,p)
generates N independent samples from a binomial(n,p) distribution.

simulation1 <- rbinom (100, 100, 0.25)
hist (simulation1)

simulation2 <- rbinom (100, 100, 0.05)
hist (simulation2)

simulation3 <- rbinom (100, 100, 0.01)
hist (simulation3)

## Problem 7.5

#While we are at it, we should learn the ”right” way to write functions. We should be able to
modify n the number of trials and p the success probability in our function.
#The format for the variable is n=100 this says that n is the ?rst variable given to the function, by
default it is 100, p is the second by default it is p=.5. Now we would call simple.sim as
#simple.sim(1000,f,n,p)

###
```

```
f=function(n=1,a=0,b=1) { mu=(b+a)/2 sigma=(b-a)/sqrt(12) (mean(runif(n,a,b))-mu)/(sigma/sqrt(n)) }
simulation4 = simple.sim(100,f,1)
hist(simulation4,breaks=10,prob=TRUE)

simulation5 = simple.sim(100,f,5)
hist(simulation5,breaks=10,prob=TRUE)

simulation6 = simple.sim(100,f,10)
hist(simulation6,breaks=10,prob=TRUE)

simulation7 = simple.sim(100,f,25)
hist(simulation7,breaks=10,prob=TRUE)

###

# uniform function

sim4 <- runif(n=1,min=0,max=1)
hist (sim4)

sim5 <- runif(n=5,min=0,max=1)
hist (sim5)

sim6 <- runif(n=10,min=0,max=1)
hist (sim6)

sim7 <- runif(n=25,min=0,max=1)
hist (sim7)

### Problem 7.10
#comparing wether the mean or median are the better options

install.packages("UsingR")
```

Normal function

```
median.normal = function(n=100) median(rnorm(100,0,1))
mean.normal = function(n=100) mean(rnorm(100,0,1))
var(simple.sim(100,mean.normal)) / var(simple.sim(100,median.normal))
```

Exponential function

```
mean.exp = function(n=100) mean(rexp(n,1/10))
median.exp = function(n=100) median(rexp(n,1/10))
var(simple.sim(100,mean.exp)) / var(simple.sim(100,median.exp))
```

t distribution

```
mean.t = function(n=100) mean(rt(n,2))
median.t = function(n=100) median(rt(n,2))
var(simple.sim(100,mean.t)) / var(simple.sim(100,median.t))

####################################################################################
####################################################################################

### PUBH 7445 assignment 2

#Problem 8.6
data(trees)
names(trees)
hist(trees$Girth)
boxplot(trees$Girth)
hist(trees$Height)
boxplot(trees$Height)
hist(trees$Volume)
boxplot(trees$Volume)

#Problem 9.1
x=rnorm(15,mean=10,sd=5)
t.test(x)

#One sample z test aren't in R, you have to create your own function.

n = length(x)
sigma = 5
sem = sigma/sqrt(n) ##standard error of the mean
moe = qnorm(0.95)*sem ##margin of error
xbar = mean (x)
ci = xbar + c(-moe, moe) ##95% CI for a one sided z-test

#Problem 9.2

#loop a simulation 100 times
install.packages("UsingR")
library(UsingR)

##this version did not work for building a custom function
f= function (m,SE) {m=mean(rnorm(15,mean=10,sd=5)) SE = 5/sqrt(15)}
xbar = simple.sim(100,f)
alpha = 0.05
zstar = qnorm(1-alpha/2)
```

```
sum(abs(xbar-10) < zstar*SE

# the back arrow and equal sign are the same
f <- function() mean(rnorm(15, mean=10, sd=5))
SE <- 5/sqrt(15)
xbar <- simple.sim(100, f)
alpha = 0.05; zstar = qnorm(1-alpha/2); sum(abs(xbar-10) < zstar*SE)
```

Problem 9.8

```
##Uniform function
f1 <- function()  y = (runif(n, min = 0, max = 1))
f2 <- function()  t = (mean(y) - 0) / (sqrt(var(y))/sqrt(n))
sample1 = simple.sim(100,f1)
sample2 = simple.sim(100,f2)
qqplot(sample,rt(100,df=9),main="sample vs. t");qqline(sample)
qqnorm(sample,main="sample vs. normal");qqline(sample)
hist(sample)

### t distribution
f1 <- function()  y = (rt(n, df, ncp))
f2 <- function()  t = (mean(y) - 0) / (sqrt(var(y))/sqrt(n))
sample1 = simple.sim(100,f1)
sample2 = simple.sim(100,f2)
qqplot(sample,rt(100,df=9),main="sample vs. t");qqline(sample)
qqnorm(sample,main="sample vs. normal");qqline(sample)
hist(sample)

###exponential uniform function
f1 <- function()  y = (rexp(n, rate = 1))
f2 <- function()  t = (mean(y) - 0) / (sqrt(var(y))/sqrt(n))
sample1 = simple.sim(100,f1)
sample2 = simple.sim(100,f2)
qqplot(sample,rt(100,df=9),main="sample vs. t");qqline(sample)
qqnorm(sample,main="sample vs. normal");qqline(sample)
hist(sample)

###Custom Problem 1
data(Theoph)
names(Theoph)
table(Theoph$Dose,Theoph$conc>10)
3/129
prop.test(3,132,p=.5)
binom.test(3, 132, p = 0.5)

###problem 10.2
```

```r
data(smokyph)
names(smokyph)
hist(smokyph$waterph)
summary(smokyph$waterph)
t.test(smokyph$waterph,mu=7) # Ho: mu=7

###problem 11.2
data(corn)
names(corn)
boxplot (corn$New,corn$Standard)
t.test(corn$New,corn$Standard,alt="two.sided",var.equal=TRUE) ## two sided ttest
t.test(corn$New,corn$Standard,alt="two.sided",var.equal=TRUE, paired=TRUE) ## Matched two sided ttest

##Problem 12.3
#report the distribution
freq = c(53,22,49)
# specify probabilities (they must sum up to 1, so divide by n)
probs = c(5,3,4)/12
chisq.test(freq,p=probs)

##problem 12.4
data(UCBAdmissions)
names(UCBAdmissions)
x = ftable(UCBAdmissions)
x
x[1:2,]
admitted_m = c(x[1,])
admitted_f = c(x[2,])
rejected_m = c(x[3,])
rejected_f = c(x[4,])
##test for homogeneity among the admitted
chisq.test(rbind(admitted_m,admitted_f))
##test for homogeneity among the rejected
chisq.test(rbind(rejected_m,rejected_f))

##problem 13.4
elevation = c(600, 1000, 1250, 1600, 1800, 2100, 2500, 2900)
temperature= c(56, 54, 56, 50, 47, 49, 47, 45)

plot(elevation,temperature) # make a plot
abline(lm(temperature ~ elevation)) # plot the regression line

lm.result=simple.lm(elevation,temperature)
summary(lm.result)
lm.res = resid(lm.result)
```

```
summary(lm.res) ## check the summary to make sure that the model is appropriate for the data

##hypothesis testing
es = resid(lm.result) # the residuals lm.result
lm.result=simple.lm(elevation,temperature)
coefficients(lm.result)
t = ((-.0051146) - (-0.00534) )/(0.0009214) # the null is 5.34 degrees per 1000 feet or (-0.00534)
pt(t,6,lower.tail=TRUE) # find the right tail for this value of t # and 8-2 d.f

##Problem 14.5 ##go to the CRAN website for variable descriptions
data(babies)
names(babies)
babywt = c(babies$wt<999)
babywt
babies$wt
summary(lm(babywt ~ babies$age + babies$ht + babies$wt1))

##problem 15.2
data(failrate)

####################################################################################
####################################################################################

### PUBH 7445 assignment 3

# Specify the web location of the data file as follows

fmsURL <- "http://www.biostat.umn.edu/~ cavanr/FMS data.txt"


# First way did not work, Use the read.delim() function to pull the data into R directly from the
textbook website:

fms <- read.delim("http://www.biostat.umn.edu/~cavanr/FMS_data.txt",
header=T, sep="\t")

# Looking at variables
names (fms)
summary(fms$Race)

#Overall Minor Allele Frequency

#Identifying the minor allele frequency
GenoCount <- summary(fms$actn3_1671064)
GenoCount
```

```
#selecting individuals with only complete data for our minor allele frequency denominator
NumbObs <- sum(!is.na(fms$actn3_1671064))
NumbObs

# Get the genotype frequency for AA GA GG and NA's
GenoFreq <- as.vector(GenoCount/NumbObs)
GenoFreq

# Calculate the A Frequency
FreqA <- (2*GenoFreq[1] + GenoFreq[2])/2
FreqA

# Calculate the G Frequency
FreqG <- (GenoFreq[2] + 2*GenoFreq[3])/2
FreqG

# Using the genetics package to get the frequencies stratified by race
install.packages("genetics")
library(genetics)

Geno <- genotype(fms$actn3_1671064,sep="")
summary(Geno)
summary(subset Geno, if fms$race=="African Am")
summary(Geno) if fms$race=="Am Indian"
summary(Geno) if fms$race=="Asian"
summary(Geno) if fms$race=="Caucasian"
summary(Geno) if fms$race=="Hispanic"
summary(Geno) if fms$race=="Other"

install.packages("doBy")
library(doBy)
summaryBy(geno, fms$race)
table(geno,fms$Race)

#Problem 1.5

vircoURL <- "http://www.biostat.umn.edu/~cavanr/Virco_data.csv"
virco <- read.csv(file=vircoURL)
dim(virco)
virco[1:5,c(1,6,11,32,85,93,104,112,122)]

names(virco)
P1Count <-summary(virco$P1)
P1num <- sum(!is.na(!as.numeric(virco$P1)))
P1num
P1denom <- sum(!is.na(virco$P1))
```

```
P1denom
P1Freq <- as.vector(P1num/P1denom)
P1Freq

count(virco$P1)
table(virco$P1)

test <- 100 * length(which(virco$P1 == "-")) / length(virco$p1)
perc (virco$P1, "-")

P10Count <-summary(virco$P10)
P10Obs <- sum(!is.na(virco$P10))
P10Freq <- as.vector(P10Count/P10Obs)
P10Freq

P30Count <-summary(virco$P30)
P30Obs <- sum(!is.na(virco$P30))
P30Freq <- as.vector(P30Count/P30Obs)
P30Freq

P71Count <-summary(virco$P71)
P71Obs <- sum(!is.na(virco$P71))
P71Freq <- as.vector(P71Count/P71Obs)
P30Freq

P82Count <-summary(virco$P82)
P82Obs <- sum(!is.na(virco$P82))
P82Freq <- as.vector(P82Count/P82Obs)
P82Freq

P90Count <-summary(virco$P90)
P90Obs <- sum(!is.na(virco$P90))
P90Freq <- as.vector(P90Count/P90Obs)
P90Freq

#manually calculating the proportions
summary(virco$P1)
length(!is.na(virco$P1))

summary(virco$P10)
length(!is.na(virco$P10))

summary(virco$P30)
length(!is.na(virco$P30))

summary(virco$P71)
```

```
length(!is.na(virco$P71))

summary(virco$P82)
length(!is.na(virco$P82))

summary(virco$P90)
length(!is.na(virco$P90))

#Problem 3.2

FMSAkt1Snp1 <- genotype(fms$akt1_t10726c_t12868c,sep="")
HWE.chisq(FMSAkt1Snp1)

summary(fms$Race)

#Checking HWE by race
HWEGeoArea <- tapply(FMSAkt1Snp1,INDEX=fms$Race,HWE.chisq)
HWEGeoArea$"African Am"
HWEGeoArea$"Am Indian"
HWEGeoArea$"Asian"
HWEGeoArea$"Caucasian"
HWEGeoArea$"Hispanic"
HWEGeoArea$"Other"

#Problem 3.4
hgdpURL <- "http://www.biostat.umn.edu/~cavanr/HGDP_AKT1.txt"
hgdp <- read.delim(file=hgdpURL)
head(hgdp)
dim(hgdp)
summary(hgdp)

#make a dataframe for all snps
Akt1Snp1 <- genotype(hgdp$AKT1.C0756A,sep="")
Akt1Snp2 <- genotype(hgdp$AKT1.C6024T,sep="")
Akt1Snp3 <- genotype(hgdp$AKT1.G2347T,sep="")
Akt1Snp4 <- genotype(hgdp$AKT1.G2375A,sep="")
Akt1AllSnps <- data.frame(Akt1Snp1,Akt1Snp2,Akt1Snp3,Akt1Snp4)

Akt1Snp1[1:10]
Akt1Snp2[1:10]
Akt1Snp3[1:10]
Akt1Snp4[1:10]

#Pairwise LD
LD(Akt1AllSnps)$"D'"
```

```
#By race
hgdppakistan <-hgdp[hgdp$Geographic.area=="Pakistan", ]
summary (hgdppakistan)

Akt1Snp1 <- genotype(hgdppakistan$AKT1.C0756A,sep="")
Akt1Snp2 <- genotype(hgdppakistan$AKT1.C6024T,sep="")
Akt1Snp3 <- genotype(hgdppakistan$AKT1.G2347T,sep="")
Akt1Snp4 <- genotype(hgdppakistan$AKT1.G2375A,sep="")
Akt1AllSnps <- data.frame(Akt1Snp1,Akt1Snp2,Akt1Snp3,Akt1Snp4)

Akt1Snp1[1:10]
Akt1Snp2[1:10]
Akt1Snp3[1:10]
Akt1Snp4[1:10]

#Pairwise LD
LD(Akt1AllSnps)$"D'"

hgdpchina <-hgdp[hgdp$Geographic.area=="China", ]
summary (hgdpchina)

Akt1Snp1 <- genotype(hgdpchina$AKT1.C0756A,sep="")
Akt1Snp2 <- genotype(hgdpchina$AKT1.C6024T,sep="")
Akt1Snp3 <- genotype(hgdpchina$AKT1.G2347T,sep="")
Akt1Snp4 <- genotype(hgdpchina$AKT1.G2375A,sep="")
Akt1AllSnps <- data.frame(Akt1Snp1,Akt1Snp2,Akt1Snp3,Akt1Snp4)

Akt1Snp1[1:10]
Akt1Snp2[1:10]
Akt1Snp3[1:10]
Akt1Snp4[1:10]

#Pairwise LD
LD(Akt1AllSnps)$"D'"

hgdpisrael <-hgdp[hgdp$Geographic.area=="Israel", ]
summary (hgdpisrael)

Akt1Snp1 <- genotype(hgdpisrael$AKT1.C0756A,sep="")
Akt1Snp2 <- genotype(hgdpisrael$AKT1.C6024T,sep="")
Akt1Snp3 <- genotype(hgdpisrael$AKT1.G2347T,sep="")
Akt1Snp4 <- genotype(hgdpisrael$AKT1.G2375A,sep="")
Akt1AllSnps <- data.frame(Akt1Snp1,Akt1Snp2,Akt1Snp3,Akt1Snp4)

Akt1Snp1[1:10]
Akt1Snp2[1:10]
```

```
Akt1Snp3[1:10]
Akt1Snp4[1:10]

#Pairwise LD
LD(Akt1AllSnps)$"D'"

hgdpse <-hgdp[hgdp$Geographic.area=="Southern Europe", ]
summary (hgdpse)

Akt1Snp1 <- genotype(hgdpse$AKT1.C0756A,sep="")
Akt1Snp2 <- genotype(hgdpse$AKT1.C6024T,sep="")
Akt1Snp3 <- genotype(hgdpse$AKT1.G2347T,sep="")
Akt1Snp4 <- genotype(hgdpse$AKT1.G2375A,sep="")
Akt1AllSnps <- data.frame(Akt1Snp1,Akt1Snp2,Akt1Snp3,Akt1Snp4)

Akt1Snp1[1:10]
Akt1Snp2[1:10]
Akt1Snp3[1:10]
Akt1Snp4[1:10]

#Pairwise LD
LD(Akt1AllSnps)$"D'"

hgdpca <-hgdp[hgdp$Geographic.area=="Central Africa", ]
summary (hgdpca)

Akt1Snp1 <- genotype(hgdpca$AKT1.C0756A,sep="")
Akt1Snp2 <- genotype(hgdpca$AKT1.C6024T,sep="")
Akt1Snp3 <- genotype(hgdpca$AKT1.G2347T,sep="")
Akt1Snp4 <- genotype(hgdpca$AKT1.G2375A,sep="")
Akt1AllSnps <- data.frame(Akt1Snp1,Akt1Snp2,Akt1Snp3,Akt1Snp4)

Akt1Snp1[1:10]
Akt1Snp2[1:10]
Akt1Snp3[1:10]
Akt1Snp4[1:10]

#Pairwise LD
LD(Akt1AllSnps)$"D'"

hgdprussia <-hgdp[hgdp$Geographic.area=="Russia", ]
summary (hgdprussia)

Akt1Snp1 <- genotype(hgdprussia$AKT1.C0756A,sep="")
Akt1Snp2 <- genotype(hgdprussia$AKT1.C6024T,sep="")
Akt1Snp3 <- genotype(hgdprussia$AKT1.G2347T,sep="")
```

```
Akt1Snp4 <- genotype(hgdprussia$AKT1.G2375A,sep="")
Akt1AllSnps <- data.frame(Akt1Snp1,Akt1Snp2,Akt1Snp3,Akt1Snp4)

Akt1Snp1[1:10]
Akt1Snp2[1:10]
Akt1Snp3[1:10]
Akt1Snp4[1:10]

#Pairwise LD
LD(Akt1AllSnps)$"D'"


##### test LD code  #####
#Individual LD
LD(Akt1Snp1,Akt1Snp2)$"D'"
LD(Akt1Snp1,Akt1Snp3)$"D'"
LD(Akt1Snp1,Akt1Snp4)$"D'"
LD(Akt1Snp2,Akt1Snp3)$"D'"
LD(Akt1Snp2,Akt1Snp4)$"D'"
LD(Akt1Snp3,Akt1Snp4)$"D'"

LDGeoArea <- tapply(Akt1AllSnps,INDEX=hgdp$Geographic.area,LD)
LDGeoArea$"Pakistan"
LDGeoArea$"China"
LDGeoArea$"Israel"
LDGeoArea$"Southern Europe"
LDGeoArea$"Central Africa"
LDGeoArea$"Russia"
LDGeoArea$"(Other)"

#Problem 3.6
names(fms)
summary(fms$Race)
fmsam <-fms[fms$Race=="African Am", ]

NamesAkt1Snps <- names(fmsam)[substr(names(fmsam),1,4)=="akt1"]
NamesAkt1Snps

FMSgeno <- fmsam[,is.element(names(fmsam),NamesAkt1Snps)]
FMSgenoNum <- data.matrix(FMSgeno)
FMSgenoNum[is.na(FMSgenoNum)] <- 4
FMSgenoNum

DistFmsGeno <- as.matrix(dist(FMSgenoNum))
DistFmsGeno[1:5,1:5]
```

```
plot(cmdscale(DistFmsGeno),xlab="C1",ylab="C2")
abline(v=0,lty=2)
abline(h=4,lty=2)

###################################################################################
###################################################################################

### PUBH 7445 assignment 4

##wilcoxon sum rank test simulation
y1=rnorm(40, mean =1, sd=2)
y2=rnorm(40, mean=2,sd=2)
pval=wilcox.test(y1, y2)$p.value
pval

##two sample ttest with equal variance
y1=rnorm(40, mean =1, sd=2)
y2=rnorm(40, mean=2,sd=2)
pval=t.test(y1, y2, var.equal=TRUE)
pval

##wilcoxon simulation version 1
##When writing your own functions with multiple arguments in R, you need to hit enter/return in
between each arguments or R will not process the function!! (This is what the + sign in the R
console indicate
pval <- rep(NA,1000)
for(i in 1:1000){y1=rnorm(40, mean =1, sd=2)
y2=rnorm(40, mean=2,sd=2)
pval[i]=wilcox.test(y1, y2)$p.value}
sum(pval<.05)/1000

##wilcoxon simulation version 2 ##Ignore this
pval <- rep(NA,1000)
for(i in 1:1000){y1=rnorm(40,mean=1,sd=2)}
for(i in 1:1000){y2=rnorm(40, mean=2,sd=2)}
for(i in 1:1000){pval[i]=wilcox.test(y1, y2)$p.value}
sum(pval<.05)/1000

##ttest simulation version 1
##When writing your own functions with multiple arguments in R, you need to hit enter/return in
between each arguments or R will not process the function!! (This is what the + sign in the R
console indicate
pval <- rep(NA,1000)
for(i in 1:1000){y1=rnorm(40, mean =1, sd=2)
y2=rnorm(40, mean=2,sd=2)
pval[i]=t.test(y1, y2, var.equal=TRUE)$p.value}
```

```
sum(pval<.05)/1000

##ttest simulation version 2  ## Ignore this
pval <- rep(NA,1000)
for(i in 1:1000){y1=rnorm(40,mean=1,sd=2)}
for(i in 1:1000){y2=rnorm(40, mean=2,sd=2)}
for(i in 1:1000){pval[i]=t.test(y1, y2, var.equal=TRUE)$p.value}
sum(pval<.05)/1000

##Problem 4.2

#Loading the dataset
fms <- read.delim("http://www.biostat.umn.edu/~cavanr/FMS_data.txt",
header=T, sep="\t")
names (fms)

#Making binary indicator variables
Actn3Bin <-
data.frame(fms$actn3_r577x!="TT",fms$actn3_rs540874!="AA",fms$actn3_rs1815739!="TT",f
ms$actn3_1671064!="GG")
Actn3Bin

#building the unadjusted p values model
Mod <- summary(lm(fms$HDL_C~.,data=Actn3Bin))
Mod

pvector <- c(0.2078,0.5557,0.0765,0.5701)

bonferroni <- p.adjust(pvector, method="bonferroni")
bonferroni

BH <-p.adjust(pvector, method="BH")
BH

##Testing models for getting p-values in problem 4.2
Test <- lm(fms$NDRM.CH~.,data=Actn3Bin)$"p.value"
Test

##Problem 4.3
#Tukey Adjustments
TukeyHSD(aov(fms$CHOL~fms$resistin_c180g))

table(fms$Gender)
males <- subset(fms, fms$Gender=="Male")
dim(fms)
dim(males)
```

```
TukeyHSD(aov(males$CHOL~males$resistin_c180g))

females <- subset(fms, fms$Gender=="Female")
dim(fms)
dim(females)
TukeyHSD(aov(females$CHOL~females$resistin_c180g))

##Problem 4.6
#Let us begin by making a multivariable linear model that includes four indicators for the
presence of two variant alleles at each of the corresponding SNPs

#Loading the dataset
fms <- read.delim("http://www.biostat.umn.edu/~cavanr/FMS_data.txt",
header=T, sep="\t")
names (fms)

#Making binary indicator variables
Actn3Bin <-
data.frame(fms$actn3_r577x!="TT",fms$actn3_rs540874!="AA",fms$actn3_rs1815739!="TT",f
ms$actn3_1671064!="GG")

#Categorizing the outcome

table(fms$NDRM.CH)
quantile(fms$NDRM.CH, 0.75, na.rm="TRUE")
muscstr <- cut(fms$NDRM.CH, c(0,66.7,250))
table(muscstr)
factormuscstr <- factor(muscstr)
table (factormuscstr)
muscstr2 <- (as.numeric(muscstr))
table(muscstr2)

#building the unadjusted p values model
Mod <- summary(lm(muscstr2~.,data=Actn3Bin))
Mod

#The first step of the FSDR approach is to record the “observed” test statistics.

TestStatObs <- Mod$coefficients[-1,3]
TestStatObs
Tobs <- as.vector(sort(abs(TestStatObs)))
Tobs

#Before applying the resampling procedure, we need to subset the data that went into the
analysis above. Recall from the modeling output that n = 801
```

observationsweredeletedduetomissingnessinthegenotypeortraitvariables. We also need to record the ordering of our original test statistics. We do these two steps as follows

```
#the is.na function return values as "True" if any of the components

MissDat <- apply(is.na(Actn3Bin),1,any) | is.na(fms$NDRM.CH)
Actn3BinC <- Actn3Bin[!MissDat,]
Ord <- order(abs(TestStatObs))

#The second step of the FSDR requires resampling from the residuals and arriving at test
statistics under the null generating distribution using a for loop
#Making a data frame of size M x NSnps
M <- 1000
NSnps <- 4
Nobs <- sum(!MissDat)
TestStatResamp <- matrix(nrow=M, ncol=NSnps)
dim(TestStatResamp)

for (i in 1:M){Ynew <- sample(Mod$residuals, size=Nobs, replace=T)
ModResamp <- summary(lm(Ynew~., data=Actn3BinC))
TestStatResamp[i,] <- abs(ModResamp$coefficients[-1,3])[Ord]}

#We see that in each iteration of the for loop we (1) take a sample from the model residuals with
replacement, (2) re?t the model using this sample as the new outcome and (3) record the test
statistics corresponding to our ordered observed statistics, given by Tobs. The result is a matrix
of test statistics, called TestStatResamp, corresponding to the null distribution.
#The Final step is to compare our observed test statistics with the distribution of test statistics we
generated. To do this, we ?rst arrive at successive maxima by applying the cummax() function to
each row of our matrix of resampling-based statistics

Qmat <- t(apply(TestStatResamp, 1, cummax))

# Adjusted p-values

Padj <- apply(t(matrix(rep(Tobs,M), NSnps)) < Qmat, 2, mean)
Padj

#Problem 4.7

hgdpURL <- "http://www.biostat.umn.edu/~cavanr/HGDP_AKT1.txt"
hgdp <- read.delim(file=hgdpURL)
head(hgdp)
dim(hgdp)
names(hgdp)
summary(hgdp)

#Making binary indicator variables
```

```
table(hgdp$AKT1.C0756A) ##CC seems to be the major combination
table(hgdp$AKT1.C6024T) ##CC seems to be the major combination
table(hgdp$AKT1.G2347T) ##GG seems to be the major combination
table(hgdp$AKT1.G2375A) ##GG seems to be the major combination

Akt1Bin <-
data.frame(hgdp$AKT1.C0756A!="CC",hgdp$AKT1.C6024T!="CC",hgdp$AKT1.G2347T!="
GG",hgdp$AKT1.G2375A!="GG")
table(Akt1Bin)
summary (Akt1Bin)
dim(Akt1Bin)

##comparing these resulst to the 4.2 excercise
table(Actn3Bin)
summary (Actn3Bin)
dim(Actn3Bin)

#Need to make my outcome into a numeric variable for the model
Gender2 <- as.numeric(hgdp$Gender)
table(Gender2)

#Building the model
Mod2 <- summary(lm(Gender2~.,data=Akt1Bin))
Mod2

#Making the Binary Matrix excluding missing values
MissDat <- apply(is.na(Akt1Bin),1,any)
Akt1BinC <- Akt1Bin[!MissDat,]
table(Akt1BinC)
summary(Akt1BinC)
dim(Akt1BinC)

##comparing these resulst to the 4.2 excercise to see if there is any mistakes in my new variable
table(Actn3BinC)
summary(Actn3BinC)
dim(Actn3BinC)

Actn3Bin
Akt1Bin
Actn3BinC
Akt1BinC

##The akt1 binary matrice correlation structure did not work, so I tried to manually pull the
columns from the hgdp dataset, convert the SNPs to numeric values, then obtain the correlation
for the new data frame
```

```
test <- data.frame(hgdp[7:10])
test
names(test)
test$akt11 <- as.numeric(test$AKT1.C0756A)
test$akt12 <- as.numeric(test$AKT1.C6024T)
test$akt13 <- as.numeric(test$AKT1.G2347T)
test$akt14 <- as.numeric(test$AKT1.G2375A)
test2 <- data.frame(test[5,8])

##this did not work either
corAkt1test <- cor(test2)
corAkt1test

## This worked after exiting and restarting R
## Calculating correlation matrices
corActn3 <- cor(Actn3BinC)
corActn3

corAkt1 <- cor(Akt1BinC)
corAkt1

corAkt1v2 <- cor(Akt1Bin)

#Calculating the effective number of tests: If obs are the observed eigenvalues of the correlation
matrix of a
#set of m SNPs and V^ is the variance of these eigenvalues then the formula for calculating the
meff
#effective number of tests is given by the following expression:

eigenValAkt1 <- eigen(corAkt1)$values
mEff <- 1+(4-1)*(1-var(eigenValAkt1)/4)
mEff

## Custom problem 2

#if you want a binary variable with an expectation of .75, and 1000 observations, say:
runif(1000)<=.75
x1 <- runif(1000)<=.75
x1

#simulating 2000 markers with a probability of success of 0.5
X <- rbinom(2000,1,0.5)
table (X)
X

#Simulating the outcome variable
```

```
meanX <- 10*X[1] + 9*X[2] + 8*X[3] + 7*X[4] + 6*X[5] + 5*X[6] + 4*X[7] + 3*X[8] +
2*X[9] + 1*X[10]
Y <- rnorm(100, mean=meanX, sd = 1)
hist (Y)

##Custom Problem 3

# Bootstrap example

set.seed(1)
y1 <- rnorm(100,0,1)
y2 <- rnorm(50,3,0.5)
y3 <- c(y1,y2)

y3
hist(y3)

mean(y3)-qt(.975, df=149)*sqrt(var(y3)/150)
mean(y3)+qt(.975, df=149)*sqrt(var(y3)/150)
sim <- rep(NA, 1000)
for(i in 1:1000){
  y4 <- sample(y3, replace=TRUE)
  sim[i] <- mean(y4)
}
quantile(sim, c(.025, .975))

sim <- rep(NA, 10000)
for(i in 1:10000){
  y2 <- sample(y1, replace=TRUE)
  sim[i] <- mean(y2)
}
quantile(sim, c(.025, .975))

###################################################################################
###################################################################################

### PUBH 7445 assignment 5

#Problem 5.1

#Loading the fms dataset
fms <- read.delim("http://www.biostat.umn.edu/~cavanr/FMS_data.txt",
header=T, sep="\t")
attach(fms)

#Finding the names of the Resistin SNPs
```

```
names (fms)

#Making binary indicator variables
table(resistin_c30t) ##CC seems to be the major combination
table(resistin_c398t) ##CC seems to be the major combination
table(resistin_g540a) ##GG seems to be the major combination
table(resistin_c980g) ##CG seems to be the major combination
table(resistin_a537c) ##AA seems to be the major combination

##We begin by calling the haplo.stats package and creating a genotype matrix. The genotype
matrix has a pair of adjacent columns for each SNP such that each column corresponds to one of
the two observed alleles at the corresponding site
## The order of the columns is assumed to correspond to the order of the sites on the
chromosome. Recall that we start with four SNPs within the actn3 gene and so the following
code is required

install.packages("haplo.stats")
library(haplo.stats)

## This step should not be needed
resistin <- c(resistin_c30t, resistin_c398t, resistin_g540a, resistin_c980g, resistin_a537c)
resistin

##Resume here
Geno <- cbind(substr(resistin_c30t,1,1), substr(resistin_c30t,2,2),
substr(resistin_c398t,1,1), substr(resistin_c398t,2,2),
substr(resistin_g540a,1,1), substr(resistin_g540a,2,2),
substr(resistin_c980g,1,1), substr(resistin_c980g,2,2),
substr(resistin_a537c,1,1), substr(resistin_a537c,2,2))
SNPnames <- c("resistin_c30t", "resistin_c398t", "resistin_g540a", "resistin_c980g",
"resistin_a537c")

#We then subset African Americans and Caucasians and apply the haplo.em() function to each
group
#This function applies a modi?ed version of the EM approach described above, in which sets of
loci are progressively included and in turn haplotype pairs with small estimated probabilities are
excluded.
#The haplo.em.control() function is used within the haplo.em() function call to specify the
minimum posterior probability of a haplotype pair

#In Caucasians

Geno.C <- Geno[Race=="Caucasian" & !is.na(Race),]
HaploEM <- haplo.em(Geno.C, locus.label=SNPnames,
control=haplo.em.control(min.posterior=1e-4))
HaploEM
```

```
#In African Americans
Geno.AA <- Geno[Race=="African Am" & !is.na(Race),]
HaploEM2 <- haplo.em(Geno.AA, locus.label=SNPnames,
control=haplo.em.control(min.posterior=1e-4))
HaploEM2

#Testing hypotheses about haplotype frequencies within the EM framework
#In this example, we test the null hypothesis that the frequency of haplotype  is the same in
African Americans and Caucasians.
#We do this by constructing a con?dence interval around the di?erence in the two frequencies
and checking whether it covers 0.
#First we calculate the di?erence in the estimated frequencies.
#In order to calculate the standard error of each frequency, we use the function HapFreqSE()
#We then combine the results to get an estimate of the standard error of this difference

#The most common haplotype in african americans is 1, and the most common haplotype in
caucasians is 6
#Have to use the HapFreqSE from the lecture notes

# New functions needed

###############################################################################
# Description: This function creates a design matrix with i,j
#              element equal to the conditional expectation
#              of the number of copies of haplotype j for
#              individual i based on the output from haplo.em()
# Input:       HaploEM (object resulting from haplo.em())
# Output:      XmatHap
###############################################################################

HapDesign <- function(HaploEM){
        Nobs <- length(unique(HaploEM$indx.subj)) # number of observations
        Nhap <- length(HaploEM$hap.prob) # number of haplotypes
        XmatHap <- matrix(data=0,nrow=Nobs,ncol=Nhap)
        for (i in 1:Nobs){
                IDSeq <- seq(1:sum(HaploEM$nreps))[HaploEM$indx.subj==i]
                for (j in 1:length(IDSeq)){
                        XmatHap[i,HaploEM$hap1code[IDSeq][j]] <-
                                XmatHap[i,HaploEM$hap1code[IDSeq][j]] +
                                HaploEM$post[IDSeq][j]
                        XmatHap[i,HaploEM$hap2code[IDSeq][j]] <-
                                XmatHap[i,HaploEM$hap2code[IDSeq][j]] +
                                HaploEM$post[IDSeq][j]
                        }
                }
        return(XmatHap)
}

############################################################################
# Description: This function creates a vector with jth element
#              equal to the standard error of haplotype j
#              based on the output from haplo.em()
# Input:       HaploEM (object resulting from haplo.em())
# Output:      HapSE
############################################################################

HapFreqSE <- function(HaploEM){
        HapMat <- HapDesign(HaploEM)
        Nobs <- length(unique(HaploEM$indx.subj)) # number of observations
        Nhap <- length(HaploEM$hap.prob) # number of haplotypes
        S.Full<-matrix(data=0, nrow=Nobs, ncol=Nhap-1)
        for(i in 1:Nobs){
                for(k in 1:(Nhap-1)){
                S.Full[i,k]<-HapMat[i,k]/HaploEM$hap.prob[k]-
                        HapMat[i,Nhap]/HaploEM$hap.prob[Nhap]
                }
        }
        Score<-t(S.Full)%*%S.Full
        invScore<-solve(Score)
        HapSE<-c(sqrt(diag(invScore)),
                sqrt(t(rep(1,Nhap-1))%*%invScore%*%rep(1,Nhap-1)))
        return(HapSE)
        }

###Testing for differences in resistin haplotype frequency between causasians and African
americans

FreqDiff <- HaploEM2$hap.prob[1] - HaploEM$hap.prob[6]
s1 <- HapFreqSE(HaploEM)[6]
s2 <- HapFreqSE(HaploEM2)[1]
SE <- sqrt(s1^2 + s2^2)
CI <- c(FreqDiff - 1.96*SE, FreqDiff + 1.96*SE)
CI

#Problem 5.2

library(haplo.stats)
hgdpURL <- "http://www.biostat.umn.edu/~cavanr/HGDP_AKT1.txt"
hgdp <- read.delim(file=hgdpURL)
attach(hgdp)
names(hgdp)
```

```
attach(hgdp)
table(Population)

attach(hgdp)
table (Geographic.origin)

geno=cbind(substr(AKT1.C6024T,1,1), substr(AKT1.C6024T,2,2),
  substr(AKT1.C0756A,1,1), substr(AKT1.C0756A,2,2),
  substr(AKT1.G2375A,1,1), substr(AKT1.G2375A,2,2),
  substr(AKT1.G2347T,1,1), substr(AKT1.G2347T,2,2))
SNPnames <- c("C6024T","C0756A","G2375A", "G2347T")

## AKT1 haplotype frequencies within groups de?ned by the variable Population.

i=1
id=levels(Population)[i]
dat=geno[Population==id,]
h1=haplo.em(dat, locus.label=SNPnames,
  control=haplo.em.control(min.posterior=1.0e-4))

h1
names(h1)
h1$hap.prob
h1$haplotype
maxNoHap=40
hapProb=matrix(NA, 52, maxNoHap)
hapMat=vector("list",52)

for(i in 1:52){
  id=levels(Population)[i]
  dat=geno[Population==id,]
  h1=haplo.em(dat, locus.label=SNPnames,
    control=haplo.em.control(min.posterior=1.0e-4))
  nh=dim(h1$haplotype)[1]
  hapProb[i,1:nh]=h1$hap.prob
  hapMat[[i]]=h1$haplotype
}

dat=numeric(0)
for(i in 1:52){
  nh=dim(hapMat[[i]])[1]
  for(j in 1:nh) dat=c(dat,hapMat[[i]][j,])
}

length(dat)/4
```

```
ddat1=matrix(dat, byrow=T, ncol=4)
dat2=rep(NA, 155)

for(i in 1:155) dat2[i]=paste(ddat1[i,1],ddat1[i,2],ddat1[i,3],ddat1[i,4],sep="")
table(dat2)
hapProb[1:5,1:10]
t(hapProb[1:5,1:10])
c(t(hapProb[1:5,1:10]))
hapProbPop=c(t(hapProb))[!is.na(c(t(hapProb)))]
length(hapProbPop)
i=1

names(table(dat2))[i]
hid=names(table(dat2))[i]
hapProbPop[dat2==hid]
par(mfrow=c(3,3))
for(i in 1:9){
  hid=names(table(dat2))[i]
  hist(hapProbPop[dat2==hid])
}

par(mar=c(2,2,2,1))
for(i in 1:9){
  hid=names(table(dat2))[i]
  hist(hapProbPop[dat2==hid], xlab="", main=paste("hap:", hid))
}

hapProbpop

## AKT1 haplotype frequencies within groups defined by the variable Geographic.Origin.

i=1
id=levels(Geographic.origin)[i]
dat=geno[Geographic.origin==id,]
h1=haplo.em(dat, locus.label=SNPnames,
  control=haplo.em.control(min.posterior=1.0e-4))

h1
names(h1)
h1$hap.prob
h1$haplotype
maxNoHap=40
hapProb=matrix(NA, 27, maxNoHap)
hapMat=vector("list",27)

for(i in 1:27){
  id=levels(Geographic.origin)[i]
  dat=geno[Geographic.origin==id,]
  h1=haplo.em(dat, locus.label=SNPnames,
    control=haplo.em.control(min.posterior=1.0e-4))
  nh=dim(h1$haplotype)[1]
  hapProb[i,1:nh]=h1$hap.prob
  hapMat[[i]]=h1$haplotype
}

dat=numeric(0)
for(i in 1:27){
  nh=dim(hapMat[[i]])[1]
  for(j in 1:nh) dat=c(dat,hapMat[[i]][j,])
}

length(dat)/4
ddat1=matrix(dat, byrow=T, ncol=4)
dat2=rep(NA, 155)

for(i in 1:155) dat2[i]=paste(ddat1[i,1],ddat1[i,2],ddat1[i,3],ddat1[i,4],sep="")
table(dat2)
hapProb[1:5,1:10]
t(hapProb[1:5,1:10])
c(t(hapProb[1:5,1:10]))
hapProbPop=c(t(hapProb))[!is.na(c(t(hapProb)))]
length(hapProbPop)
i=1

names(table(dat2))[i]
hid=names(table(dat2))[i]
hapProbPop[dat2==hid]
par(mfrow=c(3,3))
for(i in 1:9){
  hid=names(table(dat2))[i]
  hist(hapProbPop[dat2==hid])
}

par(mar=c(2,2,2,1))
for(i in 1:9){
  hid=names(table(dat2))[i]
  hist(hapProbPop[dat2==hid], xlab="", main=paste("hap:", hid))
}

#Problem 5.3
#Determining association between Gender and Geographic origin
```

```
hgdpURL <- "http://www.biostat.umn.edu/~cavanr/HGDP_AKT1.txt"
hgdp <- read.delim(file=hgdpURL)
attach(hgdp)
names(hgdp)

#Need to make my outcome into a numeric variable for the model
Gender2 <- as.numeric(hgdp$Gender)
table(Gender2)

#Building the model
Mod2 <- summary(lm(Gender2~Geographic.origin))
Mod2

#Problem 5.4
#Haplotype trend regression begins by creating a design matrix with elements equal to the
conditional expectation for the number of copies of each haplotype
#This is achieved using the function HapDesign()
#F-test that compares this model to the reduced model with just an intercept is straightforward
#Applying this to test an association between the resistin haplotypes and change in non-dominant
arm muscle strength within African Americans using the FAMuSS data

attach(fms)
HapMat <- HapDesign(HaploEM2)
Trait <- NDRM.CH[Race=="African Am" & !is.na(Race)]
mod1 <- (lm(Trait~HapMat))
mod2 <- (lm(Trait~1))
anova(mod2,mod1)

#Problem 5.5

#In this example, we implement the fully likelihood-based approach to estimating haplotype
frequencies and haplotype–trait association
#Again we consider data from the FAMuSS study and focus on association between haplotypes
within the actn3 gene and the percentage change in the non-dominant arm muscle strength.
#Using the genotype data matrix Geno.AA for african americans that we generated in Example
5.1, we can apply the following code, where again haplo.glm() is a function within the
haplo.stats package

attach(fms)
Geno.AA <- setupGeno(Geno.AA)
Trait <- NDRM.CH[Race=="African Am" & !is.na(Race)]
Dat <- data.frame(Geno.AA=Geno.AA, Trait=Trait)

#Additive Genetic Model
library(haplo.stats)
test1 <- haplo.glm(Trait~Geno.AA,data=Dat,
allele.lev=attributes(Geno.AA)$unique.alleles)
summary(test1)

##By default, the base haplotype is set equal to the most prevalent one in our sample, in this case CGCA.
##Another important parameter that we can control is an indicator for the type of genetic model.
##By default, the haplo.glm() function assumes an additive genetic model, so that the e?ect of having two copies of a haplotype is twice the e?ect of having a single copy of the haplotype
##We can easily specify an alternative model structure by specifying the haplo.effect parameter within haplo.glm.control.
##For example, under a dominant genetic model in which one or more copies of a haplotype cause the effect, we have

library(haplo.stats)
test2 <-haplo.glm(Trait~Geno.AA,data=Dat,
allele.lev=attributes(Geno.AA)$unique.alleles,
control=haplo.glm.control(haplo.effect="dominant"))
summary(test2)

##Custom Problem
##The following residual analyses will be done using the additive model
##Tried adding the , na.action = na.exclude option to the model as suggested online, did not work
## Did not need to plot the residuals against Xvalues, used a histogram of residual values insteas

library(haplo.stats)
haplotest <- haplo.glm(Trait~Geno.AA,data=Dat,
allele.lev=attributes(Geno.AA)$unique.alleles)
haplotest

geno.res = resid(haplotest)
geno.res
hist(geno.res)

dim(Geno.AA)
attach(fms)
length(Gender)
LogAA <- log(Geno.AA)
LogAA
Trait

##Skip this code
plot(Dat$Geno.AA, geno.res,
 ylab="Residuals", xlab="Geno",
 main="Association between Resistin and Non Dominant Arm Muscle Strength in African Americans")
```

```
##Resume Here
library(haplo.stats)
haplotest2 <- haplo.glm(log(Trait)~Geno.AA,data=Dat,
allele.lev=attributes(Geno.AA)$unique.alleles)
haplotest2

geno.res2 = resid(haplotest2)
geno.res2
hist(geno.res2)

##Interactions with Gender

## testing
Gender2 <- as.numeric(Gender)
Geno.AA <- setupGeno(Geno.AA)
Gendermk1 <- Gender2[Race=="African Am" & !is.na(Race)]
Dat <- data.frame(Geno.AA=Geno.AA, Gender=Gendermk1)
Dat

##resume coding
attach(fms)
table(Gender)
Gender2 <- as.numeric(Gender)
table(Gender2)
Geno.AA <- setupGeno(Geno.AA)
Trait <- NDRM.CH[Race=="African Am" & !is.na(Race)]
Gendermk1 <- Gender2[Race=="African Am" & !is.na(Race)]
Dat <- data.frame(Geno.AA=Geno.AA, Trait=Trait, Gender=Gendermk1)
Dat

##subsetting the Dat dataframe for males and females
DatF <- subset(Dat, Dat$Gender=="1")
DatM <- subset(Dat, Dat$Gender=="2")

##Does Gender have an impact on NDRM in african americans

impact <- glm(Dat$Trait~Dat$Gender)
summary(impact)

library(haplo.stats)
haplotest3 <- haplo.glm(Trait~Geno.AA + Gender,data=Dat,
allele.lev=attributes(Geno.AA)$unique.alleles)
summary(haplotest3)
```

```
################################################################################
################################################################################

### PUBH 7445 assignment 6

## We are interested in identifying polymorphisms within the protease region of the viral
genome that are associated with the trait

vircoURL <- "http://www.biostat.umn.edu/~cavanr/Virco_data.csv"
virco <- read.csv(file=vircoURL)
dim(virco)
names(virco)

# In this case, we begin by de?ning a dataframe, entitled VircoGeno, that contains binary
indicators for the presence of a mutation at each of the 99 sites within the protease region:

attach(virco)
VircoGeno <- data.frame(virco[,substr(names(virco),1,1)=="P"]!="-")

#We then de?ne our trait and construct a classi?cation tree as follows. Note that we de?ne our
trait as a factor variable since we are interested in creating a classi?cation tree. We additionally
specify method="class", though this is the default setting when the trait is a factor:

Trait <- as.factor(APV.Fold > IDV.Fold)
library(rpart)
ClassTree <- rpart(Trait~., method="class", data=VircoGeno)
ClassTree

#A visual representation of this output is generated with the plot() and text() functions

plot(ClassTree)
text(ClassTree)

#Pruning this tree involves determining a cost-complexity parameter value. We do this by
considering the output of the printcp() function. A visual representation is given by applying the
plotcp() function

plotcp(ClassTree)
printcp(ClassTree)

#Based on the output and ?gure, we see that the cross-validated error begins to increase between
three and 4 splits corresponding to 11 to 12 nodes

#This is achieved using the prune() function and specifying, in this example, cp=0.01 as follows:

pruneTree <- prune(ClassTree,cp=.01)
```

```
pruneTree

# Problem 6.5

#we consider SNPs within the actn3 and resistin genes and their association with the percentage
change in non-dominant arm muscle strength using the FAMuSS data.

fms <- read.delim("http://www.biostat.umn.edu/~cavanr/FMS_data.txt",
header=T, sep="\t")
dim(fms)
names(fms)

attach(fms)
Trait <- NDRM.CH
library(rpart)

## actn3 and resistin combined

RegTree <-
rpart(Trait~actn3_r577x+actn3_rs540874+actn3_rs1815739+actn3_1671064+resistin_c30t+resis
tin_c398t+resistin_g540a+resistin_c980g+resistin_c180g+resistin_a537c , method="anova")
RegTree

#resistin alone
RegTree <-
rpart(Trait~resistin_c30t+resistin_c398t+resistin_g540a+resistin_c980g+resistin_c180g+resistin
_a537c, method="anova")
RegTree

#resistin + race
RegTree <-
rpart(Trait~resistin_c30t+resistin_c398t+resistin_g540a+resistin_c980g+resistin_c180g+resistin
_a537c+Race, method="anova")
RegTree

## actn3 alone
RegTree <- rpart(Trait~actn3_r577x+actn3_rs540874+actn3_rs1815739+actn3_1671064,
method="anova")
RegTree

## actn3 + race
RegTree <- rpart(Trait~actn3_r577x+actn3_rs540874+actn3_rs1815739+actn3_1671064+Race,
method="anova")
RegTree

# Problem 7.1
```

```
vircoURL <- "http://www.biostat.umn.edu/~cavanr/Virco_data.csv"
virco <- read.csv(file=vircoURL)
dim(virco)
names(virco)

install.packages("randomForest")
library(randomForest)

#We de?ne the trait and a matrix of potential predictor variables, given by Trait and VircoGeno,
respectively, using the same code as in Example 6.3. The randomForest() function does not
permit missing data in the response variable, and so we next subset our data to include only those
individuals with complete information on the trait:

attach(virco)
Trait <- SQV.Fold
VircoGeno <- data.frame(virco[,substr(names(virco),1,1)=="P"]!="-")
Trait.c <- Trait[!is.na(Trait)]
VircoGeno.c <- VircoGeno[!is.na(Trait),]

#Finally, we ?t a random forest and plot the ordered variable importance measures, given by
mean decrease in accuracy (%IncMSE) and mean decrease in nodeimpurity(%IncMSE),usingthe
randomForest() and varImpPlot() functions.

RegRF <- randomForest(VircoGeno.c, Trait.c, importance=TRUE)
RegRF
varImpPlot(RegRF)

## Problem 7.2

##Application of logic regression

vircoURL <- "http://www.biostat.umn.edu/~cavanr/Virco_data.csv"
virco <- read.csv(file=vircoURL)
dim(virco)
names(virco)

#First, de?ne our trait variable and matrix of genotypes

attach(virco)
Trait <- SQV.Fold
VircoGeno <- data.frame(virco[,substr(names(virco),1,1)=="P"]!="-")
Trait.c <- Trait[!is.na(Trait)]
VircoGeno.c <- VircoGeno[!is.na(Trait),]

# Install and upload the LogicReg package in R
```

```
install.packages("LogicReg")
library(LogicReg)

#apply logic regression using the logreg() function. Here we ?rst specify select=1 to ?t a single tree model. A plot of the resulting decision tree is obtained using the plot() function with a logic regression object as input.

VircoLogicReg <- logreg(resp=Trait.c, bin=VircoGeno.c, select=1)
plot(VircoLogicReg)
VircoLogicReg

## Problem 7.4

##An application of MARS

fms <- read.delim("http://www.biostat.umn.edu/~cavanr/FMS_data.txt",
header=T, sep="\t")
dim(fms)
names(fms)

#Comparing the format of the FMS and Virco datasets
attach(virco)
virco$P45
table(virco$P45)
attach(fms)

fms$actn3_r577x
#First, de?ne our trait variable and matrix of genotypes: Need to subset the dataset to limit it only to the variables we need in the analysis (actn3, resistin and the outcome)

FMSMARS <-
data.frame(NDRM.CH,actn3_r577x,actn3_rs540874,actn3_rs1815739,actn3_1671064,resistin_c30t,resistin_c398t,resistin_g540a,resistin_c980g,resistin_c180g,resistin_a537c)
attach(FMSMARS)

###Tests

Trait <- NDRM.CH
Trait.c <- Trait[!is.na(Trait)]
FMSMARS.c <- FMSMARS[!is.na(Trait),]
FMSMARS.c <- FMSMARS[complete.cases(FMSMARS),]
table(Trait.c)

#We then install and upload the earth package:
install.packages("earth")
```

```
library(earth)

FMSMARS2 <- earth(Trait.c~., data=FMSMARS.c, degree=2)

###Resume code
## Use tor complete cases option to remove any rows with missing data: the is.na option gave me
errors on the lenght of NDRM.CH. Maybe it was colinear with teh outcome so it would not run
in the model?
FMS2 <- fms[,c('NDRM.CH', 'resistin_c30t', 'resistin_c398t', 'resistin_g540a',
'resistin_c980g','resistin_c180g', 'resistin_a537c', 'actn3_r577x', 'actn3_rs540874',
'actn3_rs1815739' , 'actn3_1671064')]
FMS.c <- FMS2[complete.cases(FMS2),]
FMSMARSTEST <- earth(NDRM.CH~., data=FMS.c, degree=2)
FMSMARSTEST

##Converting the SNPS to binary variables
table(fms$resistin_c30t) #CC
table(fms$resistin_c398t) #CC
table(fms$resistin_g540a) #GG
table(fms$resistin_c980g) #CC
table(fms$resistin_c180g) #CC
table(fms$resistin_a537c) #AA

##When making the dataframe, you can't have spaces between ! and =
MARSBIN <- data.frame(fms$NDRM.CH, fms$resistin_a537c!="AA",
fms$resistin_c180g!="CC", fms$resistin_c980g!="CC", fms$resistin_g540a!="GG",
fms$resistin_c30t!="CC",
fms$resistin_c398t!="CC",fms$actn3_r577x!="TT",fms$actn3_rs540874!="AA",fms$actn3_rs1
815739!="TT",fms$actn3_1671064!="GG")
MARSBIN.c <- MARSBIN[complete.cases(MARSBIN),]
MARSBIN.c
FMSMARSBIN <- earth(fms.NDRM.CH~., data=MARSBIN.c, degree=2)
FMSMARSBIN

#########################################################################################
#########################################################################################

### PUBH 7445 assignment 7

#we consider SNPs within the actn3 and resistin genes and their association with the percentage
change in non-dominant arm muscle strength using the FAMuSS data.

fms <- read.delim("http://www.biostat.umn.edu/~cavanr/FMS_data.txt",
header=T, sep="\t")
dim(fms)
names(fms)
```

```r
attach(fms)

#Making a Dataset with just the actn3 and resistin genes

#First, define our trait variable and matrix of genotypes: Need to subset the dataset to limit it only to the variables we need in the analysis (actn3, resistin and the outcome)
HW7 <- data.frame(NDRM.CH,actn3_r577x,actn3_rs540874,actn3_rs1815739,actn3_1671064,resistin_c30t,resistin_c398t,resistin_g540a,resistin_c980g,resistin_c180g,resistin_a537c)
attach(HW7)

#Better way for making the datasubset using complete cases option to remove any rows with missing data: the is.na option gave me errors on the lenght of NDRM.CH. Maybe it was colinear with teh outcome so it would not run in the model?
HW7 <- fms[,c('NDRM.CH', 'resistin_c30t', 'resistin_c398t', 'resistin_g540a', 'resistin_c980g','resistin_c180g', 'resistin_a537c', 'actn3_r577x', 'actn3_rs540874', 'actn3_rs1815739' , 'actn3_1671064')]
HW7.c <- HW7[complete.cases(HW7),]
attach(HW7.c)

#Checking SNP levels in dataset
dim(HW7.c)
q1=rep(NA,24)
for(i in 1:24) q1[i]=nlevels(HW7.c[,i])
which(q1!=3)

##Third way with binary variables for the SNPS (When making the dataframe, you can't have spaces between ! and =)
HW7BIN <- data.frame(fms$NDRM.CH, fms$resistin_a537c!="AA", fms$resistin_c180g!="CC", fms$resistin_c980g!="CC", fms$resistin_g540a!="GG", fms$resistin_c30t!="CC", fms$resistin_c398t!="CC",fms$actn3_r577x!="TT",fms$actn3_rs540874!="AA",fms$actn3_rs1815739!="TT",fms$actn3_1671064!="GG")
HW7BIN.c <- HW7BIN[complete.cases(HW7BIN),]
HW7BIN.c

attach(HW7BIN.c)
fms.NDRM.CH

#Making the outcome variable
TraitC <- fms.NDRM.CH
TraitCD <- TraitC > median(TraitC)
TraitCD

#Checking SNP levels in dataset
dim(HW7BIN.c)
```

```
q1=rep(NA,11)
for(i in 1:11) q1[i]=nlevels(HW7BIN.c[,i])
which(q1!=3)

names(HW7BIN.c)
HW7BIN.c
dim(HW7BIN.c)
```

Checking the values for SNPs

```
table(HW7BIN.c[,1])
table(HW7BIN.c[,2])
table(HW7BIN.c[,3])
table(HW7BIN.c[,4])
table(HW7BIN.c[,5])
table(HW7BIN.c[,6])
table(HW7BIN.c[,7])
table(HW7BIN.c[,8])
table(HW7BIN.c[,9])
table(HW7BIN.c[,10])
table(HW7BIN.c[,11])

#Install ROC

install.packages("pROC")
library(pROC)

#Install Random Forest
install.packages("randomForest")
library(randomForest)

## Install support vector machines
install.packages("kernlab")
library("kernlab")

#Set 100 observations aside as test cases. Make the learning sample and the test sample

set.seed(102)
testSmp <- sample(1:590,100)
testSmp
HW7BIN.ts <- HW7BIN.c[testSmp,]
Trait.ts <- TraitCD[testSmp]
HW7BIN.ls <- HW7BIN.c[-testSmp,]
Trait.ls <- TraitCD[-testSmp]

#Check the distribution of the outcome in both subset of datasets
summary(Trait.ts)
summary(Trait.ls)

##Random Forest Model
clsRF <- randomForest(HW7BIN.ls, factor(Trait.ls))
prdRF <- predict(clsRF,HW7BIN.ts)
t1=table(prdRF,Trait.ts)
t1
t1[2,2]/sum(t1[,2])
t1[1,1]/sum(t1[,1])

prdRFp <- predict(clsRF,HW7BIN.ts,type="prob")
rfROC <- roc(Trait.ts,prdRFp[,1],plot=TRUE)

# Average of ROC Curves

for(i in 1:100){
  clsRF <- randomForest(HW7BIN.ls, factor(Trait.ls))
  prdRFp <- predict(clsRF,HW7BIN.ts,type="prob")
  rfROC <- lines.roc(Trait.ts,prdRFp[,1])
}

###Support Vector Machine Model
clsSVM <- ksvm(factor(Trait.ls)~., data=data.frame(HW7BIN.ls), kernel="rbfdot",
kpar="automatic", C=60, cross=3, prob.model=TRUE)
prdSVM <- predict(clsSVM,HW7BIN.ts)
t3=table(prdSVM,Trait.ts)
t3[2,2]/sum(t3[,2])
t3[1,1]/sum(t3[,1])

##Changing the C factor in SVMs

clsSVM <- ksvm(factor(Trait.ls)~., data=data.frame(HW7BIN.ls), kernel="rbfdot",
kpar="automatic", C=10, cross=3, prob.model=TRUE)
t3=table(prdSVM,Trait.ts)
t3
t3[2,2]/sum(t3[,2])
t3[1,1]/sum(t3[,1])

clsSVM <- ksvm(factor(Trait.ls)~., data=data.frame(HW7BIN.ls), kernel="rbfdot",
kpar="automatic", C=.1, cross=3, prob.model=TRUE)
t3=table(prdSVM,Trait.ts)
t3[2,2]/sum(t3[,2])
t3[1,1]/sum(t3[,1])

prdSVMp <- predict(clsSVM,HW7BIN.ts,type="probabilities")
svmROC <- roc(Trait.ts,prdSVMp[,1],plot=TRUE)
```

```
#compare the methods in terms of confidence intervals for the AUC
ci(svmROC)
ci(rfROC)

#test for differences
roc.test(rfROC,svmROC,paired=TRUE)

######################################################################################
######################################################################################

### PUBH 7445 assignment 8

#Use the CLLbatch dataset for this homework. Set up this affyBatch object as we did in class
(i.e. match disease status and create an annotated data frame to hold the phenotypic data and
metadata).

source("http://www.bioconductor.org/biocLite.R")
biocLite("affy")
library(affy)

biocLite("CLL")
library(CLL)

#Matching The disease statys and creating and annotated data frame.
data("CLLbatch")
sampleNames(CLLbatch)

data(disease)
rownames(disease)=disease$SampleID

#This removes the CEL extension to the CLL batch rows
sampleNames(CLLbatch)=sub("\\.CEL$","",sampleNames(CLLbatch))

#This matches the row names in the CLL batch and disease dataset
mt=match(rownames(disease),sampleNames(CLLbatch))

#This is our combined dataset with the metadata included
vmd=data.frame(labelDescription=c("Sample ID", "Disease status: progressive or stable"))
phenoData(CLLbatch)=new("AnnotatedDataFrame",data=disease[mt,], varMetadata=vmd)
CLLbatch=CLLbatch[,!is.na(CLLbatch$Disease)]

#Delete the microarrays labeled CLL1 and CL11
badArray1=match("CLL1", sampleNames(CLLbatch))
badArray2=match("CLL11", sampleNames(CLLbatch))
```

```
CLLB1=CLLbatch[,-badArray1]
CLLB=CLLB1[,-badArray2]

#Use the RMA algorithm to get probe level summaries of gene expression.

CLLrma=rma(CLLB)
CLLrma

#Loading packages for the filtering and ttests
biocLite("genefilter")
library("genefilter")

biocLite ('hgu95av2.db')
library ('hgu95av2.db')

biocLite('mogene21sttranscriptcluster.db')
library ('mogene21sttranscriptcluster.db')

biocLite("limma")
library("limma")

##########################################################################################
##########################################################################################

#Excluding probesets which are Affymetrix controls
annotation(CLLrma) <- "mogene21sttranscriptcluster.db"

CLLAFFXNEG = nsFilter(CLLrma, remove.dupEntrez=FALSE, var.cutof =0.5,
feature.exclude="^AFFX")$eset

#ttest
CLLAFFXNEGTT = rowttests(CLLAFFXNEG, "Disease")
sum(CLLAFFXNEGTT$p.value<0.05 & abs(CLLAFFXNEGTT$dm)>0.5)

#Moderated ttest

design = model.matrix(~CLLAFFXNEG$Disease)
CLLAFFXNEGLM = lmFit(CLLAFFXNEG, design)
CLLAFFXNEGEB = eBayes(CLLAFFXNEGLM)

#Comparing the 2 methods
table(p.adjust(CLLAFFXNEGTT[,3],method="BH")<0.5,
p.adjust(CLLAFFXNEGEB$p.value[,2],method="BH")<0.5)

##########################################################################################
##########################################################################################
```

```
#Cutoff at 20, 40, 60, 80

CLLNEG2 = nsFilter(CLLrma, remove.dupEntrez=FALSE, var.cutoff=0.2,
require.GOBP=TRUE, feature.exclude="^AFFX")$eset

#ttest
CLLNEG2TT = rowttests(CLLNEG2, "Disease")
sum(CLLNEG2TT$p.value<0.05 & abs(CLLNEG2TT$dm)>0.5)

lod = -log10(CLLtt$p.value)
plot(CLLtt$dm, lod, pch=".", xlab="log-ratio", ylab=expression(-log[10]~p))
abline(h=2)

#Moderated ttest
design = model.matrix(~CLLNEG2$Disease)
CLLNEG2LM = lmFit(CLLNEG2, design)
CLLNEG2EB = eBayes(CLLNEG2LM)

#Comparing the 2 methods
table(p.adjust(CLLNEG2TT[,3],method="BH")<0.5,
p.adjust(CLLNEG2EB$p.value[,2],method="BH")<0.5)

CLLNEG4 = nsFilter(CLLrma, remove.dupEntrez=FALSE, var.cutoff=0.4,
require.GOBP=TRUE, feature.exclude="^AFFX")$eset

#ttest
CLLNEG4TT = rowttests(CLLNEG4, "Disease")
sum(CLLNEG4TT$p.value<0.05 & abs(CLLNEG4TT$dm)>0.5)

#Moderated ttest
design = model.matrix(~CLLNEG4$Disease)
CLLNEG4LM = lmFit(CLLNEG4, design)
CLLNEG4EB = eBayes(CLLNEG4LM)

#Comparing the 2 methods
table(p.adjust(CLLNEG4TT[,3],method="BH")<0.5,
p.adjust(CLLNEG4EB$p.value[,2],method="BH")<0.5)

CLLNEG6 = nsFilter(CLLrma, remove.dupEntrez=FALSE, var.cutoff=0.6,
require.GOBP=TRUE, feature.exclude="^AFFX")$eset

#ttest
CLLNEG6TT = rowttests(CLLNEG6, "Disease")
sum(CLLNEG6TT$p.value<0.05 & abs(CLLNEG6TT$dm)>0.5)
```

```
#Moderated ttest
design = model.matrix(~CLLNEG6$Disease)
CLLNEG6LM = lmFit(CLLNEG6, design)
CLLNEG6EB = eBayes(CLLNEG6LM)

#Comparing the 2 methods
table(p.adjust(CLLNEG6TT[,3],method="BH")<0.5,
p.adjust(CLLNEG6EB$p.value[,2],method="BH")<0.5)

CLLNEG8 = nsFilter(CLLrma, remove.dupEntrez=FALSE, var.cutof =0.8,
require.GOBP=TRUE, feature.exclude="^AFFX")$eset

#ttest
CLLNEG8TT = rowttests(CLLNEG8, "Disease")
sum(CLLNEG8TT$p.value<0.05 & abs(CLLNEG8TT$dm)>0.5)

#Moderated ttest
design = model.matrix(~CLLNEG8$Disease)
CLLNEG8LM = lmFit(CLLNEG8, design)
CLLNEG8EB = eBayes(CLLNEG8LM)

#Comparing the 2 methods
table(p.adjust(CLLNEG8TT[,3],method="BH")<0.5,
p.adjust(CLLNEG8EB$p.value[,2],method="BH")<0.5)

#Making Volcano Plots
#abline is set at 1.301 because log(0.05) = 1.301 for the delimeter of p values

lod = -log10(CLLNEG2TT$p.value)
plot(CLLNEG2TT$dm, -log10(CLLNEG2TT$p.value), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
o1 = abs(CLLNEG2TT$dm)>1.5
points(CLLNEG2TT$dm[o1], lod[o1], pch=18, col="blue")

lod = -log10(CLLNEG2EB$p.value[,2])
plot(CLLNEG2TT$dm, -log10(CLLNEG2EB$p.value[,2]), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
o1 = abs(CLLNEG2TT$dm)>1.5
points(CLLNEG2TT$dm[o1], lod[o1], pch=18, col="blue")

lod = -log10(CLLNEG4TT$p.value)
plot(CLLNEG4TT$dm, -log10(CLLNEG4TT$p.value), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
```

```
o1 = abs(CLLNEG4TT$dm)>1.5
points(CLLNEG4TT$dm[o1], lod[o1], pch=18, col="blue")

lod = -log10(CLLNEG4EB$p.value[,2])
plot(CLLNEG4TT$dm, -log10(CLLNEG4EB$p.value[,2]), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
o1 = abs(CLLNEG4TT$dm)>1.5
points(CLLNEG4TT$dm[o1], lod[o1], pch=18, col="blue")

lod = -log10(CLLNEG6TT$p.value)
plot(CLLNEG6TT$dm, -log10(CLLNEG6TT$p.value), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
o1 = abs(CLLNEG6TT$dm)>1.5
points(CLLNEG6TT$dm[o1], lod[o1], pch=18, col="blue")

lod = -log10(CLLNEG6EB$p.value[,2])
plot(CLLNEG6TT$dm, -log10(CLLNEG6EB$p.value[,2]), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
o1 = abs(CLLNEG6TT$dm)>1.5
points(CLLNEG6TT$dm[o1], lod[o1], pch=18, col="blue")

lod = -log10(CLLNEG8TT$p.value)
plot(CLLNEG8TT$dm, -log10(CLLNEG8TT$p.value), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
o1 = abs(CLLNEG8TT$dm)>1.5
points(CLLNEG8TT$dm[o1], lod[o1], pch=18, col="blue")

lod = -log10(CLLNEG8EB$p.value[,2])
plot(CLLNEG8TT$dm, -log10(CLLNEG8EB$p.value[,2]), pch=".", xlab="log-ratio",
ylab=expression(log[10]~p))
abline(h=1.301)
o1 = abs(CLLNEG8TT$dm)>1.5
points(CLLNEG8TT$dm[o1], lod[o1], pch=18, col="blue")

library(annotate)
library ("hgu95av2.db")

annotation(CLLNEG2) <- "hgu95av2.db"
annotation(CLLNEG2)

genenames2tt=featureNames(CLLNEG2)[p.adjust(CLLNEG2TT$p.value,method="BH")<.05]
ll=getEG(genenames2tt,"hgu95av2")
```

```
sym=getSYMBOL(genenames2tt,"hgu95av2")

genenames2eb=featureNames(CLLNEG2)[p.adjust(CLLNEG2EB$p.value[,2],method="BH")<.
05]
ll2=getEG(genenames2eb,"hgu95av2")
sym2=getSYMBOL(genenames2eb,"hgu95av2")

ll2
sym2

tab=topTable(CLLNEG2EB, coef=2, adjust.method="BH", n=10)
tab=data.frame(sym,signif(tab[,-1],3))
htmlpage(list(ll),othernames=tab,, filename="GeneList2.html",
  title="HTML Report", table.center=T,
  table.head=c("Entrez ID",colnames(tab)))


genenames4eb=featureNames(CLLNEG4)[p.adjust(CLLNEG4EB$p.value[,2],method="BH")<.
05]
ll4=getEG(genenames4eb,"hgu95av2")
sym4=getSYMBOL(genenames4eb,"hgu95av2")

ll4
sym4

genenames6eb=featureNames(CLLNEG6)[p.adjust(CLLNEG6EB$p.value[,2],method="BH")<.
05]
ll6=getEG(genenames6eb,"hgu95av2")
sym6=getSYMBOL(genenames6eb,"hgu95av2")

ll6
sym6

genenames8eb=featureNames(CLLNEG8)[p.adjust(CLLNEG8EB$p.value[,2],method="BH")<.
05]
ll8=getEG(genenames8eb,"hgu95av2")
sym8=getSYMBOL(genenames8eb,"hgu95av2")

ll8
sym8

########################################################################################
########################################################################################

#RMA
CLLrma=rma(CLLB)
```

```
# filtering
CLLf=nsFilter(CLLrma, remove.dupEntrez=F,var.cutof=0.5)$eset

# testing for DE
CLLtt=rowttests(CLLf, "Disease")

library(limma)
design=model.matrix(~CLLf$Disease)
CLLlim=lmFit(CLLf, design)
CLLeb=eBayes(CLLlim)

# annotation
library(annotate)
annotation(CLLf)
library(hgu95av2.db)
genenames=featureNames(CLLf)[p.adjust(CLLeb$p.value[,2],method="BH")<.05]

ll=getEG(genenames,"hgu95av2")
sym=getSYMBOL(genenames,"hgu95av2")

###################################################################################
###################################################################################

#Using Hypergeometric tests to determine overepresentation of gene
source("http://www.bioconductor.org/biocLite.R")
biocLite("GOstats")
library("GOstats")
library(annotate)
library ("hgu95av2.db")

CLL2=nsFilter(CLLrma, require.entrez=T,require.GOBP=T, var.cutoff=0.2)
CLL2BP=CLL2$eset
rt=rowttests(CLL2BP,"Disease")

affyUniverse=featureNames(CLL2BP)
uniId=hgu95av2ENTREZID[affyUniverse]
entrezUniverse=unique(as.character(uniId))
EGsub=featureNames(CLL2BP)[p.adjust(rt$p,method="holm")<.05]
EGsub1=as.character(hgu95av2ENTREZID[EGsub])
params=new("GOHyperGParams",geneIds=EGsub1,
universeGeneIds=entrezUniverse,annotation="hgu95av2",ontology="BP",
  pvalueCutoff=0.05, conditional=F, testDirection="over")
mfhyper=hyperGTest(params)
```

```
###################################################################################
###################################################################################

biocLite("biomaRt")
library("biomaRt")
head(listMarts())
head(listDatasets(mart))

#http://www.ensembl.org/info/data/biomart/biomart_r_package.html; Based on google, the genes
are from this dataset: btaurus_gene_ensembl
listDatasets(ensembl)


mart=useMart("ensembl")
ensembl = useEnsembl(biomart="ensembl", dataset="btaurus_gene_ensembl")


##The "listAttributes" function will give you the list of the available attributes for a given mart
and species
listAttributes(ensembl)
listFilters(ensembl)

info=getBM(c('ensembl_gene_id', 'go_id'), filters="ENSBTAG00000000005",
values="ENSBTAG00000000005",mart=ensembl)
info=getBM(c('ensembl_gene_id', 'go_id'), filters="ENSBTAG00000000008",
values="ENSBTAG00000000008",mart=ensembl)
info=getBM(c('ensembl_gene_id', 'go_id'), filters="ENSBTAG00000000009",
values="ENSBTAG00000000009",mart=ensembl)
info=getBM(c('ensembl_gene_id', 'go_id'), filters="ENSBTAG00000000009",
values="ENSBTAG00000000009",mart=ensembl)
info=getBM(c('ensembl_gene_id', 'go_id'), filters="ENSBTAG00000000009",
values="ENSBTAG00000000009",mart=ensembl)

###################################################################################
###################################################################################

source("http://www.bioconductor.org/biocLite.R")
biocLite("hgu133a.db")
library("hgu133a.db")
dba = hgu133a_dbconn()
length(dba)
phenoData("hgu133a.db")

## Bimap interface:
x <- hgu133aACCNUM
# Get the probe identifiers that are mapped to an ACCNUM
```

```
mapped_probes <- mappedkeys(x)
# Convert to a list
xx <- as.list(x[mapped_probes])
length(xx)

biocLite("hgu133b.db")
library("hgu133b.db")
dbb = hgu133b_dbconn()

######################################################################################
######################################################################################

#Look at the columns in the chip database
columns(hgu133b.db)

## list supported key types (also the column names)
keytypes(hgu133b.db)

## get a default result (captures only the 1st element)
keys <- head(keys(hgu133b.db, 'ENTREZID'))
mapIds(hgu133b.db, keys=keys, column='ALIAS', keytype='ENTREZID')

#Listing all the packages available for the database
ls("package:hgu133b.db")

#Then making a count of the probes based on the ENTREZID
EG = as.character(hgu133bENTREZID[featureNames(hgu133b)])

######################################################################################
######################################################################################

## Bimap interface:
x <- hgu133bACCNUM
# Get the probe identifiers that are mapped to an ACCNUM
mapped_probes <- mappedkeys(x)
# Convert to a list
xx <- as.list(x[mapped_probes])
length(xx)

###Number of Ensembl Ids

## Bimap interface:
x <- hgu133aENSEMBL
# Get the probe identifiers that are mapped to an ENSEMBL
mapped_probes <- mappedkeys(x)
# Convert to a list
```

```
xx <- as.list(x[mapped_probes])
length(xx)

## Bimap interface:
x <- hgu133bENSEMBL
# Get the probe identifiers that are mapped to an ENSEMBL
mapped_probes <- mappedkeys(x)
# Convert to a list
xx <- as.list(x[mapped_probes])
length(xx)

###Number of Gene symbols

x <- hgu133aSYMBOL
# Get the probe identifiers that are mapped to an ENSEMBL
mapped_probes <- mappedkeys(x)
# Convert to a list
xx <- as.list(x[mapped_probes])
length(xx)

## Bimap interface:
x <- hgu133bSYMBOL
# Get the probe identifiers that are mapped to an ENSEMBL
mapped_probes <- mappedkeys(x)
# Convert to a list
xx <- as.list(x[mapped_probes])
length(xx)


##Gene present in a but not b
x <- hgu133aSYMBOL
# Get the probe identifiers that are mapped to an ENSEMBL
mapped_probes <- mappedkeys(x)
# Convert to a list
xxa <- as.list(x[mapped_probes])
length(xxa)

## Bimap interface:
x <- hgu133bSYMBOL
# Get the probe identifiers that are mapped to an ENSEMBL
mapped_probes <- mappedkeys(x)
# Convert to a list
xxb <- as.list(x[mapped_probes])
length(xxb)

Mismatch=xxa[!is.na(xxb)]
```

```
length(Mismatch)

########################################################################################
########################################################################################

### PUBH 7445 Microarray review code

# Definitions

#Microarrays are physical platforms that have nucleotide sequences bound to their surface.
#Since nucleotide sequences hybridize to their complements we can use these bound sequences
to fish out their complements from a mixture of nucleotide sequences.
#They are used for a) genotyping b) measuring gene expression c) determining DNA copy
number d) determining transcription factor binding sites

#Here are they steps for the processing:
#a) the patient and control DNA are labeled with fluorescent dye and applied with the microarray
#b) after hybridization, they bound to the matching probe on the chip and the unbound pairs are
washed away
#c) the microarray measures the strength of the resulting signial from the attached probes, which
is then analyzed using specialized computer software

#Here is information on the microarray probes
#each probe is a specific sequence of single stranded DNA/RNA which is complementary to the
genetic sequence we are interested in
#each probe pair contain a pair of perfect match and missmatch probes for the genetic sequence
of interest, which are paired up on the cell
#each probe set contains a certain number of pair specifically picked to interrogate a specific
sequence by matching up to it at different spots of the sequence.

##The ALL Dataset

#The ALL data consist of microarrays from 128 different individuals with acute lymphoblastic
leukemia(ALL.
#There are 95 samples with B-cell ALL and 33 with T-cell ALL and because these are different
tissues and quite different diseases we consider them separately, and typically focus on the B-cell
ALL tumors.
#A number of covariates are stored along with data, describing more general properties of the
patients such as age or sex but also a whole range of clinical parameters about type and stage of
the disease
#The data have been jointly normalized using rma and stored in the form of an ExpressionSet

#For the Biobase data packages, you cannot use the library function directly, you need to go
through the biocilite custom application
```

#To make the installation of Bioconductor packages as easy as possible, we provide a Web-accessible script called biocLite that you can use to install any Bioconductor package along with its dependencies. You can also use biocLite to install packages hosted on CRAN.

```
source("https://bioconductor.org/biocLite.R")
biocLite("Biobase")
biocLite("ALL")
biocLite("genefilter")
```

#The command update.packages can be used to check for and install new versions of already installed packages. Note that you need to supply update.packages with the URL to a Bioconductor repository in order to update Bioconductor packages as well. The recommended way of updating all your installed packages is:

```
source("http://bioconductor.org/biocLite.R")
update.packages(repos=biocinstallRepos(), ask=FALSE)
```

#Once you have used the biocLite function, you can resume using the Library function as usual

```
library("Biobase")
library("ALL")
library("genefilter")
```

#Data subsetting

#An interesting subset, with two groups having approximately the same number of samples in each group, is the comparison of the B-cell tumors found to carry the BCR/ABL mutation to those B-cell tumors with no observed cytogenetic abnormalities
#These samples are labeled BCR/ABL and NEG in the mol.biol covariate.

#We ?rst load the ALL package and attach the data to our work space

```
data("ALL")
```

#Then, we select those samples originating from B-cell tumors by searching the BT variable (which distinguishes the B-cell from the T-cell tumors) for entries beginning with the letter B using a regular expression.

```
bcell = grep("^B", as.character(ALL$BT))
```

#Next, we want to know which of the samples are of molecular types BCR/ABL or NEG.

```
types = c("NEG", "BCR/ABL")
moltyp = which(as.character(ALL$mol.biol) %in% types)
```

#Combining these two selection criteria gives us a data set for carrying out comparisons between tumors harboring the BCR/ABL translocation and those that did not have any of the tested molecular abnormalities

```
ALL_bcrneg = ALL[, intersect(bcell, moltyp)]
```

#One more step is required. Some of the sample annotation data is kept in variables of type factor. These are variables that can only take a number of discrete categorical values.
#Because we have reduced the set of samples, we only need fewer factor levels than were present in ALL, and the most succint way to reduce the set of levels of a factor to those that are actually present is by calling the constructor function factor on it again.

```
ALL_bcrneg$mol.biol = factor(ALL_bcrneg$mol.biol)
ALL_bcrneg$BT = factor(ALL_bcrneg$BT)
```

##Nonspeci?c ?ltering
#Some fraction of genes were not expressed at all in the cells that were assayed, at least not to a level that we could detect with the microarrays used here.
#For further genes, the data did not show enough variation to allow any reliable detection of differential expression.
#It is a good idea to remove probe sets for these genes before further analysis, and we can do that on the basis of variance
#Here, we show how to proceed using the function nsFilter from the gene?lter package to ?lter for a number of different criteria, all controlled by the function's various parameters.
#Setting feature.exclude="^AFFX" removes the control probes, which can be identi?ed by the pre?x AFFX in their name. As a measure of dispersion for the variance ?ltering step, we use the interquartile range (IQR), and we choose the 0.5 quantile of the IQR values as a cutoff

```
varCut = 0.5
filt_bcrneg = nsFilter(ALL_bcrneg, require.entrez=TRUE, require.GOBP=TRUE,
remove.dupEntrez=TRUE, var.func=IQR, var.cutoff=varCut, feature.exclude="^AFFX")
filt_bcrneg$filter.log
```

#Checking which variables got removed from the dataset

```
$numDupsRemoved
$numLowVar
$feature.exclude
$numNoGO.BP
$numRemoved.ENTREZID
ALLfilt_bcrneg = filt_bcrneg$eset
```

#Here is another example of subsetting your data for a dataset comparing cancer of the BCR/ABL mutation to the ALL1/F4 Mutation.
#All the steps for the extraction are combined at once

```
types = c("ALL1/AF4", "BCR/ABL")
```

```
moltyp = which(ALL$mol.biol %in% types)
ALL_af4bcr = ALL[, intersect(bcell, moltyp)]
ALL_af4bcr$mol.biol = factor(ALL_af4bcr$mol.biol)
ALL_af4bcr$BT = factor(ALL_af4bcr$BT)
filt_af4bcr = nsFilter(ALL_af4bcr,require.entrez=TRUE, require.GOBP=TRUE,
remove.dupEntrez=TRUE, var.func=IQR, var.cutoff=varCut)
ALLfilt_af4bcr = filt_af4bcr$eset
```

##The apply family of functions

#In R a great deal of work is done by applying some function to all elements of a list, matrix, or array. There are several functions available for you to use; apply, lapply, sapply are the most commonly used.
#The function eapply is also available for applying a function to each element of an environment.

#The hgu95av2MAP environment contains the mappings between A?ymetrix identi?ers and chromosome band locations.
#For example, in the code below we ?nd the chromosome band to which the gene, for probe 1001_at (TIE1), maps.

```
library("hgu95av2")
hgu95av2MAP$"1001_at
```

#We can extract all of the map locations for a particular chromosome or part of a chromosome by using regular expressions and the apply family of functions.
#say we want to ?nd all genes that map to the p arm of chromosome 17. Then we know that their map positions will all start with the characters 17p.
# This is a simple regular expression, ^17p, where the caret, ^, means that we should match the start of the word. We do this in two steps: ?rst we use eapply and grep and ask for grep to return the value that matched.

```
myPos = eapply(hgu95av2MAP, function(x) grep("^17p", x, value=TRUE))
myPos = unlist(myPos)
length(myPos)
```

# Note that here we used an anonymous function to process each element of the hgu95av2MAP environment. We could have named it and then used it.

```
f17p = function(x) grep("^17p", x, value=TRUE)
myPos2 = eapply(hgu95av2MAP, f17p)
myPos2 = unlist(myPos2)
```

##Environments

#In R, an environment is a set of symbol–value pairs. These are similar to lists, but there is no natural ordering of the values and so you cannot make use of numeric indices.

#Here, we show how to work with your own environments.

#We ?rst create an environment and carry out some simple tasks, such as storing things in it, removing things from it, and listing the contents.

```
e1 = new.env(hash=TRUE)
e1$a = rnorm(10)
e1$b = runif(20)
ls(e1)
xx = as.list(e1)
names(xx)
rm(a, envir=e1)
```

#Genomic data can be very complex, usually consisting of a number of different bits and pieces.

#In Bioconductor we have taken the approach that these pieces should be stored in a single structure to easily manage the data.

#The package Biobase contains standardized data structures to represent genomic data.

#TheExpressionSet classisdesignedtocombineseveral di?erent sources of information into a single convenient structure. An ExpressionSet can be manipulated (e.g., subsetted, copied), and is the input to or output of many Bioconductor functions.

#The data in an ExpressionSet consist of assay data, metadata and experiment data

#assayData:Expressiondatafrommicroarrayexperiments(assayData is used to hint at the methods used to access different data components, as we show below).

#metadata:Adescriptionofthesamplesintheexperiment(phenoData), metadataaboutthefeaturesonthechiportechnologyusedfortheexperiment (featureData), and further annotations for the features, for example gene annotations from biomedical databases (annotation).

#experimentData: A ?exible structure to describe the experiment

#The ExpressionSet class coordinates all of these data, so that you do not usually have to worry about the details. However, an ExpressionSet needs to be created in the ?rst place, because it will be the starting point for many of the analyses using Bioconductor software.

#Building an ExpressionSet from .CEL and other ?les

#Many users have access to .CEL or other ?les produced by microarray chip manufacturer hardware.

#Usually the strategy is to use a Bioconductor package such as a?yPLM, a?y, oligo, limma, or arrayMagic to read these ?les.

#These Bioconductor packages have functions (e.g., ReadAffy, expresso, or justRMA in a?y) to read CEL ?les and perform preliminary preprocessing, and to represent the resulting data as an ExpressionSet or other type of object.

#Building an ExpressionSet from scratch

#The data from many high-throughput genomic experiments, such as microarray experiments, usually consist of several conceptually distinct parts: assay data, sample annotations, feature annotations, and an overall description of the experiment. We construct each of these components, and then assemble them into an ExpressionSet.

#Assay data
#One important part of the experiment is a matrix of “expression” values.
#The matrix has F rows and S columns, where F is the number of features on the chip and S is the number of samples.
#A likely scenario is that your assay data are in a “tab-delimited” text ?le (as exported from a spreadsheet, for instance) with rows corresponding to features and columns to samples.
#The strategy is to read this ?le into R using the read.table command, converting the result to a matrix. A typical command to read a tab-delimited ?le that includes column headers is

#The ?rst two lines create a ?le path pointing to where the assay data are stored; replace these with a character string pointing to your own ?le, for example

```
dataDirectory = system.file("extdata", package="Biobase")
exprsFile = file.path(dataDirectory, "exprsData.txt")
exprs = as.matrix(read.table(exprsFile, header=TRUE, sep="\t", row.names=1, as.is=TRUE))
```

#(Windows users: note the use of / rather than \; this is because R treats the \ character as an “escape” sequence to change the meaning of the subsequent character.) See the help pages for read.table for more detail. A common variant is that the character separating columns is a comma (“comma-separated values”, or “csv” ?les), in which case the sep argument might be sep=",")
#It is always important to verify that the data you have read match your expectations. At a minimum, check the class and dimensions of geneData and take a peek at the ?rst several rows.

```
class(exprs)
dim(exprs)
colnames(exprs)
head(exprs)
```

#Sample annotation
#The information about the samples (e.g., experimental conditions or parameters, or attributes of the subjects such as sex, age, and diagnosis) is often referred to as covariates.
#The information describing the samples can be represented as a table with S rows and V columns, where V is the number of covariates.

```
pDataFile = file.path(dataDirectory, "pData.txt")
pData = read.table(pDataFile, row.names=1, header=TRUE, sep="\t")
dim(pData)
rownames(pData)
summary(pData)
```

#Note that the rows of the sample data table align with the columns of the expression data matrix
#This is an essential feature of the relationship between the assay and sample data; ExpressionSet will complain if these names do not match.

```
all(rownames(pData) == colnames(exprs))
```

#Metadata
#Investigators often ?nd that the meaning of simple column names does not provide enough information about the covariate
#We can create a data frame containing such metadata (or read the information from a ?le using read.table)
#Here we create a data.frame object with a single column called“labelDescription”, and with row names identical to the column names of the data.frame containing the sample annotation data. The column labelDescription must be present; other columns are optional.

```
metadata = data.frame(labelDescription=c("Patient gender", "Case/control status", "Tumor progress on XYZ scale"), row.names=c("gender", "type", "score"))
```

#Alternatively, Bioconductor’s Biobase package provides a class called AnnotatedDataFrame that conveniently stores and manipulates tabular data in a coordinated fashion.

```
adf = new("AnnotatedDataFrame", data=pData, varMetadata=metadata)
adf
```

#Some useful operations on an AnnotatedDataFrame include sampleNames, pData (to extract the original pData data.frame), and varMetadata. In addition, AnnotatedDataFrame objects can be subset much as a data.frame

```
head(pData(adf)
adf[c("A","Z"), "gender"]
pData(adf[adf$score > 0.8,])
```

##Annotations and feature data
#Metadata on features are as important as metadata on samples, and can be very large and diverse. The idea is to construct specialized metadata packages for each type of chip or instrument. The annotate package provides basic data manipulation tools for the metadata packages
#The appropriate way to create annotation data for features is very straight forward: we provide a character string identifying the type of chip used in the experiment. For instance, the data we are using are from the A?ymetrix HGU95Av2 GeneChip

```
annotation = "hgu95av2"
```

#It is also possible to record information about features that are unique to the experiment (e.g., ?agging particularly relevant features). This is done by creating or modifying an AnnotatedDataFrame like that for adf but with row names of the AnnotatedDataFrame matching rows of the assay data.

##Assembling an ExpressionSet

#An ExpressionSet object is created by assembling its component parts, and after all this work the ?nal assembly is disappointingly easy

```
exampleSet = new("ExpressionSet", exprs=exprs, phenoData=adf,
experimentData=experimentData, annotation="hgu95av2")
```

#Note that the names on the right of each equal sign can refer to any object of appropriate class for the argument. See the help page for ExpressionSet for more information.
#We created a rich data object to coordinate diverse sources of information. Less rich objects can be created by providing less information. A minimal expression set can be created with.

```
minimalSet = new("ExpressionSet", exprs=exprs)
```

##ExpressionSet basics
#When you print an ExpressionSet object, a brief summary of the contents of the object is displayed (displaying the entire object would ?ll your screen with numbers)

```
exampleSet
```

#Accessing data elements
#A number of accessor functions are available to extract data from an ExpressionSet instance. You can access the columns of the sample data (an AnnotatedDataFrame) using $

```
exampleSet$gender[1:5]
exampleSet$gender[1:5] == "Female"
```

#You can retrieve the names of the features using featureNames. For many microarray datasets, the feature names are the probe set identi?ers.

```
featureNames(exampleSet)[1:5]
```

#The unique identi?ers of the samples in the dataset are available via the sampleNames method. The varLabels method lists the column names of the sample data:

```
sampleNames(exampleSet)[1:5]
varLabels(exampleSet)
```

#Extract the expression matrix and the AnnotatedDataFrame of sample information using exprs and phenoData, respectively:

```
mat = exprs(exampleSet)
adf = phenoData(exampleSet)
```

##Subsetting
#Probably the most useful operation to perform on ExpressionSet objects is subsetting. Subsetting an ExpressionSet is very similar to subsetting the expression matrix that is contained within the ExpressionSet: the ?rst argument subsets the features and the second argument subsets the samples. Here are some examples.
#Create a new ExpressionSet consisting of the ?ve features and the ?rst three samples

```
vv = exampleSet[1:5, 1:3]
dim(vv)
featureNames(vv)
sampleNames(vv)
```

#Create a subset consisting of only the male samples

```
males = exampleSet[, exampleSet$gender == "Male"]
males
```

#The function plot can be used to produce dot plots: In this example, we use the plot function to compare the gene expression intensities of two samples from our dataset on a log–log scale.

```
x = exprs(exampleSet[, 1])
y = exprs(exampleSet[, 3])
plot(x=x, y=y, log="xy"
```

#Proper visualization can help to detect possible problems or inconsistencies in the data. In the simplest case one can spot such problems by looking at distribution summaries. A good example for this is the dependency of the measurement intensity of a microarray probe on its GC-content.
#To demonstrate this, we need to load a more extended data set from the CLL package which includes the raw measurement values for each probe from an experiment using the Affymetrix HG-U95Av2 GeneChip. The basecontent function from package matchprobes calculates the base frequencies for each probe based on a sequence vector.

```
library("CLL")
library("matchprobes")
library("hgu95av2probe")
library("hgu95av2cdf")
library("RColorBrewer")
data("CLLbatch")
bases = basecontent(hgu95av2probe$sequence)
```

#We now need to match the probes via their position on the array to positions in the data matrix of the CLLbatch object.
#Because we have several samples in the set, we use the rowMeans function to compute, for each probe, the average of its expression values across arrays.

```
iab = with(hgu95av2probe, xy2indices(x, y, cdf="hgu95av2cdf"))
probedata = data.frame( int=rowMeans(log2(exprs(CLLbatch)[iab, ])), gc=bases[, "C"] + bases[, "G"])
```

### The input data: CEL ?les
#The most widely used and generally most useful format in which A?ymetrix data can be obtained is the CEL format.

#To import these data into Bioconductor, most users will be served by the ReadAffy function from the a?y package.

```
library("affy")
myAB = ReadAffy()
```

#The function ReadAffy imports CEL ?les into R objects of class A?yBatch. These are the basic containers for A?ymetrix datasets in Bioconductor.
#Alternatively, you can supply the names of the ?les that you want to import in the filenames argument, as shown in the next code chunk.

```
myAB = ReadAffy(filenames=c("a1.cel", "a2.cel", "a3.cel"))
```

#In this chapter we make use of data from a cohort of patients with chronic lymphocytic leukemia (CLL). The data are contained in the CLL package. The author of the package has already performed the task of reading the CEL ?les into an A?yBatch object (this saves you time and disk space), so here we can start with that object already. The commands needed to load the object are given next

```
library("CLL")
data("CLLbatch")
CLLbatch
```

#The sample annotation

#First let us look at the names of the 24 samples in CLLbatch.

```
sampleNames(CLLbatch)
```

#These data come from 24 patients with CLL. Although a great number of clinical covariates were collected, we restrict our attention to only one, disease status, which is either progressive (abbreviated by progres.), stable (stable), or missing (NA).
#In practice, such data will often be provided to you in a spreadsheetlike format. In the package CLL, they are provided by the data.frame object disease.

```
data("disease")
head(disease)
```

#As is often the case, we need to do a little data reorganization before starting with the analysis. First, the row names of the data.frame should be a suitable set of identi?ers.
#Here we use the SampleID column for this purpose.

```
rownames(disease) = disease$SampleID
```

#Next, we remove the (uninformative) suffixes .CEL from the sample names annotation of CLLbatch, in order to make them match the row names of the disease data.frame.

```
sampleNames(CLLbatch) = sub("\\.CEL$", "", sampleNames(CLLbatch))

#We construct a vector mt that contains the matching of the rows of disease with the samples of CLLbatch.

mt = match(rownames(disease), sampleNames(CLLbatch))

#Next, we want to provide longer descriptions of the variables so that if we return to the analysis in a few months (or years) we will still be able to see what the variables were

vmd = data.frame(labelDescription = c("Sample ID", "Disease status: progressive or stable disease")

#Finally, we are ready to construct an AnnotatedDataFrame object which we can insert into CLLbatch for the annotation of the samples.

phenoData(CLLbatch) = new("AnnotatedDataFrame", data = disease[mt, ], varMetadata = vmd)

#We drop one array for which we have no information on the disease status, because for the analysis we intend this array will not be informative.

CLLbatch = CLLbatch[, !is.na(CLLbatch$Disease)]

###How to remove the bad arrays

badArray = match("CLL1", sampleNames(CLLbatch)) > CLLB = CLLbatch[, -badArray]

###Expression microarray preprocessing comprises three major tasks:
#background correction, which aims to adjust the intensity readings for nonspeci?c signal and hence to increase the array’s sensitivity;
#between-array normalization, which aims to adjust the intensity readings for technical variability between arrays due to subtle di?erences in handling, labeling, hybridization, and scanning;
#reporter summarization, which computes a summary gene expression value for each gene from all the features on the array that target its transcripts.

#A comprehensive method that provides all three of the preprocessing tasks is RMA. It is provided by the function rma in the affy package
#It accepts an instance of the A?yBatch class and returns an ExpressionSet object that can be used in downstream analyses

CLLrma = rma(CLLB)

#The expression values calculated by rma are in log2 scale. A matrix with the estimate expression values can be obtained by using the exprs method.
```

```
# The following code extracts the expression data and obtains the dimensions of the matrix containing the data. The same information can also be obtained by calling the dim function directly on the ExpressionSet.

e = exprs(CLLrma)
dim(e)

#or

dim(CLLrma)

#The sample information that was stored with the raw data CLLB has been transferred to the ExpressionSet. This picks the first 3 rows for us to look at

pData(CLLrma)[1:3,]

#You can conveniently use $ to access the sample annotation variables.

table(CLLrma$Disease)

#At this stage of the analysis, we have obtained expression estimates for each gene and each sample.
#The sensitivity of microarrays is such that we do not expect to reliably detect expression, and di?erential expression, for more than, say, 50% of the genes on a genome-wide array such as the HG-U95Av2.
#The signal for the remaining genes will essentially consist of noise and will only aggravate the multiple testing problem.
#Thissuggeststhatsomeformofnonspeci?c ?ltering of noninformative probe sets will have potentially great bene?ts in the downstream analyses.

#The nsFilter function in the gene?lter package can be used to ?lter out probe sets on a variety of criteria. The one we use here is based on variability.

CLLf = nsFilter(CLLrma, remove.dupEntrez=FALSE, var.cutof =0.5)$eset

#Summary statistics and tests for ranking
#Log fold-change

#A naive ?rst choice for comparing two groups is the average log fold-change. It can be computed by using R base functions such as rowMeans applied to the appropriate columns of the matrix exprs(CLLf). We can also use the more convenient function rowttests.

CLLtt = rowttests(CLLf, "Disease")
names(CLLtt)
```

#The function computes for each row the average log-ratio between the two disease groups, the t-statistic, and the corresponding p-value by using the Student’s t-distribution.

#The variability of log-ratios often depends on the overall intensity of the probe set in question. We can estimate the overall intensity by the average log expression, which is computed in the code chunk below

```
a = rowMeans(exprs(CLLf))
```

#t-statistic

#The t-statistic measures the di?erence in mean divided by an estimate of the variance. When there are few replicates, the variance is not well estimated and the t-statistic can perform poorly

# Alternative statistics that borrow information across genes often provide better results. An instance of such a modi?ed t-statistic is based on an empirical Bayes moderation approach, implemented in the eBayes function in the limma package.

```
library("limma")
design = model.matrix(~CLLf$Disease)
CLLlim = lmFit(CLLf, design)
CLLeb = eBayes(CLLlim)
```

#Visualization of di?erential expression

#The volcano plot is a useful way to see the estimate of the log fold-change and statistic you choose to rank the genes simultaneously

```
lod = -log10(CLLtt$p.value)
plot(CLLtt$dm, lod, pch=".", xlab="log-ratio", ylab=expression(-log[10]~p))
abline(h=2)
```

#Suppose we consider all genes attaining p-values less than 0.01. In a single experiment, a p-value of less than 0.01 would be regarded as highly signi?cant.

#However, here we are testing many hypotheses simultaneously, and the p-values no longer have the conventional meaning.

#We are testing 6098 null hypotheses (this is the number of probe sets in CLLf) in the example dataset. If they are all true(i.e.,if no genes are differentially expressed) we expect 0.01×6098=61 hypothesis rejections, which all would be false positives.

#Various approaches have been suggested for dealing with the multiple testing problem.

#The multtest package provides implementations of many oftheoptions.Alternatively,thefunction topTable in the limma package provides multiple testing adjustment methods, including Benjamini and Hochberg’s false discovery rate (FDR), simple Bonferroni correction, and several others.

#The following example lists the top ten genes and creates a report. Here, coef=2 speci?es that we care for the second coe?cient in the linear model?t,thatis,thebetween-groupdi?erence,astheparameterofinterest. The ?rst coe?cient is the intercept. The parameter n indicates how many genes should be selected.

```
tab = topTable(CLLeb, coef=2, adjust.method="BH", n=10)
genenames = as.character(tab$ID)

#Annotation
#First we load the annotate package

library("annotate")

#?nd out which metadata package we need,

annotation(CLLf)

#and load it

library("hgu95av2")

#Now we can retrieve more annotation information about the genes that interest us, for example,
their EntrezGene ID and gene symbol.

ll = getEG(genenames, "hgu95av2")
sym = getSYMBOL(genenames, "hgu95av2")

#We can use the following code to create an HTML page, useful for instance to share results
with collaborators

tab = data.frame(sym, signif(tab[,-1], 3))
htmlpage(list(ll), othernames=tab, filename="GeneList1.html", title="HTML report",
table.center=TRUE, table.head=c("Entrez ID",colnames(tab)))
browseURL("GeneList1.html")

#Advanced preprocessing

#The “raw” feature intensity data are typically provided in CEL ?les, one for each scanned array,
whereas the assignment of features to target genes is made in the so-called CDF ?le, of which
there is only one for each type of array. In this section, we demonstrate how you can work with
these data, in order to construct your own custom analysis methods.

#PM and MM probes

#Many Affymetrix arrays have both perfect match (PM) and mismatch (MM) probes on the
arrays. The PM probes are intended to be complementary to the mRNA being probed, while the
MM probes have the same sequence except in the thirteenth position, where the base is changed
to its complementary base (e.g. if A in the PM, then T in the MM).
#The PMs are designed to hybridize speci?cally to their target cDNA of interest, however,
hybridization of short oligonucleotide tends not to be perfectly speci?c, and there is also a certain
```

amount of nonspeci?c (or less speci?c) hybridization from various other cDNA molecules. The intention of the MMs is to measure this nonspeci?c hybridization.
#If that were the case, then each MM should generally have lower intensities than its PM partner, and its value could be subtracted from the PM value in order to obtain a more accurate estimation of the signal due to speci?c hybridization of the target.

#The PM and MM values can be extracted using the functions pm and mm.

```
pms = pm(CLLB)
mms = mm(CLLB)
```

##Background-correction

#The feature intensities from A?ymetrix microarrays are always positive, even if the abundance of the intended target gene is zero. This is due to optical noise and, in many cases, cross-hybridization. Background-correction of the intensities is essential to obtain good sensitivity from the data

#In this section, we explore the background-correction method of RMA and compare it to an alternative method.
#In RMA, background-correction is done by ?tting a Normal-Exponential mixture model and subtracting a background estimate from the PM value of each probe that is estimated in such a way that the result is guaranteed to remain positive. Subsequently, the data are logarithm transformed
#InVSN,onlyoneoverallbackgroundestimate is computed for the whole array, and this estimate can be larger than some of the smaller feature intensities on the array. Hence, some of the background-subtracted values can be zero or negative. Subsequently, the so-called generalized logarithm transformation.

#The RMA background-correction is embedded within the function rma, and it is not easy to get at the backgroundcorrected intensities before the probe set summarization, but we can use the following code to produce them.

```
bgrma = bg.correct.rma(CLLB)
exprs(bgrma) = log2(exprs(bgrma))
```

#The following code obtains the VSN background correction.

```
library("vsn")
bgvsn = justvsn(CLLB)
```

#The return value of the call to justvsn above is an A?yBatch with values that have been background-corrected, normalized between arrays, and log2transformed. In order to do the summarization of probe sets, we could call the function rma with arguments normalize=FALSE and background=FALSE, or we can use the function vsnrma, which is a wrapper that performs all of these steps.

```
CLLvsn = vsnrma(CLLB)

#Then, We can repeat the same analysis as in Section 3.4 to do nonspeci?c ?ltering and testing for di?erential expression

##Summarization

#We next show how to explore other probe set summarization methods. First, we create a list of indices. Each element of the list indices corresponds to a probe set and contains the row indices of the matrices pms and mms corresponding to the probe set's probes. These data are obtained from the CDF ?le.

pns = probeNames(CLLB)
indices = split(seq(along=pns), pns)
length(indices)

#Let us try out a naive summary method: for each sample, we take the median of differences between the PM and MM values for each probe set.
#The code yielded a matrix with one row for each probe set and one column for each sample.

newsummary = t(sapply(indices, function(j) rowMedians(t(pms[j,]-mms[j,]))))
dim(newsummary)

######################################################################################
######################################################################################

### PUBH 7445 Final

#one useful thing to do is set the folder that R copies and draws files from. You can either do it from under file, or change the working directory using a command

#Set working directory for download: you have to use forward slashes in R pathways
getwd()
setwd("C:/Users/guillaumeo/Desktop/AFFY_FINAL")

##read.table wants to return a data.frame, which must have an element in each column. Therefore R expects each row to have the same number of elements and it doesn't fill in empty spaces by default. Try read.table("/PathTo/file.csv" , fill = TRUE ) to fill in the blanks.

read.table("AnalysisData_code.txt", header=TRUE, fill= TRUE)

test <- read.table("AnalysisData_code.txt",  fill= TRUE)

# The test dataset with the analysis code of ALL dataset will have 25 rows and 521 columns
```

```
dim(test)

#Importing the CSV file with the classification of resistant/intermediate/sensitive for each of the
173 patients in the dataset
#The supplementary web site for Holleman et al. [2], http://www.stjuderesearch.org/data/ALL4,
includes a “key.xls”file.
#There are 2 sheets in the CSV file, describing which samples are allocated to which categories,
and how these samples were posted to GEO.
# I extracted the main file, then save each sheet individually. Thanks to that, we extracted both
sheets as individual csv files for easier loading.

hollemanKeyGSM <- read.table("key_gsm.csv",  fill= TRUE)
dim(hollemanKeyGSM)
head(hollemanKeyGSM)

hollemanKeyGSE <- read.table("key_gse.csv",  fill= TRUE)
dim(hollemanKeyGSE)
head(hollemanKeyGSE)

#Processing Affymetrix Gene Expression Arrays
#Analyzing Affy microarrays with Bioconductor is "relatively" easy, particularly if all you want
is to get the gene expression matrix.  Once you have the gene expression values, much of the
analysis techniques that can be used for RNA-Seq analysis can also be used for microarrays.
Below is some basic information to get you started.
#This mini-tutorial assumes you are trying to analyze affymetrix arrays from CEL files.  CEL
files the 'raw' data files produced at the end of the array scan, and are normally deposited in gene
expression databases like GEO.  If you do not have the CEL files, make every attempt to find
them (if it's your data)!
# the affy package allows us to set up and annotate affy datasets
# the geoquery package allows us to pull the affy data from the GEO website directly
# the other method to get the .CEL file would be to either use the gunzip function in R, or
download the TAR file on your windows machine, use 7-zip to unpack the files, and load all of
them using the read affy funtion.


#Getting the necessary libraries to work with Affymetrix CEL files

# get the necessary libraries.  The first six lines only need to be executed the first time you run
this code

source("http://www.bioconductor.org/biocLite.R")
biocLite("affy")
biocLite("affyPLM")
biocLite("GEOquery")
biocLite("annotate")
biocLite("limma")
biocLite("hgu133a")
biocLite("simpleaffy")

library("affy")
library("affyPLM")
library("GEOquery")
library("annotate")
library("limma")
library("hgu133a")
library("simpleaffy")

# Read in probe level data

# you can also use the getGEO function from the GEOQuery dataset to get the data directly from the GEO database to play with it
GSE635<-getGEO("GSE635")

#Once you have taken the dataset directly from GEO, use the untar function to extract them into a new data folder, change the data directory to access the individual CEL files

untar("GSE635_RAW.tar", exdir="data")
cels = list.files("data/", pattern = "CEL")
sapply(paste("data", cels, sep="/"), gunzip)
cels = list.files("data/", pattern = "CEL")

setwd("C:/Users/onyea005/Desktop/AFFY_FINAL/data")

Meta(GSE635)
Columns(GSE635)
head(Table(GSE635))

##########################################################################################
##########################################################################################

#Downloading the GEO dataset directly from the WBOB console does not work

getwd()
source("http://www.bioconductor.org/biocLite.R")
biocLite("GEOquery")
library("GEOquery")

#Download GDS file, put it in the current directory, and load it:
GSE635 <- getGEO("GSE635")

#The meta function and GDS2eSet will not work because getGEO defaults to GSEMatrix=True. It works with GSEMatrix = False
```

```
#According to the documentation, the GSEMatrix=true is an ExpressionSet. Unless there is a specific reason to get the information from the full GSE software, it is better to use the defaults.

#functions dtah did not work with the GSE object, probably becuase it is arleady and expression set?
Meta(GSE635)$channel_count
eset <-GDS2eSet(GSE635, do.log2=TRUE)
geneNames(GSE635)
varLabels(GSE635)
pData(GSE635)
colnames(Table(GSE635))
ei <- exprs(GSE635)
Meta(GSE635)
Columns(GSE635)
head(Table(GSE635))
phenoData(GSE635)

#Just use the getGEO to generate your expressionset automatically

GSE635
sampleNames(GSE635)

#Loading a GPL annotation file with GEOquery
#You can find the annotation type by looking at the dataset (type GSE635)

GPL96 <- getGEO("GPL96")
colnames(Table(GPL96))

#You can also load the HGU133A annotation information from GPL96 directly from the bioconductor package

source("http://www.bioconductor.org/biocLite.R")
biocLite("hgu133a")
library("hgu133a")

#####################################################################################
#####################################################################################
#################################################################################

#Importing the CSV file with the classification of resistant/intermediate/sensitive for each of the 173 patients in the dataset
#The supplementary web site for Holleman et al. [2], http://www.stjuderesearch.org/data/ALL4, includes a “key.xls”file.
#There are 2 sheets in the CSV file, describing which samples are allocated to which categories, and how these samples were posted to GEO.
```

```
# I extracted the main file, then save each sheet individually. Thanks to that, we extracted both sheets as individual csv files for easier loading.

hollemanKeyGSM <- read.table("key_gsm.csv",  fill= TRUE)
dim(hollemanKeyGSM)
head(hollemanKeyGSM)
hollemanKeyGSM

hollemanKeyGSE <- read.table("key_gse.csv",  fill= TRUE)
dim(hollemanKeyGSE)
head(hollemanKeyGSE)

# I converted the excel file from the St Judes website to a CSV file and ultimately a txt file to make my vmd work
# I also converted the cell names to text and sorted them to make sure they match the affybatch format

KEYGSM2 <-read.table("key_gsm_GZ.txt", header=TRUE, fill= TRUE)
KEYGSM2

#in order for the affy funtions to work in importing the variables, you have to use 7-zip to unpack the TAR files in your directory, then load all of them using the read affy function

setwd("C:/Users/onyea005/Desktop/AFFY_FINAL")

#The affy package will automatically download the appropriate array annotation when you require it.
#Take a quick look at the raw affy data
#read the data

data <- ReadAffy()
data

phenoData(data)
pData(data)

# raw expression data
ed <- exprs(data)
samp <- sampleNames(data)
probes <- featureNames(data)

ed[1:10,]
probes[1:10]
samp

library("simpleaffy")
```

```
qc <- qc(data)
plot(qc)

# 2 Normalizing Data
# The Affy package has implementations of a number of normalization methods
# for single-channel arrays. This includes (among others): rma() is the 'Robust Multi-Chip'
average
# the normalized data is on the log-scale

nvals <- rma(data)
nvals

phenoData(nvals)
pData(nvals)

# normalised expression data
ned <- exprs(nvals)
nsamp <- sampleNames(nvals)
nprobes <- featureNames(nvals)

##Making the phenodata to go with the normalized dataset
####These data come from 173 patients with ALL. Although a great number of clinical
covariates were collected, we restrict our attention to the outcome, drug resistance status. In
practice, such data will often be provided to you in a spreadsheetlike format.

getwd()
GSM2 <- data.frame(KEYGSM2)
GSM2

#As is often the case, we need to do a little data reorganization before starting with the analysis.
First, the row names of the data.frame should be a suitable set of identifiers.
rownames(GSM2) = GSM2$gsm

#Next, we remove the (uninformative) suffixes .CEL from the sample names annotation of our
GEO data, in order to make them match the row names of the outcome data frame
samp2 = sampleNames(nvals)

#We construct a vector mt that contains the matching of the rows of disease with the samples of
CLLbatch.
mt = match(rownames(GSM2), samp2)
mt

#Next, we want to provide longer descriptions of the variables so that if we return to the analysis
in a few months (or years) we will still be able to see what the variables were
vmd = data.frame(labelDescription = c("gsm", "IPT", "ASP_score", "DNR_score",
"PRED_score", "VCR_score"))
```

```
vmd
```

#Finally, we are ready to construct an AnnotatedDataFrame object which we can insert for the annotation of the samples.

```
phenoData(nvals) = new("AnnotatedDataFrame", data = GSM2[mt, ], varMetadata = vmd)
adf = phenoData(nvals)
adf
pData(adf)
pData(nvals)
```

#Note that the rows of the sample data table align with the columns of the expression data matrix: This is an essential feature of the relationship between the assay and sample data; ExpressionSet will complain if these names do not match.

```
all(rownames(adf) == colnames(nvals))
```

#A comprehensive method that provides all three of the preprocessing tasks is RMA. It is provided by the function rma in the affy package: It accepts an instance of the AffyBatch class and returns an ExpressionSet object that can be used in downstream analyses

#the command phenoData(nvals) automatically replaced the phenodata from my rma dataset with the file with the drug resistance status I wanted

#The sample information that was stored with the raw data has been transferred to the ExpressionSet. This picks the first 3 rows for us to look at

```
pData(nvals)[1:3,]
```

#You can conveniently use $ to access the sample annotation variables.

```
table(nvals$ASP_score)
```

#At this stage of the analysis, we have obtained expression estimates for each gene and each sample. The sensitivity of microarrays is such that we do not expect to reliably detect expression, and differential expression, for more than, say, 50% of the genes on a genome-wide array such as the HG-U95Av2. The signal for the remaining genes will essentially consist of noise and will only aggravate the multiple testing problem. This suggests that some form of nonspecific filtering of noninformative probe sets will have potentially great benefits in the downstream analyses. The nsFilter function in the genefilter package can be used to filter out probe sets on a variety of criteria. The one we use here is based on variability.

```
ALLf = nsFilter(nvals, remove.dupEntrez=FALSE, var.cutof =0.5)$eset
nvals
ALLf
```

#t-statistic would not work, because we have 3 levels for the outcomes

```
ALLtt = rowttests(ALLf, "ASP_score")
names(ALLtt)
```

```
#Visualization of differential expression (doesn't work because we did not use the row ttest function)
#The volcano plot is a useful way to see the estimate of the log fold-change and statistic you choose to rank the genes simultaneously
lod = -log10(ALLtt$p.value)
plot(ALLtt$dm, lod, pch=".", xlab="log-ratio", ylab=expression(-log[10]~p))
abline(h=2)

#The t-statistic measures the difference in mean divided by an estimate of the variance. When there are few replicates, the variance is not well estimated and the t-statistic can perform poorly. Alternative statistics that borrow information across genes often provide better results. An instance of such a modified t-statistic is based on an empirical Bayes moderation approach, implemented in the eBayes function in the limma package.
library("limma")

#we have to put the scores as ordinal variables before including them in the design for the limma package

#ASP_SCORE

ASPresistance <- ordered(ALLf$ASP_score, levels=c("sensitive", "intermediate", "resistant"))
ASPresistance <- ordered(ALLf$ASP_score, levels=c("1", "2", "3"))
ALLf$ASP_score
ASPresistance

design = model.matrix(~ASPresistance)
ALLlim = lmFit(ALLf, design)
ALLeb = eBayes(ALLlim)
ALLeb

#Various approaches have been suggested for dealing with the multiple testing problem. The multtest package provides implementations of many of the options. Alternatively, the function topTable in the limma package provides multiple testing adjustment methods, including Benjamini and Hochberg’s false discovery rate (FDR), simple Bonferroni correction, and several others. The following example lists the top ten genes and creates a report. Here, coef=2 specifies that we care for the second coefficient in the linear model fit, that is, the between-group difference, as the parameter of interest. The first coefficient is the intercept. The parameter n indicates how many genes should be selected.
tab = topTable(ALLeb, coef=2, adjust.method="BH", n=25)
tab
colnames(tab)
rownames(tab)

#modified method to get the list of the top 25 gene names
genenames = as.character(rownames(tab))
genenames
```

```
#Annotation
#First we load the annotate package
library("annotate")

#Find out which metadata package we need,
annotation(ALLf)

#and load it
biocLite("hgu133a.db")
library("hgu133a.db")

#Now we can retrieve more annotation information about the genes that interest us, for example, their EntrezGene ID and gene symbol.

LLALLeb=getEG(genenames,"hgu133a.db")
symALLeb=getSYMBOL(genenames,"hgu133a.db")

LLALLeb
symALLeb

Finally, make a table of the top 25 significantly associated genes
tableADP <- data.frame(genenames, LLALLeb, symALLeb)
tableADP
summary(table)

#Making Volcano Plots

plotMA(ALLeb)
volcanoplot(ALLeb, coef=2, highlight=25)

#DNR_score

DNRresistance <- ordered(ALLf$DNR_score, levels=c("1", "2", "3"))
DNRresistance[is.na(DNRresistance)] <- "1"
DNRresistance

design = model.matrix(~DNRresistance)
ALLlim = lmFit(ALLf, design)
ALLeb = eBayes(ALLlim)
ALLeb

tab = topTable(ALLeb, coef=2, adjust.method="BH", n=25)

#modified method to get the list of the top 25 gene names
genenames = as.character(rownames(tab))
```

```
genenames

#Now we can retrieve more annotation information about the genes that interest us, for example,
their EntrezGene ID and gene symbol.
LLALLeb=getEG(genenames,"hgu133a.db")
symALLeb=getSYMBOL(genenames,"hgu133a.db")

Finally, make a table of the top 25 significantly associated genes
tableADP <- data.frame(genenames, LLALLeb, symALLeb)
tableADP

#Making Volcano Plots
plotMA(ALLeb)
volcanoplot(ALLeb, coef=2, highlight=25)

#PRED_score

PREDresistance <- ordered(ALLf$PRED_score, levels=c("1", "2", "3"))
PREDresistance[is.na(PREDresistance)] <- "1"
PREDresistance

design = model.matrix(~PREDresistance)
ALLlim = lmFit(ALLf, design)
ALLeb = eBayes(ALLlim)
ALLeb

tab = topTable(ALLeb, coef=2, adjust.method="BH", n=25)

#modified method to get the list of the top 25 gene names
genenames = as.character(rownames(tab))
genenames

#Now we can retrieve more annotation information about the genes that interest us, for example,
their EntrezGene ID and gene symbol.
LLALLeb=getEG(genenames,"hgu133a.db")
symALLeb=getSYMBOL(genenames,"hgu133a.db")

Finally, make a table of the top 25 significantly associated genes
tableADP <- data.frame(genenames, LLALLeb, symALLeb)
tableADP

#Making Volcano Plots
plotMA(ALLeb)
volcanoplot(ALLeb, coef=2, highlight=25)

#VCR_score
```

```
VCRresistance <- ordered(ALLf$VCR_score, levels=c("1", "2", "3"))
VCRresistance[is.na(VCRresistance)] <- "1"
VCRresistance

design = model.matrix(~VCRresistance)
ALLlim = lmFit(ALLf, design)
ALLeb = eBayes(ALLlim)
ALLeb

tab = topTable(ALLeb, coef=2, adjust.method="BH", n=25)

#modified method to get the list of the top 25 gene names
genenames = as.character(rownames(tab))
genenames

#Now we can retrieve more annotation information about the genes that interest us, for example,
their EntrezGene ID and gene symbol.
LLALLeb=getEG(genenames,"hgu133a.db")
symALLeb=getSYMBOL(genenames,"hgu133a.db")

Finally, make a table of the top 25 significantly associated genes
tableADP <- data.frame(genenames, LLALLeb, symALLeb)
tableADP

#Making Volcano Plots
plotMA(ALLeb)
volcanoplot(ALLeb, coef=2, highlight=25)

#http://bioinformatics.knowledgeblog.org/2011/06/20/analysing-microarray-data-in-
bioconductor/
#http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual#TOC-Analysis-of-
Differentially-Expressed-Genes
#https://www.bioconductor.org/help/workflows/arrays/

#Running some QC on the CHIP

source("https://bioconductor.org/biocLite.R")
biocLite("simpleaffy")
library(simpleaffy)

nvals
nvals.qc <- qc(nvals)

######################################################################################
######################################################################################
```

ANOVA and Regression

https://www.theanalysisfactor.com/why-anova-is-really-linear-regression-notation/

In this article, we're going to focus on notation, as that is the most fundamental part.

Here is the typical regression model with two predictors. Let's assume one predictor is a treatment variable (treatment vs. control) and the other is some other kind of grouping variable, like whether subjects are bilingual or monolingual:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \varepsilon_i$$

where:

Y_i = Response for individual i
β_0 = Intercept—the mean of Y when all Xs=0
β_j = Coefficient of $X_{j,}$ the jth predictor
X_{ji} = jth predictor for individual i
ε_i = Residual for individual i

And here is the typical ANOVA model with two predictors:

$$Y_{ijk} = \mu + \alpha_j + \beta_k + \varepsilon_{ijk}$$

Y_{ijk} = Response for individual i, who is in treatment j and group k
μ = the grand mean

ε_{ijk} = Residual for individual i who is in treatment j and group k
α = effect of treatment

β = effect of group

In the ANOVA model, the predictors are often called factors or grouping variables, but just like in the regression case, we *can* call them the more generic "predictors." For simplicity, there is no interaction, though that would be simple to add to both models.
Y, the response variable, is on the left hand side of both equations. The subscript i indicates that each case has an individual value of Y.

Likewise, both models have an error term, denoted by ε. This too has an i subscript because there is one value per case.

In between these on both models are three terms and these don't look the same. At all.

In the linear regression model, we use Xs to indicate the value of the predictor variables. This is super flexible — if X is numerical, we just plug in the numerical values of X. If X is categorical, we simply indicate which group someone was in with coded values of X1.

The simplest and most common would have a 1 for the treatment group and a 0 for the control group.

β_j measures the effect on Y of the treatment effect.

Because ANOVA assumes that all the predictors are categorical (aka factors or grouping variables), those predictors have a limited number of values. These are values like: treatment vs. control group. Two values.

Because of the limited number of values, the ANOVA model uses subscripts to indicate if someone is in the treatment or control group. Subscript j would have values of 1 for the treatment group and 1 for the control.

α measures the effect on Y of the treatment effect.

Even those these X values aren't written directly into the ANOVA model, they exist. Your software is actually creating those X values for you when you indicate that X is categorical.

One other term that looks different, but is essentially the same thing: the constant. In the regression model, this is called the intercept and denoted β_0 and in the ANOVA model, this is called the grand mean and denoted μ.

Generalized Linear Models Part 1

https://www.theanalysisfactor.com/r-tutorial-glm1/

Ordinary Least Squares regression provides linear models of continuous variables. However, much data of interest to statisticians and researchers are not continuous and so other methods must be used to create useful predictive models.

The glm() command is designed to perform generalized linear models (regressions) on binary outcome data, count data, probability data, proportion data and many other data types.Below, we explore the use of R's glm() command on one such data type. Let's take a look at a simple example where we model binary data.

In the mtcars data set, the variable vs indicates if a car has a V engine or a straight engine. We want to create a model that helps us to predict the probability of a vehicle having a V engine or a straight engine given a weight of 2100 lbs and engine displacement of 180 cubic inches.

First we fit the model:

We use the glm() function, include the variables in the usual way, and specify a binomial error distribution, as follows:

```
model <- glm(formula= vs ~ wt + disp, data=mtcars, family=binomial)


summary(model)


Call: glm(formula = vs ~ wt + disp, family = binomial, data = mtcars)
```

We see from the estimates of the coefficients that *weight* influences *vs* positively, while *displacement* has a slightly negative effect.

The model output is somewhat different from that of an ordinary least squares model. I will explain the output in more detail in the next article, but for now, let’s continue with our calculations.

Remember, our goal here is to calculate a predicted probability of a V engine, for specific values of the predictors: a weight of 2100 lbs and engine displacement of 180 cubic inches. To do that, we create a data frame called newdata, in which we include the desired values for our prediction.

```
newdata = data.frame(wt = 2.1, disp = 180)
```

Now we use the predict() function to calculate the predicted probability. We include the argument type=”response” in order to get our prediction.

```
predict(model, newdata, type="response") 0.2361081
```

The predicted probability is 0.24.

Generalized Linear Models Part 2

we saw how to create a simple Generalized Linear Model on binary data using the glm() command. We continue with the same glm on the mtcars data set(modeling the vs variable on the weight and engine displacement).

```
model <- glm(formula= vs ~ wt + disp, data=mtcars, family=binomial)
```

```
summary(model)


Call: glm(formula = vs ~ wt + disp, family = binomial, data = mtcars)
```

We see that *weight* influences *vs* positively, while displacement has a slightly negative effect. We also see that the coefficient of weight is non-significant ($p > 0.05$), while the coefficient of displacement is significant. Later we will see how to investigate ways of improving our model. In fact, the estimates (coefficients of the predictors *weight* and *displacement*) are now in units called logits. We will define the logit in a later blog.

Deviance

We see the word Deviance twice over in the model output. Deviance is a measure of goodness of fit of a generalized linear model. Or rather, it's a measure of badness of fit–higher numbers indicate worse fit.
R reports two forms of deviance – the null deviance and the residual deviance. The null deviance shows how well the response variable is predicted by a model that includes only the intercept (grand mean).

For our example, we have a value of 43.9 on 31 degrees of freedom. Including the independent variables (weight and displacement) decreased the deviance to 21.4 points on 29 degrees of freedom, a significant reduction in deviance.

The Residual Deviance has reduced by 22.46 with a loss of two degrees of freedom.

Fisher Scoring

What about the Fisher scoring algorithm? Fisher's scoring algorithm is a derivative of Newton's method for solving maximum likelihood problems numerically.

For model1 we see that Fisher's Scoring Algorithm needed six iterations to perform the fit.

This doesn't really tell you a lot that you need to know, other than the fact that the model did indeed converge, and had no trouble doing it.

Information Criteria

The Akaike Information Criterion (AIC) provides a method for assessing the quality of your model through comparison of related models. It's based on the Deviance, but penalizes you for making the model more complicated. Much like adjusted R-squared, it's intent is to prevent you from including irrelevant predictors.

However, unlike adjusted R-squared, the number itself is not meaningful. If you have more than one similar candidate models (where all of the variables of the simpler model occur in the more complex models), then you should select the model that has the smallest AIC.

So it’s useful for comparing models, but isn’t interpretable on its own.

Hosmer-Lemeshow Goodness of Fit

How well our model fits depends on the difference between the model and the observed data. One approach for binary data is to implement a Hosmer Lemeshow goodness of fit test.

To implement this test, first install the ResourceSelection package, a follows.

```
install.packages("ResourceSelection")
```

Then load the package using the library() function. The test is available through the hoslem.test() function.

```
library(ResourceSelection)

hoslem.test(mtcars$vs, fitted(model)) Hosmer and Lemeshow goodness of fit (GOF) test

data: mtcars$vs, fitted(model) X-squared = 6.4717, df = 8, p-value = 0.5945
```

Our model appears to fit well because we have no significant difference between the model and the observed data (i.e. the p-value is above 0.05).

As with all measures of model fit, we’ll use this as just one piece of information in deciding how well this model fits. It doesn’t work well in very large or very small data sets, but is often useful nonetheless.

Generalized Linear Models Part 3

We continue with the same glm on the mtcars data set (regressing the *vs* variable on the *weight* and *engine displacement*). Now we want to plot our model, along with the observed data.

Although we ran a model with multiple predictors, it can help interpretation to plot the predicted probability that *vs*=1 against each predictor separately. So first we fit a glmfor only one of our predictors, *wt*.

```
model_weight

summary(model_weight)

Call: glm(formula = vs ~ wt, family = binomial, data = mtcars)
```

To plot our model we need a range of values of *weight* for which to produce fitted values. This range of values we can establish from the actual range of values of *wt*.

```
range(mtcars$wt)  [1] 1.513 5.424
```

A range of *wt* values between 0 and 6 would be ideal. So we create a sequence of values between 0 and 6 in increments of 0.01. Joining such a large number of closely spaced points will give a smooth appearance to our model.

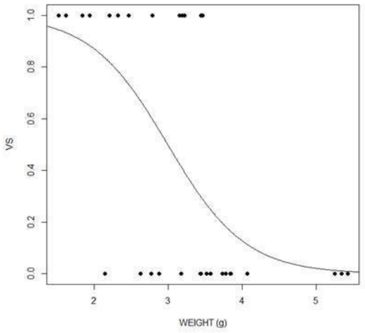
```
xweight <- seq(0, 6, 0.01)
```

Now we use the predict() function to create the model for all of the values of xweight.

```
yweight <- predict(model_weight, list(wt = xweight),type="response")
```

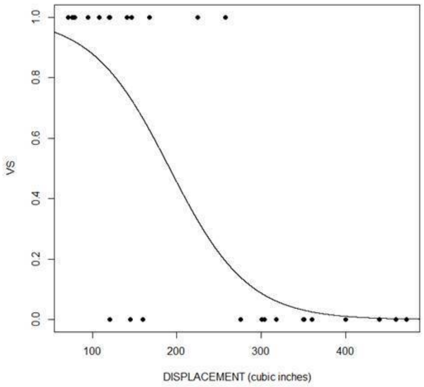
Now we plot.

```
plot(mtcars$wt, mtcars$vs, pch = 16, xlab = "WEIGHT (g)", ylab = "VS")

lines(xweight, yweight)
```



We can do the same for displacement.

```
model_disp

summary(model_disp)

Call: glm(formula = vs ~ disp, family = binomial, data = mtcars)

ydisp

plot(mtcars$disp, mtcars$vs, pch = 16, xlab = "DISPLACEMENT (cubic inches)", ylab = "VS")

lines(xdisp, ydisp)
```



We can see that for both predictors, there is a negative relationship between the probability that *vs*=1 and the predictor variable. As the predictor increases, the probability decreases.

Generalized Linear Models Part 4

As a reminder, Generalized Linear Models are an extension of linear regression models that allow the dependent variable to be non-normal. In our example for this week we fit a GLM to a set of education-related data.

Let’s read in a data set from an experiment consisting of numeracy test scores (numeracy), scores on an anxiety test (anxiety), and a binary outcome variable (success) that records whether or not the students eventually succeeded in gaining admission to a prestigious university through an admissions test.

We will use the glm() command to run a logistic regression, regressing success on the numeracy and anxiety scores.

```
attach(A)
names(A) [1] "numeracy" "anxiety"  "success"
head(A)
```

The variable ‘success’ is a binary variable that takes the value 1 for individuals who succeeded in gaining admission, and the value 0 for those who did not. Let’s look at the mean values of numeracy and anxiety.

```
mean(numeracy) [1] 10.722
mean(anxiety) [1] 13.954
```

We begin by fitting a model that includes interactions through the asterisk formula operator. The most commonly used link for binary outcome variables is the logit link, though other links can be used.

```
model1 <- glm(success ~ numeracy * anxiety, binomial)
```

glm() is the function that tells R to run a generalized linear model.
Inside the parentheses we give R important information about the model. To the left of the ~ is the dependent variable: success. It must be coded 0 & 1 for glm to read it as binary.
After the ~, we list the two predictor variables. The * indicates that not only do we want each main effect, but we also want an interaction term between numeracy and anxiety.
And finally, after the comma, we specify that the distribution is binomial. The default link function in glm for a binomial outcome variable is the logit. More on that below.
We can access the model output using summary().

```
summary(model1) Call: glm(formula = success ~ numeracy * anxiety, family = binomial)
```

The estimates (coefficients of the predictors – numeracy and anxiety) are now in logits. The coefficient of numeracy is: 1.94556, so that a one unit change in numeracy produces approximately a 1.95 unit change in the log odds (i.e. a 1.95 unit change in the logit).
From the signs of the two predictors, we see that numeracy influences admission positively, but anxiety influences survival negatively.

We can’t tell much more than that as most of us can’t think in terms of logits. Instead we can convert these logits to odds ratios. We do this by exponentiating each coefficient. (This means raise the value e –approximately 2.72–to the power of the coefficient. e^b).

So, the odds ratio for numeracy is:

$$\text{OR} = \exp(1.94556) = 6.997549$$

However, in this version of the model the estimates are non-significant, and we have a non-significant interaction. Model1 produces the following relationship between the logit (log odds) and the two predictors:

$$\text{logit}(p) = 0.88 + 1.95 * \text{numeracy} - 0.45 * \text{anxiety} - 1.0 * \text{interaction term}$$

The output produced by glm() includes several additional quantities that require discussion. We see a z value for each estimate. The z value is the Wald statistic that tests the hypothesis that the estimate is zero. The null hypothesis is that the estimate has a normal distribution with mean zero and standard deviation of 1. The quoted p-value, P(>|z|), gives the tail area in a two-tailed test.

For our example, we have a Null Deviance of about 68.03 on 49 degrees of freedom. This value indicates poor fit (a significant difference between fitted values and observed values). Including the independent variables (numeracy and anxiety) decreased the deviance by nearly 40 points on 3 degrees of freedom. The Residual Deviance is 28.2 on 46 degrees of freedom (i.e. a loss of three degrees of freedom).

Generalized Linear Models Part 5

We used the glm() command in R to fit a logistic model with binomial errors to investigate the relationships between the numeracy and anxiety scores and their eventual success.
Now we will create a plot for each predictor. This can be very helpful for helping us understand the effect of each predictor on the probability of a 1 response on our dependent variable.
We wish to plot each predictor separately, so first we fit a separate model for each predictor. This isn't the only way to do it, but one that I find especially helpful for deciding which variables should be entered as predictors.

```
model_numeracy <- glm(success ~ numeracy, binomial)

summary(model_numeracy)
```

We do the same for anxiety**.**

```
model_anxiety <- glm(success ~ anxiety, binomial)

summary(model_anxiety)
```

Now we create our plots. First we set up a sequence of length values which we will use to plot the fitted model. Let’s find the range of each variable.

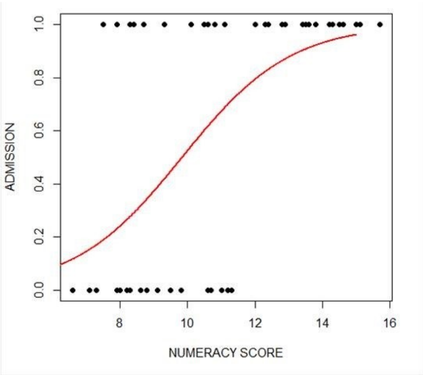
```
range(numeracy)

range(anxiety)
```

Given the range of both numeracy and anxiety. A sequence from 0 to 15 is about right for plotting numeracy, while a range from 10 to 20 is good for plotting anxiety.

```
xnumeracy <-seq (0, 15, 0.01)

ynumeracy <- predict(model_numeracy, list(numeracy=xnumeracy),type="response")
```

Now we use the predict() function to set up the fitted values. The syntax type = “response” back-transforms from a linear logit model to the original scale of the observed data (i.e. binary).

```
plot(numeracy, success, pch = 16, xlab = "NUMERACY SCORE", ylab = "ADMISSION")

lines(xnumeracy, ynumeracy, col = "red", lwd = 2)
```

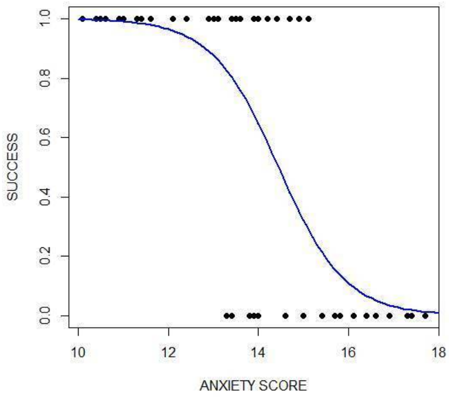


The model has produced a curve that indicates the probability that success = 1 to the numeracy score. Clearly, the higher the score, the more likely it is that the student will be accepted.

Now we plot for anxiety.

```
xanxiety <- seq(10, 20, 0.1)
```

```
yanxiety <- predict(model_anxiety, list(anxiety=xanxiety),type="response")

plot(anxiety, success, pch = 16, xlab = "ANXIETY SCORE", ylab = "SUCCESS")

lines(xanxiety, yanxiety, col= "blue", lwd = 2)
```



Clearly, those who score high on anxiety are unlikely to be admitted, possibly because their admissions test results are affected by their high level of anxiety.

Fixed Effects vs Mixed Effect Models

Analytical issues in GRTS

Let Y i:k:l the outcome observation for the ith participant nested within the kth group nested in the lth condition. Since the groups are non-random, we have a portion of the variation in the outcome due to the group. We call this the ICC (intra class correlation: between group variance / [between group variance + within group variance])

And the group variance is $\sigma_{yg} = (\sigma_e/m) + \sigma_{g:c} = \sigma_y/m\ (1 + (m-1)*ICC)$
The condition variance is $\sigma_{yc} = \sigma_{yg} / g = \sigma_y /mg\ (1 + (m-1)*ICC)$
σ_y /mg would be the variance in the outcome if all observation within the groups were independent

(1 + (m-1)*ICC) if the variance inflation factor or design effect, which is the adjustment due to the positive correlation in the outcomes.

If the ICC > 0 then $\sigma_{yc} > \sigma_y$ so by assuming independence, our standard errors will be too small, leading to larger test statistics and false positives (statistically significant results when we shouldn't have one)

Planning the analysis

Fundamentals:

- Use the research question to determine the primary and secondary endpoints
- Use the design to determine the # of conditions, # of levels per conditions, a priori group matching variables, and main effect vs factorial analyses.
- The endpoint can be continuous, categorical, count, rate or survival data
- You can use regression adjustment of covariates and model time variables to reduce bias and improve precision

Threats to the validity of the analysis
Misspecification of the model: can be due to missing measurable sources of variation (ignoring the clustering effect) or to using the wrong correlation structure for group correlation (not modeling the cluster as a RV)

GLM assumptions

- Normally distributed errors (ok if we have similar sample size of groups per condition)
- Homogeneity of errors (ok if we have similar sample size of groups per condition)
- Independence of errors (Very serious violation, because our SE is too small when we ignore the positive ICCs in our data due to clustering)

To protect the internal validity in our study, we identify all possible sources of random variation in our data, check the variance assumptions using model fits, and use robust methods in the analysis.

Statistical models

Fixed effect: we want to draw inference on specific levels of the variables on the outcome
Random effect: want to generalize the inference to a larger population (group level variable)

- General Linear Model: intercept + slope + residual
- Generalized linear model: GLM for non-normal outcomes (use the Gaussian, binomial and poission distribution in conjunction with identity, logit and log link functions respectively)
- General Linear mixed model: GLM for tow or more random variables (clusters or repeated measures
- Generalized linear mixed model: GLMM for non-normal outcomes (use the Gaussian, binomial and poission distribution in conjunction with identity, logit and log link functions respectively)

Estimation of outcomes

- Ordinary least squares: minimize the sum of squares deviation between data and predicted values in the data: best for independent residual errors
- Maximum likelihood: maximize the probability that the predicted values match the data. Can be used for both fixed and random effects. The restricted ML does not estimate the

fixed and random effects at the same time, rather it estimates fixed effects first then uses the to inform random effects estimation.

- Parametric methods make assumptions on the distribution of the standard errors in our model, but non parametric methods do not.
- Sampling distribution and DF: find the post hoc condition, find the ICC, calculate the VIFF then make the sampling distribution and use the t-test with c(g-1) df to estimate your statistical significance.
- Units of analysis in the model: there are units of assignment, intervention and observations in GRTS. The units of analysis for an effect are those and only those for which this effect is assessed against the variation among these units.

Overview of GRTs

The unit of randomization is a group, not individuals within a group

The unit of treatment application can either be the whole groups (limits our ability for sub-analyses) or all individuals within the group (difference from RCT's)

If you randomize within a cluster, you have a multi-site RCT, whereas randomization of the whole cluster to either treatment arms: this means that for multi-site RCTs we have the option of using either the individual effect or group level effect in the analysis, whereas for GRTs we can only estimate the group level effect

Group specific effects are important to estimate because we need to account for nesting (group level error + individual level error)

When we talk about nesting, the strata used within each treatment arm should be comparable themselves (I.e, you cannot compare three school within one city to three schools in three cities, because the true cluster in this case is the city, which is not comparable across treatment arms.)

The measurements are done at the individual level

The analysis is also done at the individual level, while taking the clustering effect as a specification into the model

In ecological GRTs, we average the individual measurements we take at the group level : Y= ax + b + U (group error) + e (individual error) becomes Y = aX + b + U (Group error)

There are two sample size considerations in GRTs: The total # of clusters (K), and the total number of individuals per cluster (increasing the number of groups is more statistically efficient than increasing the number of individuals per group)

You can increase the power of your analysis by adjusting for baseline values of your outcome, however, you add another correlation (pre-post) into consideration

The clusters need to have identifiable common characteristics, but watch out for a large cluster effect (the clusters in one treatment arm have a shared characteristic, whereas those in the other arm do not, resulting in a lack of exchangeability (ie comparing three schools from the same city to three schools from three cities)

During the data collection, you can use complete enumeration of all individuals in the cluster, or use simple random samples

Overview of GRTs continued

The ICC (Intra-class correlation) is an estimate of the proportion of the total variance due to the group. It is a measure of clustering, and is calculated by ICC= (Between group variance)/(Between group variance + Within Group Variance). The within group variance in this case is the individual error, and the total variance is the sum of the two.

In GRT's, because the treatment is at the group level, we need to determine is the treatment effect is attributable to individual variations, or the group itself: this means that even small ICC can have a large impact on the inference for our effect.

Naturally, there is a covariance (or correlation) between our group level error and our individual error, when we look at any single group (That is, knowing something about the group can tell us something about its individual members and vice versa). Using randomization in GRT's we break that link and with enough groups randomized, the covariance between the group error and individual error becomes 0.

In order to estimate our effect, we use B/SeB for our effect estimate: B is the signal, and Se is the noise around the signal. If SeB is too small, we will pick up larger signals than we should (false positive). This can happen if we only consider the individual level error in GRT's, and must be corrected by adding the group level variability using our ICC-> This is the design effect, and is possible because there is no covariance between U and e thanks to the randomization.

Ignoring the cluster specification means ignoring the ICC (the proportion of the variance attributable to clustering i.e. between group variance / total variance): Even though the ICC is small, it has large impacts on the statistical inference.

GEE is better to assess between group comparisons, whereas GLMM is better suited for within group comparisons.

ITT is better for evaluating the biological/behavioral effect of an intervention(mechanism/Policy), whereas ATT is better suited for effect in practice (implementation in current practical settings).

Interference, Contamination and Spillover effects in GRT's: Interference is when we have spillover within a group ie when we assign treatment to some members within a group, its effects will spread to other members within the group (this is good!)

Spillover is when we have interference across groups within the same treatment arm. This is not desirable

Contamination is when we have interference across treatment arms. This is the worst case scenario of interference as it limits our ability to identify a treatment effect.

ANOVA comparisons in GRT's

ANOVA	σ^2 (Variance)	Meaning
TX	Effect of interest	The true GRT Comparison is between TX and groups (ie group means across txt conditions) This drives the df in the analysis for the treatment effect = (#groups -1)*(#conditions)
Group	Error attributable to the group	
Individual	Error attributable to the individuals	We usually ignore the individual effect in the analysis, but more individuals increases the precision of the group means

Posttest analysis in GRT's

Using simple regression (treating our observations as independent) will yield a correct effect estimate despite being misspecified. What will be wrong is the small standard error (variance over square root of n) because it will be deflated. This leads to a bigger test statistic (larger signal or false positive) and a smaller confidence interval than when the clustering is taken into account.

In more practical terms, when you assume independent assumptions, you use too many degrees of freedom (Z distribution with infinite degrees of freedom) so it is very easy to have a significant value, on top of having a deflated standard error. So even if we specify the correlation, we also have to specify the t-distribution in our model (because we rarely have more than 30 groups per condition to make the Z distribution applicable) so that we properly interpret our results.

BLUE vs BLUP

How to evaluate group level (level 2) effects in our model:

The first approach we might try is to include the group level effect as indicator variables for each clusters in our model. In this case, the coefficient estimates for each indicator variables (cluster) is called the Best Linear Unbiased Estimate (BLUE).

- This accounts for clustering & uses the proper df for the clusters, but it is incorrect for GRTs because the indicator variables are based on the assumption that individual observations within each cluster are independent-> this approach would work for a multi-site RCT, because we randomize at the individual level within each site/cluster, removing any correlation between individual participants within the cluster.

Another approach would involve specifying the cluster effects as a random variable: The coefficient estimates given by the random variable are called the Best Linear Unbiased Predictors (BLUP).

$Y = \alpha + \beta*X + Z*\mu + \varepsilon$ (ε is the individual error, and μ is the group level error)

- How are these different from BLUE? They are calculated differently: In brief, the model fits a grand mean for the effect due to the cluster as a fixed effect, then it superimposes on that grand mean the individual values for each clusters to create a normal distribution for our random variable, centered around the grand mean it just calculated.
- if the value for a particular cluster is very far from the center of the distribution (towards either tail of the distribution), it gets adjusted or "smushed" towards the center of the distribution (this phenomenon "shrinkage"). Values which are already close to the center of the distribution don't get adjusted as much.
- Using BLUP in our model costs only one degree of freedom (since it is a random variable), but it does cost us one additional assumption, that we are drawing our random variable based on a random distribution.

Pre-Post Analyses

Fixed effects: effect of the variable is replicable

Random effects: effect comes from a super population so it will not be replicable
Choosing your testing options: posttest only would be useful to avoid social desirability bias
Pre-post analysis involves making pre measurements prior to randomization, and then calculate the difference (txt – ctl) in differences (post – pre): by using the control arm as a comparison group, we remove the natural change in trend over time (because it should be the same across condition for the outcome)

- Y2 = cond + Y1 (Baseline adjusted)
- Y2-Y1 = cond (Differences in differences)
- Yi = Condi (Repeated measured analysis)

The pre-post analysis compares the results from before the intervention (preferably taken after randomization) to those after the intervention.
Baseline adjustment methods (Y2 = cond + Y1) give us the difference in the predicted value of the outcome when everyone's baseline value is the same ("adjusted" for baseline).
Change in score method (Delta, or Y2-Y1 = condition) gives the difference in the person change on average between treatment and control arms.
Member cohort GRT follow the same person at baseline and follow up, whereas member cross section analyses follow the same group/clusters, but measures different individuals at each time point
Degrees of freedom in a pre-post analysis: by adding time, we go from c(g-1) to (t-1)*c*(g-1). As you can notice, for a simple prepost analysis, the degrees of freedom are the same as a post test only analysis.

Understanding Random Effects in Mixed Models

https://www.theanalysisfactor.com/understanding-random-effects-in-mixed-models/

In fixed-effects models (e.g., regression, ANOVA, generalized linear models), there is only one source of random variability. This source of variance is the random sample we take to measure our variables.
It may be patients in a health facility, for whom we take various measures of their medical history to estimate their probability of recovery. Or random variability may come from individual students in a school system, and we use demographic information to predict their grade point averages.

We call the variability across individuals' "residual" variance (in linear models, this is the estimate of σ^2, also called the mean squared error). It's the variability that was unexplained by the predictors in the model (the fixed effects).

Multiple Sources of Random Variability

Mixed effects models—whether linear or generalized linear—are different in that there is more than one source of random variability in the data.
In addition to patients, there may also be random variability across the doctors of those patients. In addition to students, there may be random variability from the teachers of those students.

Some doctors' patients may have a greater probability of recovery, and others may have a lower probability, even after we have accounted for the doctors' experience and other measurable traits. Some teachers' students will have higher GPAs than other teachers' students, even after we account for teaching methods.

Random Effects: Intercepts and Slopes

We account for these differences through the incorporation of random effects. Random intercepts allow the outcome to be higher or lower for each doctor or teacher; random slopes allow fixed effects to vary for each doctor or teacher.
What do these random effects mean? How do we interpret them? We usually talk about them in terms of their variability, instead of focusing on them individually.

Using the patient/doctor data as an example, this allows us to make "broad level" inferences about the larger population of patients, which do not depend on a particular doctor. In other words, we can now incorporate (instead of ignore) doctor-to-doctor variability in patient recovery, and improve our ability to describe how fixed effects relate to outcomes.

Variance of Random Effects

We can also talk directly about the variability of random effects, similar to how we talk about residual variance in linear models.

There is no general measure of whether variability is large or small, but subject-matter experts can consider standard deviations of random effects relative to the outcomes.

For example, if teacher-averaged GPAs only vary from the overall average with an SD of 0.02 GPA points, the teachers may be considered rather uniform; however, if teacher-averaged GPAs varied from the overall average with an SD of 0.5 GPA points, it would seem as if individual teachers could make a large difference in their students' success.

(For an additional way to look at variability in linear mixed effects models, check out Karen's blog post on ICC here.)

Individual random effects

Finally, we *can* talk about individual random effects, although we usually don't. This was not the original purpose of mixed effects models, although it has turned out to be useful in certain applications. Software programs do provide access to the random effects (best linear unbiased predictors, or BLUPs) associated with each of the random subjects.
BLUPs are the differences between the intercept for each random subject and the overall intercept (or slope for each random subject and the overall slope). In some software, such as SAS, these are accompanied by standard errors, t-tests, and p-values.

In the case of the patient/doctor data set (assuming no random slopes for easier interpretation), a small p-value for an individual doctor's random intercept would indicate that the doctor's typical patient recovery probability is significantly different from an average doctor's typical patient recovery probability.

These standard errors and p-values are adjusted so that they account for all of the fixed effects in the model as well as the random variability among patients. Clearly, this information could be of interest to the doctor's place of work, or to a patient who is choosing a doctor.

Here is an example analysis for Linear Mixed Effect Models from the ASMIC study
# otu analysis
# Started 11-Dec-2017

```
#one useful thing to do is set the folder that R copies and draws files from. You can either do it from under file, or change the working directory using a command

#Set working directory for download: you have to use forward slashes in R pathways
getwd()
setwd("C:/Users/guillaumeo/Desktop/AFFY_FINAL")

##read.table wants to return a data.frame, which must have an element in each column. Therefore R expects each row to have the same number of elements and it doesn't fill in empty spaces by default. Try read.table("/PathTo/file.csv" , fill = TRUE ) to fill in the blanks.

read.table("AnalysisData_code.txt", header=TRUE, fill= TRUE)

test <- read.table("AnalysisData_code.txt",  fill= TRUE)

# The test dataset with the analysis code of ALL dataset will have 25 rows and 521 columns

dim(test)

# clear memory -----------------------------------------------------------

rm(list = ls())

# libraries --------------------------------------------------------------

library(data.table)
library(readxl)
library(readr)
library(ggplot2)

# read in data -----------------------------------------------------------

CountTB <- read_excel("N:/BDAC/Projects/menk/prizment/otu/Aspirin OTU Taxonomy.xlsx", sheet = "Genera", n_max = 351)

ProportionTB <- read_excel("N:/BDAC/Projects/menk/prizment/otu/Aspirin OTU Taxonomy.xlsx", sheet = "Genera", skip = 351, n_max = 351)

TB <- read_excel("N:/BDAC/Projects/menk/prizment/otu/Aspirin OTU Taxonomy.xlsx", sheet = "Genera", skip = 703, n_max = 243)

# make a data.table ------------------------------------------------------
```

```
CountM <- as.matrix(CountTB)
CountMT <- t(CountM[, -1])
colnames(CountMT) <- CountM[, 1]
value <- rownames(CountMT)
CountDM <- cbind(value, CountMT)
CountDT <- data.table(CountDM)

DT <- data.table(TB)

# Randomization and corrections ------------------------------------------

# read in randomization assignment
Group.df <- read_excel("N:/BDAC/Projects/menk/prizment/out/randomization/AsmicRandomizationForForm.xlsx")
AgeSex.df <- read_csv("N:/BDAC/Projects/menk/prizment/data/ASMIC_DATA_2018-02-19_1125.csv")
names(AgeSex.df)[grep("stool", names(AgeSex.df))]
AgeSex.df <- AgeSex.df[, c("record_id_es", "random_bottle_number", "clinic_collect_kit", "clinic_collect_kit2", "stool_1_kit", "stool_2_kit", "stool_3_kit", "stool_4_kit2", "stool_5_kit2")]

# correct the randomization error per email from Jen Stromberg 22-Feb-2017

# As per our conversation this morning, I'm sending you the information about the REDCap issue I encountered.  I issued bottle number 0010 to record ID 180 (participant A0010) on 2/16/17, but neglected to hit the randomize button in that record in REDCap.  I was unaware that I had not completed the randomization in the system for this individual until I issued bottle number 0011 to record ID 409 (participant A0011) on 2/21/17.  When I hit the randomize button in record 409, the bottle number that came up was 0010, which I knew I had dispensed to A0010 the previous week.  I aborted the randomization form without saving at that time and went back into record 180.  I saw that the randomization had not been completed, so thinking that since I had aborted randomization on record 409, that when I randomized record 180, the appropriate bottle (0010) would be assigned.  Bottle 0011 was assigned and I realized that once you hit the randomize button, there's no going back even if the form entry is aborted.  The problem now is that although I dispensed the proper bottles to the proper participants in the proper order, REDCap shows the wrong bottle number for those 2 individuals because of my careless mistake.  Bottle 0010 should be associated with record ID 180 (A0010) and bottle 0011 should be associated with record ID 409 (A0011).  Please let me know if the bottle numbers can be switched in the system to correct this discrepancy.

subset(AgeSex.df, record_id_es %in% "180")
subset(AgeSex.df, record_id_es %in% "409")

AgeSex.df$random_bottle_number[AgeSex.df$record_id_es %in% "180"] <- "0010"
AgeSex.df$random_bottle_number[AgeSex.df$record_id_es %in% "409"] <- "0011"

Stool.df <- subset(AgeSex.df, !is.na(random_bottle_number), select = c("record_id_es",
"stool_1_kit", "stool_2_kit", "stool_3_kit", "stool_4_kit2", "stool_5_kit2"))

# merge and creat randomization and bottle information
Randomization.df <- merge(Group.df, AgeSex.df, by.x = "Bottle_Number", by.y =
"random_bottle_number")

# clean up bottle kit id in Randomization
Randomization.df$clinic_collect_kit <- gsub(" ", "", Randomization.df$clinic_collect_kit)
Randomization.df$clinic_collect_kit2 <- gsub(" ", "", Randomization.df$clinic_collect_kit2)

Randomization.df$stool_1_kit <- gsub(" ", "", Randomization.df$stool_1_kit)
Randomization.df$stool_2_kit <- gsub(" ", "", Randomization.df$stool_2_kit)
Randomization.df$stool_3_kit <- gsub(" ", "", Randomization.df$stool_3_kit)
Randomization.df$stool_4_kit2 <- gsub(" ", "", Randomization.df$stool_4_kit2)
Randomization.df$stool_5_kit2 <- gsub(" ", "", Randomization.df$stool_5_kit2)

# hard code corrections
Randomization.df$stool_1_kit[35] <- "FK7058"
Randomization.df$stool_4_kit2[48] <- "FK7130"

Randomization.df$stool_1_kit <- gsub("FK", "", Randomization.df$stool_1_kit)
Randomization.df$stool_2_kit <- gsub("FK", "", Randomization.df$stool_2_kit)
Randomization.df$stool_3_kit <- gsub("FK", "", Randomization.df$stool_3_kit)
Randomization.df$stool_4_kit2 <- gsub("FK", "", Randomization.df$stool_4_kit2)
Randomization.df$stool_5_kit2 <- gsub("FK", "", Randomization.df$stool_5_kit2)

# corrections to stool kit numbers
Randomization.df$stool_4_kit2[Randomization.df$stool_4_kit2 %in% "refused"] <- NA
Randomization.df$stool_5_kit2[Randomization.df$stool_5_kit2 %in% "refused"] <- NA

# Merge randomization with stool data -------------------------------------

DT.l <- melt(Randomization.df, id.vars = c("Bottle_Number", "Contents"), measure.vars =
c("stool_1_kit", "stool_2_kit", "stool_3_kit", "stool_4_kit2", "stool_5_kit2"))
StoolDF <- merge(DT.l, DT, by.x = "value", by.y = "X__1", all = TRUE)
StoolDT <- data.table(StoolDF)

StoolDT[, visit := factor(variable, levels = c("stool_1_kit", "stool_2_kit", "stool_3_kit",
"stool_4_kit2", "stool_5_kit2"), labels = 1:5)]

setkey(StoolDT, Contents, Collection, value)
```

```
# Merge in Weights -------------------------------------------------------

names(CountDT) <- paste0(names(CountDT), ".Count")
StoolDT <- merge(StoolDT, CountDT, by.x = "value", by.y = "value.Count")

# add: Prevotella, Bacteroides, Barnesiella, Enterococcus, Escherichia/Shigella,
Faecalibacterium, Dialister, Solobacterium, Eubacterium
StoolDT.w <- dcast(StoolDT, Bottle_Number + Contents ~ Collection, value.var =
c("Fusobacterium", "Parabacteroides", "Parvimonas", "Ruminococcus", "Streptococcus",
"Porphyromonas", "Parvimonas", "Fusobacterium.Count", "Parabacteroides.Count",
"Parvimonas.Count", "Ruminococcus.Count", "Streptococcus.Count", "Porphyromonas.Count",
"Parvimonas.Count", "Prevotella", "Bacteroides", "Barnesiella", "Enterococcus",
"Escherichia/Shigella", "Faecalibacterium", "Dialister", "Solobacterium", "Eubacterium",
"Prevotella.Count", "Bacteroides.Count", "Barnesiella.Count", "Enterococcus.Count",
"Escherichia/Shigella.Count", "Faecalibacterium.Count", "Dialister.Count",
"Solobacterium.Count", "Eubacterium.Count"))

# mixed effects logistic regression with weights ---------------------------------------

library(lme4)
StoolDT[, collection.factor := factor(Collection)]
# fre.Fusobacterium <- glmer(Fusobacterium / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Fusobacterium.Count), family = "binomial")
# summary(fre.Fusobacterium)
fre.Parabacteroides <- glmer(Parabacteroides / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Parabacteroides.Count), family = "binomial")
summary(fre.Parabacteroides)
# fre.Parvimonas <- glmer(Parvimonas / 100 ~ Contents*collection.factor + (1 | Bottle_Number),
data = StoolDT, subset = Collection %in% c(1, 2, 3), weights = as.numeric(Parvimonas.Count),
family = "binomial")
# summary(fre.Parvimonas)
fre.Ruminococcus <- glmer(Ruminococcus / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Ruminococcus.Count), family = "binomial")
summary(fre.Ruminococcus)
fre.Streptococcus <- glmer(Streptococcus / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Streptococcus.Count), family = "binomial")
summary(fre.Streptococcus)
# fre.Porphyromonas <- glmer(Porphyromonas / 100 ~ Contents*Collection + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Porphyromonas.Count), family = "binomial")
# summary(fre.Porphyromonas)
# add: Prevotella, Bacteroides, Barnesiella, Enterococcus, Escherichia/Shigella,
Faecalibacterium, Dialister, Solobacterium, Eubacterium
fre.Prevotella <- glmer(Prevotella / 100 ~ Contents*collection.factor + (1 | Bottle_Number), data
= StoolDT, subset = Collection %in% c(1, 2, 3), weights = as.numeric(Prevotella.Count), family
= "binomial")
summary(fre.Prevotella)
fre.Bacteroides <- glmer(Bacteroides / 100 ~ Contents*collection.factor + (1 | Bottle_Number),
data = StoolDT, subset = Collection %in% c(1, 2, 3), weights = as.numeric(Bacteroides.Count),
family = "binomial")
summary(fre.Bacteroides)
fre.Barnesiella <- glmer(Barnesiella / 100 ~ Contents*collection.factor + (1 | Bottle_Number),
data = StoolDT, subset = Collection %in% c(1, 2, 3), weights = as.numeric(Barnesiella.Count),
family = "binomial")
summary(fre.Barnesiella)
# fre.Enterococcus <- glmer(Enterococcus / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Enterococcus.Count), family = "binomial")
# summary(fre.Enterococcus)
fre.EscherichiaShigella <- glmer(`Escherichia/Shigella` / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(`Escherichia/Shigella.Count`), family = "binomial")
summary(fre.EscherichiaShigella)
fre.Faecalibacterium <- glmer(Faecalibacterium / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Faecalibacterium.Count), family = "binomial")
summary(fre.Faecalibacterium)
fre.Dialister <- glmer(Dialister / 100 ~ Contents*collection.factor + (1 | Bottle_Number), data =
StoolDT, subset = Collection %in% c(1, 2, 3), weights = as.numeric(Dialister.Count), family =
"binomial")
summary(fre.Dialister)
# fre.Solobacterium <- glmer(Solobacterium / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Solobacterium.Count), family = "binomial")
# summary(fre.Solobacterium)
# fre.Eubacterium <- glmer(Eubacterium / 100 ~ Contents*collection.factor + (1 |
Bottle_Number), data = StoolDT, subset = Collection %in% c(1, 2, 3), weights =
as.numeric(Eubacterium.Count), family = "binomial")
# summary(fre.Eubacterium)

# library(contrast)
# summary(fre.Fusobacterium)
# model.matrix(fre.Fusobacterium)


# make a wide data set for each genera of interest ------------------------
# StoolDT.w <- dcast(StoolDT, Bottle_Number + Contents ~ visit, value.var =
c("Fusobacterium", "Parabacteroides", "Parvimonas", "Ruminococcus", "Streptococcus",
"Porphyromonas"))
```

Principal Component Analysis

Multivariate Statistics

- Wikipedia – "Multivariate statistics is a subdivision of statistics encompassing the simultaneous observation and analysis of more than one outcome variable. The application of multivariate statistics is multivariate analysis."
- Examples include MANOVA, cluster analysis, factor analysis, canonical correlation, multidimensional scaling, discriminant analysis, structural equation modeling, vector time series, and principal components analysis.

'

Principal Components Analysis

- Often have variables in our dataset that are related to one another
- This relation leads to redundancy, and it may be beneficial to remove this redundancy by creating a small number of indexes that account for most of the variance in the observed variables
- These new indexes can then be used in a model, as either predictors or an outcome
- PCA finds patterns in the data (variances/covariances)
- It is hard to find patterns using traditional or informal techniques (e.g., graphical) when the data are highly dimensional
- PCA is an analytic tool for identifying these patterns and then compressing the data from the larger number of dimensions to this smaller number of dimensions, without losing much information
- PCA will derive a small number of linear combinations (aka principal components) of a set of variables that retains a maximum amount of information in those original variables
- Each variable gets a weight that can be used to develop an overall score for that component
- Good technique when the goal is to develop a linear combination and/or data reduction

Principal Components Analysis Model

$C_1 = b_{11} (X_1) + b_{12} (X_2) + \ldots .\ b_{1p} (X_p)$

C_1 = score on principal component 1

b_{1p} = weight (regression coefficient) for observed variable p

X_p = score on observed variable p

Some Definitions

- PCA uses some mathematical terminology that is popular in linear algebra/matrix algebra
- Eigen – German. Translates to "unique" in English.
- Eigenvector - A column (vector) of numbers for each component. Each number in the column is the coefficient/weight for that component for that variable.
- Eigenvalue – A single number associated with each eigenvector. It is the amount of variance in the variables explained by that component.

- Note: each PCA has the same number of eigenvectors and eigenvalues, with the maximum equal to the number of variables

Criteria to Determine #of Components

- Eigenvalues > 1
- Scree test
- Proportion of variance accounted for
- Interpretability
    - At least 3 variables per component
    - Face validity
    - Discriminate validity
    - Simple structure

Steps in PCA

- **Initial extraction**
- **Determine number of components**
- **Interpret & label components (possibly using rotation)**
- **Create factor/component scores**

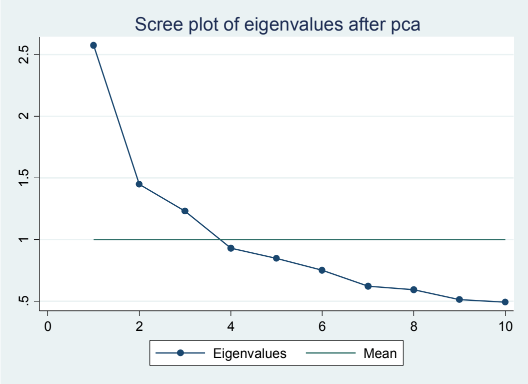
PCA Example

- Data from the MACC study examining tobacco use among youth
- Restrict the analysis to baseline, with ~4200 youth ages 12-16
- Asked respondents to report how difficult it is for youth to smoke (1=Not at all difficult, 5=Very difficult) in 10 different areas:
    - On School Property
    - In Restaurants
    - In Malls
    - In Pool Halls, Arcades, Bowling Alleys
    - In Parks/Playgrounds
    - In Your Home
    - In Best Friend's Home
    - In Coffee House
    - In Teen Dance Club
    - During School Hours

Descriptive Statistics
corr v*, means

Initial Extraction
pca v*

Scree Test
screeplot, mean



Keep 3 Components
pca v*, comp(3)

Create Component Scores
predict pc1 pc2 pc3, score
corr pc*, means

Principal Components Summary

- PCA is a statistical technique for creating weighted indexes or components from a set of observed variables
- PCA uses all of the variability in the items, not just the variability they share
- This technique is good for data reduction, showing how a smaller number of components will typically explain a fairly substantial portion of the variability in the items

Common Factor Analysis/ Exploratory Factor Analysis

- Sometimes we are only interested in using the variability that an item shares with the other items as opposed to the total variability – this is what primarily distinguishes common factors from principal components
- When the correlation between items is quite high, the difference between the total variability and the shared variability is quite small and the two approaches will yield very similar results
- However, if the overall correlation between items (as measured by the squared multiple correlation) is low, the two approaches can give fairly different answers

Common Factor Analysis

- In SEM parlance, the factor is a latent variable
- Almost all SEM techniques are confirmatory in nature; the user specifies a model, estimates this model, and then determines how well it 'fits' the data
- EFA is a precursor to this approach, developed as an exploratory technique, not requiring a user-specified model
- Many SEM users see this as a good first-pass technique, but has limitations that need to be understood when interpreting results

Principal Component vs Principal Coordinates Analysis

Principal Component Analysis and Principal Coordinates Analysis cover two of the main mathematical approaches to multivariate analyses. I think the main use of these methods is to visualize the data, but more complicated analyses can be done using the same ideas (e.g. MANOVA and factor analysis are based on the PCA approach).

Principal Component Analysis

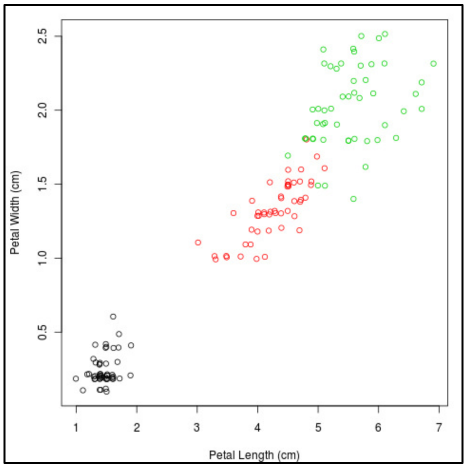
- Sometimes, we have variables in our dataset that are highly related to one another.
- This redundancy leads to correlation which creates unneeded noise in our analysis
- With PCA, we take all of these redundant variables, and make new index variables that get rid of the redundancy by grouping the more similar variables together
- Therefore, PCA is an analytical tool to find patterns in our data using their variance/covariance. It compresses the larger dimensions into similar groups without loosing information by using linear combinations: each variable in the combination gets its own weight depending on how much it contributes to the groupiness of the new index variable.

Criteria to determine the number of components vary: It can either be based on the eigenvalue (greater than 1), the scree test (visual version of the eigen value)

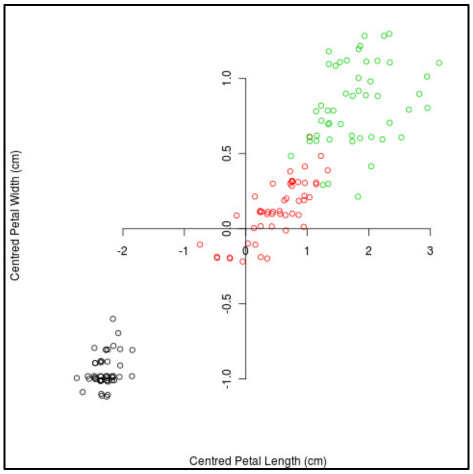
- Step 1: calculate the correlation between all the variables you want to use in the component analysis (to get a sense of the grouping)
- Step 2: run the PCA extraction in the variables
- Step3: look at the eigen values, or multiple class modeling based on AIC/BIC to determine the ideal number of classes
- Step 4: create the component scores, and check their correlation: there should be none since the redundancy was removed.
- The gamma parameter in PCA (LCA) gives you the proportion of the sample in each class, and the rho parameter gives you the proportion of each variable in your dataset that endorses the class you're looking at (variables that score more than 75% in one class should be the ones grouped together)

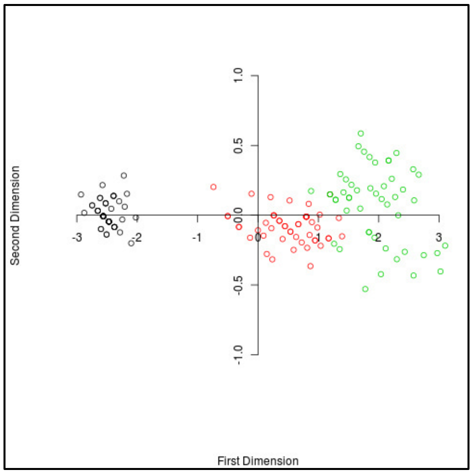
Principal Component Analysis: PCA

PCA is a statistical yoga warm-up: it's all about stretching and rotating the data. I'll illustrate it with part of a famous data set, of the size and shape of iris flowers. The original data has 4 dimensions: sepal and petal length and width. Here are the petal measurements (the different colours are different species):



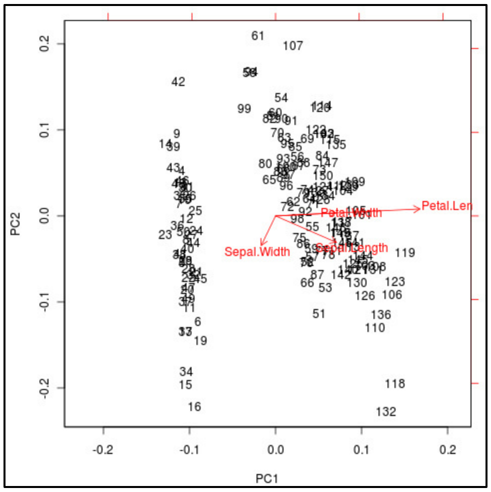
The purpose of PCA is to represent as much of the variation as possible in the first few axes. To do this we first center the variables to have a mean of zero, and then rotate the data (or rotate the axes, but I can't work out how to do that in R):





The rotation is done so that the first axis contains as much variation as possible, the second axis contains as much of the remaining variation etc. Thus if we plot the first two axes, we know that these contain as much of the variation as possible in 2 dimensions. As well as rotating the axes, PCA also re-scales them: the amount of re-scaling depends on the variation along the axis. This can be measured by the Eigenvalue, and it's common to present the proportion of total variation as the Eigenvalue divided by the sum of the Eigenvalues, e.g. for the data above the first dimension contains 99% of the total variation.

We can plot these on a graph: here we have it for the full data, with both sepals and petals:

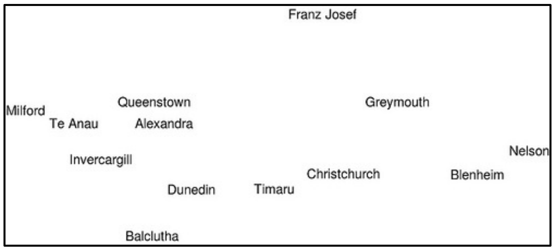


The arrows show the direction the variables point, so we can see that the petal variables are pretty much the same (i.e. there is a petal size which affects both equally). Sepal width is the only one that's different, mainly affecting the second PC. It's almost at 90° from the sepal length, suggesting they have independent effects.

It's obvious, looking at the data, that one species (I. setosa) is very different, with smaller petals, and differently shaped sepals. Mathematically, PCA is just an eigen analysis: the covariance (or correlation) matrix is decomposed into its Eigenvectors and Eigenvalues. The Eigenvectors are the rotations to the new axes, and the Eigenvalues are the amount of stretching that needs to be done. But you didn't want to know that, did you?

Principal Coordinate Analysis: PCoA

The second method takes a different approach, one based on distance. For example, we can take a map of towns in New Zealand:

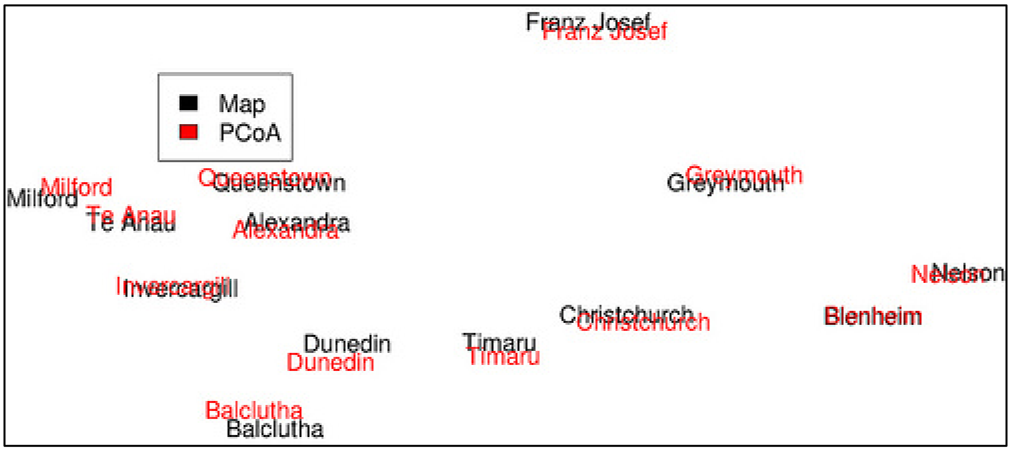


(I've rotated the map, to make it easier later)

The geographical distance between the town is Euclidean (well, almost. And near enough that you won't notice the difference). But we can also measure the distance one would travel by road between the towns. This would give us another measure of distance. Of course this may not be Euclidean in two dimensions, so we can't simply plot it onto a piece of paper. Distance-based methods are essentially about finding a "good" set of Euclidean distances from distances that are not a priori Euclidean in those dimensions.

The way principal coordinate analysis does this is to start off by projecting the distances into Euclidean space in a larger number of dimensions. This is not difficult; as long as the distances are fairly well behaved then we only need n-1 dimensions for with n data point. PCoA starts by putting the first point at the origin, and the second along the first axis the correct distance from the first point, then adds the third so that the distance to the first 2 is correct: this usually means adding a second axis. This continues until all of the points are added.

But how do we get back down to 2 dimensions? Well, simply do a PCA on these constructed points. This obviously captures the largest amount of variation from the n-1 dimensional space. So, for the New Zealand data if we do PCoA on the road distances, we get this:



And, not surprisingly, the maps are fairly close to each other, but not exact. One wrinkle for the sorts of applications we were discussing for bioinformatics (and which is also important in ecology) is the notion of a distance between two data points. In ecology we work with abundances of species, and the distances between the points need to somehow scale the abundances, so that rarer and more common species. So a plethora of diversity indices have been devised, and it's not clear which one should be used (unless one is in Canberra or New York, I guess).

For me, these methods are mainly useful for visualizing the data, so we can actually see how it's behaving, and they're not really very good for making formal inferences. But a lot of the multivariate methods for inference have the same ideas at their core, so understanding these is a good starting point. But it's still all about avoiding headaches by not having to think in 17 dimensions

The Fundamental Difference Between Principal Component Analysis and Factor Analysis

https://www.theanalysisfactor.com/the-fundamental-difference-between-principal-component-analysis-and-factor-analysis/

One of the many confusing issues in statistics is the confusion between Principal Component Analysis (PCA) and Factor Analysis (FA).
They are very similar in many ways, so it's not hard to see why they're so often confused. They appear to be different varieties of the same analysis rather than two different methods. Yet there is a fundamental difference between them that has huge effects on how to use them.

(Like donkeys and zebras. They seem to differ only by color until you try to ride one).

Both are data reduction techniques—they allow you to capture the variance in variables in a smaller set.
Both are usually run in stat software using the same procedure, and the output looks pretty much the same.

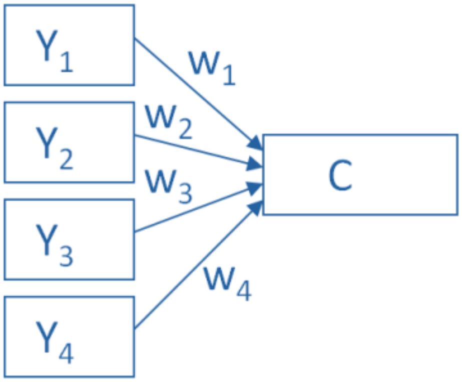
The steps you take to run them are the same—extraction, interpretation, rotation, choosing the number of factors or components.

Despite all these similarities, there is a fundamental difference between them: PCA is a linear combination of variables; Factor Analysis is a measurement model of a latent variable.

Principal Component Analysis
PCA's approach to data reduction is to create one or more index variables from a larger set of measured variables. It does this using a linear combination (basically a weighted average) of a set of variables. The created index variables are called components.

The whole point of the PCA is to figure out how to do this in an optimal way: the optimal number of components, the optimal choice of measured variables for each component, and the optimal weights.

The picture below shows what a PCA is doing to combine 4 measured (Y) variables into a single component, C. You can see from the direction of the arrows that the Y variables contribute to the component variable. The weights allow this combination to emphasize some Y variables more than others.



This model can be set up as a simple equation:

$C = w_1(Y_1) + w_2(Y_2) + w_3(Y_3) + w_4(Y_4)$

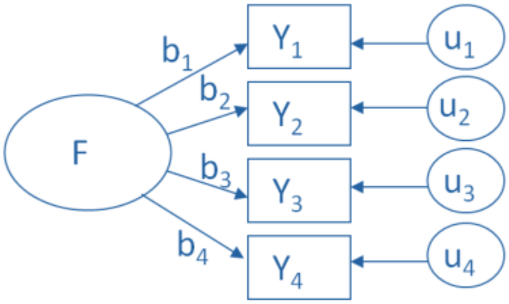
Factor Analysis

A Factor Analysis approaches data reduction in a fundamentally different way. It is a model of the measurement of a latent variable. This latent variable cannot be directly measured with a single variable (think: intelligence, social anxiety, soil health). Instead, it is seen through the relationships it causes in a set of Y variables.

For example, we may not be able to directly measure social anxiety. But we can measure whether social anxiety is high or low with a set of variables like "I am uncomfortable in large groups" and "I get nervous talking with strangers." People with high social anxiety will give similar high responses to these variables *because of* their high social anxiety. Likewise, people with low social anxiety will give similar low responses to these variables *because of* their low social anxiety.

The measurement model for a simple, one-factor model looks like the diagram below. It's counter intuitive, but F, the latent Factor, is *causing* the responses on the four measured Y variables. So the arrows go in the opposite direction from PCA. Just like in PCA, the relationships between F and each Y are weighted, and the factor analysis is figuring out the optimal weights.

In this model we have is a set of error terms. These are designated by the u's. This is the variance in each Y that is unexplained by the factor.



You can literally interpret this model as a set of regression equations:

$Y_1 = b_1*F + u_1$
$Y_2 = b_2*F + u_2$
$Y_3 = b_3*F + u_3$
$Y_4 = b_4*F + u_4$

As you can probably guess, this fundamental difference has many, many implications. These are important to understand if you're ever deciding which approach to use in a specific situation.

Part 12_QIIME Tutorial_MSI_MACQIIME_HMP

Using MSI for bioinformatics analyses
Abridged MACQIIME Microbiome analysis and MSI adaptation (MACQIIME dataset)
Unix Guide
Complete HMP QIIME 454 microbiome analysis tutorial (HMP dataset)
Complete MACQIIME microbiome analysis tutorial (MACQIIME dataset)

Files needed for a microbiome analysis

Raw Sequencing Data (.sff) for the Denoiser

This Denoiser step only applies to 454 Pyrosequencing data. It can be skipped, at your own risk. One problem with 454 pyrosequencing is that sequencing errors can give you effectively more OTUs than really are there. There are a number of strategies for dealing with this problem. The default in QIIME is to use a built-in program called Denoiser, which compares flowgrams (the raw sequencing data) of similar sequences to see if the differences between them may have been due to erroneous base calls. You can see some of this raw sequencing flowgram data by opening the Fasting_Example.sff.txt file in less. This is a text version of the "SFF" raw data format generated by the 454 pyrosequencing platform. It won't be terribly meaningful to you as it is, but it's always nice to have a feel for what all the data files look like.

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.

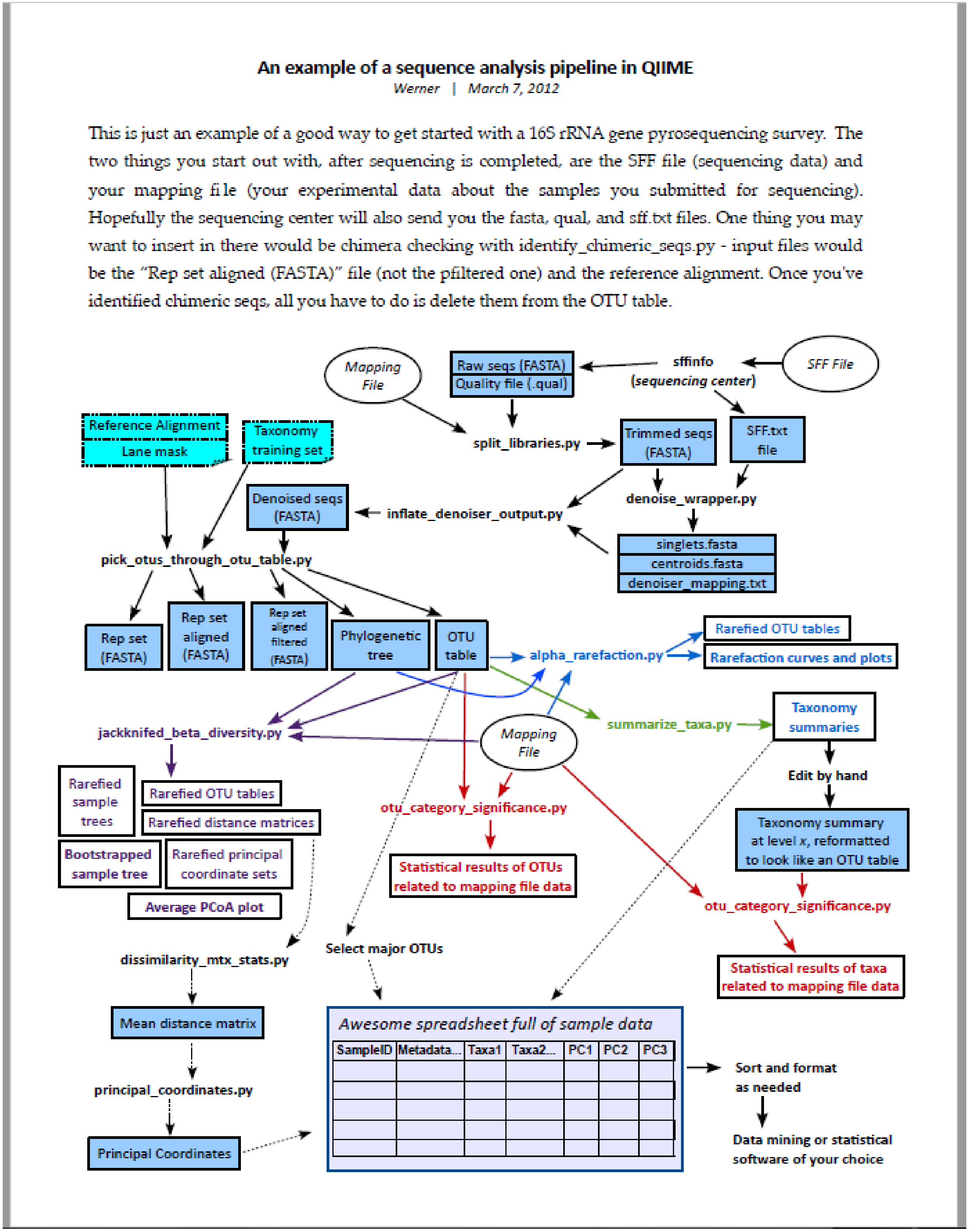
The primary file format for storing sequence data supported by QIIME is the FASTA format. The file Fasting_Example.fna is in FASTA format, indicated by the suffix ".fna" which stands for **F**ASTA **n**ucleic **a**cids (as opposed to amino acids which would have a suffix ".faa").

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.

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.



An example of a sequence analysis pipeline in QIIME

Werner | March 7, 2012

This is just an example of a good way to get started with a 16S rRNA gene pyrosequencing survey. The two things you start out with, after sequencing is completed, are the SFF file (sequencing data) and your mapping file (your experimental data about the samples you submitted for sequencing). Hopefully the sequencing center will also send you the fasta, qual, and sff.txt files. One thing you may want to insert in there would be chimera checking with identify_chimeric_seqs.py - input files would be the "Rep set aligned (FASTA)" file (not the pfiltered one) and the reference alignment. Once you've identified chimeric seqs, all you have to do is delete them from the OTU table.

SampleID	Metadata...	Taxa1	Taxa2...	PC1	PC2	PC3

Microbiome definitions

The human microbiome is our second genome, with much greater genetic diversity than our bodies

- Monocultures are very dangerous: a single strand taking over a whole community is a sign of severe dysbiosis (in extreme conditions)
- There are other, milder types of dysbiosis. Most chronic diseases (CVD, HTN, obesity, cancer, IBD and chronic inflammation) have now been linked to the human microbiome

We cannot study the microbiome directly, because most of the bacteria are hard to culture outside of the body

- We sequence DNA then align all bacteria genes to determine which sequences come from different strains
- These DNA sequences provide information on the bacterial species, genes, and functions in our microbiome
- An enterotype is a group or cluster of microbial genotype from a particular community

The goal of microbiome analysis ranges from identifying general dysbiosis (from healthy microbiomes, or epidemiological and clinical applications) to biomarker discovery (to determine a dysbiosis "fingerprint") to exploring changes in the microbiome over time and between individuals. We seek to make sense from the raw sequence data (number of species or microbial genes), then we use this to identify general dysbiosis (healthy vs unhealthy) which can be used for clinical applications, studying the changes in the microbiome of healthy people over time, or biomarker discovery (looking for a microbial fingerprint to distinguish healthy individuals from dysbiosis using bacterial genes or functional pathways)

Microbiome data generation (Microbial community > DNA sequences)

There are 2 major approaches to sequencing the microbiome:

- Shotgun sequencing: grind up the entire bacterial DNA in a sample and sequence it. However, it is hard to compare different genes from different bugs. This would be analogous to trying to compare multiple pieces from different jigsaw puzzles.
- 16S rRNA sequencing: Using a single common gene across many bacteria. 16S rRNA is ubiquitous with both highly conserved across species, + highly variable regions to differentiate bacteria. The 16 S rRNA gene fits this description because it codes for the ribosomal RNA, which is essential for all bacteria.

16S rRNA variable regions

- PCR is used to amplify DNA sequences rapidly and multiple times (Sequencing is made easier by making multiple copies of the gene). The revolution of PCR use in microbiome analysis came when DNA barcode were added to the beginning of sequences in each sample, allowing researchers to sequence multiple samples at once.
- Marker gene sequences are genes that are similar in every bug that we would like to sequence
- Ribosomal RNA is one of the most common type gene used to sequence bacteria. In accordance to the central dogma of molecular biology (DNA > RNA > Protein)

Ribosomal RNA has regions that are highly conserved (essential for ribosome function) and variable regions (mutated but still functional regions that vary across species)

- We use probes starting at the beginning of a conserved region to the end of another conserved region (so that it sandwiches a variable region in-between
- Once the sequencing is completed, we bin clusters of sequences together (called Operational Taxonomic Units) and compare these to a database to identify the species (Most people don't have most bug, so the table will be sparse).

Data processing with QUIIME

QUIIME provides OTU tables with the counts of how many times OTUs appear in each sample as table rows (will need to be rotated in the final analysis table), and the sample IDs in each column

- Tag the 16S gene from your samples with a DNA barcode joined to the probe
- Mix the sequences, amplify and sequence the V4 hypervariable sequence
- We use conserved and hyper variable regions of the 16S rRNA gene to sequence bacteria of interest
- Primers tags bookend the beginning and end sections of a variable region (by being complementary to the conserved regions) and can overlap over the variable regions in such a way that they meet in the middle and can resolve sequencing errors
- We typically use either the V1 – V3 or V3 – V5 variable regions of the 16s rRNA gene when sequencing the microbiome.

After sequencing, we obtain a file with sequences data (454 sequence data in the .fna format, or raw sequence data in the .sff format) + quality data (quality scores in the .qual format) + metadata (mapping file in the .txt format) + parameters (secquence of commands that can be run in UNIX).

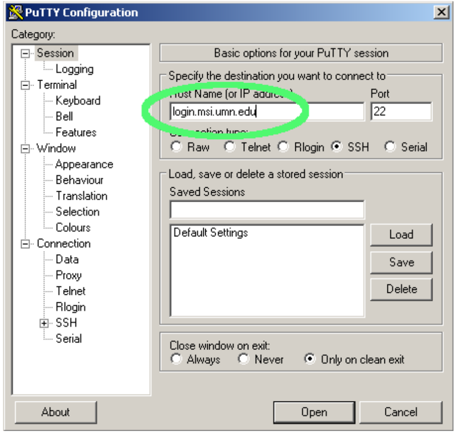
- The metadata file is a file with sample IDs + barcode sequences + primer sequences + description sequences
- The parameter file contains workflow scripts to run analyses

Installing QIIME

First Things First: Open the Terminal

Using QIIME on MSI

1. Download Putty on Windows (http://www.putty.org/) and configure it for MSI access



Host Name (or IP address): login.msi.umn.edu
Port: 22
Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial
Close window on exit: ○ Always ○ Never ● Only on clean exit

5. Connecting to Mesabi and Itasca from Login Host. Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh (QIIME is currently loaded under Itasca so use ssh Itasca for QIIME analyses)

```
ssh mesabi or ssh itasca
```

6. Alternatively, connect to MSI using Terminal on Mac/Linux, located in Applications>Utilities>Terminal.app.

```
ssh yourusername@login.msi.umn.edu
```

7. Log in to an interactive node, making sure you have enough time and memory

```
isub -n nodes=1:ppn=4 -m 16GB -w 24:00:00
```

8. Load the QIIME module

```
module load qiime/1.9.1
```

MacQIIME Users Only:

If you are using MacQIIME, you will have to use your Mac OS X terminal, located in Applications>Utilities>Terminal.app.

Normally, when you run a command in the terminal, all the packages in MacQIIME will be invisible to your computer. In order to access macqiime, after starting a new terminal session you have to source the environment variables with the "macqiime" command. To source the QIIME environment variables and load up all the MacQIIME scripts in the PATH, you have to start the MacQIIME subshell:

```
macqiime
```

You should now be looking at a dollar sign ($) with a cursor after it. This is the command line interface for your computer. Here, you can type commands in order to "execute" programs. This Unix/Linux command line is a powerful place to analyze and process data. Everything you do in QIIME is executed through this command line interface. I'm going to assume you have some familiarity with the Unix command line, but I'll try to help you along the way with commands like **cp**, **cd**, **ls**, **pwd**, **less**, **nano**, etc.

**Note: Getting MacQIIME to work in El Capitan (OS 10.11)

This section is required only if you are running OS 10.11 (El Capitan) or higher. Apple added a new security feature in El Capitan that makes it impossible to install software into /usr/bin/ folder (that was where I was putting the “macqiime” sourcing script. I'll fix this in the next release of MacQIIME. In the meantime, we can work around it. The quick solution is to use this command, instead of the “macqiime” script:

```
source /macqiime/configs/bash_profile.txt
```

## Get your dataset: you can either download the dataset from the source, then use the cd command to move to the folder where the data is located as follows, or you can download the file through the terminal window with the wget command (followed by the unzip and mv commands to get the uncompressed data). We will show the cd method here:

```
cd ~/qiime_tutorial/
```

This changes the directory to user\qiime_tutorial, Now you can look at the contents of the directory using the **ls** command:

```
ls –lh
```

You can always check the location of your current directory (aka folder) using the **pwd** command. If you type:

```
pwd
```

***** To copy any line code from the tutorial to the terminal window, select the code and hit CTRL + C, then in the control window right click and it will copy the code**
***** To quit any command that is running in the terminal, hit q**

The primary file format for storing sequence data supported by QIIME is the FASTA format. The file Fasting_Example.fna is in FASTA format, indicated by the suffix ".fna" which stands for **F**ASTA **n**ucleic **a**cids (as opposed to amino acids which would have a suffix ".faa").

One very useful command for this is ***less***. To use less to look at the contents of a text file, you just type *less* - spacebar - and then the name of the file. Try this command:

```
less Fasting_Example.fna
```

You should now see the first few lines of that FASTA file. You can scroll down and up using the arrow keys, or page down and up using the space bar and the "b" key. You can always return to the very top of the file by pressing "g". If you want to exit the less program, you can press "q".

Notice the format of these sequence entries. Each entry starts with a line of metadata, called the **header**. A header always starts with a ">" symbol, followed by a unique identifier, or ID, followed by a space and then some other data.

Press "q" to quit *less*. You can count how many sequences are in the file by searching for the ">" character in the file. This can be done using a program called **grep**, like this:

```
grep -c ">" Fasting_Example.fna
```

This should return "**1339**" -- meaning there are 1339 lines with ">" characters in them and, therefore, 1339 FASTA entries in the file. (The grep program normally searches for text lines that match the search string you enter, and the **-c** option asks grep to just count how many lines match, rather than displaying all the matching lines in the terminal.)

You also have data on the quality of each base. The quality data are stored in a separate file that ends in the ".qual" suffix. Find that file, and look at its contents using the *less* command. The first entry should look like this:

A qual file starts each entry with a header, identical to the FASTA format. Following each header is a series of numbers that denote the quality of each base (on a scale from 1 to 40). E.g., each number above corresponds with a respective base in the sequence FLP3FBN01ELBSX in the .fna file. Now, quit *less* and use *grep* to check how many entries are in this qual file -- it should also have 1339 entries.

The last file we'll look at now is the **sample mapping file**. It is called Fasting_Map.txt and you can look at it with *less* if you like, though this may not be an easy way to examine the file

The mapping file starts with one or more lines initiated by the "hash" symbol (#). The first of these initial lines should contain the headers, or titles, of each column of data, starting with the SampleID, followed by BarcodeSequence and the LinkerPrimerSequence. All the other columns are optional, and specific to the samples. The SampleID entries should be short, unique identifiers for your samples. They will get incorporated into the results data as you generate it.

You can see that this sequencing experiment included nine samples: five controls (mouse gut microbiomes) and four experimental samples (mouse gut microbiomes from fasted mice). If you want to make your own mapping file for an experiment, you could do so in a spreadsheet program, making sure to save the file as a tab-delimited text file.

Demultiplexing [need sequence file (.fna) + quality file (.qual) to make a demultiplexed sequence file (.fna)]

Demultiplexing is the process of separating sequences based on tagged barcodes. The sequences are assigned back to the original sample > phylogeny and taxonomy > statistical testing and predictive modeling

- Demultiplexing: Use the ID DNA barcode to determine which sequences come from which samples.
- OTU tables have features in the table columns (OTUs, Species, Gene Labels) and Sample IDs in the Rows
- For taxonomy assignments, we can compare our OTU clusters to a reference phylogeny tree, or build a phylogeny tree ourselves based on how similar DNA sequences are.

The QIIME script split_libraries.py accomplishes the task of checking each sequence for a barcode and primer, trimming off the barcode and primer, and then renaming the sequence so that the ID in the FASTA header indicates which sample it came from. This script needs two input files: (1.) the FASTA file of raw sequencing data, and (2.) the mapping file specifying which barcodes and primers belong to which samples. You can also tell the split_libraries.py script where to find the qual file, so it can trim out low-quality sequence data.

Here is an example of running split_libraries.py on the tutorial data (all one long line of text):

```
split_libraries.py -m Fasting_Map.txt -f Fasting_Example.fna -q Fasting_Example.qual -o split_library_output/
```

The command line options for QIIME scripts can appear in any order. The flag **-m** is followed by the name of the mapping file, the flag **-f** is followed by the name of the FASTA file, **-q** is followed by the qual file, and **-o** is followed by the name you want to give to the output folder (a new folder that will be created by the script).

The script created the new directory you specified with the **-o** option. You can verify this with "**ls -lh**" -- there should be a new directory called split_library_output. You can see the contents within this directory using the ***ls*** command as well, by specifying the name of the directory to look in:

```
ls -lh split_library_output/
```

There should be three output files in that directory: histograms.txt, seqs.fna, and split_library_log.txt. The log file tells you stats on how your analysis worked out, including a table of the number of sequences assigned to each sample. Read it with *less*, like this:

```
less split_library_output/split_library_log.txt
```

Nice; there were about 150 reads assigned to each sample. Good to know! Quit, and then look at the output FASTA file:

```
less split_library_output/seqs.fna
```

The sequences are shorter, and they now have new IDs! The new FASTA IDs created by split_libraries are formatted to indicate the sample name, followed by an underscore (_), and then a number. For example, the sequences with IDs PC.634_1 and PC.634_2 both came from the sample PC.634, and the numbers 1 and 2 are there to make sure that each sequence still has a unique FASTA ID. This ID format: SampleName_[number] is necessary for future steps in the pipeline, so QIIME will know what samples the reads came from (Remember - we've removed the barcodes and primers!).

Midprocessing

- Making an OTU table
- Assigning Taxonomy
- Building a phylogeny

Denoise Seqs with Denoiser

This Denoiser step only applies to 454 Pyrosequencing data. It can be skipped, at your own risk. One problem with 454 pyrosequencing is that sequencing errors can give you effectively more OTUs than really are there. There are a number of strategies for dealing with this problem. The default in QIIME is to use a built-in program called Denoiser, which compares flowgrams (the raw sequencing data) of similar sequences to see if the differences between them may have been due to erroneous base calls.

```
denoise_wrapper.py -i Fasting_Example.sff.txt -f split_library_output/seqs.fna -m Fasting_Map.txt -o denoiser/
```

Some of the options:
-i [filename] Specifies the sff.txt file name
-f [filename] Specifies the fasta file that was created by split_libraries.py
-m [filename] Sample mapping file with primers and barcodes
-o [name] Specifies a directory name for the output

This script created a new output directory called denoiser/ which contains a bunch of new files. The important ones are denoiser_mapping.txt, centroids.fasta and singletons.fasta. The singletons had no other sequences that matched up with them, and the centroids are representatives of clusters that collapsed into a single sequence following denoising. The denoiser_mapping.txt file maps which sequences fell into which centroids. To make use of these data, we have to "inflate" the results to create a new FASTA file that is a theoretically denoised version of the original. The script that does this for us is called inflate_denoiser_output.py

```
inflate_denoiser_output.py -c denoiser/centroids.fasta -s denoiser/singletons.fasta -f
split_library_output/seqs.fna -d denoiser/denoiser_mapping.txt -o
inflated_denoised_seqs.fna
```

Some of the options:
-c [file] The centroids.fasta file from denoise_wrapper.py
-s [file] The singletons.fasta file from denoise_wrapper.py
-f [file] The seqs.fna file from split_libraries.py
-d [file] The denoiser_mapping.txt file from denoise_wrapper.py
-o [file] The name of the new fasta file to be created

Check the contents of the newly created output file, inflated_denoised_seqs.fna. It should contain all the seqs from split_library_output/seqs.fna, but the sequences will be slightly different depending on to what extent they were denoised. (For more complex examples and information, see the Denoising Tutorial on qiime.org. Note: I tested this tutorial both with and without the denoising step, and denoising reduced the number of OTUs from 417 to 209 -- quite a big difference!)

Picking OTUs [need a demultiplexed sequencing file (.fna) to make another sequencing file where the sequences are clustered into OTUs (.fna) via the pick_otus.py command].

- Rather than matching every individual DNA sequence to a species, we use cluster to control for sequencing error and make OTUs
- OTUs can be made based on a centroid based approach: Ideally the within cluster distance is <3%, and the between centroid distance is >3%.
    - Single linkage clustering: Friend of a friend approach, where a sequence only needs to be sufficiently close to one member of a cluster to be included in said cluster
    - Complete linkage clustering: All members of a cluster need to be sufficiently close to one another to constitute a cluster
    - Average linkage clustering: A sequence needs to be within a certain average distance of a cluster to be included
- A good cluster has a small within cluster distance, and a large between cluster distances.
- One of the most common clustering approaches is UCLUST (in QIIME), which is a greedy clustering algorithm
    - Cluster the sequences into OTUs, and use a phylogenetic tree to determine how closely related each sequence within a sample is
    - Pick a closed reference OTU: Match OTUs to a reference database, and discard any mismatch sequences
    - Pick an open reference OTU: Match OTUs to a reference database, and use de novo clustering for any mismatched sequences from the database
    - Pick de novo OTUs: Create your own phylogenetic trees based on how similar your sampled DNA sequences are

Pick Operational Taxonomic Units

OTUs (operational taxonomic units) are clusters of similar sequences. Much of the QIIME pipeline is concerned with analyzing OTUs rather than the full set of sequences. In this OTU-centric process, you are looking at samples as collections of OTUs, where two different samples may contain some of the same OTUs and some different OTUs. We'll eventually assign each OTU some data, such as taxonomy data, data as to how abundant it was in each sample, and data on the evolutionary history of each OTU compared to the others. There are a number of ways to bin sequences into these "clusters" of OTUs, a process called OTU picking. The default OTU-picking method in QIIME is to use a program called uclust.

The defaults in the QIIME script pick_otus.py for OTU-picking are to use uclust, and to use an identity threshold of 97% (i.e., sequences that are 97% similar should get binned into the same OTU together). You can see how to specify different options in the help info for pick_otus.py. We're going to use all the defaults, so all we have to specify is the input file name with the -i flag:

```
pick_otus.py -i inflated_denoised_seqs.fna
```

Notice that the fasta file we used as input was the output of our previous denoising step. (If you skipped denoising, the input for the -i option in this step could instead be split_library_output/seqs.fna) Our new command should create a new directory automatically named "uclust_picked_otus." Look inside that directory with ls and you should see three new files: a .uc file, a .log file, and a file called inflated_denoised_seqs_otus.txt.

Let's look at the contents of that file:

```
less uclust_picked_otus/inflated_denoised_seqs_otus.txt
```

It's another tab-separated file. This format is called an OTU map. It is an intermediate file format, and maybe not very useful to you as raw data. It lists each OTU (the first column is the ID of the OTU, counting up from 0) followed by the IDs of all the sequences that fell into that OTU. My first few lines look like this:

```
0	PC.356_1161
1	PC.635_788
2	PC.635_779	PC.634_176	PC.634_7
```

You can see that OTU 0 and OTU 1 both contain only one sequence, while OTU 2 contains three sequences. Some of the OTUs contain many sequences! Okay, we can quit less now. Working with all of your sequences is cumbersome, if you are, for example, building an alignment, assigning taxonomy, etc. Since we have OTUs picked, what we really would like to work with are a collection of just one representative sequence per OTU. We can get this using pick_rep_set.py.

pick_rep_set.py -i uclust_picked_otus/inflated_denoised_seqs_otus.txt -f inflated_denoised_seqs.fna -o rep_set.fna

Some of the options:
-i [file] The OTU mapping file from pick_otus.py
-f [file] The fasta file that was used to pick OTUs
-o [file] The name of the output FASTA file to create

How many sequences are in our resulting FASTA file? We can do **grep -c ">" rep_set.fna** to see: mine has 209 sequences in it (one for each OTU).

Assigning Taxonomies (need the OTU sequence file from the previous step (.fna) to make a folder with a text file (.txt) which contains the taxonomic assignments for each OTU)

In order to assign taxonomies, we need to have a training dataset to create a model which maps our input data (DNA sequences) to labels (Taxonomy). Then, we can run our model on our newly sequenced data to get taxonomy assignments and confidence levels for these taxonomies. The order of classification is taxa > Phyla > Genus > Species.

Naïve Bayes Classification

Statistics Review
P(X) = Prior probability
P (A, B) = Joint probability
P (A|B) = Conditional probability
P (A|B) = P (A, B) / P (B)
P (A, B) = P (A|B)*P (B)
P (A) = P (A, B) + P (A, not B) = Law of total probability

Bayes Rule
P (A|B)*P (B) = P (A, B) = P (B, A) = P (B|A)*P (A)
P (A|B)*P (B) = P (B|A)*P (A)
P (A|B) = P (B|A)*P (A) / P(B)

P (C|X) = P (X|C) * P(C) / P (X)
Posterior = likelihood * prior / evidence
P (genus|sequence) = p (sequence|genus)* p (genus) / p (sequence)
P (Taxonomic label | data) = P (from training data) * P (genus) / P (sequence)

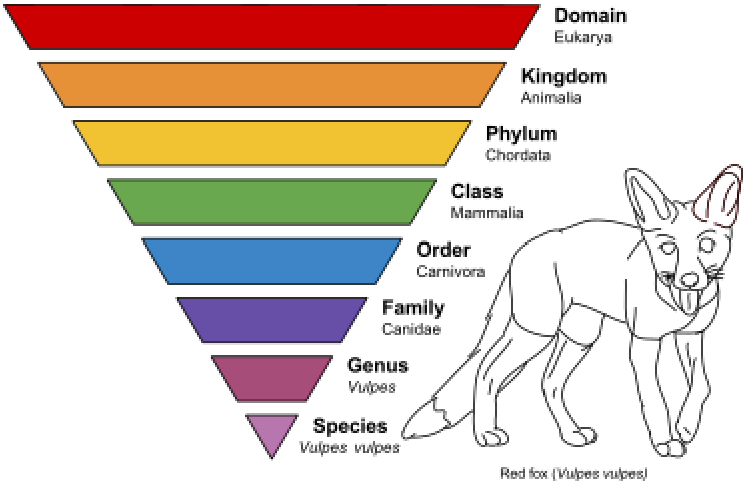
Naïve Bayes Classification procedure

- The probabilities are dependent on the length of k-mers. K-mers are short nucleotide sequences (up to 8 bases long, typically)
- The main limitation of the naïve Bayes classifier is the level of false positives, because in practice we simplify away the fraction given by P (genus)/ P (sequence) by assuming that they have the same distribution. However, this is misleading because P (genus) can vary

greatly based on the sample and microbial community, although P (sequence) becomes negligible since we are using kmers which are very short.

Assigning Taxonomy

Once you have clustered your sequences, you can use a reference database to assign various taxonomic ranks to the OTU clusters you have created



In QIIME, L2=phylum, L3 = Class, L4 = Order, L5 = Family, L6 = Genus, L7 = Species Taxonomy is assigned to high-throughput 16S rRNA gene sequences using some kind of comparison to a reference database. The most basic approach might be doing a BLAST search and taking the top hit. However, this doesn't account for redundancy or give you any idea of your confidence or specificity. The default algorithm in QIIME is the RDP Classifier. This is a Bayesian classifier that incorporates information about different places in the taxonomic tree where the sequence might fit in, and it calculates the highest probability taxonomy that can be assigned with some specified level of confidence. This still uses a reference database, in this case called a training set. In MacQIIME, the default training set is the Greengenes (GG) reference database clustered at 97% identity, other versions of GG are available for download from here. We're going to use the QIIME script assign_taxonomy.py as follows (may take a minute):

```
assign_taxonomy.py -i rep_set.fna -o taxonomy_results/
```

Some of the options:

```
   -i [file]  Name of the fasta file to classify (usually OTU representative seqs)
   -o [name]  Name of directory to create for output
```

To use a custom training set like the download from Greengenes, you have to reference it using the -t and -r options as described in the documentation.

This created a new directory called taxonomy_results. Look in there with ls and you will see that it contains two files: a log file, and a txt file of results. The rep_set_tax_assignments.txt file contains an entry for each representative sequence, listing taxonomy to the greatest depth allowed by the confidence threshold (80% by default, can be changed with the -c option), and a column of confidence values for the deepest level of taxonomy shown. These data will be really useful, once we have them inside an OTU table!

Building an OTU Table

An OTU table is a form of your sequencing results that will finally be really useful to analyze in excel, visualize, etc. It is a table giving the count of the number of sequences in each OTU, for each sample, and the taxonomy of that OTU. Super! We generate an OTU table with a script called make_otu_table.py as follows

```
make_otu_table.py -i uclust_picked_otus/inflated_denoised_seqs_otus.txt -t
taxonomy_results/rep_set_tax_assignments.txt -o otu_table.biom
```

Some of the options:
-i [file] The OTU map output from pick_otus.py
-t [file] The taxonomy file from assign_taxonomy.py
-o [name] The name of the output file

This should create an OTU table file called otu_table.biom. Check it out in less - it is in an XML/JSON format called biom (new as of QIIME 1.5.0). If you want to look at the OTU counts per sample in a spreadsheet, you'll first have to convert the biom OTU table into a normal text table. This can be done with the biom script as follows:

```
biom convert -i otu_table.biom -o otu_table_tabseparated.txt --to-tsv --header-key
taxonomy --output-metadata-id "ConsensusLineage"
```

Now, the new output file otu_table_tabseparated.txt is something you can easily import into a spreadsheet. You can also read it more easily in the terminal:

```
less otu_table_tabseparated.txt
```

Note that this tab-separated file format for OTU tables used to be supported by QIIME (v 1.4 and earlier) but is no-longer a supported format as of QIIME 1.5.0. This is due to the needs of researchers generating 16S rRNA gene surveys using the Illumina HiSeq platform, with thousands of samples and hundreds of thousands+ of OTUs. Once the OTU table gets that big, it's much more reasonable to store it in the biom format.
Also note, when analyzing these data in a spreadsheet, that you will first want use the spreadsheet functions to normalize the abundances to the total number of reads in each sample. If one sample has 100 reads and another has 200 reads, you can't directly compare raw read numbers -- it's more meaningful to look at percentages. e.g., One OTU was 10% of the abundance in Sample 1 vs 50% of the abundance in Sample 2, etc.

Summarizing Taxonomy

The OTU table is pretty useful, but we can make other tables of taxonomic summaries that we can analyze in a spreadsheet as well. To summarize taxonomy in terms of relative abundance, we can use the script summarize_taxa.py as follows:

```
summarize_taxa.py -i otu_table.biom -o taxonomy_summaries/
```

Some of the options:

```
   -i [file]  The input OTU table with included taxonomy information
   -o [name]  The name of the directory to be created and filled with goodies
```

Check out the contents of the taxonomy_summaries folder. You can open any of these files in a spreadsheet and graph taxonomy summaries for your samples! The numbers are in terms of relative abundance. In other words, they're already normalized. The code L2, L3, etc, refers to the level of taxonomy summarized in the file.

If you're impatient and don't want to dive into the spreadsheet just yet, QIIME can make some simple graphs to summarize the results. The script that does this is called plot_taxa_summary.py. For example, you can summarize taxonomy at the Class level (L3) like this:

```
plot_taxa_summary.py -i taxonomy_summaries/otu_table_L3.txt -o taxonomy_plot_L3/
```

Open up the newly created bar_charts.html file in your web browser to see some graphed data.

Build a phylogeny [need the OTU sequence file (.fna) to make a .fasta file in which our sequences have been aligned, then use that .fasta file to create a phylogenetic tree file (.tre)]

To test the evolutionary distance between your OTUs, taxonomy assignment is not the best way to go, as it depends too much on whether the taxonomy database had representatives of everything in your sample, and it depends on whether or not the taxonomic hierarchy used accurately reflects evolution. Instead, it's better to build a real phylogenetic tree.

D. Alignments and Trees

Make a Multiple Sequence Alignment

To test the evolutionary distance between your OTUs, taxonomy assignment is not the best way to go, as it depends too much on whether the taxonomy database had representatives of everything in your sample, and it depends on whether or not the taxonomic hierarchy used accurately reflects evolution. Instead, it's better to build a real phylogenetic tree, and to do that we'll need to first align our sequences. We'll be using a reference alignment to align our sequences. In MacQIIME, this reference alignment is located at /macqiime/greengenes/core_set_aligned.fasta.imputed and QIIME already knows where it is, so the command line you need is actually quite simple (Also true in the VB, but it's located in

/data/greengenes_core_sets/). The script we're using is called align_seqs.py and we use it as follows:

```
align_seqs.py -i rep_set.fna -o alignment/
```

Some of the options:
-i [file] The fasta file of queries, usually your representative seqs
-o [name] The name of the new directory that should be created
-t [file] The template file to use (used default GG core in our command above; didn't have to specify)

You can see the alignment results in that alignment/ directory, file name rep_set_aligned.fasta. This alignment contains lots of gaps, and it includes hypervariable regions that make it difficult to build an accurate tree. So, we'll filter it. Filtering an alignment of 16S rRNA gene sequences can involve a Lane mask. In MacQIIME, this Lane mask for the GG core is located at /macqiime/greengenes/lanemask_in_1s_and_0s and MacQIIME already knows where it is. The script you use to filter an alignment is filter_alignment.py as follows:

```
filter_alignment.py -i alignment/rep_set_aligned.fasta -o alignment/
```

Some of the options:

-i [file] The fasta file of queries, usually your representative seqs
-o [name] The name of the new directory that should be created
-m [file] The Lane mask file to use (used default GG Lane mask in our command above; didn't have to specify)

This created a new file in the alignment/ directory called rep_set_aligned_pfiltered.fasta -- this is the file we can use to build a phylogenetic tree! If you want to visually check the alignment, I suggest using a free program called SeaView to open the rep_set_aligned_pfiltered.fasta file.

Build a phylogenetic tree

Okay, let's make a tree out of that alignment! This is actually quite easy in QIIME, using the make_phylogeny.py script (which uses the FastTree approximately maximum likelihood program, a good model of evolution for 16S rRNA gene sequences). The input for this script is our filtered alignment.

```
make_phylogeny.py -i alignment/rep_set_aligned_pfiltered.fasta -o rep_set_tree.tre
```

How do you look at this tree? You could try something like FigTree, or TreeViewX, but actually Topiary Explorer may be a better option - it is meant to be able to import your QIIME OTU table and mapping file to display data as well as the tree - try it out (http://topiaryexplorer.sourceforge.net/)!

Diversity analysis

- Rarefaction is an indicator of whether you have sampled all of the diversity within your sample
- Alpha diversity refers to the diversity within a sample (how diverse each sample is)
- Beta diversity revers to diversity between samples (how different are your samples)

Rarefaction

You may think of diversity, or species richness, as "the number of species" that are present in the system. That *is* the general definition of diversity. However, the deeper you sequence, the more species you will find. That is problematic, especially if you gathered 500 reads from one sample and only 100 reads from another sample. You would expect to find more species if you sequenced 5x as many reads! To account for this, we perform an *in-silico* (i.e., on your computer) experiment called **rarefaction**. A rarefaction is a random collection of sequences from a sample, with a specified depth (number of sequences). For example, a rarefaction with a depth of 75 reads per sample is a simulation of what your sequencing results would look like if you sequenced exactly 75 reads from each sample. To look at alpha diversity systematically, we can perform many rarefactions: at multiple depths and repeat many times at each depth. In QIIME, this task is performed on your OTU table.

- Rarefaction is an indicator of the balance between sequencing depth versus diversity discovered. It is calculated by measuring the alpha diversity for a subset of the sequence, (mean + STD) and repeating the process for sequentially longer sequences.
- If the community was sequenced deeply enough, then the mean of the alpha diversity will stop increasing as the length of the sequences increases. This mean that increasing the sequencing depth does not add any additional information on the community's diversity

Alpha Diversity (Need to rarefy your samples first (based on the sample with the least amount of reads) then the alpha rarefaction will use the rarefied file folder + the phylogenetic tree (.tre) along with the diversity metrics to create an output folder with the metrics)

Diversity within a single sample (aka, what species are present, and how many species are present)

- For alpha diversity, we need the OTU + phylogeny + alpha rarefaction
- We use species counts (which does not take into account how closely related species are) as well as 97% OTUs to group species and account for phylogenetic diversity.
- In practical terms, it is the sum of the branches of a phylogenetic tree in a sample that are conserved across samples.
- The Chao Diversity index uses species + singleton/doubletons to give us an estimate of rarefaction (depth of sequencing of the alpha diversity): Attempts to predict the distribution of species (90 percent is one bug and 10% is 10 bugs, or evenly distributed 10% for each bug)
- Another diversity index that is often used is the Shannon Diversity Index

Example:
Sample A: Pseudomonas aeruginosa + Pseudomonas argentinensis + Pseudomonas Flavescens
Sample B: Pseudomonas aeruginosa + Pseudomonas argentinensis + E Coli
Sample C: Pseudomonas aeruginosa + Giardia Lamblia + Methanobrevibacter smithii

Just using the species count, we would see that sample A = 3, sample B = 3 and sample C = 3, thus the three samples are equally diverse. However, this does not take into account how closely related species within a sample are. By using phylogenetic data in conjunction with alpha diversity, we can see that the species in sample A would be clustered in the same area of the tree of life (all pseudomonas), and would have a low phylogenic diversity index (PD). Sample B would be a little less clustered, while sample C would be the least clustered. Thus, by taking PD into account we can see that sample A < sample B < sample C in terms of diversity

To look at alpha diversity systematically, we can perform many rarefactions: at multiple depths and repeat many times at each depth. In QIIME, this task is performed on your OTU table. The QIIME script multiple_rarefactions.py takes your OTU table and makes a folder full of many OTU tables, all of which are repeats of rarefactions at specific depths. Let's use multiple_rarefactions.py as follows:

```
multiple_rarefactions.py -i otu_table.biom -m 20 -x 100 -s 20 -n 10 -o rare_20-100/
```

Some options for multiple_rarefactions.py:

```
   -i [file]  The input OTU table file
   -m [number]  The lowest rarefaction depth in the series of depths
   -x [number]  The highest rarefaction depth in the series of depths
   -s [number]  The step size to increment from the low to high depths
   -n [number]  The number of replicates to perform at each depth
   -0 [name]   The name of the new directory to be created
```

From that command above, you can see that we are performing rarefactions starting at 20 seqs/sample, and stepping up to 100 seqs/sample in increments of 20. In other words, we'll perform rarefactions at 20, 40, 60, 80, and 100 seqs/sample. The -n option specifies that we'll do 10 replicates at each of those depths. It is important to chose the rarefaction depths based on how many total sequences per sample you have. For example, it would not make sense to do rarefactions from 20 to 100 seqs/sample if I had a larger data set with an average of 5000 seqs/sample. If I did that, I would be throwing out a lot of my data and statistical power!

Run the command, and it will make that directory named rare_20-100/. If you look inside that new directory with ls, you will see that we've created 50 new files. Each of those files is a new OTU table with all the samples rarefied at the specified level! We're going to look at alpha diversity in those rarefied OTU tables, not in our original OTU table.

Calculate Alpha Diversity

There are many measures of alpha diversity. Depending on your ecological allegiances, you may have a preference for Chao1, Simpson's Diversity, Shannon Index, etc. These all measure different things, so it's important to think about what is most meaningful for your experiment, and your question. The QIIME script for calculating alpha diversity in samples is called alpha_diversity.py. There are many options for what metrics to use, and you can chose to run a bunch of metrics all at once if you like. All the possible alpha diversity metrics available in QIIME are listed here.

alpha_diversity.py -i rare_20-100/ -o alpha_rare/ -t rep_set_tree.tre -m observed_species,chao1,PD_whole_tree

Some options:
-i [directory] The name of the directory containing rarefied OTU tables
-o [name] The name of the directory to create for output
-t [file] The file for your phylogenetic tree
-m [list] The list of metrics, separated with commas and no spaces

If you run the above command, it will calculate alpha diversity metrics for all of your rarefied OTU tables and place the results in a new directory called alpha_rare. The metric PD_whole_tree is Faith's Phylogenetic Diversity, and it is based on the phylogenetic tree. Basically, it adds up all the branch lengths as a measure of diversity. So, if you find a new OTU and it's closely related to another OTU in the sample, it will be a small increase in diversity. However, if you find a new OTU and it comes from a totally different lineage than anything else in the sample, it will contribute a lot to increasing the diversity. There are still a ton of separate files, and we need to "collate" them together into a nice, neat collection of results that are easy to graph.

The QIIME script collate_alpha.py takes the output directory from alpha_diversity.py as its input, and creates a new output directory containing files that are much easier to look at in a spreadsheet.

collate_alpha.py -i alpha_rare/ -o alpha_collated/

Take a look at the new files in the alpha_collated/ folder-- they are each organized quite nicely for spreadsheet analysis

Beta Diversity (Need to use the OTU table as a biom file (.biom) + the phylogenetic tree (.tre) to create an output folder with the metrics)

Diversity between different individuals or body sites. Can also be determine using principal coordinate analysis to determine how each community is related to one another and changes over time.

- We are comparing communities from 2 different samples, whether these are between samples are different time points for the same individual, or 2 samples for individuals in different treatment group.
- In order to determine how different two communities are, we can either use the Euclidian distance Square root (qi-pi)^2 to determine the distance between the two communities (since it is the multidimensional pairwise distance collapsed on a 2 D scale),

- Another metric that works well with beta diversity is the bray curtis metric: Beta diversity compares two different samples and asks, overall, how different are they? For each species among two samples, you find the lowest value, take the sum of the lower value and divide it by the total count for all the species in both samples. This ratio gives you an estimate of how much does one sample undershoots the other, in terms of abundance for a given species.
- We need to include an OUT table + a parameter file for rarefaction (#of sequences/sample) + phylogeny data (how OTUs in the sample are related) + a mapping file
- To visualize the beta diversity matrices, we can use a principal coordinate approach (based on principal component analysis)

Unifracs uses phylogenetics (how closely related species within a sample are vs how different) to assess Beta diversity.

- In practical terms. Beta diversity can be interpreted as the sum of the branches in a phylogenetic tree in a sample that are different across samples (in contrast to alpha diversity, which is the sum of the branches of a phylogenetic tree in a sample that are conserved across samples.
- Weighted unifracs weights the branch lengths of the phylogenetic tree pertaining to beta diversity by the abundance of each bacteria species. This tends to bias the measure towards dominant bugs
- Unweighted unifracs only uses the presence/absence of bugs as a weighting mechanism; therefore it is biased towards rare bacterial species, because if a bug is only present in a few samples, it gets assigned a large weight.
- One method to compare the diversity measure involves converting the beta diversity index into a single variable. You can plot each sample average in your intervention group against the average diversity in the control group, thus giving you a single measure of diversity which can then be used for analysis with other covariates.

Visualize the results of beta diversity analysis

Beta diversity is a term for the comparison of samples to each other. A beta diversity metric does not calculate a value for each sample. Rather, it calculates a distance between a pair of samples. If you have many samples (in this tutorial we have nine samples), a beta diversity metric will return a matrix of the distances of all samples to all other samples. If you have experience in phylogenetics, you may know that a distance matrix can be visualized as a tree. Distance matrices can also be visualized as a graph of points, a network, or any other creative method you can come up with. In this tutorial, we'll use principal coordinates analysis to visualize distances between samples on an x-y-z plot.

Sequencing depth can have an effect on beta diversity analysis, just as it does on alpha diversity. The effect of sequencing depth is not as easy to visualize, and it depends a lot on what distance metric you chose. The safest bet is always to choose a single sequencing depth that is less than the total number of sequences in your smallest sample, and rarefy all your samples at that same depth.

For now, we'll start with the workflow script that helps us with our task at hand: beta diversity analysis. Note: Once you get more advanced in QIIME, you may want to set a lot of custom options. Because of the complexity of a workflow script, many of the options have to be set by supplying an input file called a parameters file (with the -p option). You don't need a parameters file, though. If one is not specified, all the steps of the workflow will happen with relatively common default settings.

Jackknifed beta diversity analysis

We're going to use the workflow script jackknifed_beta_diversity.py to do our beta diversity analysis. This script does the following steps:

- Compute a beta diversity distance matrix from the full data set
- Perform multiple rarefactions at a single depth
- Compute distance matrices for all the rarefied OTU tables
- Build UPGMA trees for all the rarefactions
- Compare all the trees to get consensus and support values for branching
- Perform principal coordinates analysis on all the rarefied distance matrices
- Generate plots of the principal coordinates

Let's start this following script, and we'll talk about all the parts to the script while it's running:

jackknifed_beta_diversity.py -i otu_table.biom -o jackknifed_beta_diversity/ -e 90 -m Fasting_Map.txt -t rep_set_tree.tre

Some of the options for jackknifed_beta_diversity.py:

-i [file] The OTU table file
-o [name] Choose a name for the output directory to be created
-e [number] The depth for even rarefactions
-m [file] The sample mapping file
-t [file] The phylogenetic tree file

Let’s start with the -e option for the rarefaction depth. This script performs rarefaction at only one depth. The idea is that we want to create a large collection of distance matrices that we can do statistics on. For this to work, we need to choose a depth that is significantly smaller than the number of seqs in the smallest sample. If any samples have fewer seqs than the rarefaction depth, they will be left out! Also, if the rarefaction depth is too high, then every rarefied OTU table will be the same, and there won't be any differences between all our distance matrices. I chose 90 seqs/sample as our depth, because most samples have a little under 150 seqs/sample total. Going even lower may have been advisable, perhaps to 75 seqs/sample or less, but this is a pretty small data set to start with so I just made a compromise.

Let's go back and look at that parameters file... there are lots of variables in there that you can set. You'll notice that most of them are not set to anything. Also, many of them have nothing to do with our jackknifed_beta_diversity.py workflow. That's okay. The idea is that you can set up one parameters file for all your workflows in a project, and any variables you leave blank in the parameters file will just be set to their default values.

Okay, my jackknifed_beta_diversity.py workflow is done; there are LOTS of new files that it created in that jackknifed_beta_diversity/ folder! There is one sub-folder full of rarefied OTU tables, and three separate sub-folders for each of the three different distance metrics we chose! Let's look in the unweighted UniFrac folder. It contains a folder called "3d_plots," (Note: This has changed to "emperor_pcoa_plots" in QIIME 1.8 and above) and in there is an html file named index.html. Double-click on this html file to open it in your browser.

Those are the results of our principal coordinates analysis! Each point represents one of the samples. The important thing to realize here is that the axes are meaningless. This is different from principal components analysis, which you may be more familiar with. If you do principal components analysis, you get out axes that have weighted components from real variables that went into the ordination. In principal coordinates analyses, these axes were created to reproduce the distances between samples, and we've lost the variables that the distances originally came from.

Summary Code for Microbiome Analyses

QIIME Outputs

- Metadata file containing the covariates information from the samples (txt format) + OTU table with absolute or relative number of species observed per sample (biom format)
- OTU tables + Phylogeny tree > Alpha.rarefaction_py = core diversity (compares alpha diversity indices such as the observed species number or the Chao1 diversity index)
- OTU tables + Phylogeny tree > Beta.rarefaction_py = core diversity (compares relative diversity with distance matrices such as eucledian, chi square or unifracs distances)
- The file will need to be modified to contain the features (# of species / genus / function) in columns, and the sample or subject ID numbers in rows

1. Prepping the data

Loading the file

```
source /macqiime/configs/bash_profile.txt

cp -R /macqiime/QIIME/qiime_tutorial ~/qiime_tutorial

cd ~/qiime_tutorial

ls –lh

less Fasting_Example.fna

grep -c ">" Fasting_Example.fna
```

Demultiplexing & Denoising

```
split_libraries.py -m Fasting_Map.txt -f Fasting_Example.fna -q Fasting_Example.qual -o
split_library_output/

ls -lh split_library_output/

less split_library_output/seqs.fna

denoise_wrapper.py -i Fasting_Example.sff.txt -f split_library_output/seqs.fna -m
Fasting_Map.txt -o denoiser/

inflate_denoiser_output.py -c denoiser/centroids.fasta -s denoiser/singletons.fasta -f
split_library_output/seqs.fna -d denoiser/denoiser_mapping.txt -o inflated_denoised_seqs.fna
```

2. Making an OTU table (picking otus and assigning taxonomy)

Picking OTUs with UCLUST then picking representative sequences per OTUs

```
pick_otus.py -i inflated_denoised_seqs.fna
```

```
less uclust_picked_otus/inflated_denoised_seqs_otus.txt
```

```
pick_rep_set.py -i uclust_picked_otus/inflated_denoised_seqs_otus.txt -f
inflated_denoised_seqs.fna -o rep_set.fna
```

```
grep -c ">" rep_set.fna
```

Assigning Taxonomy + Making the OTU table (need the original OTU clusters & Taxonomy Results) + Making a phylogenic Tree (needed for Alpha and Beta Diversity)

```
assign_taxonomy.py -i rep_set.fna -o taxonomy_results/
```

```
make_otu_table.py -i uclust_picked_otus/inflated_denoised_seqs_otus.txt -t
taxonomy_results/rep_set_tax_assignments.txt -o otu_table.biom
```

```
less otu_table_tabseparated.txt
```

```
summarize_taxa.py -i otu_table.biom -o taxonomy_summaries/
```

```
align_seqs.py -i rep_set.fna -o alignment/
```

```
filter_alignment.py -i alignment/rep_set_aligned.fasta -o alignment/
```

```
make_phylogeny.py -i alignment/rep_set_aligned_pfiltered.fasta -o rep_set_tree.tre
```

3. Alpha and Beta Diversity Analyses

Rarefaction

```
multiple_rarefactions.py -i otu_table.biom -m 20 -x 100 -s 20 -n 10 -o rare_20-100/
```

Alpha Diversity (need the rarefaction file + the phylogenetic tree)

```
alpha_diversity.py -i rare_20-100/ -o alpha_rare/ -t rep_set_tree.tre -m
observed_species,chao1,PD_whole_tree
```

```
collate_alpha.py -i alpha_rare/ -o alpha_collated/
```

Beta Diversity (need the rarefaction file + the phylogenetic tree)

```
jackknifed_beta_diversity.py -i otu_table.biom -o jackknifed_beta_diversity/ -e 90 -m
Fasting_Map.txt -t rep_set_tree.tre
```

Connecting to and using MSI resources (From Part_12)

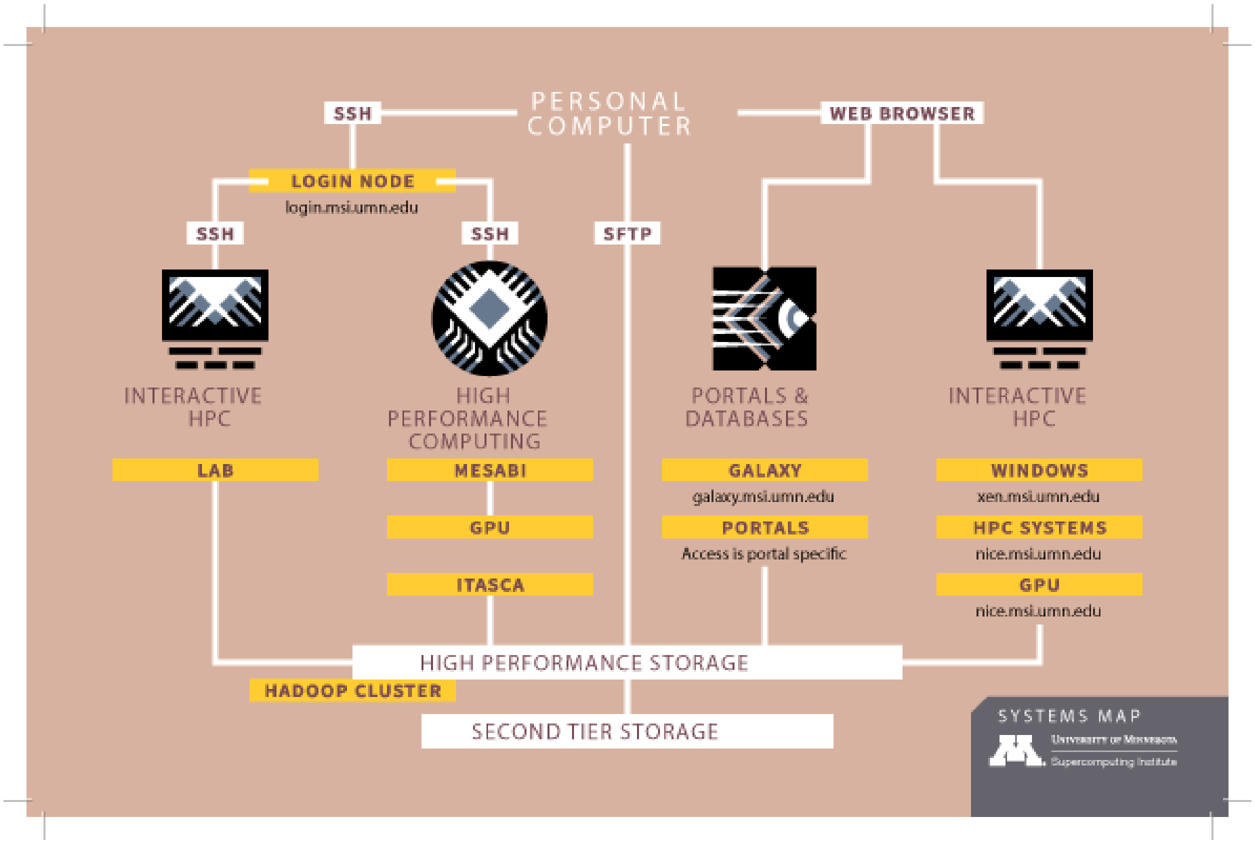
https://www.msi.umn.edu/content/connecting-hpc-resources

https://www.msi.umn.edu/support/faq

https://www.msi.umn.edu/getting-started

Summary

MSI provides access to a variety of High-Performance Computing (HPC) systems designed to tackle a wide range of computational needs. This document will show you how you can access all of MSI computational resources from your personal computer.



Skills/Software Needed

If you are connecting from a computer running Windows OS you will need to download and install PuTTY. OSX and Linux does not require additonal software to connect.

Connecting to MSI

SSH (Use Secure Shell)

SSH, also known as Secure Socket Shell, is a network protocol that provides administrators with a secure way to access a remote computer. SSH also refers to the suite of utilities that implement the protocol. Secure Shell provides strong authentication and secure encrypted data communications between two computers connecting over an insecure network such as the Internet. SSH is widely used by network administrators for managing systems and applications remotely, allowing them to log in to another computer over a network, execute commands and move files from one computer to another.

The main features of SSH include secure remote logins, terminal emulation, fully integrated secure file transfers, and secure tunneling of X11 traffic. Many University of Minnesota systems require secure telnet and FTP connections.

Install a SSH Client

Unix/Linux

- OpenSSH is freely usable and re-usable by everyone under a BSD license. OpenSSH is available from http://www.openssh.org/portable.html.

Windows

- Cygwin provides a Windows port of the most current OpenSSH tools.

- PuTTY is a full-featured, SSH compliant client for Windows. You can get Putty at http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html, as well as PSCP, a tool for encrypted remote file transfer

- WinSCP is a freeware Secure Copy (SCP) protocol client for Windows, which can be used to connect to an SSH server, mainly to UNIX machines, for file transfer using the SCP service. http://winscp.net/eng/index.php

- SecureCRT software requires you to purchase a license. http://www.vandyke.com/products/securecrt/

Mac OS X

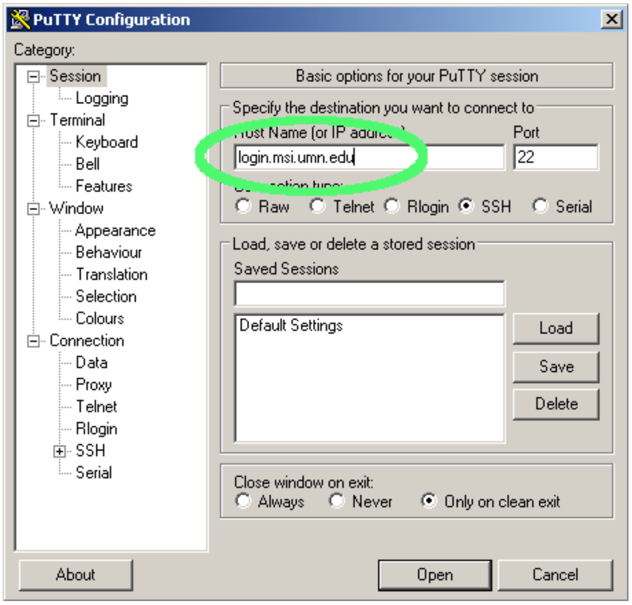
- Mac OS X comes with OpenSSH pre-installed. Open the Terminal application from inside the Utilities folder, and then type "ssh *user@server_name_or_IP_address*" to get started.

- Portable OpenSSH is available at http://www.openssh.com/portable.html at no charge. The portable OpenSSH follows development of the official version, but releases are not synchronized. Portable releases are marked with a 'p' (e.g. 2.3.0p1).

Connecting to MSI with Windows OS

Configure PuTTY to connect to MSI

- Double-click on the PuTTY icon and a configuration window will pop up.
- In the Host Name box type: **login.msi.umn.edu**



PuTTY Configuration

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

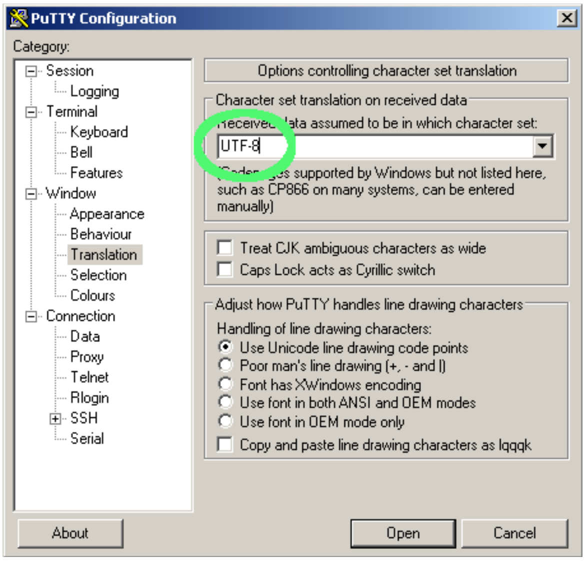
Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

- In the left panel, under Window, select Translation
- Change Receive data assumed to bin in which character set to : UTF-8



PuTTY Configuration

Options controlling character set translation

Character set translation on received data

Received data assumed to be in which character set: UTF-8

(Codepages supported by Windows but not listed here, such as CP866 on many systems, can be entered manually)

□ Treat CJK ambiguous characters as wide

□ Caps Lock acts as Cyrillic switch

Adjust how PuTTY handles line drawing characters

Handling of line drawing characters:

● Use Unicode line drawing code points

○ Poor man's line drawing (+, - and |)

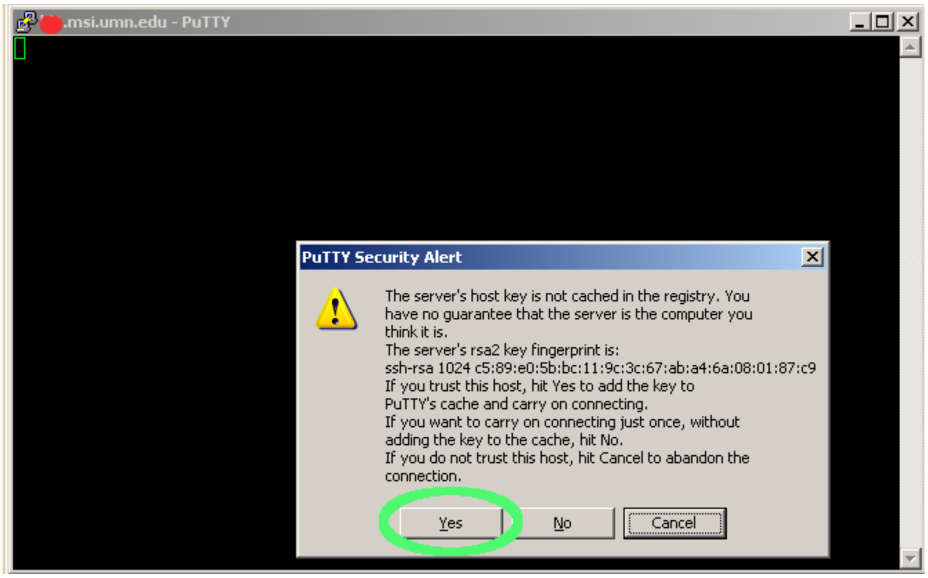
○ Font has XWindows encoding

○ Use font in both ANSI and OEM modes

○ Use font in OEM mode only

□ Copy and paste line drawing characters as lqqqk

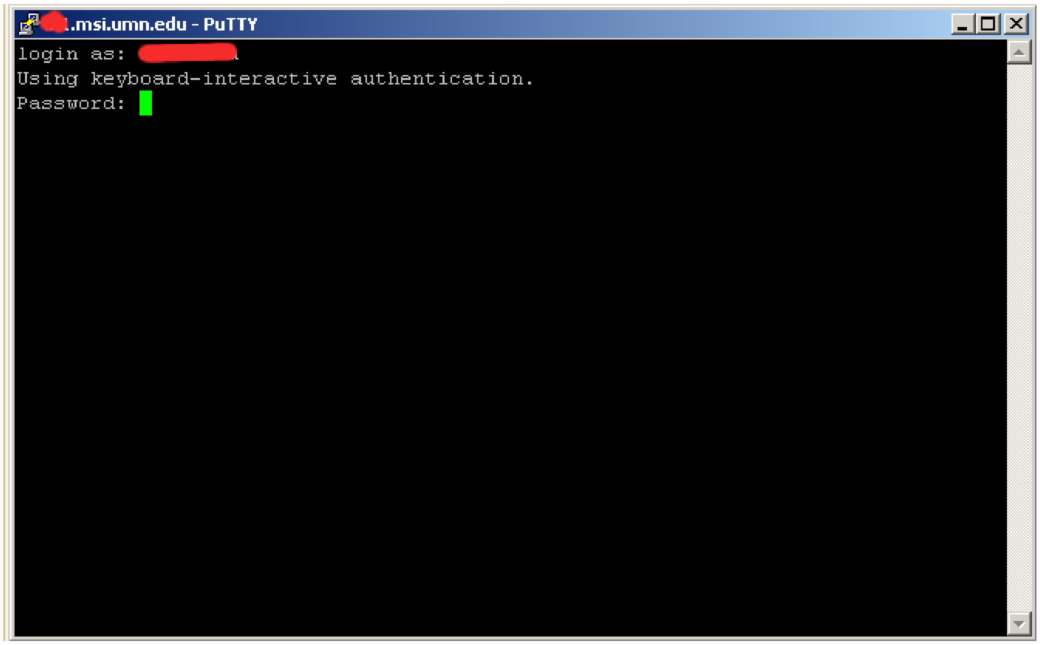
- Select Session in the left panel and in the Saved Session box type the name you would like to use to refer to this session. We suggest MSI server, then select Save.
- The next time you open PuTTY you will be able to recall a saved profile by clicking on the name and clicking Load then Open to start the session.
- The first time you connect to each MSI system you may be asked to cache the server fingerprint, select Yes.



PuTTY Security Alert

The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 1024 c5:89:e0:5b:bc:11:9c:3c:67:ab:a4:6a:08:01:87:c9
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

- When prompted enter your MSI username and password



```
login as: [redacted]
Using keyboard-interactive authentication.
Password:
```

- You are now connected to the MSI Login host (or bastion host). This host will let you view the directories but you can not submit jobs or run interactive computation on this host.

Connecting to MSI with OSX

Terminal is a preinstalled application which can be used to connect to MSI systems. Terminal can be found in the Utilities folder found in the Applications folder.

Connecting to MSI with Linux OS

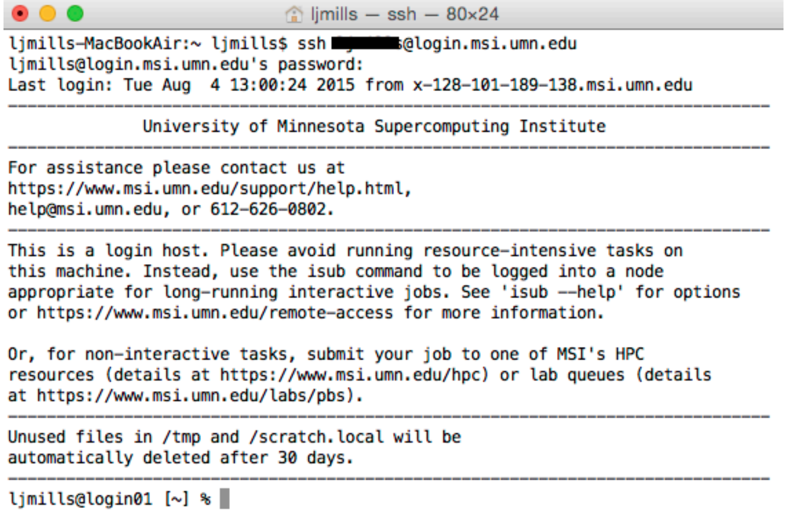
Terminal is a preinstalled application which can be used to connect to MSI systems.

For OSX and Linux OS Terminal Apps

Once you open the Terminal App type:

- ssh yourMSIusername@login.msi.umn.edu

- You will be prompted for your MSI password type it in then press enter. The terminal will not display your password as you type.
- The first time you log into MSI systems you will be asked if you would like to cache the server fingerprint, type yes and press return.



```
ljmills-MacBookAir:~ ljmills$ ssh [redacted]@login.msi.umn.edu
ljmills@login.msi.umn.edu's password:
Last login: Tue Aug  4 13:00:24 2015 from x-128-101-189-138.msi.umn.edu
-------------------------------------------------------------------------------
              University of Minnesota Supercomputing Institute
-------------------------------------------------------------------------------
For assistance please contact us at
https://www.msi.umn.edu/support/help.html,
help@msi.umn.edu, or 612-626-0802.
-------------------------------------------------------------------------------
This is a login host. Please avoid running resource-intensive tasks on
this machine. Instead, use the isub command to be logged into a node
appropriate for long-running interactive jobs. See 'isub --help' for options
or https://www.msi.umn.edu/remote-access for more information.

Or, for non-interactive tasks, submit your job to one of MSI's HPC
resources (details at https://www.msi.umn.edu/hpc) or lab queues (details
at https://www.msi.umn.edu/labs/pbs).
-------------------------------------------------------------------------------
Unused files in /tmp and /scratch.local will be
automatically deleted after 30 days.
-------------------------------------------------------------------------------
ljmills@login01 [~] % 
```

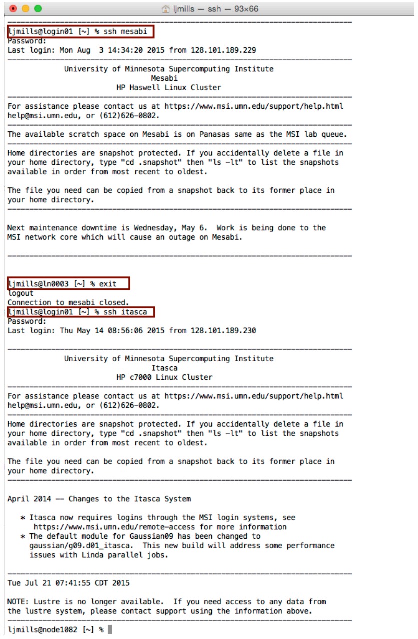
- You are now connected to the MSI Login host (or bastion host). This host will let you view the directories but you can not submit jobs or run interactive computation on this host.

Connecting to Mesabi and Itasca from Login Host

- Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh.

ssh mesabi or ssh itasca

- MSI prohibits moving directly from one system to another. If you are connected to Mesabi and would like to connect to Itasca you will first have to exit back to the Login Host, then ssh to the new system. Commands are highlighted with red boxes in the example below.



```
-----------------------------------------------------------------------------
ljmills@login01 [~] % ssh mesabi
Password:
Last login: Mon Aug  3 14:34:20 2015 from 128.101.189.229
-----------------------------------------------------------------------------
                University of Minnesota Supercomputing Institute
                                    Mesabi
                          HP Haswell Linux Cluster
-----------------------------------------------------------------------------
For assistance please contact us at https://www.msi.umn.edu/support/help.html
help@msi.umn.edu, or (612)626-0802.
-----------------------------------------------------------------------------
The available scratch space on Mesabi is on Panasas same as the MSI lab queue.
-----------------------------------------------------------------------------
Home directories are snapshot protected. If you accidentally delete a file in
your home directory, type "cd .snapshot" then "ls -lt" to list the snapshots
available in order from most recent to oldest.

The file you need can be copied from a snapshot back to its former place in
your home directory.
-----------------------------------------------------------------------------


Next maintenance downtime is Wednesday, May 6.  Work is being done to the
MSI network core which will cause an outage on Mesabi.

-----------------------------------------------------------------------------



ljmills@ln0003 [~] % exit
logout
Connection to mesabi closed.
ljmills@login01 [~] % ssh itasca
Password:
Last login: Thu May 14 08:56:06 2015 from 128.101.189.230

-----------------------------------------------------------------------------
                University of Minnesota Supercomputing Institute
                                    Itasca
                           HP c7000 Linux Cluster
-----------------------------------------------------------------------------
For assistance please contact us at https://www.msi.umn.edu/support/help.html
help@msi.umn.edu, or (612)626-0802.
-----------------------------------------------------------------------------
Home directories are snapshot protected. If you accidentally delete a file in
your home directory, type "cd .snapshot" then "ls -lt" to list the snapshots
available in order from most recent to oldest.

The file you need can be copied from a snapshot back to its former place in
your home directory.
-----------------------------------------------------------------------------

April 2014 -- Changes to the Itasca System

   * Itasca now requires logins through the MSI login systems, see
      https://www.msi.umn.edu/remote-access for more information
   * The default module for Gaussian09 has been changed to
     gaussian/g09.d01_itasca.  This new build will address some performance
     issues with Linda parallel jobs.

-----------------------------------------------------------------------------
Tue Jul 21 07:41:55 CDT 2015

NOTE: Lustre is no longer available.  If you need access to any data from
the lustre system, please contact support using the information above.
-----------------------------------------------------------------------------
ljmills@node1082 [~] % 
```

- Each time you move between systems you will see the welcome message associated with the system you are connecting to. These messages often contain useful information and should be read.
- **NOTE:** Like the login node Mesabi and Itasca have specific nomenclatures in the prompt. When connected to Mesabi your prompt will contain **ln** followed by some number when connected to Itasca your prompt will contain **node** followed by some numbers. This is a quick short cut to remind you which system you are connected to at any time.

Choosing a Job Queue

Summary

Most MSI systems use job queues to efficiently and fairly manage when computations are executed. A job queue is an automated waiting list for use of a particular set of computational hardware. When computational jobs are submitted to a job queue they wait in the queue in line until the appropriate resources become available. Different job queues have different resources and limitations. When submitting a job, it is very important to choose a job queue which has resources and limitations suitable to the particular calculation.

This document outlines factors to consider when choosing a job queue.   These factors are important when choosing where to place a job. This document is best used on all MSI systems and in conjunction with the Queues page that outlines the resource limitations for each queue.

Please note that Mesabi's "widest" queue requires special permission to use. Please submit your code for review at: **help@msi.umn.edu**.

Guidelines

There are several important factors to consider when choosing a job queue for a specific program or custom script. In most cases, jobs are submitted via PBS scripts as described in Job Submission and Scheduling.

Overall System

Each MSI system contains job queues managing sets of hardware with different resource and policy limitations. MSI currently has three primary systems: the newest supercomputer Mesabi, the supercomputer Itasca, and the Lab compute cluster. Mesabi is MSI's newest supercomputer, with the highest performance hardware, and a wide variety of queues suitable for many different job types. **Mesabi should be your first choice when doing any computation at MSI**. Itasca is a supercomputer with queues most suitable for multi-node jobs which will complete within 1-2 days. The Lab cluster is primarily for interactive software that is graphical in nature, and testing. Which system to choose depends highly on which system has queues appropriate for your software/script. The variety of queues on Mesabi will be suitable for most users, but the Queue page should be examined.

Job Walltime (walltime=)

The job walltime is the time from the start to the finish of a job (as you would measure it using a clock on a wall), not including time spent waiting to run. This is in contrast to cputime, which measures the cumulative time all cores spent working on a job. Different job queues have different walltime limits, and it is important to choose a queue with a sufficiently high walltime that enables your job to complete. Jobs which exceed the requested walltime are killed by the system to make room for other jobs. Walltime limits are maximums only, and you can always request a shorter walltime, which will reduce the amount of time you wait in the queue for your job to start. If you are unsure how much walltime your job will need start with the queues with shorter walltime limits and only move to others if needed.

Job Nodes and Cores (nodes=X:ppn=Y)

Many calculations have the ability to use multiple cores (ppn), or (less often) multiple nodes, to improve calculation speed. Certain job queues have maximum or minimum values for the number nodes and cores a job may use. If **Node Sharing** is enabled for a queue you can request fewer cores (ppn) than exist on an entire node. If Node Sharing is not enabled then you must request resources equivalent to a multiple of an entire node. All Itasca queues, and Mesabi's widest and large queues, **do not allow** Node Sharing**.**

Job Memory (mem=)

The memory which a job requires is an important factor when choosing a queue. The largest amount of memory (RAM) that can be requested for a job is limited by the memory on the hardware associated with that queue. Mesabi has two queues (ram256g and ram1t) with high memory hardware, the largest memory hardware is available through the ram1t queue. Itasca also has two queues with high memory hardware (sb128 and sb256).

User and Group Limitations

To efficiently share resources, many queues have limits on the number of jobs or cores a particular user or group may simultaneously use. If a workflow requires many jobs to complete, it can be helpful to choose queues which will allow many jobs to run simultaneously. Mesabi allows more simultaneous jobs to run than Itasca.

Special Hardware

Some queues contain nodes with special hardware, GPU accelerators and solid-state scratch drives being the most common. If a calculation needs to use special hardware, then it is important to choose a queue with the correct hardware available. Furthermore, those queues may require additional resources to be specified (e.g., GPU nodes require ":gpus=X").

Queue Congestion

At certain times particular queues may become overloaded with submitted jobs. In such a case, it can be helpful to send jobs to queues with lower utilization (node status). Sending jobs to lower utilization queues can decrease wait time and improve throughput. Care must be taken to make sure calculations will fit within queue limitations.

Accessing Software Resources

MSI provides access to approximately 400 software packages. To find out if a particular software package is installed, you can browse or search a list of the software packages that MSI provides under the Help and Documentation section of this website. The MSI webpage includes descriptions of software packages including example usage to help you get started.

Support Tiers

Software packages will be marked with a support tier -- either Primary, Secondary, or Minimal. Primary support is given to mature, popular scientific software important to a large fraction of the user base generally receive the highest level of support. These software are compiled, installed, benchmarked, and documented as a service to you. They are tested after major system upgrades, new versions are installed in a timely manner, and we expect to be able to offer advice to assist with the majority of inquiries. Conversely, Secondary or Minimally supported software are only useful to a very small number of users. These software may not be actively maintained, regularly updated, or tested after system updates. They may be removed without notice if usage is found to be too low to continue support.

How To Access Software

HPC and Interactive HPC Systems

To access software on Linux systems, use the module command to load the software into your environment. You can find the module name to load on the MSI software page for a package.

For example, if you want to use the 2014b version of MATLAB, you would type in your job script or at the command line:

```
% module load matlab/R2014b
% matlab
```

To find out what module versions are available, consult the software page or use the module avail command to search by name. For example to find out what versions of Python are available:

```
% module avail python
```

Windows Software

Windows-only software can be accessed and run via the Windows Virtual Environment (Citrix). Once a windows session is started you access software applications in the same manner as if you were using a physical Windows computer.

Need a Specific Package?

First try and install it yourself in your MSI home directory following the guidelines below:

Software Installation Guide

Installing software on any Unix platform can be challenging for inexperienced and veteran users alike.

While it is impossible to cover every issue you may experience, the following steps will allow you to compile and install in your own home directory many of the libraries and scientific applications that you will come across.

Introductory Compiling

Building a Python interpreter from its source code is straightforward, requiring only a C compiler, and will serve as an instructive example.

Installation materials are most commonly distributed as compressed tar files with the extension .tar.gz or .tar.bz.

These files can be untarred and unzipped with the tar -xzf and tar -xjf commands, respectively.

We have downloaded the file Python-2.7.2.tar.gz from www.python.org to our home directory, so the appropriate command is:

```
tar -xzf Python-2.7.2.tar.gz

cd Python-2.7.2
```

Your next step should always be to review the distributed files for a README file, which can be viewed in a text editor, or some other file that is named to indicate it contains installation instructions.

In many cases, and as is the case with Python, a configure script is included. In the simplest case, all that is necessary to compile the code is to run the configurescript, then make.

By default many codes will try to install to system directories you cannot access. The recommended installation method is to create a directory in your home directory for software installation.

Pwd (It will display a result such as /home/xe2/nlabello/Python-2.7.2)

mkdir /home/xe2/nlabello/software/python

./configure --prefix=/home/xe2/nlabello/software/python/

make install

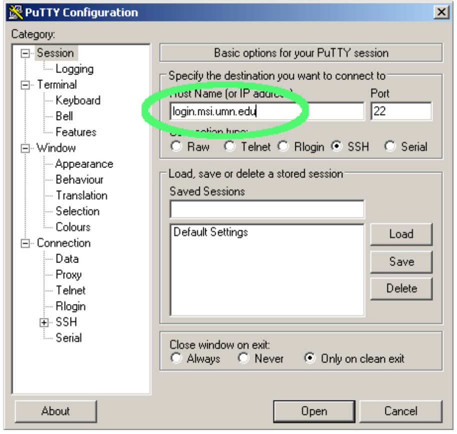
When the make install step completes you will have access to binary in /home/xe2/nlabello/software/python/bin

Installing QIIME

First Things First: Open the Terminal

Using QIIME on MSI

1. Download Putty on Windows (http://www.putty.org/) and configure it for MSI access



PuTTY Configuration

Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Saved Sessions: Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

9. Connecting to Mesabi and Itasca from Login Host. Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh

```
ssh mesabi or ssh itasca
```

10. Alternatively, connect to MSI using Terminal on Mac/Linux, located in Applications>Utilities>Terminal.app.

```
ssh yourusername@login.msi.umn.edu
```

11. Log in to an interactive node, making sure you have enough time and memory

```
isub -n nodes=1:ppn=4 -m 16GB -w 24:00:00
```

12. Load the QIIME module

```
module load qiime/1.9.1
```

MacQIIME Users Only:

If you are using MacQIIME, you will have to use your Mac OS X terminal, located in Applications>Utilities>Terminal.app.
Normally, when you run a command in the terminal, all the packages in MacQIIME will be invisible to your computer. In order to access macqiime, after starting a new terminal session you have to source the environment variables with the "macqiime" command. To source the QIIME environment variables and load up all the MacQIIME scripts in the PATH, you have to start the MacQIIME subshell:

```
macqiime
```

You should now be looking at a dollar sign ($) with a cursor after it. This is the command line interface for your computer. Here, you can type commands in order to "execute" programs. This Unix/Linux command line is a powerful place to analyze and process data. Everything you do in QIIME is executed through this command line interface. I'm going to assume you have some familiarity with the Unix command line, but I'll try to help you along the way with commands like **cp**, **cd**, **ls**, **pwd**, **less**, **nano**, etc.

**Note: Getting MacQIIME to work in El Capitan (OS 10.11)

This section is required only if you are running OS 10.11 (El Capitan) or higher. Apple added a new security feature in El Capitan that makes it impossible to install software into /usr/bin/ folder (that was where I was putting the “macqiime” sourcing script. I'll fix this in the next release of MacQIIME. In the meantime, we can work around it. The quick solution is to use this command, instead of the “macqiime” script:

```
source /macqiime/configs/bash_profile.txt
```

How do I transfer data between a Mac or Windows computer and MSI Unix computers?

For transferring data from Mac / Windows computers to MSI Unix computers, any **SFTP** or **SCP** software can be used. Connect to login.msi.umn.edu using your MSI username and password and you will be able to transfer files to and from your home directory and group shared directory. For Windows, we recommend WinSCP. For Mac OS X, we recommend using FileZilla.

If you are on the Unix side and need to pull data from your Windows U: drive, you can use smbclient:

$ smbclient -U 'MSI\*username*' -D *username* //falcon2/Users

From the prompt:

smb: \username\>

You can issue ftp style commands ('ls', 'get', 'cd', etc.). To get an entire directory recursively, use:

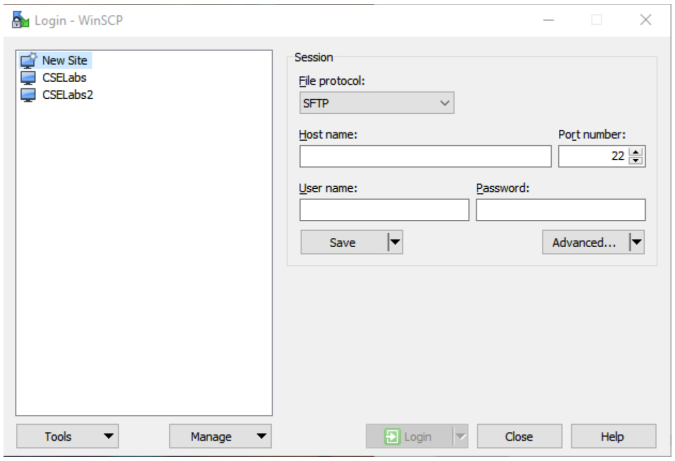
smb: \username\> prompt
smb: \username\> recurse
smb: \username\> mget *DirectoryName*

How do I upload files to my MSI home directory from Windows?

To upload files from your Windows machine to your home directory on a Unix machine, use WinSCP.

How do I use WinSCP to transfer data?

17. These directions explain how to transfer files back and forth between a Windows client and a Unix server. Please see the directions for remote access for an MSI Unix machine from a Windows client if you need to log in interactively.
18. Download and install WinSCP, a graphical SCP (secure copy protocol) file transfer client for Windows. These directions are based on version 5.7.5; upgrade if needed.
19. Double-click the WinSCP icon to reach this screen. If you just want to transfer files to or from your group home or lab scratch, you can skip steps 4 and 5 and proceed to step 6.



File protocol: SFTP

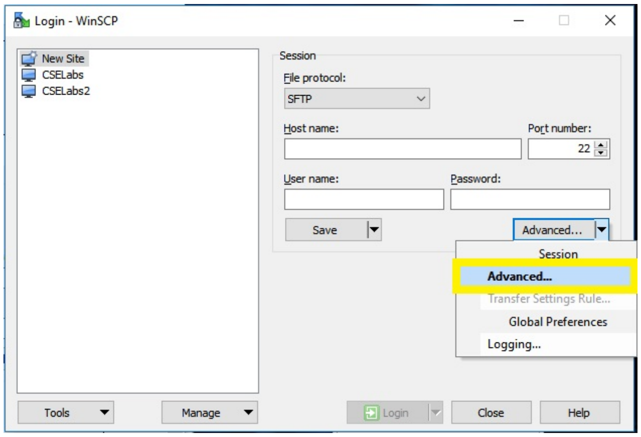
Host name:

Port number: 22

User name:

Password:

20. (**Optional:** if you want to transfer files elsewhere other than login.msi) Click the "Advanced Options" box in the lower left-hand corner:



File protocol: SFTP

Host name:

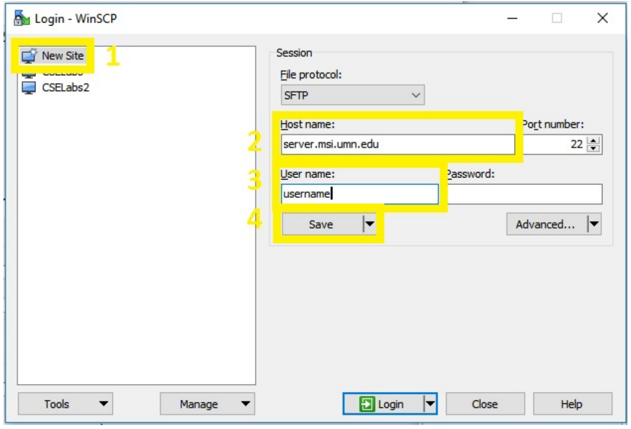
Port number: 22

User name:

Password:

21. (**Optional:** if you want to transfer files elsewhere other than login.msi) Under the Connection section, click on "Tunnel" to get to this screen and click "Connect through SSH tunnel". Fill in **login.msi.umn.edu** and your MSI username. Then click back in the left column on Session at the top.

22. (Mandatory for all cases) Enter the name of the MSI server you wish to use, such as itasca, cascade, hosting, etc, in this format: *server*.msi.umn.edu. (Do not use the actual name server.msi.umn.edu, which does not exist.) **If you skipped steps 4 and 5, the only option here is login.msi.umn.edu.** Enter your MSI username. Click "Save".



File protocol: SFTP

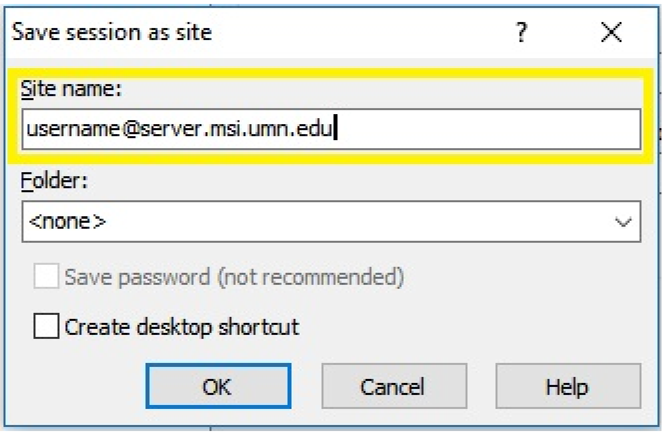
Host name: server.msi.umn.edu

Port number: 22

User name: username

Password:

23. Make sure the session is called something you want it to be called, then click "OK" on this screen:



Save session as site

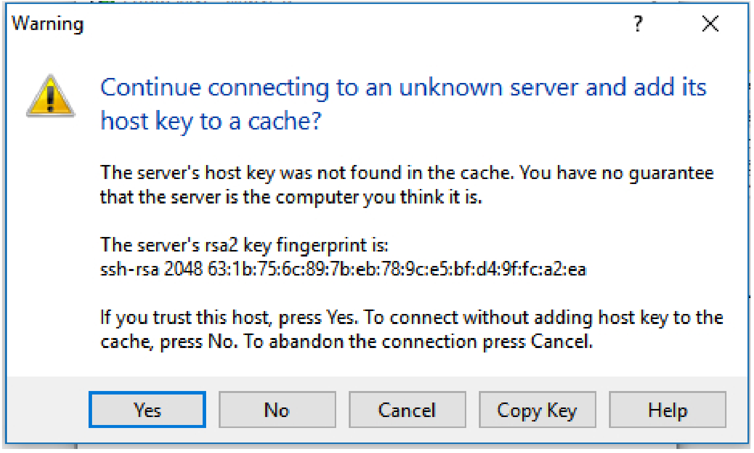
Site name:
username@server.msi.umn.edu

Folder:
<none>

☐ Save password (not recommended)

☐ Create desktop shortcut

24. The first time you connect you will get a "Warning" window concerning the server's key fingerprint. This is normal. Click "Yes". You may get this window twice.



Warning

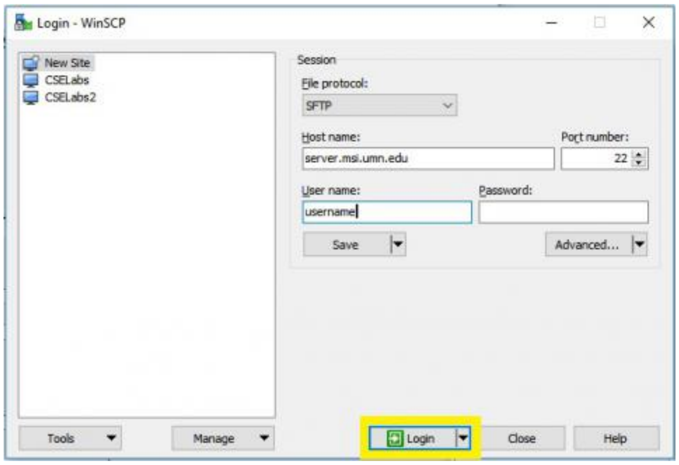
Continue connecting to an unknown server and add its host key to a cache?

The server's host key was not found in the cache. You have no guarantee that the server is the computer you think it is.

The server's rsa2 key fingerprint is:
ssh-rsa 2048 63:1b:75:6c:89:7b:eb:78:9c:e5:bf:d4:9f:fc:a2:ea

If you trust this host, press Yes. To connect without adding host key to the cache, press No. To abandon the connection press Cancel.

25. Login to the session



Login - WinSCP

Session

File protocol: SFTP

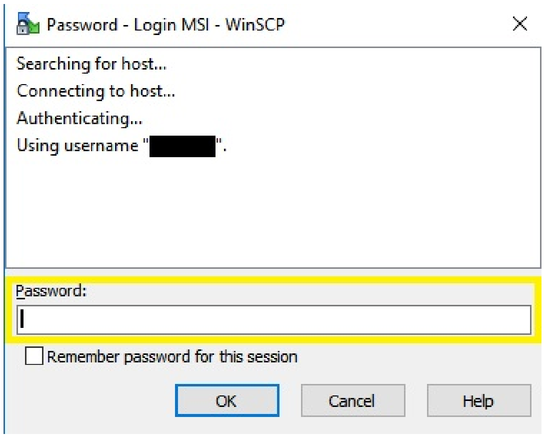
Host name: server.msi.umn.edu

Port number: 22

User name: username

Password:

26. You will now get a window asking for your password. After entering your password click "OK". Do not save your password! For alternatives to saving your password using ssh keys contact help@msi.umn.edu.

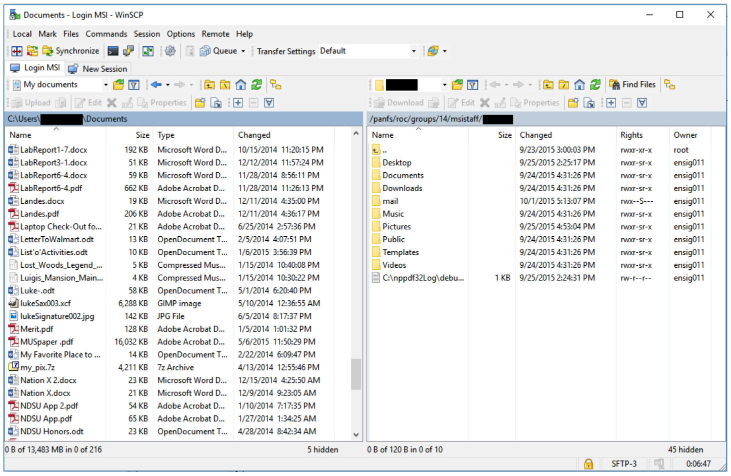


Searching for host...
Connecting to host...
Authenticating...
Using username "[redacted]".

Password:

□ Remember password for this session

27. You may get the "Warning" message again at this point.
28. You may be prompted for the password a second time.
29. Once connected to the remote system, a window will appear showing the local and remote systems.



Transfer Settings Default

C:\Users\[redacted]\Documents

Name	Size	Type	Changed
LabReport1-7.docx	192 KB	Microsoft Word D...	10/15/2014 11:20:15 PM
LabReport3-1.docx	51 KB	Microsoft Word D...	12/12/2014 11:57:24 PM
LabReport6-4.docx	59 KB	Microsoft Word D...	11/28/2014 8:56:11 PM
LabReport6-4.pdf	662 KB	Adobe Acrobat D...	11/28/2014 11:26:13 PM
Landes.docx	19 KB	Microsoft Word D...	12/11/2014 4:35:00 PM
Landes.pdf	206 KB	Adobe Acrobat D...	12/11/2014 4:36:17 PM
Laptop Check-Out fo...	21 KB	Adobe Acrobat D...	6/25/2014 2:57:36 PM
LetterToWalmart.odt	13 KB	OpenDocument T...	2/5/2014 4:07:51 PM
List'o'Activities.odt	10 KB	OpenDocument T...	1/6/2015 3:56:39 PM
Lost_Woods_Legend_...	5 KB	Compressed Mus...	1/15/2014 10:40:08 PM
Luigis_Mansion_Main...	4 KB	Compressed Mus...	1/15/2014 10:30:22 PM
Luke-.odt	58 KB	OpenDocument T...	5/1/2014 6:20:40 PM
lukeSax003.xcf	6,288 KB	GIMP image	5/10/2014 12:36:55 AM
lukeSignature002.jpg	142 KB	JPG File	6/5/2014 8:17:37 PM
Merit.pdf	128 KB	Adobe Acrobat D...	1/5/2014 1:01:32 PM
MUSpaper .pdf	16,032 KB	Adobe Acrobat D...	5/6/2015 11:50:29 PM
My Favorite Place to ...	14 KB	OpenDocument T...	2/22/2014 6:09:47 PM
my_pix.7z	4,211 KB	7z Archive	4/13/2014 12:55:46 PM
Nation X 2.docx	23 KB	Microsoft Word D...	12/15/2014 4:25:50 AM
Nation X.docx	21 KB	Microsoft Word D...	12/9/2014 9:23:05 AM
NDSU App 2.pdf	54 KB	Adobe Acrobat D...	1/10/2014 7:17:35 PM
NDSU App.pdf	65 KB	Adobe Acrobat D...	1/27/2014 1:34:25 AM
NDSU Honors.odt	23 KB	OpenDocument T...	4/28/2014 8:42:34 AM

0 B of 13,483 MB in 0 of 216   5 hidden

[redacted]

/panfs/roc/groups/14/msistaff/[redacted]

Name	Size	Changed	Rights	Owner
..		9/23/2015 3:00:03 PM	rwxr-xr-x	root
Desktop		9/25/2015 2:25:17 PM	rwxr-sr-x	ensig011
Documents		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Downloads		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
mail		10/1/2015 5:13:07 PM	rwx--S---	ensig011
Music		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Pictures		9/25/2015 4:53:04 PM	rwxr-sr-x	ensig011
Public		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Templates		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Videos		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
C:\nppdf32Log\debu...	1 KB	9/25/2015 2:24:31 PM	rw-r--r--	ensig011

0 B of 120 B in 0 of 10   45 hidden

SFTP-3   0:06:47

30. To move to the desired directories, use the navigation bars at the top. The local system is on the left and the remote system on the right. The toolbar at the bottom of the window contains function keys for performing copies, deletes, etc.
31. To perform an action, highlight the desired files or directories and press the appropriate function key (F5 for a copy). It will then prompt you for confirmation. Click "OK". Depending on the size of the file(s) and the speed of your network connection, an upload or download could take anywhere from a few seconds to several minutes.
32. When you are ready to disconnect from the remote system, press F10 or just exit from the WinSCP program.

*****Note**
I saved the microbiome analysis files under the following paths

C:\Users\onyea005\Documents\mice_tutorial
C:\Users\onyea005\Documents\qiime_tutorial

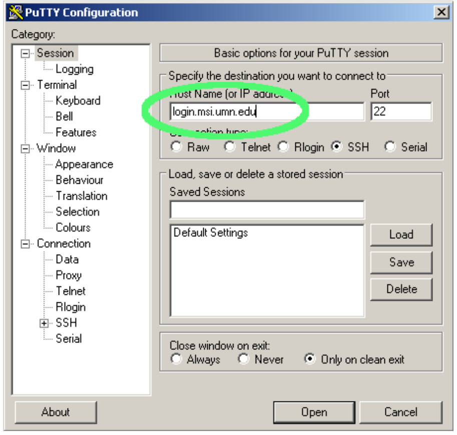
QIIME WORKFLOW QUICK GUIDE

Installing QIIME

First Things First: Open the Terminal

Using QIIME on MSI

1. Download Putty on Windows (http://www.putty.org/) and configure it for MSI access



Specify the destination you want to connect to

Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Close window on exit: ○ Always ○ Never ● Only on clean exit

13. Connecting to Mesabi and Itasca from Login Host. Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh (QIIME is currently loaded under Itasca so use ssh Itasca for QIIME analyses)

```
ssh mesabi or ssh itasca
```

14. Alternatively, connect to MSI using Terminal on Mac/Linux, located in Applications>Utilities>Terminal.app.

```
ssh yourusername@login.msi.umn.edu
```

15. Log in to an interactive node, making sure you have enough time and memory

```
isub -n nodes=1:ppn=4 -m 16GB -w 24:00:00
```

16. Load the QIIME module

```
module load qiime/1.9.1
```

MacQIIME Users Only:

If you are using MacQIIME, you will have to use your Mac OS X terminal, located in Applications>Utilities>Terminal.app.

Normally, when you run a command in the terminal, all the packages in MacQIIME will be invisible to your computer. In order to access macqiime, after starting a new terminal session you have to source the environment variables with the "macqiime" command. To source the QIIME environment variables and load up all the MacQIIME scripts in the PATH, you have to start the MacQIIME subshell:

```
macqiime
```

You should now be looking at a dollar sign ($) with a cursor after it. This is the command line interface for your computer. Here, you can type commands in order to "execute" programs. This Unix/Linux command line is a powerful place to analyze and process data. Everything you do in QIIME is executed through this command line interface. I'm going to assume you have some familiarity with the Unix command line, but I'll try to help you along the way with commands like **cp**, **cd**, **ls**, **pwd**, **less**, **nano**, etc.

**Note: Getting MacQIIME to work in El Capitan (OS 10.11)

This section is required only if you are running OS 10.11 (El Capitan) or higher. Apple added a new security feature in El Capitan that makes it impossible to install software into /usr/bin/ folder (that was where I was putting the “macqiime” sourcing script. I'll fix this in the next release of MacQIIME. In the meantime, we can work around it. The quick solution is to use this command, instead of the “macqiime” script:

```
source /macqiime/configs/bash_profile.txt
```

## Get your dataset: you can either download the dataset from the source, then use the cd command to move to the folder where the data is located as follows, or you can download the file through the terminal window with the wget command (followed by the unzip and mv commands to get the uncompressed data). We will show the cd method here:

```
cd ~/qiime_tutorial/
```

This changes the directory to user\qiime_tutorial, Now you can look at the contents of the directory using the **ls** command:

```
ls –lh
```

You can always check the location of your current directory (aka folder) using the **pwd** command. If you type:

```
pwd
```

*** **To copy any line code from the tutorial to the terminal window, select the code and hit CTRL + C, then in the control window right click and it will copy the code**
*** **To quit any command that is running in the terminal, hit q**

Files needed for a microbiome analysis

Raw Sequencing Data (.sff) for the Denoiser

This Denoiser step only applies to 454 Pyrosequencing data. It can be skipped, at your own risk. One problem with 454 pyrosequencing is that sequencing errors can give you effectively more OTUs than really are there. There are a number of strategies for dealing with this problem. The default in QIIME is to use a built-in program called Denoiser, which compares flowgrams (the raw sequencing data) of similar sequences to see if the differences between them may have been due to erroneous base calls. You can see some of this raw sequencing flowgram data by opening the Fasting_Example.sff.txt file in less. This is a text version of the "SFF" raw data format generated by the 454 pyrosequencing platform. It won't be terribly meaningful to you as it is, but it's always nice to have a feel for what all the data files look like.

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.

The primary file format for storing sequence data supported by QIIME is the FASTA format. The file Fasting_Example.fna is in FASTA format, indicated by the suffix ".fna" which stands for **F**ASTA **n**ucleic **a**cids (as opposed to amino acids which would have a suffix ".faa").

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.

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.

Microbiome Analysis Steps
(http://www.wernerlab.org/)

Validate the mapping file

First, you should ensure that your mapping file is formatted correctly with the validate_mapping_file.py script:

```
validate_mapping_file.py -m Fasting_Map.txt -o mapping_output
```

A file ending with _corrected.txt will also be created in the output directory, which will have a copy of the mapping file with invalid characters replaced by underscores. Reverse primers can be specified in the mapping file, for removal during the demultiplexing step. This is not required, but it is **strongly** recommended, as leaving in sequences following primers, such as sequencing adapters, can interfere with OTU picking and taxonomic assignment.

Demultiplex and quality filter reads

The next task is to assign the multiplexed reads to samples based on their nucleotide barcode (this is known as *demultiplexing*). This step also performs quality filtering based on the characteristics of each sequence, removing any low quality or ambiguous reads. To perform these steps we'll use split_libraries.py (Split libraries: http://qiime.org/scripts/split_libraries.html)

```
split_libraries.py -m Fasting_Map.txt -f Fasting_Example.fna -q Fasting_Example.qual -o
split_library_output/
```

The command line options for QIIME scripts can appear in any order. The flag **-m** is followed by the name of the mapping file, the flag **-f** is followed by the name of the FASTA file, **-q** is followed by the qual file, and **-o** is followed by the name you want to give to the output folder (a new folder that will be created by the script). *Please note: avoid using file names with spaces in them. In the command line, a space separates one argument from another! If you want to use files with spaces in the names, then you have to put quotes around the file name.*

Run that above command; it should only take a few seconds, after which you will see a new command line prompt appear. It is now done! *What did it do?* The script created the new directory you specified with the **-o** option. You can verify this with "**ls -lh**" -- there should be a new directory called split_library_output. You can see the contents within this directory using the ***ls*** command as well, by specifying the name of the directory to look in:

```
ls -lh split_library_output/
```

There should be three output files in that directory: histograms.txt, seqs.fna, and split_library_log.txt. The log file tells you stats on how your analysis worked out, including a table of the number of sequences assigned to each sample. Read it with *less*, like this:

```
less split_library_output/split_library_log.txt
```

*****Special Note*****

Reverse primer removal can be accomplished by adding the -z option. An example command using the mapping file with reverse primers described above:

```
split_libraries.py -m Fasting_Map_reverse_primers.txt -f Fasting_Example.fna -q Fasting_Example.qual -z truncate_only -o split_library_output_revprimers
```

If the number of sequences where the reverse primer is not identifiable is high, you should check the primer sequence to make sure it is in 5’->3’ orientation, or increase the number of mismatches allowed with --reverse_primer_mismatches. Data that are already demultiplexed can have reverse primers removed using the standalone script truncate_reverse_primer.py.

***** End Special Note*****

Denoise Seqs with Denoiser

This Denoiser step only applies to 454 Pyrosequencing data. It can be skipped, at your own risk.

One problem with 454 pyrosequencing is that sequencing errors can give you effectively more OTUs than really are there. There are a number of strategies for dealing with this problem. The default in QIIME is to use a built-in program called Denoiser, which compares flowgrams (the raw sequencing data) of similar sequences to see if the differences between them may have been due to erroneous base calls.
Our command (run this) is to use the script **denoise_wrapper.py** (*Warning: this step could take up to an hour. Once you start it, you may want to go get coffee or do something else for a bit*).

```
denoise_wrapper.py -i Fasting_Example.sff.txt -f split_library_output/seqs.fna -m Fasting_Map.txt -o denoiser/
```

Some of the options:
-i [filename] Specifies the sff.txt file name
-f [filename] Specifies the fasta file that was created by split_libraries.py
-m [filename] Sample mapping file with primers and barcodes
-o [name] Specifies a directory name for the output
This script created a new output directory called **denoiser**/ which contains a bunch of new files. The important ones are denoiser_mapping.txt, centroids.fasta and singletons.fasta. The singletons had no other sequences that matched up with them, and the centroids are representatives of clusters that collapsed into a single sequence following denoising. The denoiser_mapping.txt file maps which sequences fell into which centroids. To make use of these data, we have to "inflate" the results to create a new FASTA file that is a theoretically denoised version of the original. The script that does this for us is called **inflate_denoiser_output.py**

```
# Inflate denoiser output: http://qiime.org/scripts/inflate_denoiser_output.html
inflate_denoiser_output.py -c denoiser/centroids.fasta -s denoiser/singletons.fasta -f split_library_output/seqs.fna -d denoiser/denoiser_mapping.txt -o inflated_denoised_seqs.fna
```

Some of the options:
-c [file] The centroids.fasta file from denoise_wrapper.py
-s [file] The singletons.fasta file from denoise_wrapper.py
-f [file] The seqs.fna file from split_libraries.py
-d [file] The denoiser_mapping.txt file from denoise_wrapper.py
-o [file] The name of the new fasta file to be created

Check the contents of the newly created output file, inflated_denoised_seqs.fna. It should contain all the seqs from split_library_output/seqs.fna, but the sequences will be slightly different depending on to what extent they were denoised.

Pick Operational Taxonomic Units

OTUs (operational taxonomic units) are clusters of similar sequences. Much of the QIIME pipeline is concerned with analyzing OTUs rather than the full set of sequences. In this OTU-centric process, you are looking at samples as collections of OTUs, where two different samples may contain some of the same OTUs and some different OTUs. We'll eventually assign each OTU some data, such as taxonomy data, data as to how abundant it was in each sample, and data on the evolutionary history of each OTU compared to the others. There are a number of ways to bin sequences into these "clusters" of OTUs, a process called OTU picking. (Some folks may colloquially speak of OTUs with 97% similarity threshold as being "species," though this is just for the sake of communication where technical lingo would bog down the conversation). The default default OTU-picking method in QIIME is to use a program called **uclust**. The defaults in the QIIME script pick_otus.py for OTU-picking are to use uclust, and to use an identity threshold of 97% (i.e., sequences that are 97% similar should get binned into the same OTU together). You can see how to specify different options in the help info for pick_otus.py. We're going to use all the defaults, so all we have to specify is the input file name with the -i flag:

pick_otus.py -i inflated_denoised_seqs.fna

Notice that the fasta file we used as input was the output of our previous denoising step. (If you skipped denoising, the input for the -i option in this step could instead be **split_library_output/seqs.fna**) Our new command should create a new directory automatically named "uclust_picked_otus." Look inside that directory with *ls* and you should see three new files: a .uc file, a .log file, and a file called inflated_denoised_seqs_otus.txt. Let's look at the contents of that file:

ls ~/qiime_tutorial/uclust_picked_otus
less uclust_picked_otus/inflated_denoised_seqs_otus.txt

It's another tab-separated file. This format is called an OTU map. It is an intermediate file format, and maybe not very useful to you as raw data. It lists each OTU (the first column is the ID of the OTU, counting up from 0) followed by the IDs of all the sequences that fell into that OTU. Working with all of your sequences is cumbersome, if you are, for example, building an alignment, assigning taxonomy, etc. Since we have OTUs picked, what we really would like to work with are a collection of just one representative sequence per OTU. We can get this using pick_rep_set.py.

```
pick_rep_set.py -i uclust_picked_otus/inflated_denoised_seqs_otus.txt -f
inflated_denoised_seqs.fna -o rep_set.fna
```

Some of the options:
-i [file] The OTU mapping file from pick_otus.py
-f [file] The fasta file that was used to pick OTUs
-o [file] The name of the output FASTA file to create
How many sequences are in our resulting FASTA file? We can do **grep -c ">" rep_set.fna** to see: mine has 209 sequences in it (one for each OTU). What does the file format look like? Check it out using *less*:

Assigning Taxonomy

Taxonomy is assigned to high-throughput 16S rRNA gene sequences using some kind of comparison to a reference database. The most basic approach might be doing a BLAST search and taking the top hit. However, this doesn't account for redundancy or give you any idea of your confidence or specificity. The default algorithm in QIIME is the RDP Classifier. This is a Bayesian classifier that incorporates information about different places in the taxonomic tree where the sequence might fit in, and it calculates the highest probability taxonomy that can be assigned with some specified level of confidence. This still uses a reference database, in this case called a training set. In MacQIIME, the default training set is the Greengenes (GG) reference database clustered at 97% identity. We're going to use the QIIME script assign_taxonomy.py as follows (may take a minute):

```
assign_taxonomy.py -i rep_set.fna -o taxonomy_results/
```

Some of the options:
  -i [file] Name of the fasta file to classify (usually OTU representative seqs)
  -o [name] Name of directory to create for output

To use a custom training set like the download from Greengenes, you have to reference it using the -t and -r options as described in the documentation (http://qiime.org/scripts/assign_taxonomy.html).
This created a new directory called taxonomy_results. Look in there with *ls* and you will see that it contains two files: a log file, and a txt file of results. The rep_set_tax_assignments.txt file contains an entry for each representative sequence, listing taxonomy to the greatest depth allowed by the confidence threshold (80% by default, can be changed with the -c option), and a column of confidence values for the deepest level of taxonomy shown. These data will be really useful, once we have them inside an OTU table!

Building an OTU Table

An OTU table is a form of your sequencing results that will finally be really useful to analyze in excel, visualize, etc. It is a table giving the count of the number of sequences in each OTU, for each sample, and the taxonomy of that OTU. Super! We generate an OTU table with a script called **make_otu_table.py** as follows

```
make_otu_table.py -i uclust_picked_otus/inflated_denoised_seqs_otus.txt -t taxonomy_results/rep_set_tax_assignments.txt -o otu_table.biom
```

Some of the options:
-i [file] The OTU map output from pick_otus.py
-t [file] The taxonomy file from assign_taxonomy.py
-o [name] The name of the output file

This should create an OTU table file called otu_table.biom. Check it out in *less* - it is in an XML/JSON format called biom (new as of QIIME 1.5.0). If you want to look at the OTU counts per sample in a spreadsheet, you'll first have to convert the biom OTU table into a normal text table. This can be done with the biom script as follows:

```
biom convert -i otu_table.biom -o otu_table_tabseparated.txt --to-tsv --header-key taxonomy --output-metadata-id "ConsensusLineage"
```

Now, the new output file otu_table_tabseparated.txt is something you can easily import into a spreadsheet. You can also read it more easily in the terminal:

```
less otu_table_tabseparated.txt
```

Summarizing Taxonomy

The OTU table is pretty useful, but we can make other tables of taxonomic summaries that we can analyze in a spreadsheet as well. To summarize taxonomy in terms of relative abundance, we can use the script **summarize_taxa.py** as follows:

```
summarize_taxa.py -i otu_table.biom -o taxonomy_summaries/
```

Some of the options:
-i [file] The input OTU table with included taxonomy information
-o [name] The name of the directory to be created and filled with goodies

Check out the contents of the taxonomy_summaries folder. You can open any of these files in a spreadsheet and graph taxonomy summaries for your samples! The numbers are in terms of relative abundance. In other words, they're already normalized. The code L2, L3, etc, refers to the level of taxonomy summarized in the file. If you're impatient and don't want to dive into the spreadsheet just yet, QIIME can make some simple graphs to summarize the results. The script that does this is called **plot_taxa_summary.py**. For example, you can summarize taxonomy at the Class level (L3) like this:

```
plot_taxa_summary.py -i taxonomy_summaries/otu_table_L3.txt -o taxonomy_plot_L3/
```

Open up the newly created bar_charts.html file in your web browser to see some graphed data.

Make a Multiple Sequence Alignment

To test the evolutionary distance between your OTUs, taxonomy assignment is not the best way to go, as it depends too much on whether the taxonomy database had representatives of everything in your sample, and it depends on whether or not the taxonomic hierarchy used accurately reflects evolution. Instead, it's better to build a real phylogenetic tree, and to do that we'll need to first align our sequences. We'll be using a reference alignment to align our sequences. In MacQIIME, this reference alignment is located at /macqiime/greengenes/core_set_aligned.fasta.imputed and QIIME already knows where it is, so the command line you need is actually quite simple (Also true in the VB, but it's located in /data/greengenes_core_sets/). The script we're using is called **align_seqs.py** and we use it as follows:

```
align_seqs.py -i rep_set.fna -o alignment/
```

Some of the options:
-i [file] The fasta file of queries, usually your representative seqs
-o [name] The name of the new directory that should be created
-t [file] The template file to use (*used default GG core in our command above; didn't have to specify)*

You can see the alignment results in that alignment/ directory, file name rep_set_aligned.fasta. This alignment contains lots of gaps, and it includes hypervariable regions that make it difficult to build an accurate tree. So, we'll filter it. Filtering an alignment of 16S rRNA gene sequences can involve a Lane mask. In MacQIIME, this Lane mask for the GG core is located at /macqiime/greengenes/lanemask_in_1s_and_0s and MacQIIME already knows where it is. (Also true in the VB, but it's located in /data/greengenes_core_sets/). The script you use to filter an alignment is **filter_alignment.py** as follows:

```
filter_alignment.py -i alignment/rep_set_aligned.fasta -o alignment/
```

Some of the options:
-i [file] The fasta file of queries, usually your representative seqs
-o [name] The name of the new directory that should be created
-m [file] The Lane mask file to use (*used default GG Lane mask in our command above; didn't have to specify*)

This created a new file in the alignment/ directory called rep_set_aligned_pfiltered.fasta -- this is the file we can use to build a phylogenetic tree! If you want to visually check the alignment, I suggest using a free program called SeaView (http://pbil.univ-lyon1.fr/software/seaview.html) to open the rep_set_aligned_pfiltered.fasta file.

Build a phylogenetic tree

Okay, let's make a tree out of that alignment! This is actually quite easy in QIIME, using the **make_phylogeny.py** script (which uses the FastTree approximately maximum likelihood program, a good model of evolution for 16S rRNA gene sequences). The input for this script is our filtered alignment.

```
make_phylogeny.py -i alignment/rep_set_aligned_pfiltered.fasta -o rep_set_tree.tre
```

How do you look at this tree? You could try something like FigTree, or TreeViewX, but actually Topiary Explorer (http://topiaryexplorer.sourceforge.net/) may be a better option - it is meant to be able to import your QIIME OTU table and mapping file to display data as well as the tree.

Perform Multiple Rarefactions

You may think of diversity, or species richness, as "the number of species" that are present in the system. That *is* the general definition of diversity. However, the deeper you sequence, the more species you will find. That is problematic, especially if you gathered 500 reads from one sample and only 100 reads from another sample. You would expect to find more species if you sequenced 5x as many reads! To account for this, we perform an *in-silico* (i.e., on your computer) experiment called **rarefaction**. A rarefaction is a random collection of sequences from a sample, with a specified depth (number of sequences). For example, a rarefaction with a depth of 75 reads per sample is a simulation of what your sequencing results would look like if you sequenced exactly 75 reads from each sample. To look at alpha diversity systematically, we can perform many rarefactions: at multiple depths and repeat many times at each depth. In QIIME, this task is performed on your OTU table. The QIIME script multiple_rarefactions.py takes your OTU table and makes a folder full of many OTU tables, all of which are repeats of rarefactions at specific depths. Let's use **multiple_rarefactions.py** as follows:

```
multiple_rarefactions.py -i otu_table.biom -m 20 -x 100 -s 20 -n 10 -o rare_20-100/
```

Some options for multiple_rarefactions.py:

```
   -i [file]  The input OTU table file
   -m [number]  The lowest rarefaction depth in the series of depths
   -x [number]  The highest rarefaction depth in the series of depths
   -s [number]  The step size to increment from the low to high depths
   -n [number]  The number of replicates to perform at each depth
   -0 [name]   The name of the new directory to be created
```

From that command above, you can see that we are performing rarefactions starting at 20 seqs/sample, and stepping up to 100 seqs/sample in increments of 20. In other words, we'll perform rarefactions at 20, 40, 60, 80, and 100 seqs/sample. The **-n** option specifies that we'll do

10 replicates at each of those depths. It is important to chose the rarefaction depths based on how many total sequences per sample you have. For example, it would *not* make sense to do rarefactions from 20 to 100 seqs/sample if I had a larger data set with an average of 5000 seqs/sample. If I did that, I would be throwing out a lot of my data and statistical power! Run the command, and it will make that directory named **rare_20-100/**. If you look inside that new directory with *ls*, you will see that we've created 50 new files. Each of those files is a new OTU table with all the samples rarefied at the specified level! We're going to look at alpha diversity in those **rarefied** OTU tables, **not** in our original OTU table.

Calculate Alpha Diversity

There are many measures of alpha diversity. Depending on your ecological allegiances, you may have a preference for Chao1, Simpson's Diversity, Shannon Index, etc. These all measure different things, so it's important to think about what is most meaningful for your experiment, and your question. The QIIME script for calculating alpha diversity in samples is called **alpha_diversity.py**. There are many options for what metrics to use, and you can chose to run a bunch of metrics all at once if you like. All the possible alpha diversity metrics available in QIIME are listed here.

```
alpha_diversity.py -i rare_20-100/ -o alpha_rare/ -t rep_set_tree.tre -m
observed_species,chao1,PD_whole_tree
```

Some options:

```
   -i [directory]  The name of the directory containing rarefied OTU tables
   -o [name]  The name of the directory to create for output
   -t [file]  The file for your phylogenetic tree
   -m [list]  The list of metrics, separated with commas and no spaces
```

If you run the above command, it will calculate alpha diversity metrics for all of your rarefied OTU tables and place the results in a new directory called alpha_rare. The metric PD_whole_tree is Faith's Phylogenetic Diversity, and it is based on the phylogenetic tree. Basically, it adds up all the branch lengths as a measure of diversity. So, if you find a new OTU and it's closely related to another OTU in the sample, it will be a small increase in diversity. However, if you find a new OTU and it comes from a totally different lineage than anything else in the sample, it will contribute a lot to increasing the diversity. There are still a ton of separate files, and we need to "collate" them together into a nice, neat collection of results that are easy to graph.

Summarize the Alpha Diversity Data

The QIIME script collate_alpha.py takes the output directory from alpha_diversity.py as its input, and creates a *new* output directory containing files that are much easier to look at in a spreadsheet.

```
collate_alpha.py -i alpha_rare/ -o alpha_collated/
```

Take a look at the new files in the **alpha_collated/** folder-- they are each organized quite nicely for spreadsheet analysis. The observed_species and chao1 metrics don't seem to show us much in this particular experiment, but check out the data in PD_whole_tree.txt

Beta Diversity

##**Compare Samples to Each Other & Visualize the results of beta diversity analysis**
Beta diversity is a term for the comparison of samples to each other. A beta diversity metric does not calculate a value for each sample. Rather, it calculates a distance between a pair of samples. If you have many samples (in this tutorial we have nine samples), a beta diversity metric will return a matrix of the distances of all samples to all other samples. If you have experience in phylogenetics, you may know that a distance matrix can be visualized as a tree. Distance matrices can also be visualized as a graph of points, a network, or any other creative method you can come up with. In this tutorial, we'll use principal coordinates analysis to visualize distances between samples on an x-y-z plot.

Sequencing depth can have an effect on beta diversity analysis, just as it does on alpha diversity. The effect of sequencing depth is not as easy to visualize, and it depends a lot on what distance metric you chose. The safest bet is always to chose a single sequencing depth that is less than the total number of sequences in your smallest sample, and rarefy all your samples at that same depth. For now, we'll start with the workflow script that helps us with our task at hand: beta diversity analysis. Note: Once you get more advanced in QIIME, you may want to set a lot of custom options. Because of the complexity of a workflow script, many of the options have to be set by supplying an input file called a parameters file (with the -p option). You don't need a parameters file, though. If one is not specified, all the steps of the workflow will happen with relatively common default settings.

We're going to use the workflow script jackknifed_beta_diversity.py to do our beta diversity analysis. This script does the following steps:

- Compute a beta diversity distance matrix from the full data set
- Perform multiple rarefactions at a single depth
- Compute distance matrices for all the rarefied OTU tables
- Build UPGMA trees for all the rarefactions
- Compare all the trees to get consensus and support values for branching
- Perform principal coordinates analysis on all the rarefied distance matrices
- Generate plots of the principal coordinates

Let's start this following script, and we'll talk about all the parts to the script while it's running:

```
jackknifed_beta_diversity.py -i otu_table.biom -o jackknifed_beta_diversity/ -e 90 -m Fasting_Map.txt -t rep_set_tree.tre
```

Some of the options for jackknifed_beta_diversity.py:

```
   -i [file]  The OTU table file
   -o [name]  Choose a name for the output directory to be created
```

-e [number] The depth for even rarefactions
-m [file] The sample mapping file
-t [file] The phylogenetic tree file

Let’s start with the -e option for the rarefaction depth. This script performs rarefaction at only one depth. The idea is that we want to create a large collection of distance matrices that we can do statistics on. For this to work, we need to choose a depth that is significantly smaller than the number of seqs in the smallest sample. If any samples have fewer seqs than the rarefaction depth, they will be left out! Also, if the rarefaction depth is too high, then every rarefied OTU table will be the same, and there won't be any differences between all our distance matrices. I chose 90 seqs/sample as our depth, because most samples have a little under 150 seqs/sample total. Going even lower may have been advisable, perhaps to 75 seqs/sample or less, but this is a pretty small data set to start with so I just made a compromise.

Let's go back and look at that parameters file... there are lots of variables in there that you can set. You'll notice that most of them are not set to anything. Also, many of them have nothing to do with our jackknifed_beta_diversity.py workflow. That's okay. The idea is that you can set up one parameters file for all your workflows in a project, and any variables you leave blank in the parameters file will just be set to their default values.

Okay, my jackknifed_beta_diversity.py workflow is done; there are LOTS of new files that it created in that jackknifed_beta_diversity/ folder! There is one sub-folder full of rarefied OTU tables, and three separate sub-folders for each of the three different distance metrics we chose! Let's look in the unweighted UniFrac folder. It contains a folder called "3d_plots," (Note: This has changed to "emperor_pcoa_plots" in QIIME 1.8 and above) and in there is an html file named index.html. Double-click on this html file to open it in your browser.

Those are the results of our principal coordinates analysis! Each point represents one of the samples. The important thing to realize here is that the axes are meaningless. This is different from principal components analysis, which you may be more familiar with. If you do principal components analysis, you get out axes that have weighted components from real variables that went into the ordination. In principal coordinates analyses, these axes were created to reproduce the distances between samples, and we've lost the variables that the distances originally came from. You'll notice that samples PC.634, PC.635, and PC.636 are all really close to each other. Because the distances between samples were calculated using unweighted UniFrac, this means that those three samples have communities with very similar overall phylogenetic trees. And, looking back into our mapping file (Fasting_Map.txt), we can see that these three samples were all experimental samples (fasted mice microbiomes)!

How can we visually look for patterns in this graph? In the upper right corner of the KiNG viewer, you'll see a list of all the variables that were in your mapping file. The variable you select will determine the color scheme for the graph. Let's select the Treatment variable, and we'll select the "unscaled" version for now (see inset image). (The unscaled plot is squared off, while the scaled version would make the higher inertia axes longer). It looks like there is definitely a separation of the fasted mice microbiomes from the controls. Do you see an image like the one below on your system?

Notice that each data point consists of a central point, surrounded by a semi-transparent cloud. The clouds represent the variation in UniFrac results from all those rarefactions we did. This demonstrates that the separation of these samples holds up to subsampling, which is a nice thing to check. That means it isn't driven by one or two low abundance singlets. If you drag the graph around with your mouse, it will rotate in 3D (adding a third principal coordinate) - give it a try! There are a bunch of other scripts you can use to analyze these distance matrices. Let's try a few.

##Distance Statistics

Let's use the script dissimilarity_mtx_stats.py to calculate means and standard deviations for our unweighted unifrac distance matrices. Remember that we created 100 distance matrices! I'm going to set our output to a new folder that is closer to the root, so we can access it more easily. It's important to chose descriptive names, to you know later on which folders contain what kinds of data.

```
dissimilarity_mtx_stats.py -i jackknifed_beta_diversity/unweighted_unifrac/rare_dm/ -o
unweighted_unifrac_stats/
```

The input (-i) directory is the directory containing the distance matrices - this directory is named rare_dm, and is located under the corresponding metric's folder. I named our output directory "unweighted_unifrac_stats" so that, later on, I will know what the data are in that directory. Notice that there are three files in there: means.txt, medians.txt, and stdevs.txt. If you wanted to know the mean and s.d. of the distance between two samples, you could get those values from these matrices. In our particular experiment, however, we have two categories, and several samples in each category. What we'd really like to know is: are the samples in an individual category closer to each other than they are to samples outside the category? To test this question, QIIME has a script called make_distance_boxplots.py - let's run it on our "means.txt" distance matrix!

```
make_distance_boxplots.py -m Fasting_Map.txt -o distance_boxplots -d
unweighted_unifrac_stats/means.txt -f Treatment --save_raw_data
```

The --save_raw_data flag is nice; you may want to graph the data yourself. In this case, there are only two Treatment categories, so the "Control vs Fast" comparison is the same as the "All between Treatment" comparison. There are more details and examples on how you can compare distances in this tutorial on qiime.org.

##Workflow Scripts

QIIME has some special built-in scripts that are categorized as **workflow scripts**. These workflow scripts take an analysis that would normally be a long series of steps, and performs all the steps in a series. All you have to do, as a user, is run the one workflow script. For example, for our previous section on alpha diversity analysis we had to run three different scripts in a series. There is actually a workflow script that can do all these steps for us in one command: alpha_rarefaction.py. And, for our process [tutorial pages C and D] of picking OTUs, picking a reference set of sequences, assigning taxonomy, building an OTU table, aligning our sequences, filtering the alignment, and building a phylogenetic tree are all combined into another single workflow script called pick_otus_through_otu_table.py. (Note that this workflow script does *not* do the denoising step, though. Therefore, you could use the following workflow commands to speed up the analysis

I'm also assuming you still have that qiime_parameters.txt file with you in the ~/qiime_tutorial/ directory (downloaded from http://www.wernerlab.org/teaching/qiime/overview/f) and saved a copy into the qiime folder on PC. Now, let's run the OTU table workflow. It doesn't need much as input: just the inflated_denoised_seqs.fna file that you created from the inflate denoiser output process. Before you start analyzing your own sequences, look closely at all those variables in the qiime_parameters.txt file. You may want to tweak them! For example, if you want to set up your own taxonomy training set, perhaps based on the Greengenes OTUs, there are a couple of variables that will need to be set for that. Or, if you want to use a different alignment method, tree building method, etc, it will all be controlled through giving values to the variables in that parameters file. Good luck!

# Split libraries: http://qiime.org/scripts/split_libraries.html

split_libraries.py -m Fasting_Map.txt -f Fasting_Example.fna -q Fasting_Example.qual -o split_library_output/

# Denoise: http://qiime.org/scripts/denoise_wrapper.html

denoise_wrapper.py -i Fasting_Example.sff.txt -f split_library_output/seqs.fna -m Fasting_Map.txt -o denoiser/

# Inflate denoiser output: http://qiime.org/scripts/inflate_denoiser_output.html

inflate_denoiser_output.py -c denoiser/centroids.fasta -s denoiser/singletons.fasta -f split_library_output/seqs.fna -d denoiser/denoiser_mapping.txt -o inflated_denoised_seqs.fna

# OTU Table WORKFLOW: http://qiime.org/scripts/pick_otus_through_otu_table.html
pick_otus_through_otu_table.py -i inflated_denoised_seqs.fna -p qiime_parameters.txt -o PickOTUsWorkflow/

I've set the output directory to be called PickOTUsWorkflow/. Look in that directory and you'll find a whole bunch of files that should look very familiar. Alignments, a tree, an OTU table, taxonomy information, the whole nine yards. Awesome!

Compute alpha diversity and generate alpha rarefaction plots

Community ecologists are often interested in computing *alpha* (or the *within-sample*) diversity for samples or groups of samples in their study. Here, we will determine the level of alpha diversity in our samples using QIIME's alpha_rarefaction.py workflow, which performs the following steps:

1. Generate rarefied OTU tables (multiple_rarefactions.py)
2. Compute measures of alpha diversity for each rarefied OTU table (alpha_diversity.py)
3. Collate alpha diversity results (collate_alpha.py)
4. Generate alpha rarefaction plots (make_rarefaction_plots.py)

We can next run alpha_rarefaction.py, which requires the OTU table (-i) and phylogenetic tree (-t) from above, and the parameters file we just created:

```
alpha_rarefaction.py -i otus/otu_table.biom -m Fasting_Map.txt -o arare -p
alpha_params.txt -t otus/rep_set.tre
```

Compute beta diversity and generate ordination plots

In addition to *alpha* (or *within-sample*) diversity, community ecologists are often interested in computing *beta* (or the *between-sample*) diversity between all pairs of samples in their study. Beta diversity represents the explicit comparison of microbial (or other) communities based on their composition. Beta diversity metrics thus assess the differences between microbial communities. The fundamental output of these comparisons is a square, hollow matrix where a "distance" or dissimilarity is calculated between every pair of community samples, reflecting the dissimilarity between those samples. The data in this distance matrix can be visualized with analyses such as Principal Coordinates Analysis (PCoA) and hierarchical clustering.

Here, we will calculate beta diversity between our 9 microbial communities using the default beta diversity metrics of weighted and unweighted UniFrac, which are phylogenetic measures used extensively in recent microbial community sequencing projects. To perform this analysis, we will use the beta_diversity_through_plots.py workflow, which performs the following steps:

1. Rarefy OTU table to remove sampling depth heterogeneity (single_rarefaction.py)
2. Compute beta diversity (beta_diversity.py)
3. Run Principal Coordinates Analysis (principal_coordinates.py)
4. Generate Emperor PCoA plots (make_emperor.py)

We can run the beta_diversity_through_plots.py workflow with the following command, which requires the OTU table (-i) and tree file (-t) from above, the metadata mapping file (-m), and the number of sequences per sample (-e, even sampling depth):

```
beta_diversity_through_plots.py -i otus/otu_table.biom -m Fasting_Map.txt -o
bdiv_even146 -t otus/rep_set.tre -e 146
```

Jackknifed beta diversity and hierarchical clustering

The jackknifed_beta_diversity.py workflow uses jackknife replicates to estimate the uncertainty in PCoA plots and hierarchical clustering of microbial communities. Many of the same concepts relevant to beta diversity and PCoA are used here. jackknifed_beta_diversity.py performs the following steps:

1. Compute beta diversity distance matrix from full OTU table and tree (beta_diversity.py). These are only used if the user passed `--master_tree full`.
2. Build UPGMA tree from full distance matrix (upgma_cluster.py). This is only used if the user passed `--master_tree full`.
3. Build rarefied OTU tables (multiple_rarefactions_even_depth.py).
4. Compute distance matrices from rarefied OTU tables (beta_diversity.py).
5. Build UPGMA trees from rarefied distance matrices (upgma_cluster.py).
6. Compare rarefied UPGMA trees and determine jackknife support for tree nodes (tree_compare.py and consensus_tree.py).
7. Compute PCoA on each rarefied distance matrix (principal_coordinates.py).
8. Compare rarefied PCoA plots from each rarefied distance matrix (make_emperor.py).

We can run the workflow with the following command:

jackknifed_beta_diversity.py -i otus/otu_table.biom -t otus/rep_set.tre -m Fasting_Map.txt -o jack -e 110

Statistical Analysis of Microbiomes

- Our goal is to identify enterotypes representative of dysbiosis associated with disease conditions. Enterotypes are not discrete clusters of bacteria, but rather modes in the relative distribution of bacteria within the community that can act as a fingerprint.
- We go from a data table of beta diversity counts (derived from unifracs using phylogenetic information) to a distance matrix (relative distances for beta diversity) to a principal component analysis for ordination.

Types of statistical analyses.

- T-test: compare 2 groups (i.e. abundance in OTUs between treatment and control groups)
- Anova: Compare more than 3 groups (Does the abundance in OTUs differ between any of the groups)
- Correlation: Compare the abundance in OTUs in relation to a continuous variable
- Regression: Compare the abundance in OTUs in relation to multiple variables

We have to be cautious about the distribution that the relative counts of bacteria (OUT counts) in our samples take, because using the wrong distribution can lead to false positive associations.

- Most microbiome OTU counts take a zero inflated negative binomial distribution.
- The zero inflated negative binomial distribution measures the number of successes in a sequence of independent Bernoulli trials before we reach a predetermined number of failures (called r). The greater the r value, the more normal the zero inflated negative binomial distribution will be.
- You can either run the korotkov statistical test to check whether the residuals for OUT counts are normally distributed, or you can log transform your counts to make the distribution look more normal.

Microbiome Analyses Notes

The areas of investigations for the human microbiome ae 1) the microbiota (what species are there) 2) the metagenome (what genes to these organisms have) and 3) the microbiome (how do the communities interact overall). We can use either 16S sequencing or Shotgun sequencing for these analyses.

Currently the field is investigating the association between the microbiota on the mucosal surface of the digestive tract (oral, esophageal, gut) and an inflammatory phenotype (systemic or localized inflammation)

- Common inflammatory markers in microbiome analyses include IL 10, CRP, IL6 TNF-alpha.
- Common Microbial biomarkers in microbiome analyses include LPS.

Beta diversity refers to the diversity between samples, individuals, or sample sites. You could either hold the sample site constant (i.e. gut) and look at differences between individuals, or you could hold the individuals constant and look at samples across sample sites (Pairs). One approach would be to use a Bray Curtis PCoA to get the principal coordinate axes, and run a linear regression on PC1 and PC2, or check whether the same changes in relative abundance occur in two different sites (oral and gut, for example).

Part 13_QIIME and R Microbiome Analysis_MSI_GUERRERO_NEGRO

Using MSI for bioinformatics analyses

Abridged Discovering the Microbiome analysis tutorial and MSI adaptation (GUERRERO_NEGRO dataset)

Complete Discovering the Microbiome analysis tutorial and MSI adaptation (GUERRERO_NEGRO dataset)

Statistical Analysis Procedures (Using correlation, GLM and the EdgeR statistical package)

Step 1: Extract the OTU table from the QIIME command line to R

Before loading data into R, this QIIME command must be run on the command line to collapse OTU counts into genus (L6) count tables. Our goal is to get the finalized table below:

Genus table (extract OTU counts: you will have multiple rows as outcome variable, one for each OTU corresponding to a particular genus, with counts for how many times the OTU was observed per sample) + Metadata (covariate information)

	OTU 1 (Bacteroides)	OTU 2 (Firnicutes)	OTU 3 (Fusobacterium)	Age	Sex	treatment
Sample 1	1	1	0			
Sample 2	0	1	1			
Sample 3	0	1	1			
Total	1	3	2			

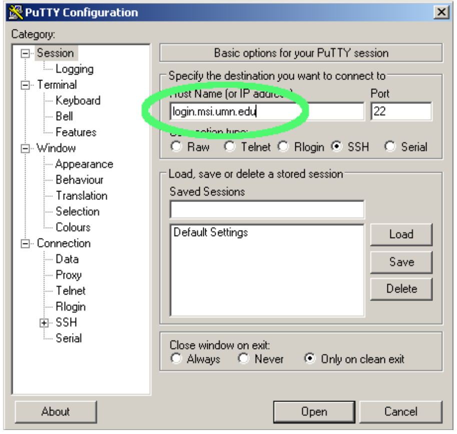
Accessing the Discovering The Microbiome Course Dataset

Installing QIIME

First Things First: Open the Terminal

Using QIIME on MSI

1. Download Putty on Windows (http://www.putty.org/) and configure it for MSI access



Specify the destination you want to connect to

Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Close window on exit: ○ Always ○ Never ● Only on clean exit

17. Connecting to Mesabi and Itasca from Login Host. Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh

```
ssh mesabi or ssh itasca
```

18. Alternatively, connect to MSI using Terminal on Mac/Linux, located in Applications>Utilities>Terminal.app.

```
ssh yourusername@login.msi.umn.edu
```

19. Log in to an interactive node, making sure you have enough time and memory

```
isub -n nodes=1:ppn=4 -m 16GB -w 24:00:00
```

20. Load the QIIME module

```
module load qiime/1.9.1
```

MacQIIME Users Only:

If you are using MacQIIME, you will have to use your Mac OS X terminal, located in Applications>Utilities>Terminal.app.
Normally, when you run a command in the terminal, all the packages in MacQIIME will be invisible to your computer. In order to access macqiime, after starting a new terminal session you have to source the environment variables with the "macqiime" command. To source the QIIME environment variables and load up all the MacQIIME scripts in the PATH, you have to start the MacQIIME subshell:

macqiime

You should now be looking at a dollar sign ($) with a cursor after it. This is the command line interface for your computer. Here, you can type commands in order to "execute" programs. This Unix/Linux command line is a powerful place to analyze and process data. Everything you do in QIIME is executed through this command line interface. I'm going to assume you have some familiarity with the Unix command line, but I'll try to help you along the way with commands like **cp**, **cd**, **ls**, **pwd**, **less**, **nano**, etc.

***Note: Getting MacQIIME to work in El Capitan (OS 10.11)*

This section is required only if you are running OS 10.11 (El Capitan) or higher. Apple added a new security feature in El Capitan that makes it impossible to install software into /usr/bin/ folder (that was where I was putting the “macqiime” sourcing script. I'll fix this in the next release of MacQIIME. In the meantime, we can work around it. The quick solution is to use this command, instead of the “macqiime” script:

source /macqiime/configs/bash_profile.txt

Running QIIME on class data

Go to https://github.com/danknights/mice8992-2016

The course code browser is available at

http://metagenome.cs.umn.edu/microbiomecodebrowser/doc/index.html

You can also run QIIME on the global gut data in the Course repository. First download the repository and data:

Copy the file to /Users/chigu142000/mice_tutorial
For example, move to the globalgut-66-adults data directory:

```
cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd globalgut-66-adults
```

* You can always check the location of your current directory (aka folder) using the **pwd** command. If you type:

```
pwd
```

This directory contains sequences from 66 adults in 3 countries (Yatsunenko et. al. Nature 2012). Then run OTU picking and core QIIME analyes:

```
# pick closed reference OTUs time (need to extract the seq.fna file)

pick_closed_reference_otus.py -i seqs.fna -o closedref

# run core QIIME diversity analyses on closed-reference OTU table

time core_diversity_analyses.py -i closedref/otu_table.biom -m map.txt -o closedref/corediv -e 500 --tree_fp closedref/97_otus.tree -v

# pick de novo OTUs

time pick_de_novo_otus.py -i seqs.fna -o denovo

# run core QIIME diversity analyses on de novo OTU table

time core_diversity_analyses.py -i denovo/otu_table.biom -m map.txt -o denovo/corediv -e 500 --tree_fp denovo/rep_set.tre
```

You can then open the “index.html” file in the output folder in your favorite web browser, and click to see the results of the analysis. In Chrome you may need to open Chrome first, and then choose “File–>Open File…”. For the Global Gut data set, click on “index.html” next to PCoA plot (weighted_unifrac). Then click the “Colors” tab in the upper right and select “COUNTRY” by which to color the points.

Getting usage info from QIIME scripts

First let’s move to the Guerrero Negro data directory. This directory contains sequences from different depths in the Guerrero Negro microbial in Mexico (Harris JK **et al.** ISME J. 2013 Jan;7(1):50-60).

We are going to run closed reference OTU picking, but first let’s get usage information and list of options (be sure QIIME is loaded first)

```
pick_closed_reference_otus.py -h
```

pick_closed_reference_otus.py is a “workflow” script that runs other scripts.

Ask pick_closed_reference_otus.py to print the other commands it would run. The -f tells it to force overwriting the directory closedref, in case that directory is already there. The -r tells it where the reference sequences are. The -t tells it where the reference taxonomy map is.

```
pick_closed_reference_otus.py -i seqs.fna -o closedref -f -r ../ref/greengenes/97_otus.fasta -t ../ref/greengenes/97_otu_taxonomy.txt -w
```

```
## pick_otus.py -i seqs.fna -o closedref/uclust_ref_picked_otus -r ../ref/greengenes/97_otus.fasta -m uclust_ref  --suppress_new_clusters

## make_otu_table.py -i closedref/uclust_ref_picked_otus/seqs_otus.txt -t ../ref/greengenes/97_otu_taxonomy.txt -o closedref/otu_table.biom
```

Picking OTUs

Move to the Guerrero Negro data directory. This directory contains sequences from different depths in the Guerrero Negro microbial in Mexico (Harris JK **et al.** ISME J. 2013 Jan;7(1):50-60).

```
 # To be run on the command line

cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd guerreronegro
```

Picking closed reference OTUs with QIIME

Now actually run the script. The -v tells it to be “verbose”, printing updates as it runs.
Use time to report how long it takes.

```
time pick_closed_reference_otus.py -i seqs.fna -o closedref -r ../ref/greengenes/97_otus.fasta -t ../ref/greengenes/97_otu_taxonomy.txt -f –v

##Pick OTUs (needed to unzip the 97_OTU file first)

pick_otus.py -i seqs.fna -o closedref/uclust_ref_picked_otus -r /Users/chigu142000/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otus.fasta -m uclust_ref  --suppress_new_clusters

## Make OTU table

make_otu_table.py -i closedref/uclust_ref_picked_otus/seqs_otus.txt -t /Users/chigu142000/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otu_taxonomy.txt -o closedref/otu_table.biom
```

Find out how many sequences were assigned (“Total count”) by uclust_ref (QIIME default)

```
biom summarize-table -i closedref/otu_table.biom > closedref/stats.txt

head closedref/stats.txt
```

Picking closed-reference OTUs with NINJA-OPS

Compare to NINJA-OPS. Installation instructions are at https://github.com/GabeAl/NINJA-OPS. (move to Users/chigu142000/mice_tutorial/NinjaOps/)

***Note NINJA-OPS is much faster.

```
time python Users/chigu142000/mice_tutorial/NinjaOps/NINJA-OPS-master/bin/ninja.py -i seqs.fna -o ninja
```

Find out how many sequences were assigned (“Total count”) by NinjaOps

```
biom summarize-table -i ninja/ninja_otutable.biom > ninja/stats.txt

head ninja/stats.txt
```

Try NINJA-OPS on the Global Gut data set. It can process 1 million sequences in the Global Gut data set in under 20 seconds on a Macbook.

```
cd ../globalgut time python Users/chigu142000/mice_tutorial/NinjaOps/NINJA-OPS-master/bin/ninja.py -i seqs.fna -o ninja
```

Picking de novo OTUs with QIIME

Run the workflow script, pick_de_novo_otus.py with -w to print the basic commands it would run.

```
cd ../guerreronegro

pick_de_novo_otus.py -i seqs.fna -o openref -f -w
```

Now actually run the script. Note that the OTU picking step is quick on this data set. The other steps will take a long time (more than 10 minutes). So instead, kill the script by typing ctrl-c.

```
pick_de_novo_otus.py -i seqs.fna -o openref -f -v
```

Instead let's just run the pick_otus.py step, with several different OTU picking methods.

Method 1: uclust

```
time pick_otus.py -i seqs.fna -o uclust -m uclust

make_otu_table.py -i uclust/seqs_otus.txt -o uclust/otu_table.biom

biom summarize-table -i uclust/otu_table.biom | head -n 5
```

Method 2: swarm

```
time pick_otus.py -i seqs.fna -o swarm -m swarm make_otu_table.py -i swarm/seqs_otus.txt -o s
warm/otu_table.biom biom summarize-table -i swarm/otu_table.biom | head -n 5
```

Method 3: cdhit

```
time pick_otus.py -i seqs.fna -o cdhit -m cdhit make_otu_table.py -i cdhit/seqs_otus.txt -o cdhit/
otu_table.biom biom summarize-table -i cdhit/otu_table.biom | head -n 5
```

Method 4: sumaclust

```
time pick_otus.py -i seqs.fna -o sumaclust -m sumaclust make_otu_table.py -i sumaclust/seqs_ot
us.txt -o sumaclust/otu_table.biom biom summarize-table -i sumaclust/otu_table.biom | head -n 5
```

Customizing workflow scripts

We can customize workflow scripts using a parameter file. You will need some kind of text editor to edit the parameter file, such as Notepad or Notepad++ for Windows, TextWrangler or Sublime for Mac, or Emacs or vi for the Mac/Linux command line. For more information, please see the official QIIME explaining parameter files here: http://qiime.org/documentation/qiime_parameters_files.html.

For example, let’s assume we want to customize the OTU picking method used by (pick_otus.py) as part of the pick_de_novo_otus.pyworkflow, changing it from the default uclust to swarm. To find the parameter name, we can either run pick_otus.py -h or read the description at qiime.org: http://qiime.org/scripts/pick_otus.html (found by Googling **QIIME pick_otus.py**). We see that the parameter we need is called otu_picking_method, and that the valid options are sortmerna, mothur, trie, uclust_ref, usearch, usearch_ref, blast, usearch61, usearch61_ref, sumaclust, swarm, prefix_suffix, cdhit, uclust.

Therefore we need to add pick_otus:otu_picking_method swarm to a text file. This can be done with emacs as follows, if you have emacs on your system:

```
emacs parameters-swarm.txt
```

Then add these lines to the top of the file:

```
pick_otus:otu_picking_method swarm  pick_otus:similarity .99
```

Then save the file with ctrl-x ctrl-s, then exit with ctrl-x ctrl-c.

Now we can run the workflow script with the custom parameters file:

```
pick_de_novo_otus.py -i seqs.fna -o swarm99 -p parameters-swarm.txt
```

Making OTU tables human-readable

You can convert OTU tables to tab-delimited output using biom convert:

```
biom convert -i closedref/otu_table.biom -o closedref/otu_table.txt --to-tsv
```

The output file can now be opened in Excel.

Follow-up exercises

1. How can you use filter_otus_from_otu_table.py to determine the number of singleton OTUs produced by each OTU picking method?
2. How many OTUs remain from each method after filtering singletons?
3. Run OTU picking on another data set from EBI or QIITA.
4. Try (loading OTU tables into R)[]

Loading Microbiome Data

All of the code in this page is meant to be run in **R** unless otherwise specified.

After you have installed QIIME and generated an OTU table, you can load the OTU table into R.

Install biom package if not installed.

```
install.packages('biom',repo='http://cran.wustl.edu')
```

R - ***Installing R on your system and adding the following libraries has become much more important in the latest QIIME versions 1.9+.*** There are now many neat features in QIIME that require R. If you don't have R installed, you can get it from here. *Please note* that even if you installed R and these libraries previously for MacQIIME 1.8.0, you should still upgrade to/install the latest version of R, at least version **3.1.2**, and re-install all these R packages to get everything working. Make sure the R executable is in your PATH. You will have to open R and install additional packages: in the R command line, do the following, in the following order, one line at a time. **(*** Figure out how to pull out the script window in R for MAC: Open R > Open a document editor, type your code there, then copy and paste into the R code window as needed)**

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("DESeq2")
biocLite("biomformat")
biocLite("metagenomeSeq")

install.packages('randomForest')
install.packages('ape')
install.packages('vegan')
install.packages('optparse')
install.packages('gtools')
install.packages('klaR')
install.packages('RColorBrewer')
```

*** it seems that you need to load the **biocLite("Biomformat") package instead of the install.packages("biom") or the** install.packages('biom',repo='http://cran.wustl.edu')

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("biomformat")
```

Load biom package

```
library('biomformat')
```

Getting Information on a Dataset

There are a number of functions for listing the contents of an object or dataset.

```
# list objects in the working environment
ls()
# list the variables in mydata
names(mydata)
# list the structure of mydata
str(mydata)
# list levels of factor v1 in mydata
levels(mydata$v1)
# dimensions of an object
dim(object)
# class of an object (numeric, matrix, data frame, etc)
class(object)
# print mydata
mydata
# print first 10 rows of mydata
head(mydata, n=10)
# print last 5 rows of mydata
tail(mydata, n=5)
```

Convert BIOM file to JSON format.

If you have data in a "new" BIOM format (HDF5), you first need to convert to JSON format first.

The following code is to be run on the command line.

```
# make a JSON-formatted OTU table for loading into R

cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd globalgut-66-adults

biom convert -i otu_table.biom -o otu_table_json.biom --to-json
```

Load global gut data using biom package

```
gg.otus.biom <- read_biom('../data/globalgut-66-adults/otu_table_json.biom')

*** The fix was to convert the file to an HDF5 format instead of the JSON format using

biom convert -i otu_table.biom -o otu_table_hdf5.biom --to-hdf5

***Then instead of using the basic read_biom R function in the tutorial
```

```
g.otus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/data/globalgut-66-adults/otu_table_hdf5.biom')

use the read_hdf5_biom import file in R

gg.otus.biom <- read_hdf5_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_hdf5.biom')

***or you can use this version too! (with the full path name)

gg.otus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_json2.biom')
```

Extract data matrix (OTU counts) from biom table (need the json file, as the HDF5 file bugs out using biom_data)

```
gg.otus <- as.matrix(biom_data(gg.otus.biom))

# transpose so that rows are samples and columns are OTUs

gg.otus <- t(gg.otus)
```

Plot histogram of sample depths

```
depths <- rowSums(gg.otus)

hist(depths,breaks=30)
```

Plot histogram of OTU frequencies

```
otu.counts <- colSums(gg.otus > 0)

hist(otu.counts,breaks=30)
```

Remove OTUs present in < 10% of samples

```
gg.otus <- gg.otus[,colMeans(gg.otus > 0) >= .1]

depths <- rowSums(gg.otus)

dim(gg.otus)
```

Re-plot histogram of OTU frequencies now that we removed singletons

```
otu.counts <- colSums(gg.otus > 0)

hist(otu.counts,breaks=30)
```

Remove any samples with very low depth

```
sort(depths)[1:10]
```

Load mapping file

```
gg.map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/map.txt',sep='\t',head=T,row=1,check=F,comment='')
```

Ensure those mapping file and OTU tables contain the sample samples in the same order

```
sample.ids <- intersect(rownames(gg.otus), rownames(gg.map))

# might as well put the samples in alphabetical order

sample.ids <- sort(sample.ids)

# in R you can subset using sample IDs or numerical indices. Most languages only use indices.

gg.otus <- gg.otus[sample.ids,]

gg.map <- gg.map[sample.ids,]

dim(gg.otus)

dim(gg.map)
```

Beta Diversity (Guerrero Negro)

All of the code in this page is meant to be run in **R** unless otherwise specified.

Install biom package and vegan package if not installed.

*** it seems that you need to load the **biocLite("Biomformat") package instead of the install.packages("biom") or the** install.packages('biom',repo='http://cran.wustl.edu')

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("biomformat")
install.packages('vegan')
```

Load the packages, load data

```
library('biomformat')

library('vegan')

# load biom file

otus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerrero negro/otu_table_json.biom')

# Extract data matrix (OTU counts) from biom table
```

```
otus <- as.matrix(biom_data(otus.biom))

# transpose so that rows are samples and columns are OTUs

otus <- t(otus)

# load mapping file

map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerreronegro/
map.txt',sep='\t',head=T,row=1,check=F,comment='')
```

It is extremely important to ensure that your OTU table and metadata table sample IDs are lined up correctly.

```
# see rownames of map and otus

rownames(map)

rownames(otus)

# find the overlap

common.ids <- intersect(rownames(map), rownames(otus))

 # get just the overlapping samples

otus <- otus[common.ids,]

map <- map[common.ids,]
```

See dimensions of OTU table

```
dim(otus)
```

See dimensions of mapping file

```
dim(map)
```

Get three different distances metrics

```
# get Euclidean distance

d.euc <- dist(otus)

# get Bray-Curtis distances (default for Vegan)

d.bray <- vegdist(otus)

# get Chi-square distances using vegan command

# we will extract chi-square distances from correspondence analysis

my.ca <- cca(otus)
```

```
d.chisq <- as.matrix(dist(my.ca$CA$u[,1:2]))
```

Now run principal coordinates embedding on the distance metrics

```
# Run PCoA (not PCA)
pc.euc <- cmdscale(d.euc, k=2)
# Bray-Curtis principal coords
pc.bray <- cmdscale(d.bray,k=2)
# get first two dimensions of chi-square coordinates:
pc.chisq <- my.ca$CA$u[,1:2]
```

Plot Euclidean distances with gradient colors

```
# makes a gradient from red to blue
my.colors <- colorRampPalette(c('red','blue'))(10)
# plot Euclidean PCoA coords using color gradient
# based on layer (1...10) *** There was no layer variable in the mapping file (PS it did, I initially
 pulled the global gut files instead of the guerreronegro ones)
names(map)
layer <- map[,'LAYER']
plot(pc.euc[,1], pc.euc[,2], col=my.colors[layer], cex=3, pch=16)
```

Plot Bray-Curtis distances with gradient colors

```
# Plot Bray-Curtis PCoA
plot(pc.bray[,1], pc.bray[,2], col=my.colors[layer], cex=3, pch=16)
```

Plot Chi-square distances with gradient colors

```
# Plot Chi-square PCoA
plot(pc.chisq[,1], pc.chisq[,2], col=my.colors[layer], cex=3, pch=16)
```

Visualizing UniFrac distances

Calculate UniFrac distances in QIIME

```
# Note: This command is on the command line, not in R # (load macqiime if necessary)
```

```
beta_diversity.py -i otu_table.biom -o beta -t /Users/chigu142000/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otus.tree
```

Load UniFrac distances, calculate PCoA

```
# load unweighted and weighted unifrac

d.uuf <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerreronegro/beta/unweighted_unifrac_otu_table.txt', sep='\t',head=T,row=1, check=F)

d.wuf <- read.table('//Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerreronegro/beta/weighted_unifrac_otu_table.txt', sep='\t',head=T,row=1, check=F)

# ensure that these last two matrices have the same samples in the  # same order as the metadata table

d.uuf <- d.uuf[common.ids, common.ids]

d.wuf <- d.wuf[common.ids, common.ids]

# get first two dimensions of unifrac PCoA:

pc.uuf <- cmdscale(d.uuf, k=2)

pc.wuf <- cmdscale(d.wuf, k=2)
```

Plot unweighted UniFrac distances with gradient colors (***There was no Layer variable in the mapping file)

```
plot(pc.uuf[,1], pc.uuf[,2], col=my.colors[layer], cex=3, pch=16)
```

Plot weighted UniFrac distances with gradient colors

```
plot(pc.wuf[,1], pc.wuf[,2], col=my.colors[layer], cex=3, pch=16)
```

Note: to make a PDF:

```
pdf("chisq.pdf",width=5,height=5) plot(pc.chisq[,1], pc.chisq[,2], col=my.colors[map[,'LAYER']], cex=3, pch=16) dev.off()
```

Let’s plot pairwise comparisons of the different distance metrics

```
d.vector.matrix <- cbind(as.numeric(d.euc), as.numeric(d.bray), as.numeric(as.dist(d.chisq)), as.numeric(as.dist(as.matrix(d.uuf))), as.numeric(as.dist(as.matrix(d.wuf)))) colnames(d.vector.matrix) <- c('Euc','BC','ChiSq','UUF','WUF') pairs(d.vector.matrix)
```

And display the pairwise pearson correlations

```
cor(d.vector.matrix)
```

Which distance metric best recovered physical sample distances based on END_DEPTH?

*** There was no endepth variable in the map file (PS there was, I initially pulled the global gut fules instead of the guerrero negro ones)

```
# y is the euclidean distance matrix based on ending depth of each layer
y <- as.vector(dist(map$END_DEPTH))
# Test the correlation of END_DEPTH distance and ecological distance
# for each metric
metrics <- list(d.euc, d.bray, d.chisq, d.uuf, d.wuf)
names(metrics) <- colnames(d.vector.matrix)
# reuse pairwise column names
for(i in 1:length(metrics)){     d.name <- names(metrics)[i]
# convert distance matrix to vector form
d <- as.vector(as.dist(metrics[[i]]))
cat('Correlation of ',d.name,':','\n',sep='')
print(cor.test(d, y, method='spear', exact=FALSE)) }
```

Here is the one-liner version if you just want to test one distance metric vs. your continuous variable of interest. There are two ways to do this:

1. ask whether the **overall** distance metric is correlated with your gradient:

```
# y is the euclidean distance matrix based on your variable of interest (here depth)
# note: euclidean diatnace makes sense in the Guerrero Negro sampling depth because
# it is measuring physical distance (in millimeters)
# Note: the "as.vector" stretches it out to a single numeric vector (no longer a matrix)
y <- as.vector(dist(map$END_DEPTH))
# Stretch out the d
d <- as.vector(as.dist(d.chisq)) cor.test(d, y, method='spear', exact=FALSE)
```

2. You could ask whether PC1, the **first principal axis of variation** (not overall distance), is significantly correlated with your variable of interest. This is an even stronger result if significant.

```
cor.test(map$END_DEPTH, pc.chisq[,1], method='spear', exact=FALSE)
```

Statistical Analysis (Global Gut)

All of the code in this page is meant to be run in **R** unless otherwise specified.

Loading a genus table and the metadata into R

Before loading data into R, this QIIME command must be run on the command line to collapse OTU counts into genus (L6) and phylum (L2) count tables:

```
cd ~/mice_tutorial
cd mice8992-2016-master
cd data
cd globalgut
# (run on command line)
summarize_taxa.py -i otu_table.biom -L 6
summarize_taxa.py -i otu_table.biom -L 2
# convert to JSON BIOM format to load into R using R biom package:
biom convert -i otu_table_L6.biom -o otu_table_L6_json.biom --to-json
biom convert -i otu_table_L2.biom -o otu_table_L2_json.biom --to-json
```

Inside R, Install and load the updated biom package and the vegan package

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("biomformat")
install.packages('vegan')
library('biomformat')
library('vegan')
```

```
# load biom file
genus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/global
gut/otu_table_L6_json.biom')
# Extract data matrix (genus counts) from biom table
genus <- as.matrix(biom_data(genus.biom))
# transpose so that rows are samples and columns are genera
genus <- t(genus)
```

```
# load mapping file

map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut/map.txt', sep='\t', comment='', head=T, row.names=1)
```

It is extremely important to ensure that your genus table and metadata table sample IDs are lined up correctly.

```
# find the overlapping samples

common.ids <- intersect(rownames(map), rownames(genus))

# get just the overlapping samples

genus <- genus[common.ids,]

map <- map[common.ids,]
```

See dimensions of genus table. Then drop genera present in < 10% of samples.

```
dim(genus)

genus <- genus[,colMeans(genus > 0) >= .1]

dim(genus)
```

Show only the first ten genera in genus table

```
colnames(genus)[1:10]
```

Abbreviate the taxonomic names to make them easier to display.

```
colnames(genus) <- sapply(strsplit(colnames(genus),';'),function(xx) paste(paste(substr(xx[-c(1,length(xx))],4,7),collapse=';'),substring(xx[length(xx)],4),sep=';'))
```

Show the first 10 rows and first 2 columns of the genus table

```
genus[1:10,1:2]
```

See available columns in the metadata

```
colnames(map)
```

Show how many samples are from each Country

```
table(map$COUNTRY)
```

Step 2: Basic Association Testing (Y = single Genus or OTU, X = Covariates)

Individual Hypothesis Testing

- Descriptive statistics (histograms of the OTU counts)
- Basic association testing (Y= bacteria OTU count) vs X(Covariate from metadata)

Let's run some tests on Prevotella. First extract the Prevotella column and save it to a variable prevotella for convenience.

```
# find out what column Prevotella is in # the "$" tells grep to find only column names that end wi
th "Prevotella".

grep('Prevotella$',colnames(genus))

# save that column in a variable

prevotella <- genus[,grep('Prevotella$',colnames(genus))]
```

Visualize the distribution of Prevotella

```
# find out what column Prevotella is in

hist(prevotella, br=30)
```

Run a test of Pearson's correlation of Prevotella and age. Note that the result is not quite significant (p=0.0531).

```
cor.test(prevotella, map$AGE)
```

Now run a linear regression of prevotella against age. Notice that statistically this is equivalent to running the Pearson's correlation. The p-value in row 2 column 4 of the "Coefficients" table is the same as the p-value from the correlation test.

```
# fit a linear model. The "~" means "as a function of"

fit <- lm(prevotella ~ map$AGE)

# print a summary of the results

summary(fit)

# A nice way to get the exact p-value for the age regression coefficient using the anova function

pval <- anova(fit)['map$AGE','Pr(>F)']

pval
```

Testing for normally distributed data

We can test whether the residuals are normally distributed Kolmogorov-Smirnov test. If $p < 0.05$, we can reject the null hypothesis that the data came from a normal distribution, meaning that the linear test is not appropriate.

```
# Make a quantile-quantile plot of the (studentized) residuals vs. a normal distribution

qqnorm(rstudent(fit)); abline(0,1)

# Kolmogorov-Smirnov test

ks.test(rstudent(fit), pnorm, mean=mean(rstudent(fit)), sd=sd(rstudent(fit)))
```

Controlling for confounders

Perhaps country of origin is a confounder that is obscuring the association of Prevotella and Age. Using lm() we can add confounders to the regression. Now after removing the effects of country, there is a strong association of Prevotella and age.

```
# fit a linear model. The "~" means "as a function of"

fit <- lm(prevotella ~ map$AGE + map$COUNTRY)

# print a summary of the results

summary(fit)
```

Step 3: Multiple Association Testing using Linear Regression assuming a normal distribution of residuals (Y = Multiple Genus or OTUs, X = Covariates)

Multiple hypotheses testing (testing multiple Genuses at the same time)

- A single test has an alpha level of 0.05. However, when testing 100 hypotheses simultaneously, we can expect that 5 out of the 100 will be statistically significant due to chance
- We can either use the bonferonni correction by dividing the alpha level by the number of tests (here, number of OTUs or Genuses)
- We can also control the expected rate of false positive using a False Discovery Rate Adjustment.
- We can use a for loop in the EdgeR package to simultaneously test multiple Genuses for an association with covariates. The template is glm.edger(X=covar, Y=genus). The LogFC value from our models compare whether a particular covariate X is associated with the presence versus absence of a particular genus (1 vs 0). Once the model is processed, we can adjust for multiple comparisons using a false discovery rate approach.

Testing multiple hypotheses

We have so far only tested one genus. Let’s test them all using a loop.

```
# pvals is a vector initialized with zeroes # with enough slots for the different genera

pvals <- numeric(ncol(genus))

# "name" the pvalues after the genera

names(pvals) <- colnames(genus)

# Loop through the columns of the genus table, testing each one

for(i in 1:ncol(genus)) {     fit <- lm(genus[,i] ~ map$AGE + map$COUNTRY)     pvals[i] <-  anova(fit)['map$AGE','Pr(>F)'] }


(*** You have to make sure to break up the loop at operators so that R reads it all at once, and the breakdown below worked for me, but you need to hit enter and R will add the plus sign at the beginning of each line)

for(i in 1:ncol(genus)) {

+ fit <- lm(genus[,i] ~ map$AGE + map$COUNTRY)

+      pvals[i] <-

+ anova(fit)['map$AGE','Pr(>F)'] }
```

Note, you could put this all on one line with this:

```
# "apply" with genus, 2 means do something to every column of genus

# ("apply" with genus, 1 would mean do something every row)

# the last part defines a new function to do to each column, which

# will be passed in the temporary variable named "xx"

pvals <- apply(genus, 2, function(xx) anova(lm(xx ~ map$AGE + map$COUNTRY))['map$AGE','Pr(>F)'])

# print the 10 smallest p-values:

sort(pvals)[1:10]
```

Looks like there are some significant p-values. But did they happen just by chance? There are 97 columns in the genus table, so that means we did 97 tests, and about 5% of them should be p < 0.05 just by chance. To correct for this, we can adjust the p-values using the p.adjustfunction. Here we are correcting for multiple hypothesis testing use False Discovery Rate (FDR). The adjusted p-values are often called “q-values.”

```
qvals <- p.adjust(pvals,'fdr')

# print the lowest 10 q-values

sort(qvals)[1:10]
```

Step 4: Multiple Association Testing using GLM with a negative binomial distribution (Y = Multiple Genus or OTUs, X = Covariates)

Testing with generalized linear regression.

Let’s test whether the residuals from linear regression were normally distributed for all of the taxa. To follow along, you can put the following code in a separate text file called stats.r (or whatever you want to call it, and then call it using source('stats.r').

```
ks.pvals <- numeric(ncol(genus))

# "name" the pvalues after the genera

names(ks.pvals) <- colnames(genus)

# turn annoying warnings off  options(warn=-1)

# Loop through the columns of the genus table, testing each one

for(i in 1:ncol(genus)) {     fit <- lm(genus[,i] ~ map$AGE + map$COUNTRY)     ks.pvals[i] <- ks.test(rstudent(fit), pnorm, mean=mean(rstudent(fit)), sd=sd(rstudent(fit)),exact=FALSE)$p.value }
```

*(*** You have to make sure to break up the loop at operators so that R reads it all at once, and the breakdown below worked for me, but you need to hit enter and R will add the plus sign at the beginning of each line)*

```
for(i in 1:ncol(genus)) {

+ fit <- lm(genus[,i] ~ map$AGE + map$COUNTRY)

+ ks.pvals[i] <

+ -ks.test(rstudent(fit), pnorm, mean=mean(rstudent(fit)), sd=sd(rstudent(fit)),exact=FALSE)$p.value }

# turn warnings back on options(warn=0)

# Now since we ran 97 tests we should correct for multiple hypothesis testing

ks.qvals <- p.adjust(ks.pvals,'fdr')

# print the lowest 10 q-values

sort(ks.qvals)[1:10]
```

Let’s use a negative binomial distribution instead. We will use the edgeR package. If you don’t have it, you can install it with:

```
source("https://bioconductor.org/biocLite.R")

# If the previous command doesn't work, try http://

biocLite("edgeR")

library(limma)

library(edgeR)
```

Note: the negative binomial uses raw counts of sequences (rarefied or not), not the relative abundances. Therefore we must re-run summarize_taxa.py with the -a flag to use absolute abundance. These commands are run on the command line (not in R)

```
# (run on command line)

summarize_taxa.py -i otu_table.biom -L 6 -a -o taxa-absolute

# convert to JSON BIOM format to load into R using R biom package:

biom convert -i taxa-absolute/otu_table_L6.biom -o taxa-absolute/otu_table_L6_json.biom --to-json

# load biom file

genus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut/taxa-absolute/otu_table_L6_json.biom')

# load mapping file (Don’t need to do this step since it is the same as the one from the relative OTU tables)

map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut/map.txt', sep='\t', comment='', head=T, row.names=1)
```

Now we need to process the genus table again. (Extract, transpose, cross-reference to mapping file, and remove OTUs with less than 10% presence in our samples)

```
genus.a <- as.matrix(biom_data(genus.biom))

genus.a <- t(genus.a)

common.ids <- intersect(rownames(map), rownames(genus.a))




# get just the overlapping samples
```

```
genus.a <- genus.a[common.ids,]

map <- map[common.ids,]

dim(genus.a)

dim(map)

genus.a <- genus.a[,colMeans(genus.a > 0) >= .1]

colnames(genus.a) <- sapply(strsplit(colnames(genus.a),';'),function(xx) paste(paste(substr(xx[-c(1,length(xx))],4,7),collapse=';'),substring(xx[length(xx)],4),sep=';'))
```

First let us run the regression without covariates. The main function in this script is glm.edgeR(). This function will perform multiple hypothesis testing on all columns of a data matrix. It has the following main parameters:
- x: the independent variable. If discrete must be binary; OK to be continuous. - Y: A matrix with samples in rows and dependent variables in columns - covariates: a matrix of additional covariates you want to control for (default NULL)

The glm.edgeR function seems to be from the custom wrapper designed by Dr. Dan Knights. I did not find it in the functions from the edgeR packade on the biocoductor website

You have to source the following file in the SRC folder from the mice_tutorial

source('/Users/chigu142000/mice_tutorial/mice8992-2016-master/src/wrap.edgeR.r')

*** Make sure that your mapping file and your genus file are of the same dimensions!!!

```
result <- glm.edgeR(x=map$AGE, Y=genus.a)
```

We can print the top “tags” (genera) using the topTags function. Note that two genera are significant even after correction for multiple hypothesis testing.

```
topTags(result)
```

If we plot all p-values coming from edgeR in a quantile-quantile plot, we see that they mostly follow the null (uniform) distribution:

```
pvals <- topTags(result,n=Inf)$table[,'PValue'] plot(-log10(seq(0,1,length=98)[-1]), -log10(sort(pvals))); abline(0,1)
```

However, we have not controlled for COUNTRY, which may be a confounder. We can do this with edgeR.

```
result <- glm.edgeR(x=map$AGE, Y=genus.a, covariates=map$COUNTRY)

topTags(result)
```

Note that no genera are significantly associated with age after controlling for country, and using the negative binomial as the assumed distribution for the residuals.

We can also pass in a matrix of covariates like this, although note that SEX is not a good variable here because there are only 6 males and there are 6 with unknown gender:

```
result <- glm.edgeR(x=map$AGE, Y=genus.a, covariates=map[ , c('COUNTRY','SEX')])
```

Let us test whether any genera are significantly associated with being from the USA, while controling for age:

```
# make a vector that is TRUE if sample is from USA, FALSE otherwise
# the "==" means "test if equal"
is.USA <- map$COUNTRY == "GAZ:United States of America"
# run with is.USA as the independent variable result <- glm.edgeR(x=is.USA, Y=genus.a, covari
ates=map$AGE)
topTags(result)
```

Write the results to a tab-delimited file (to open in Excel) with:

```
write.table(topTags(result, n=Inf)$table, file='edgeR_results.txt',sep='\t',quote=FALSE, col.name
s=NA)
```

How many genera were significantly associated with the USA?

```
sum(topTags(result,n=Inf)$table$FDR <= 0.05)
```

Non-parametric tests

Whenever we do not need to control for confounding variables, we can use non-parametric tests. These are the safest because they don’t rely on any null distribution (e.g. normal). They typically consider only the **ranks** of values (order of values), not the actual values themselves.

For testing differences between two categories, we can use the Mann-Whitney U test, sometimes called the Wilcoxon Signed Rank test or Wilcoxon Rank Sum test. There are slight differences between these tests. Here we will test the difference in **Prevotella** abundance between USA and non-USA. Make sure to use the relative abundances, not the absolute abundances.

```
wilcox.test(prevotella ~ is.USA, exact=FALSE)
# get the exact p-value
wilcox.test(prevotella ~ is.USA, exact=FALSE)$p.value
```

We can do a test for differentiation across multiple categories, analogous to ANOVA, using the Kruskal-Wallis test.

```
kruskal.test(prevotella ~ map$COUNTRY)
```

For continuous variables, we can use Spearman correlation instead of Pearson correlation.

```
cor.test(prevotella, map$AGE, method='spearman', exact=FALSE)
```

Using MSI for bioinformatics analyses

A Minnesota Supercomputing Institute (MSI) account was created for
Guillaume Onyeaghala in the group prizm001 with the username onyea005.

In addition, you have been granted access to the prizm001 group's Service Units.
This access will allow you to run jobs on MSI's High-Performance Computing resources.

Passwords

Your password at MSI is the same as the password for your University Internet ID.
If you ever need to change or reset your password, go to https://www.umn.edu/dirtools.

Getting Started

Quick Start Guides for new MSI users are available on our website
at https://www.msi.umn.edu/quick-start-guides

Frequently Asked Questions

You can find answers to a range of frequently asked questions
at https://www.msi.umn.edu/support/faq

Getting Help

Our help desk is staffed 8:30 to 5:00 Monday through Friday. Send an email
to help@msi.umn.edu, call (612) 626-0802, or visit 587 Walter Library.

Connecting to and using MSI resources (From part 13)

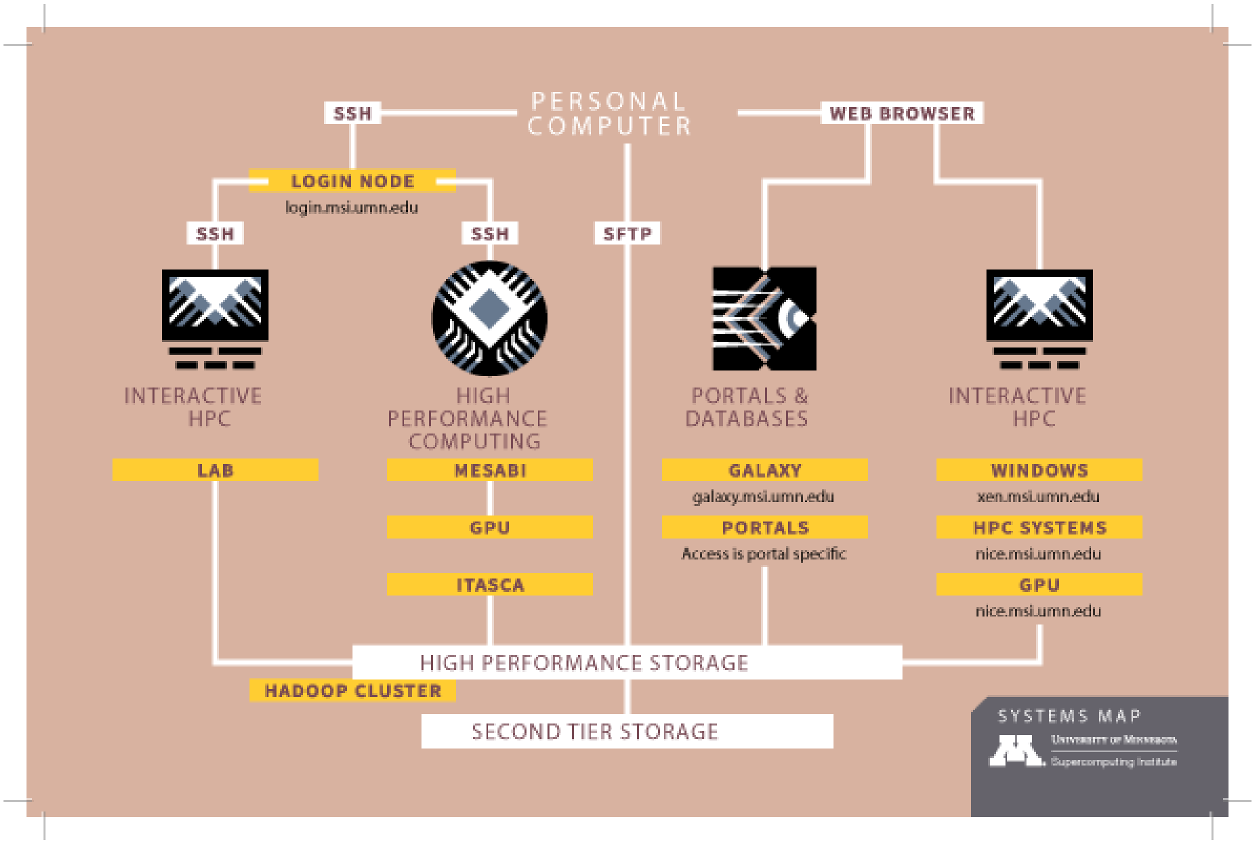
https://www.msi.umn.edu/content/connecting-hpc-resources

https://www.msi.umn.edu/support/faq

https://www.msi.umn.edu/getting-started

Summary

MSI provides access to a variety of High-Performance Computing (HPC) systems designed to tackle a wide range of computational needs. This document will show you how you can access all of MSI computational resources from your personal computer.



Skills/Software Needed

If you are connecting from a computer running Windows OS you will need to download and install PuTTY. OSX and Linux does not require additonal software to connect.

Connecting to MSI

SSH (Use Secure Shell)

SSH, also known as Secure Socket Shell, is a network protocol that provides administrators with a secure way to access a remote computer. SSH also refers to the suite of utilities that implement the protocol. Secure Shell provides strong authentication and secure encrypted data communications between two computers connecting over an insecure network such as the Internet. SSH is widely used by network administrators for managing systems and applications remotely, allowing them to log in to another computer over a network, execute commands and move files from one computer to another.

The main features of SSH include secure remote logins, terminal emulation, fully integrated secure file transfers, and secure tunneling of X11 traffic. Many University of Minnesota systems require secure telnet and FTP connections.

Install a SSH Client

Unix/Linux

- OpenSSH is freely usable and re-usable by everyone under a BSD license. OpenSSH is available from http://www.openssh.org/portable.html.

Windows

- Cygwin provides a Windows port of the most current OpenSSH tools.
- PuTTY is a full-featured, SSH compliant client for Windows. You can get Putty at http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html, as well as PSCP, a tool for encrypted remote file transfer
- WinSCP is a freeware Secure Copy (SCP) protocol client for Windows, which can be used to connect to an SSH server, mainly to UNIX machines, for file transfer using the SCP service. http://winscp.net/eng/index.php
- SecureCRT software requires you to purchase a license. http://www.vandyke.com/products/securecrt/

Mac OS X

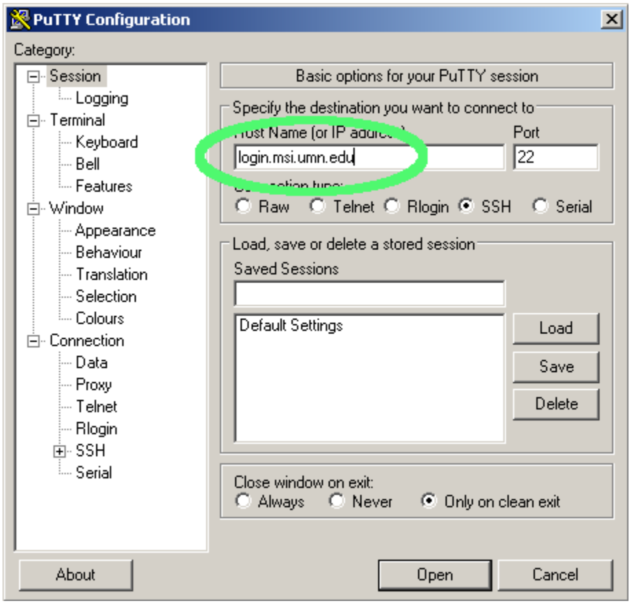
- Mac OS X comes with OpenSSH pre-installed. Open the Terminal application from inside the Utilities folder, and then type "ssh *user@server_name_or_IP_address*" to get started.

- Portable OpenSSH is available at http://www.openssh.com/portable.html at no charge. The portable OpenSSH follows development of the official version, but releases are not synchronized. Portable releases are marked with a 'p' (e.g. 2.3.0p1).

Connecting to MSI with Windows OS

Configure PuTTY to connect to MSI

- Double-click on the PuTTY icon and a configuration window will pop up.
- In the Host Name box type: **login.msi.umn.edu**



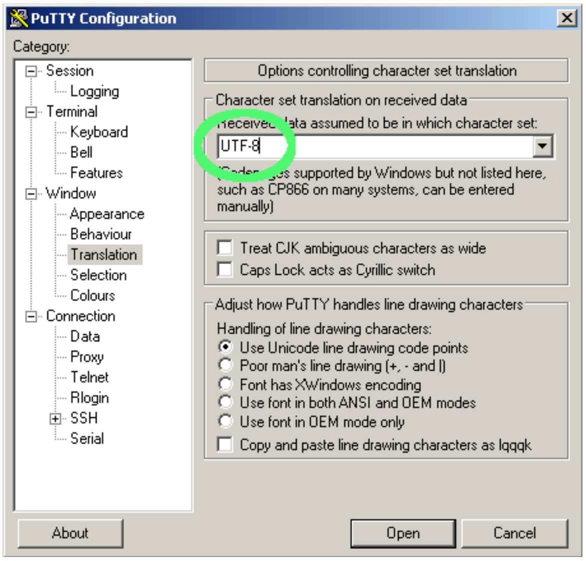
Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Close window on exit: ○ Always ○ Never ● Only on clean exit

- In the left panel, under Window, select Translation
- Change Receive data assumed to bin in which character set to : UTF-8



Received data assumed to be in which character set: UTF-8

(Codepages supported by Windows but not listed here, such as CP866 on many systems, can be entered manually)

□ Treat CJK ambiguous characters as wide

□ Caps Lock acts as Cyrillic switch

Handling of line drawing characters:

● Use Unicode line drawing code points

○ Poor man's line drawing (+, - and |)

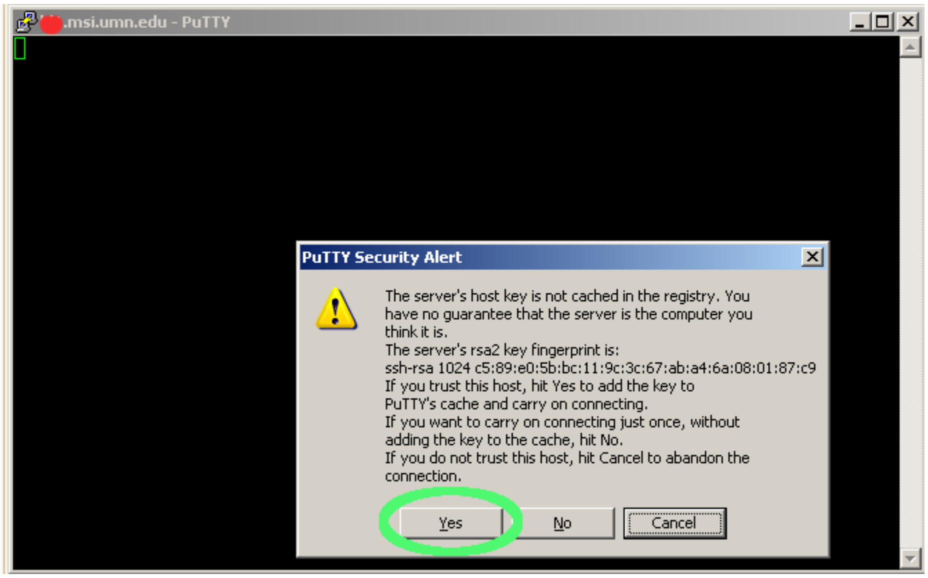
○ Font has XWindows encoding

○ Use font in both ANSI and OEM modes

○ Use font in OEM mode only

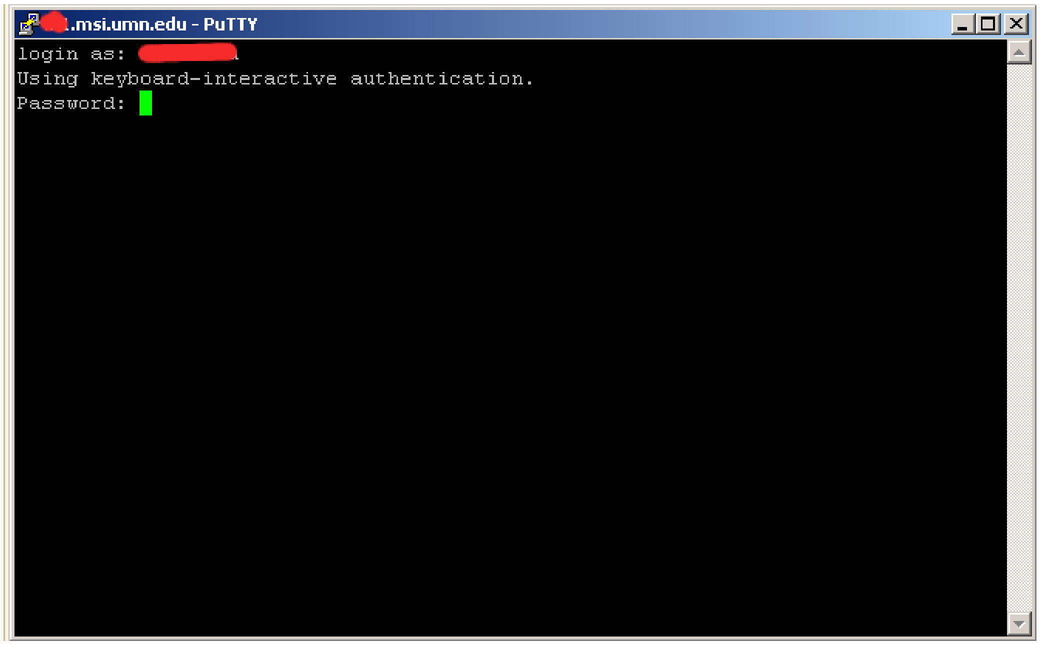
□ Copy and paste line drawing characters as lqqqk

- Select Session in the left panel and in the Saved Session box type the name you would like to use to refer to this session. We suggest MSI server, then select Save.
- The next time you open PuTTY you will be able to recall a saved profile by clicking on the name and clicking Load then Open to start the session.
- The first time you connect to each MSI system you may be asked to cache the server fingerprint, select Yes.



The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 1024 c5:89:e0:5b:bc:11:9c:3c:67:ab:a4:6a:08:01:87:c9
If you trust this host, hit Yes to add the key to PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without adding the key to the cache, hit No.
If you do not trust this host, hit Cancel to abandon the connection.

- When prompted enter your MSI username and password



```
login as: [redacted]
Using keyboard-interactive authentication.
Password:
```

- You are now connected to the MSI Login host (or bastion host). This host will let you view the directories but you can not submit jobs or run interactive computation on this host.

Connecting to MSI with OSX

Terminal is a preinstalled application which can be used to connect to MSI systems. Terminal can be found in the Utilities folder found in the Applications folder.

Connecting to MSI with Linux OS

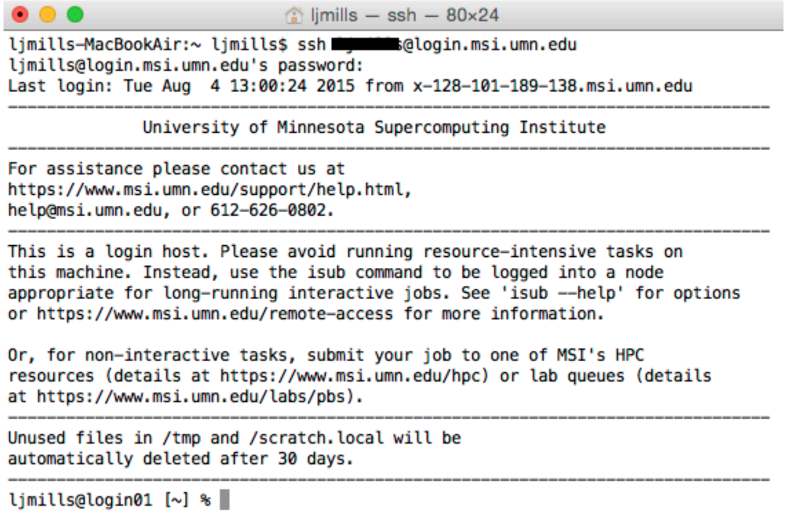
Terminal is a preinstalled application which can be used to connect to MSI systems.

For OSX and Linux OS Terminal Apps

Once you open the Terminal App type:

- ssh yourMSIusername@login.msi.umn.edu

- You will be prompted for your MSI password type it in then press enter. The terminal will not display your password as you type.
- The first time you log into MSI systems you will be asked if you would like to cache the server fingerprint, type yes and press return.



```
ljmills-MacBookAir:~ ljmills$ ssh [redacted]@login.msi.umn.edu
ljmills@login.msi.umn.edu's password:
Last login: Tue Aug  4 13:00:24 2015 from x-128-101-189-138.msi.umn.edu
-------------------------------------------------------------------------------
              University of Minnesota Supercomputing Institute
-------------------------------------------------------------------------------
For assistance please contact us at
https://www.msi.umn.edu/support/help.html,
help@msi.umn.edu, or 612-626-0802.
-------------------------------------------------------------------------------
This is a login host. Please avoid running resource-intensive tasks on
this machine. Instead, use the isub command to be logged into a node
appropriate for long-running interactive jobs. See 'isub --help' for options
or https://www.msi.umn.edu/remote-access for more information.

Or, for non-interactive tasks, submit your job to one of MSI's HPC
resources (details at https://www.msi.umn.edu/hpc) or lab queues (details
at https://www.msi.umn.edu/labs/pbs).
-------------------------------------------------------------------------------
Unused files in /tmp and /scratch.local will be
automatically deleted after 30 days.
-------------------------------------------------------------------------------
ljmills@login01 [~] %
```

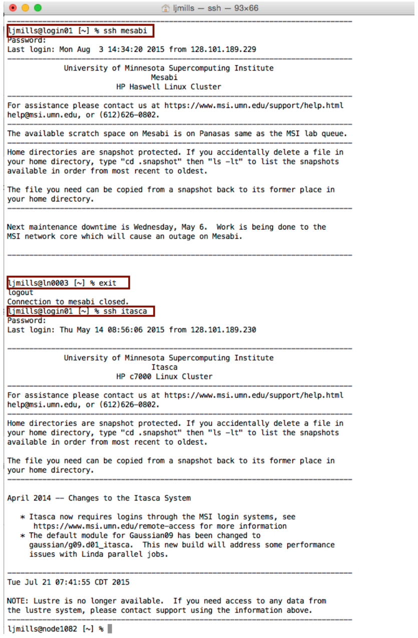
- You are now connected to the MSI Login host (or bastion host). This host will let you view the directories but you can not submit jobs or run interactive computation on this host.

Connecting to Mesabi and Itasca from Login Host

- Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh.

ssh mesabi or ssh itasca

- MSI prohibits moving directly from one system to another. If you are connected to Mesabi and would like to connect to Itasca you will first have to exit back to the Login Host, then ssh to the new system. Commands are highlighted with red boxes in the example below.



```
--------------------------------------------------------------------------------
ljmills@login01 [~] % ssh mesabi
Password:
Last login: Mon Aug  3 14:34:20 2015 from 128.101.189.229
--------------------------------------------------------------------------------
                University of Minnesota Supercomputing Institute
                                    Mesabi
                          HP Haswell Linux Cluster
--------------------------------------------------------------------------------
For assistance please contact us at https://www.msi.umn.edu/support/help.html
help@msi.umn.edu, or (612)626-0802.
--------------------------------------------------------------------------------
The available scratch space on Mesabi is on Panasas same as the MSI lab queue.
--------------------------------------------------------------------------------
Home directories are snapshot protected. If you accidentally delete a file in
your home directory, type "cd .snapshot" then "ls -lt" to list the snapshots
available in order from most recent to oldest.

The file you need can be copied from a snapshot back to its former place in
your home directory.
--------------------------------------------------------------------------------


Next maintenance downtime is Wednesday, May 6.  Work is being done to the
MSI network core which will cause an outage on Mesabi.

--------------------------------------------------------------------------------


ljmills@ln0003 [~] % exit
logout
Connection to mesabi closed.
ljmills@login01 [~] % ssh itasca
Password:
Last login: Thu May 14 08:56:06 2015 from 128.101.189.230

--------------------------------------------------------------------------------
                University of Minnesota Supercomputing Institute
                                    Itasca
                          HP c7000 Linux Cluster
--------------------------------------------------------------------------------
For assistance please contact us at https://www.msi.umn.edu/support/help.html
help@msi.umn.edu, or (612)626-0802.
--------------------------------------------------------------------------------
Home directories are snapshot protected. If you accidentally delete a file in
your home directory, type "cd .snapshot" then "ls -lt" to list the snapshots
available in order from most recent to oldest.

The file you need can be copied from a snapshot back to its former place in
your home directory.
--------------------------------------------------------------------------------

April 2014 -- Changes to the Itasca System

   * Itasca now requires logins through the MSI login systems, see
     https://www.msi.umn.edu/remote-access for more information
   * The default module for Gaussian09 has been changed to
     gaussian/g09.d01_itasca.  This new build will address some performance
     issues with Linda parallel jobs.

--------------------------------------------------------------------------------
Tue Jul 21 07:41:55 CDT 2015

NOTE: Lustre is no longer available.  If you need access to any data from
the lustre system, please contact support using the information above.
--------------------------------------------------------------------------------
ljmills@node1082 [~] %
```

- Each time you move between systems you will see the welcome message associated with the system you are connecting to. These messages often contain useful information and should be read.
- **NOTE:** Like the login node Mesabi and Itasca have specific nomenclatures in the prompt. When connected to Mesabi your prompt will contain **ln** followed by some number when connected to Itasca your prompt will contain **node** followed by some numbers. This is a quick short cut to remind you which system you are connected to at any time.

Choosing a Job Queue

Summary

Most MSI systems use job queues to efficiently and fairly manage when computations are executed. A job queue is an automated waiting list for use of a particular set of computational hardware. When computational jobs are submitted to a job queue they wait in the queue in line until the appropriate resources become available. Different job queues have different resources and limitations. When submitting a job, it is very important to choose a job queue which has resources and limitations suitable to the particular calculation.

This document outlines factors to consider when choosing a job queue.   These factors are important when choosing where to place a job. This document is best used on all MSI systems and in conjunction with the Queues page that outlines the resource limitations for each queue.

Please note that Mesabi's "widest" queue requires special permission to use. Please submit your code for review at: help@msi.umn.edu.

Guidelines

There are several important factors to consider when choosing a job queue for a specific program or custom script. In most cases, jobs are submitted via PBS scripts as described in Job Submission and Scheduling.

Overall System

Each MSI system contains job queues managing sets of hardware with different resource and policy limitations. MSI currently has three primary systems: the newest supercomputer Mesabi, the supercomputer Itasca, and the Lab compute cluster. Mesabi is MSI's newest supercomputer, with the highest performance hardware, and a wide variety of queues suitable for many different job types. **Mesabi should be your first choice when doing any computation at MSI**. Itasca is a supercomputer with queues most suitable for multi-node jobs which will complete within 1-2 days. The Lab cluster is primarily for interactive software that is graphical in nature, and testing. Which system to choose depends highly on which system has queues appropriate for your software/script. The variety of queues on Mesabi will be suitable for most users, but the Queue page should be examined.

Job Walltime (walltime=)

The job walltime is the time from the start to the finish of a job (as you would measure it using a clock on a wall), not including time spent waiting to run. This is in contrast to cputime, which measures the cumulative time all cores spent working on a job. Different job queues have different walltime limits, and it is important to choose a queue with a sufficiently high walltime that enables your job to complete. Jobs which exceed the requested walltime are killed by the system to make room for other jobs. Walltime limits are maximums only, and you can always request a shorter walltime, which will reduce the amount of time you wait in the queue for your job to start. If you are unsure how much walltime your job will need start with the queues with shorter walltime limits and only move to others if needed.

Job Nodes and Cores (nodes=X:ppn=Y)

Many calculations have the ability to use multiple cores (ppn), or (less often) multiple nodes, to improve calculation speed. Certain job queues have maximum or minimum values for the number nodes and cores a job may use. If **Node Sharing** is enabled for a queue you can request fewer cores (ppn) than exist on an entire node. If Node Sharing is not enabled then you must request resources equivalent to a multiple of an entire node. All Itasca queues, and Mesabi's widest and large queues, **do not allow** Node Sharing**.**

Job Memory (mem=)

The memory which a job requires is an important factor when choosing a queue. The largest amount of memory (RAM) that can be requested for a job is limited by the memory on the hardware associated with that queue. Mesabi has two queues (ram256g and ram1t) with high memory hardware, the largest memory hardware is available through the ram1t queue. Itasca also has two queues with high memory hardware (sb128 and sb256).

User and Group Limitations

To efficiently share resources, many queues have limits on the number of jobs or cores a particular user or group may simultaneously use. If a workflow requires many jobs to complete, it can be helpful to choose queues which will allow many jobs to run simultaneously. Mesabi allows more simultaneous jobs to run than Itasca.

Special Hardware

Some queues contain nodes with special hardware, GPU accelerators and solid-state scratch drives being the most common. If a calculation needs to use special hardware, then it is important to choose a queue with the correct hardware available. Furthermore, those queues may require additional resources to be specified (e.g., GPU nodes require ":gpus=X").

Queue Congestion

At certain times particular queues may become overloaded with submitted jobs. In such a case, it can be helpful to send jobs to queues with lower utilization (node status). Sending jobs to lower utilization queues can decrease wait time and improve throughput. Care must be taken to make sure calculations will fit within queue limitations.

Accessing Software Resources

MSI provides access to approximately 400 software packages. To find out if a particular software package is installed, you can browse or search a list of the software packages that MSI provides under the Help and Documentation section of this website. The MSI webpage includes descriptions of software packages including example usage to help you get started.

Support Tiers

Software packages will be marked with a support tier -- either Primary, Secondary, or Minimal. Primary support is given to mature, popular scientific software important to a large fraction of the user base generally receive the highest level of support. These software are compiled, installed, benchmarked, and documented as a service to you. They are tested after major system upgrades, new versions are installed in a timely manner, and we expect to be able to offer advice to assist with the majority of inquiries. Conversely, Secondary or Minimally supported software are only useful to a very small number of users. These software may not be actively maintained, regularly updated, or tested after system updates. They may be removed without notice if usage is found to be too low to continue support.

How To Access Software

HPC and Interactive HPC Systems

To access software on Linux systems, use the module command to load the software into your environment. You can find the module name to load on the MSI software page for a package. For example, if you want to use the 2014b version of MATLAB, you would type in your job script or at the command line:

```
% module load matlab/R2014b
% matlab
```

To find out what module versions are available, consult the software page or use the module avail command to search by name. For example to find out what versions of Python are available:

```
% module avail python
```

Windows Software

Windows-only software can be accessed and run via the Windows Virtual Environment (Citrix). Once a windows session is started you access software applications in the same manner as if you were using a physical Windows computer.

Need a Specific Package?

First try and install it yourself in your MSI home directory following the guidelines below:

Software Installation Guide

Installing software on any Unix platform can be challenging for inexperienced and veteran users alike.

While it is impossible to cover every issue you may experience, the following steps will allow you to compile and install in your own home directory many of the libraries and scientific applications that you will come across.

Introductory Compiling

Building a Python interpreter from its source code is straightforward, requiring only a C compiler, and will serve as an instructive example.

Installation materials are most commonly distributed as compressed tar files with the extension .tar.gz or .tar.bz.

These files can be untarred and unzipped with the tar -xzf and tar -xjf commands, respectively.

We have downloaded the file Python-2.7.2.tar.gz from www.python.org to our home directory, so the appropriate command is:

```
tar -xzf Python-2.7.2.tar.gz

cd Python-2.7.2
```

Your next step should always be to review the distributed files for a README file, which can be viewed in a text editor, or some other file that is named to indicate it contains installation instructions.

In many cases, and as is the case with Python, a configure script is included. In the simplest case, all that is necessary to compile the code is to run the configurescript, then make.

By default many codes will try to install to system directories you cannot access. The recommended installation method is to create a directory in your home directory for software installation.

Pwd (It will display a result such as /home/xe2/nlabello/Python-2.7.2)

mkdir /home/xe2/nlabello/software/python

./configure --prefix=/home/xe2/nlabello/software/python/

make install

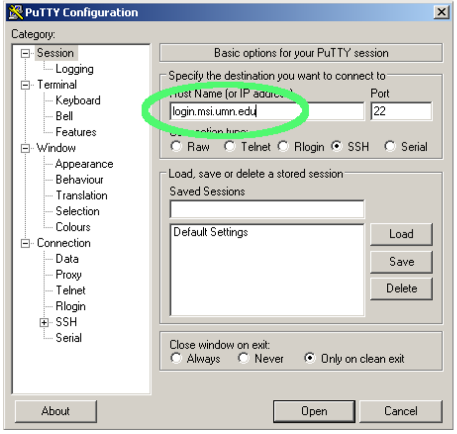
When the make install step completes you will have access to binary in /home/xe2/nlabello/software/python/bin

Installing QIIME

First Things First: Open the Terminal

Using QIIME on MSI

1. Download Putty on Windows (http://www.putty.org/) and configure it for MSI access



Specify the destination you want to connect to

Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Saved Sessions: Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

21. Connecting to Mesabi and Itasca from Login Host. Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh (QIIME is currently loaded under Itasca so use ssh Itasca for QIIME analyses)

```
ssh mesabi or ssh itasca
```

22. Alternatively, connect to MSI using Terminal on Mac/Linux, located in Applications>Utilities>Terminal.app.

```
ssh yourusername@login.msi.umn.edu
```

23. Log in to an interactive node, making sure you have enough time and memory

```
isub -n nodes=1:ppn=4 -m 16GB -w 24:00:00
```

24. Load the QIIME module

```
module load qiime/1.9.1
```

MacQIIME Users Only:

If you are using MacQIIME, you will have to use your Mac OS X terminal, located in Applications>Utilities>Terminal.app.
Normally, when you run a command in the terminal, all the packages in MacQIIME will be invisible to your computer. In order to access macqiime, after starting a new terminal session you have to source the environment variables with the "macqiime" command. To source the QIIME environment variables and load up all the MacQIIME scripts in the PATH, you have to start the MacQIIME subshell:

macqiime

You should now be looking at a dollar sign ($) with a cursor after it. This is the command line interface for your computer. Here, you can type commands in order to "execute" programs. This Unix/Linux command line is a powerful place to analyze and process data. Everything you do in QIIME is executed through this command line interface. I'm going to assume you have some familiarity with the Unix command line, but I'll try to help you along the way with commands like **cp**, **cd**, **ls**, **pwd**, **less**, **nano**, etc.

**Note: Getting MacQIIME to work in El Capitan (OS 10.11)

This section is required only if you are running OS 10.11 (El Capitan) or higher. Apple added a new security feature in El Capitan that makes it impossible to install software into /usr/bin/ folder (that was where I was putting the “macqiime” sourcing script. I'll fix this in the next release of MacQIIME. In the meantime, we can work around it. The quick solution is to use this command, instead of the “macqiime” script:

source /macqiime/configs/bash_profile.txt

How do I transfer data between a Mac or Windows computer and MSI Unix computers?

For transferring data from Mac / Windows computers to MSI Unix computers, any **SFTP** or **SCP** software can be used. Connect to login.msi.umn.edu using your MSI username and password and you will be able to transfer files to and from your home directory and group shared directory. For Windows, we recommend WinSCP. For Mac OS X, we recommend using FileZilla.

If you are on the Unix side and need to pull data from your Windows U: drive, you can use smbclient:

$ smbclient -U 'MSI\*username*' -D *username* //falcon2/Users

From the prompt:

smb: \username\>

You can issue ftp style commands ('ls', 'get', 'cd', etc.). To get an entire directory recursively, use:

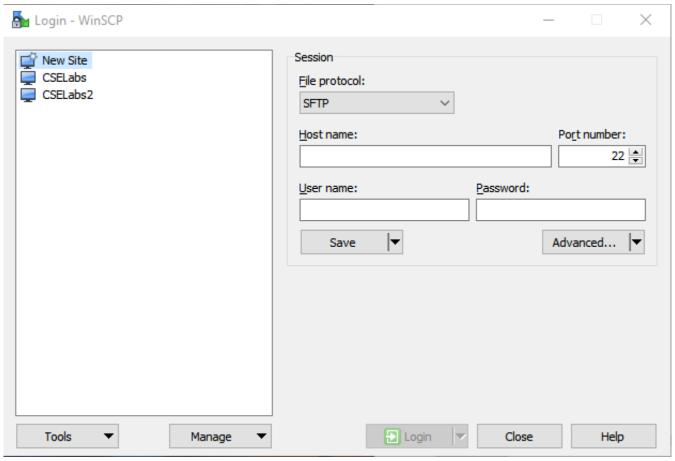
smb: \username\> prompt
smb: \username\> recurse
smb: \username\> mget *DirectoryName*

How do I upload files to my MSI home directory from Windows?

To upload files from your Windows machine to your home directory on a Unix machine, use WinSCP.

How do I use WinSCP to transfer data?

33. These directions explain how to transfer files back and forth between a Windows client and a Unix server. Please see the directions for remote access for an MSI Unix machine from a Windows client if you need to log in interactively.
34. Download and install WinSCP, a graphical SCP (secure copy protocol) file transfer client for Windows. These directions are based on version 5.7.5; upgrade if needed.
35. Double-click the WinSCP icon to reach this screen. If you just want to transfer files to or from your group home or lab scratch, you can skip steps 4 and 5 and proceed to step 6.



File protocol: SFTP

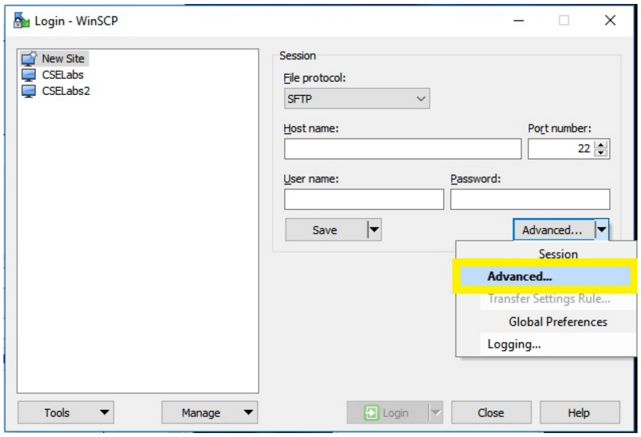
Host name:

Port number: 22

User name:

Password:

36. (**Optional:** if you want to transfer files elsewhere other than login.msi) Click the "Advanced Options" box in the lower left-hand corner:



File protocol: SFTP

Host name:

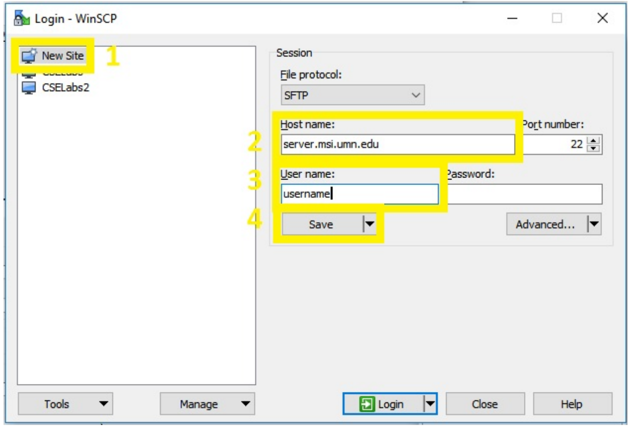
Port number: 22

User name:

Password:

37. (**Optional:** if you want to transfer files elsewhere other than login.msi) Under the Connection section, click on "Tunnel" to get to this screen and click "Connect through SSH tunnel". Fill in **login.msi.umn.edu** and your MSI username. Then click back in the left column on Session at the top.

38. (Mandatory for all cases) Enter the name of the MSI server you wish to use, such as itasca, cascade, hosting, etc, in this format: *server*.msi.umn.edu. (Do not use the actual name server.msi.umn.edu, which does not exist.) **If you skipped steps 4 and 5, the only option here is login.msi.umn.edu.** Enter your MSI username. Click "Save".



File protocol: SFTP

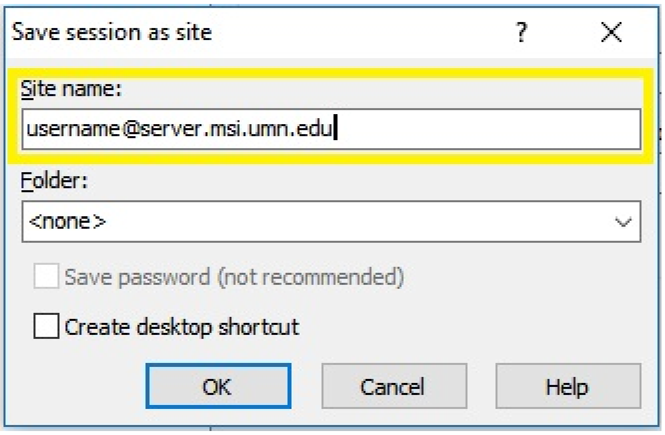
Host name: server.msi.umn.edu

Port number: 22

User name: username

Password:

39. Make sure the session is called something you want it to be called, then click "OK" on this screen:



Save session as site

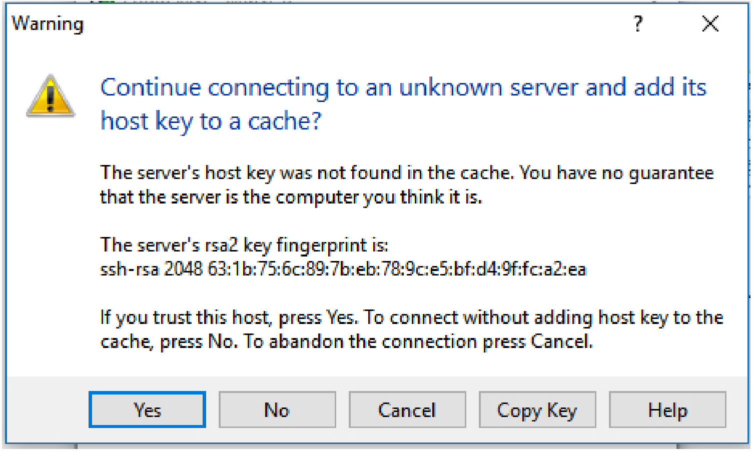
Site name: username@server.msi.umn.edu

Folder: <none>

□ Save password (not recommended)

□ Create desktop shortcut

40. The first time you connect you will get a "Warning" window concerning the server's key fingerprint. This is normal. Click "Yes". You may get this window twice.



Warning

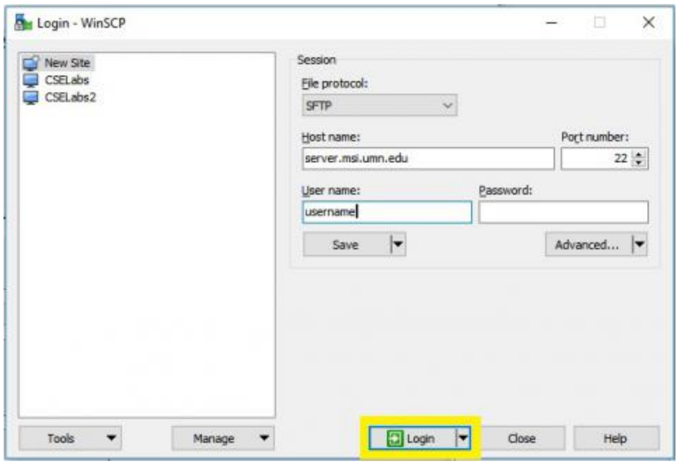
Continue connecting to an unknown server and add its host key to a cache?

The server's host key was not found in the cache. You have no guarantee that the server is the computer you think it is.

The server's rsa2 key fingerprint is:
ssh-rsa 2048 63:1b:75:6c:89:7b:eb:78:9c:e5:bf:d4:9f:fc:a2:ea

If you trust this host, press Yes. To connect without adding host key to the cache, press No. To abandon the connection press Cancel.

41. Login to the session



Login - WinSCP

New Site
CSELabs
CSELabs2

Session

File protocol: SFTP

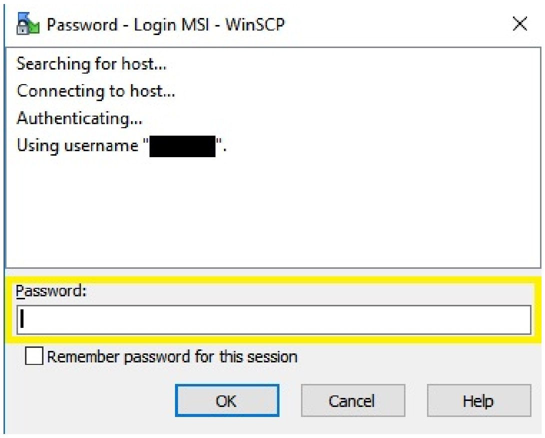
Host name: server.msi.umn.edu

Port number: 22

User name: username

Password:

42. You will now get a window asking for your password. After entering your password click "OK". Do not save your password! For alternatives to saving your password using ssh keys contact help@msi.umn.edu.

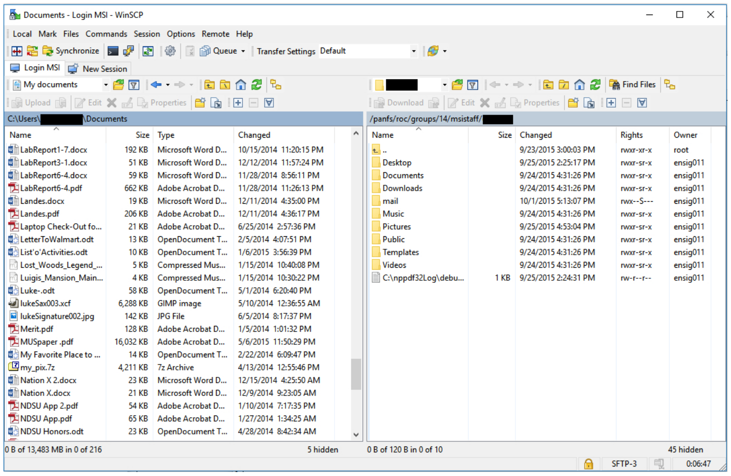


Searching for host...
Connecting to host...
Authenticating...
Using username "[redacted]".

Password:

□ Remember password for this session

43. You may get the "Warning" message again at this point.
44. You may be prompted for the password a second time.
45. Once connected to the remote system, a window will appear showing the local and remote systems.



Transfer Settings Default

C:\Users\[redacted]\Documents

Name	Size	Type	Changed
LabReport1-7.docx	192 KB	Microsoft Word D...	10/15/2014 11:20:15 PM
LabReport3-1.docx	51 KB	Microsoft Word D...	12/12/2014 11:57:24 PM
LabReport6-4.docx	59 KB	Microsoft Word D...	11/28/2014 8:56:11 PM
LabReport6-4.pdf	662 KB	Adobe Acrobat D...	11/28/2014 11:26:13 PM
Landes.docx	19 KB	Microsoft Word D...	12/11/2014 4:35:00 PM
Landes.pdf	206 KB	Adobe Acrobat D...	12/11/2014 4:36:17 PM
Laptop Check-Out fo...	21 KB	Adobe Acrobat D...	6/25/2014 2:57:36 PM
LetterToWalmart.odt	13 KB	OpenDocument T...	2/5/2014 4:07:51 PM
List'o'Activities.odt	10 KB	OpenDocument T...	1/6/2015 3:56:39 PM
Lost_Woods_Legend_...	5 KB	Compressed Mus...	1/15/2014 10:40:08 PM
Luigis_Mansion_Main...	4 KB	Compressed Mus...	1/15/2014 10:30:22 PM
Luke-.odt	58 KB	OpenDocument T...	5/1/2014 6:20:40 PM
lukeSax003.xcf	6,288 KB	GIMP image	5/10/2014 12:36:55 AM
lukeSignature002.jpg	142 KB	JPG File	6/5/2014 8:17:37 PM
Merit.pdf	128 KB	Adobe Acrobat D...	1/5/2014 1:01:32 PM
MUSpaper .pdf	16,032 KB	Adobe Acrobat D...	5/6/2015 11:50:29 PM
My Favorite Place to ...	14 KB	OpenDocument T...	2/22/2014 6:09:47 PM
my_pix.7z	4,211 KB	7z Archive	4/13/2014 12:55:46 PM
Nation X 2.docx	23 KB	Microsoft Word D...	12/15/2014 4:25:50 AM
Nation X.docx	21 KB	Microsoft Word D...	12/9/2014 9:23:05 AM
NDSU App 2.pdf	54 KB	Adobe Acrobat D...	1/10/2014 7:17:35 PM
NDSU App.pdf	65 KB	Adobe Acrobat D...	1/27/2014 1:34:25 AM
NDSU Honors.odt	23 KB	OpenDocument T...	4/28/2014 8:42:34 AM

0 B of 13,483 MB in 0 of 216 5 hidden

[redacted]

/panfs/roc/groups/14/msistaff/[redacted]

Name	Size	Changed	Rights	Owner
..		9/23/2015 3:00:03 PM	rwxr-xr-x	root
Desktop		9/25/2015 2:25:17 PM	rwxr-sr-x	ensig011
Documents		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Downloads		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
mail		10/1/2015 5:13:07 PM	rwx--S---	ensig011
Music		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Pictures		9/25/2015 4:53:04 PM	rwxr-sr-x	ensig011
Public		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Templates		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
Videos		9/24/2015 4:31:26 PM	rwxr-sr-x	ensig011
C:\nppdf32Log\debu...	1 KB	9/25/2015 2:24:31 PM	rw-r--r--	ensig011

0 B of 120 B in 0 of 10 45 hidden

SFTP-3 0:06:47

46. To move to the desired directories, use the navigation bars at the top. The local system is on the left and the remote system on the right. The toolbar at the bottom of the window contains function keys for performing copies, deletes, etc.
47. To perform an action, highlight the desired files or directories and press the appropriate function key (F5 for a copy). It will then prompt you for confirmation. Click "OK". Depending on the size of the file(s) and the speed of your network connection, an upload or download could take anywhere from a few seconds to several minutes.
48. When you are ready to disconnect from the remote system, press F10 or just exit from the WinSCP program.

*****Note**
I saved the microbiome analysis files under the following paths

C:\Users\onyea005\Documents\mice_tutorial
C:\Users\onyea005\Documents\qiime_tutorial

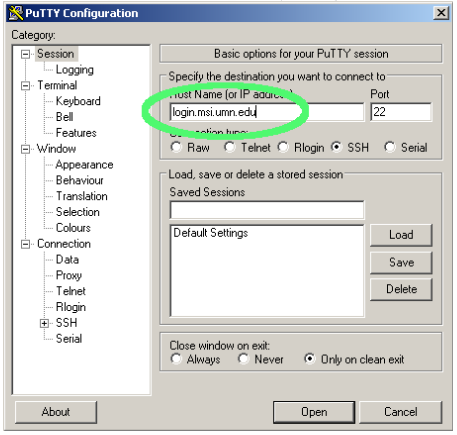
Accessing the Discovering The Microbiome Course Dataset

Installing QIIME

First Things First: Open the Terminal

Using QIIME on MSI

1. Download Putty on Windows (http://www.putty.org/) and configure it for MSI access



Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): login.msi.umn.edu

Port: 22

Connection type: ○ Raw ○ Telnet ○ Rlogin ● SSH ○ Serial

Load, save or delete a stored session

Saved Sessions: Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

25. Connecting to Mesabi and Itasca from Login Host. Once connected to the Login host, note the word login in your terminal prompt, you can connect to Mesabi or Itasca via ssh (QIIME is currently loaded under Itasca so use ssh Itasca for QIIME analyses)

```
ssh mesabi or ssh itasca
```

26. Alternatively, connect to MSI using Terminal on Mac/Linux, located in Applications>Utilities>Terminal.app.

```
ssh yourusername@login.msi.umn.edu
```

27. Log in to an interactive node, making sure you have enough time and memory

```
isub -n nodes=1:ppn=4 -m 16GB -w 24:00:00
```

28. Load the QIIME module

```
module load qiime/1.9.1
```

MacQIIME Users Only:

If you are using MacQIIME, you will have to use your Mac OS X terminal, located in Applications>Utilities>Terminal.app.
Normally, when you run a command in the terminal, all the packages in MacQIIME will be invisible to your computer. In order to access macqiime, after starting a new terminal session you have to source the environment variables with the "macqiime" command. To source the QIIME environment variables and load up all the MacQIIME scripts in the PATH, you have to start the MacQIIME subshell:

```
macqiime
```

You should now be looking at a dollar sign ($) with a cursor after it. This is the command line interface for your computer. Here, you can type commands in order to "execute" programs. This Unix/Linux command line is a powerful place to analyze and process data. Everything you do in QIIME is executed through this command line interface. I'm going to assume you have some familiarity with the Unix command line, but I'll try to help you along the way with commands like **cp**, **cd**, **ls**, **pwd**, **less**, **nano**, etc.

**Note: Getting MacQIIME to work in El Capitan (OS 10.11)

This section is required only if you are running OS 10.11 (El Capitan) or higher. Apple added a new security feature in El Capitan that makes it impossible to install software into /usr/bin/ folder (that was where I was putting the “macqiime” sourcing script. I'll fix this in the next release of MacQIIME. In the meantime, we can work around it. The quick solution is to use this command, instead of the “macqiime” script:

```
source /macqiime/configs/bash_profile.txt
```

## Get your dataset: you can either download the dataset from the source, then use the cd command to move to the folder where the data is located as follows, or you can download the file through the terminal window with the wget command (followed by the unzip and mv commands to get the uncompressed data). We will show the cd method here:

```
cd ~/qiime_tutorial/
```

This changes the directory to user\qiime_tutorial, Now you can look at the contents of the directory using the **ls** command:

```
ls –lh
```

You can always check the location of your current directory (aka folder) using the **pwd** command. If you type:

```
pwd
```

***** To copy any line code from the tutorial to the terminal window, select the code and hit CTRL + C, then in the control window right click and it will copy the code**

***** To quit any command that is running in the terminal, hit q**

Running QIIME on class data

Go to https://github.com/danknights/mice8992-2016
The course code browser is available at
http://metagenome.cs.umn.edu/microbiomecodebrowser/doc/index.html
You can also run QIIME on the global gut data in the Course repository. First download the repository and data:

Copy the file to /Users/chigu142000/mice_tutorial
For example, move to the globalgut-66-adults data directory:

```
cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd globalgut-66-adults

* You can always check the location of your current directory (aka folder) using the pwd command.

pwd
```

This directory contains sequences from 66 adults in 3 countries (Yatsunenko et. al. Nature 2012). Then run OTU picking and core QIIME analyes:

```
# pick closed reference OTUs time (need to extract the seq.fna file)

pick_closed_reference_otus.py -i seqs.fna -o closedref

# run core QIIME diversity analyses on closed-reference OTU table

time core_diversity_analyses.py -i closedref/otu_table.biom -m map.txt -o closedref/corediv -e 500 --tree_fp closedref/97_otus.tree -v

# pick de novo OTUs

time pick_de_novo_otus.py -i seqs.fna -o denovo

# run core QIIME diversity analyses on de novo OTU table

time core_diversity_analyses.py -i denovo/otu_table.biom -m map.txt -o denovo/corediv -e 500 --tree_fp denovo/rep_set.tre
```

You can then open the “index.html” file in the output folder in your favorite web browser, and click to see the results of the analysis. In Chrome you may need to open Chrome first, and then choose “File–>Open File…”. For the Global Gut data set, click on “index.html” next to PCoA plot (weighted_unifrac). Then click the “Colors” tab in the upper right and select “COUNTRY” by which to color the points.

Getting usage info from QIIME scripts

First let’s move to the Guerrero Negro data directory. This directory contains sequences from different depths in the Guerrero Negro microbial in Mexico (Harris JK **et al.** ISME J. 2013 Jan;7(1):50-60).

We are going to run closed reference OTU picking, but first let’s get usage information and list of options (be sure QIIME is loaded first)

```
pick_closed_reference_otus.py -h
```

pick_closed_reference_otus.py is a “workflow” script that runs other scripts.

Ask pick_closed_reference_otus.py to print the other commands it would run. The -f tells it to force overwriting the directory closedref, in case that directory is already there. The -r tells it where the reference sequences are. The -t tells it where the reference taxonomy map is.

```
pick_closed_reference_otus.py -i seqs.fna -o closedref -f -r ../ref/greengenes/97_otus.fasta -t ../re
f/greengenes/97_otu_taxonomy.txt -w
```

```
## pick_otus.py -i seqs.fna -o closedref/uclust_ref_picked_otus -r ../ref/greengenes/97_otus.fasta
 -m uclust_ref  --suppress_new_clusters

## make_otu_table.py -i closedref/uclust_ref_picked_otus/seqs_otus.txt -t ../ref/greengenes/97_o
tu_taxonomy.txt -o closedref/otu_table.biom
```

Picking OTUs

Move to the Guerrero Negro data directory. This directory contains sequences from different depths in the Guerrero Negro microbial in Mexico (Harris JK **et al.** ISME J. 2013 Jan;7(1):50-60).

```
 # To be run on the command line

cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd guerreronegro
```

Picking closed reference OTUs with QIIME

Now actually run the script. The -v tells it to be “verbose”, printing updates as it runs.
Use time to report how long it takes.

```
time pick_closed_reference_otus.py -i seqs.fna -o closedref -r ../ref/greengenes/97_otus.fasta -t ..
/ref/greengenes/97_otu_taxonomy.txt -f –v
```

```
##Pick OTUs (needed to unzip the 97_OTU file first)

pick_otus.py -i seqs.fna -o closedref/uclust_ref_picked_otus -r /Users/chigu142000/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otus.fasta -m uclust_ref  --suppress_new_clusters

## on the MSI server (use pwd to check the path)

pick_otus.py -i seqs.fna -o closedref/uclust_ref_picked_otus -r /home/prizm001/onyea005/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otus.fasta -m uclust_ref  --suppress_new_clusters

## Make OTU table

make_otu_table.py -i closedref/uclust_ref_picked_otus/seqs_otus.txt -t /Users/chigu142000/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otu_taxonomy.txt -o closedref/otu_table.biom

## on the MSI server (use pwd to check the path)

make_otu_table.py -i closedref/uclust_ref_picked_otus/seqs_otus.txt -t /home/prizm001/onyea005/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otu_taxonomy.txt -o closedref/otu_table.biom
```

Find out how many sequences were assigned (“Total count”) by uclust_ref (QIIME default)

```
biom summarize-table -i closedref/otu_table.biom > closedref/stats.txt head closedref/stats.txt
```

Picking closed-reference OTUs with NINJA-OPS

Compare to NINJA-OPS. Installation instructions are at https://github.com/GabeAl/NINJA-OPS. (move to Users/chigu142000/mice_tutorial/NinjaOps/)

**You need to install NINJA OPS

Where to get Bowtie2

https://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.2.8/

1. Download the appropriate package for your system from the above URL (not source).

2. Extract the bowtie2-align-s file from the archive into the main NINJA directory (the directory that contains this file).

**Then I used WINSCP to transfer the extracted files from my Windows desktop to the MSI server

3. Make sure it's executable by running 'chmod +x bowtie2-align-s' in the directory into which you extracted it.(**Could not get the ninjaops package to run after placing bowtie 2 in the folder

as directed, it could work if placed in another folder but I would rather not spend the time testing this manually since QIIME works already. It might also be because I have downloaded the WIN64 version of bowtie 2 and the MSI server cannot use that since it is technically a unix machine? Also loaded the linux version just in case)

```
cd ~/mice_tutorial

cd NinjaOps

cd NINJA-OPS-master

chmod +x bowtie2-align-s.exe (WIN64 version)

chmod +x bowtie2-align-s(Linux version)
```

***Note NINJA-OPS is much faster.

```
cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd guerreronegro

time python Users/chigu142000/mice_tutorial/NinjaOps/NINJA-OPS-master/bin/ninja.py -i seqs.fna -o ninja
```

```
## on the MSI server (use pwd to check the path)

time python /home/prizm001/onyea005/mice_tutorial/NinjaOps/NINJA-OPS-master/bin/ninja.py -i seqs.fna -o ninja
```

Find out how many sequences were assigned ("Total count") by NinjaOps

```
biom summarize-table -i ninja/ninja_otutable.biom > ninja/stats.txt head ninja/stats.txt
```

Try NINJA-OPS on the Global Gut data set. It can process 1 million sequences in the Global Gut data set in under 20 seconds on a Macbook.

```
cd ../globalgut time python Users/chigu142000/mice_tutorial/NinjaOps/NINJA-OPS-master/bin/ninja.py -i seqs.fna -o ninja

## on the MSI server (use pwd to check the path)

cd ../globalgut time python /home/prizm001/onyea005/mice_tutorial/NinjaOps/NINJA-OPS-master/bin/ninja.py -i seqs.fna -o ninja
```

Picking de novo OTUs with QIIME

Run the workflow script, pick_de_novo_otus.py with -w to print the basic commands it would run.

```
cd ../guerreronegro

pick_de_novo_otus.py -i seqs.fna -o openref -f -w
```

Now actually run the script. Note that the OTU picking step is quick on this data set. The other steps will take a long time (more than 10 minutes). So instead, kill the script by typing ctrl-c.

```
pick_de_novo_otus.py -i seqs.fna -o openref -f -v
```

Instead let's just run the pick_otus.py step, with several different OTU picking methods.

Method 1: uclust

```
time pick_otus.py -i seqs.fna -o uclust -m uclust

make_otu_table.py -i uclust/seqs_otus.txt -o uclust/otu_table.biom

biom summarize-table -i uclust/otu_table.biom | head -n 5
```

Method 2: swarm

```
time pick_otus.py -i seqs.fna -o swarm -m swarm make_otu_table.py -i swarm/seqs_otus.txt -o s
warm/otu_table.biom biom summarize-table -i swarm/otu_table.biom | head -n 5
```

Method 3: cdhit

```
time pick_otus.py -i seqs.fna -o cdhit -m cdhit make_otu_table.py -i cdhit/seqs_otus.txt -o cdhit/
otu_table.biom biom summarize-table -i cdhit/otu_table.biom | head -n 5
```

Method 4: sumaclust

```
time pick_otus.py -i seqs.fna -o sumaclust -m sumaclust make_otu_table.py -i sumaclust/seqs_ot
us.txt -o sumaclust/otu_table.biom biom summarize-table -i sumaclust/otu_table.biom | head -n 5
```

Customizing workflow scripts

We can customize workflow scripts using a parameter file. You will need some kind of text editor to edit the parameter file, such as Notepad or Notepad++ for Windows, TextWrangler or Sublime for Mac, or Emacs or vi for the Mac/Linux command line. For more information, please see the official QIIME explaining parameter files here: http://qiime.org/documentation/qiime_parameters_files.html.

For example, let’s assume we want to customize the OTU picking method used by (pick_otus.py) as part of the pick_de_novo_otus.pyworkflow, changing it from the default uclust to swarm. To find the parameter name, we can either run pick_otus.py -h or read the description at qiime.org: http://qiime.org/scripts/pick_otus.html (found by Googling **QIIME pick_otus.py**). We see that the parameter we need is called otu_picking_method, and that the valid options are sortmerna, mothur, trie, uclust_ref, usearch, usearch_ref, blast, usearch61, usearch61_ref, sumaclust, swarm, prefix_suffix, cdhit, uclust.

Therefore we need to add pick_otus:otu_picking_method swarm to a text file. This can be done with emacs as follows, if you have emacs on your system:

```
emacs parameters-swarm.txt
```

Then add these lines to the top of the file:

```
pick_otus:otu_picking_method swarm  pick_otus:similarity .99
```

Then save the file with ctrl-x ctrl-s, then exit with ctrl-x ctrl-c.

Now we can run the workflow script with the custom parameters file:

```
pick_de_novo_otus.py -i seqs.fna -o swarm99 -p parameters-swarm.txt
```

Making OTU tables human-readable

You can convert OTU tables to tab-delimited output using biom convert:

```
biom convert -i closedref/otu_table.biom -o closedref/otu_table.txt --to-tsv
```

The output file can now be opened in Excel.

Follow-up exercises

5. How can you use filter_otus_from_otu_table.py to determine the number of singleton OTUs produced by each OTU picking method?
6. How many OTUs remain from each method after filtering singletons?
7. Run OTU picking on another data set from EBI or QIITA.
8. Try (loading OTU tables into R)[]

Loading Microbiome Data

All of the code in this page is meant to be run in R unless otherwise specified.

After you have installed QIIME and generated an OTU table, you can load the OTU table into R.

Install biom package if not installed.

```
install.packages('biom',repo='http://cran.wustl.edu')
```

R - ***Installing R on your system and adding the following libraries has become much more important in the latest QIIME versions 1.9+.*** There are now many neat features in QIIME that require R. If you don't have R installed, you can get it from here. *Please note* that even if you installed R and these libraries previously for MacQIIME 1.8.0, you should still upgrade to/install the latest version of R, at least version **3.1.2**, and re-install all these R packages to get everything working. Make sure the R executable is in your PATH. You will have to open R and install additional packages: in the R command line, do the following, in the following order, one line at a time. **(*** Figure out how to pull out the script window in R for MAC: Open R > Open a document editor, type your code there, then copy and paste into the R code window as needed)**

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("DESeq2")
biocLite("biomformat")
biocLite("metagenomeSeq")

install.packages('randomForest')
install.packages('ape')
install.packages('vegan')
install.packages('optparse')
install.packages('gtools')
install.packages('klaR')
install.packages('RColorBrewer')
```

*** it seems that you need to load the **biocLite("Biomformat") package instead of the install.packages("biom") or the** install.packages('biom',repo='http://cran.wustl.edu')

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("biomformat")
```

Load biom package

```
library('biomformat')
```

*** I saved the R editor File in the mice_tutorial folder

Getting Information on a Dataset

There are a number of functions for listing the contents of an object or dataset.

```
# list objects in the working environment
ls()
# list the variables in mydata
names(mydata)
# list the structure of mydata
str(mydata)
# list levels of factor v1 in mydata
levels(mydata$v1)
# dimensions of an object
dim(object)
# class of an object (numeric, matrix, data frame, etc)
class(object)
# print mydata
mydata
# print first 10 rows of mydata
head(mydata, n=10)
# print last 5 rows of mydata
tail(mydata, n=5)
```

Convert BIOM file to JSON format.

If you have data in a “new” BIOM format (HDF5), you first need to convert to JSON format first

The following code is to be run on the command line.

```
# make a JSON-formatted OTU table for loading into R

cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd globalgut-66-adults

biom convert -i otu_table.biom -o otu_table_json.biom --to-json

biom convert -i otu_table.biom -o otu_table_json2.biom --to-json

**Don’t use these options**

biom convert -i otu_table.biom -o otu_table_hdf5.biom --to-hdf5

biom convert -i otu_table.biom -o otu_table_hdf5.HDF5 --to-hdf5

biom convert -i otu_table.biom -o otu_table_json3.biom --table-type="OTU table" --to-json
```

Load global gut data using biom package

** Whenever you want to run something inR on the windows side of things from MSI, you will need to use WINSCP to copy the file from the MSI server into windows first.

```
gg.otus.biom <- read_biom('../data/globalgut-66-adults/otu_table_json.biom')
```

```
*** The fix was to convert the file to an HDF5 format instead of the JSON format using

biom convert -i otu_table.biom -o otu_table_hdf5.biom --to-hdf5

***Then instead of using the basic read_biom R function in the tutorial

g.otus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/data/globalgut-66-adults/otu_table_hdf5.biom')

use the read_hdf5_biom import file in R

gg.otus.biom <- read_hdf5_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_hdf5.biom')

***or you can use this version too! (with the full path name)

gg.otus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_json2.biom')

**or on PC after MSI transfer(**remember to use the '/' instead of the '\' in R!!!)

gg.otus.biom <- read_biom('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_json2.biom')
```

Extract data matrix (OTU counts) from biom table (need the json file, as the HDF5 file bugs out using biom_data)

```
gg.otus <- as.matrix(biom_data(gg.otus.biom))

# transpose so that rows are samples and columns are OTUs

gg.otus <- t(gg.otus)
```

Plot histogram of sample depths

```
depths <- rowSums(gg.otus)

hist(depths,breaks=30)
```

Plot histogram of OTU frequencies

```
otu.counts <- colSums(gg.otus > 0)

hist(otu.counts,breaks=30)
```

Remove OTUs present in < 10% of samples

```
gg.otus <- gg.otus[,colMeans(gg.otus > 0) >= .1]

depths <- rowSums(gg.otus)
```

```
dim(gg.otus)
```

Re-plot histogram of OTU frequencies now that we removed singletons

```
otu.counts <- colSums(gg.otus > 0)

hist(otu.counts,breaks=30)
```

Remove any samples with very low depth

```
sort(depths)[1:10]
```

Load mapping file

```
gg.map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/map.txt',sep='\t',head=T,row=1,check=F,comment='')

**or on PC after MSI transfer(**remember to use the ‘/’ instead of the ‘\’ in R!!!)

gg.map <- read.table('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/map.txt',sep='\t',head=T,row=1,check=F,comment='')
```

Ensure those mapping file and OTU tables contain the sample samples in the same order

```
sample.ids <- intersect(rownames(gg.otus), rownames(gg.map))

# might as well put the samples in alphabetical order

sample.ids <- sort(sample.ids)

# in R you can subset using sample IDs or numerical indices. Most languages only use indices.

gg.otus <- gg.otus[sample.ids,]

gg.map <- gg.map[sample.ids,]

dim(gg.otus)

dim(gg.map)
```

Beta Diversity (Guerrero Negro)

All of the code in this page is meant to be run in R unless otherwise specified.

Install biom package and vegan package if not installed.

*** it seems that you need to load the **biocLite("Biomformat") package instead of the install.packages("biom") or the** install.packages('biom',repo='http://cran.wustl.edu')

source("http://bioconductor.org/biocLite.R") biocLite ()

biocLite("biomformat")

install.packages('vegan')

Load the packages, load data

```
library('biomformat')

library('vegan')

# load biom file

otus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerrero
negro/otu_table_json.biom')

**or on PC after MSI transfer(**remember to use the ‘/’ instead of the ‘\’ in R!!!)

otus.biom <- read_biom('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data
/guerreronegro/otu_table_json.biom')

# Extract data matrix (OTU counts) from biom table

otus <- as.matrix(biom_data(otus.biom))

# transpose so that rows are samples and columns are OTUs

otus <- t(otus)

# load mapping file

map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerreronegro/
map.txt',sep='\t',head=T,row=1,check=F,comment='')

**or on PC after MSI transfer(**remember to use the ‘/’ instead of the ‘\’ in R!!!)

map <- read.table('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/guerr
eronegro/map.txt',sep='\t',head=T,row=1,check=F,comment='')
```

It is extremely important to ensure that your OTU table and metadata table sample IDs are lined up correctly.

```
# see rownames of map and otus

rownames(map)

rownames(otus)

# find the overlap

common.ids <- intersect(rownames(map), rownames(otus))

 # get just the overlapping samples
```

```
otus <- otus[common.ids,]
map <- map[common.ids,]
```

See dimensions of OTU table

```
dim(otus)
```

See dimensions of mapping file

```
dim(map)
```

Get three different distances metrics

```
# get Euclidean distance
d.euc <- dist(otus)
# get Bray-Curtis distances (default for Vegan)
d.bray <- vegdist(otus)
# get Chi-square distances using vegan command
# we will extract chi-square distances from correspondence analysis
my.ca <- cca(otus)
d.chisq <- as.matrix(dist(my.ca$CA$u[,1:2]))
```

Now run principal coordinates embedding on the distance metrics

```
# Run PCoA (not PCA)
pc.euc <- cmdscale(d.euc, k=2)
# Bray-Curtis principal coords
pc.bray <- cmdscale(d.bray,k=2)
# get first two dimensions of chi-square coordinates:
pc.chisq <- my.ca$CA$u[,1:2]
```

Plot Euclidean distances with gradient colors

```
# makes a gradient from red to blue
my.colors <- colorRampPalette(c('red','blue'))(10)
# plot Euclidean PCoA coords using color gradient
# based on layer (1...10) *** There was no layer variable in the mapping file (PS it did, I initially
 pulled the global gut files instead of the guerreronegro ones)
names(map)
layer <- map[,'LAYER']
plot(pc.euc[,1], pc.euc[,2], col=my.colors[layer], cex=3, pch=16)
```

Plot Bray-Curtis distances with gradient colors

```
# Plot Bray-Curtis PCoA
plot(pc.bray[,1], pc.bray[,2], col=my.colors[layer], cex=3, pch=16)
```

Plot Chi-square distances with gradient colors

```
# Plot Chi-square PCoA
plot(pc.chisq[,1], pc.chisq[,2], col=my.colors[layer], cex=3, pch=16)
```

Visualizing UniFrac distances

Calculate UniFrac distances in QIIME on the guerrero negro dataset

```
# Note: This command is on the command line, not in R # (load macqiime if necessary)
cd ~/mice_tutorial
cd mice8992-2016-master
cd data
cd guerreronegro
beta_diversity.py -i otu_table.biom -o beta -t /Users/chigu142000/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otus.tree
**or on MSI
beta_diversity.py -i otu_table.biom -o beta -t /home/prizm001/onyea005/mice_tutorial/mice8992-2016-master/data/ref/greengenes/97_otus.tree
```

Load UniFrac distances, calculate PCoA

```
# load unweighted and weighted unifrac
d.uuf <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerreronegro/beta/unweighted_unifrac_otu_table.txt', sep='\t',head=T,row=1, check=F)
```

```
d.wuf <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/guerreronegro/beta/weighted_unifrac_otu_table.txt', sep='\t',head=T,row=1, check=F)

**or on PC after MSI Transfer

d.uuf <- read.table(‘/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/guerreronegro/beta/unweighted_unifrac_otu_table.txt', sep='\t',head=T,row=1, check=F)

d.wuf <- read.table('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/guerreronegro/beta/weighted_unifrac_otu_table.txt', sep='\t',head=T,row=1, check=F)

# ensure that these last two matrices have the same samples in the  # same order as the metadata
 table

d.uuf <- d.uuf[common.ids, common.ids]

d.wuf <- d.wuf[common.ids, common.ids]

# get first two dimensions of unifrac PCoA:

pc.uuf <- cmdscale(d.uuf, k=2)

pc.wuf <- cmdscale(d.wuf, k=2)
```

Plot unweighted UniFrac distances with gradient colors (***There was no Layer variable in the mapping file)

```
plot(pc.uuf[,1], pc.uuf[,2], col=my.colors[layer], cex=3, pch=16)
```

Plot weighted UniFrac distances with gradient colors

```
plot(pc.wuf[,1], pc.wuf[,2], col=my.colors[layer], cex=3, pch=16)
```

Note: to make a PDF:

```
pdf("chisq.pdf",width=5,height=5) plot(pc.chisq[,1], pc.chisq[,2], col=my.colors[map[,'LAYER']], cex=3, pch=16) dev.off()
```

Let’s plot pairwise comparisons of the different distance metrics

```
d.vector.matrix <- cbind(as.numeric(d.euc), as.numeric(d.bray), as.numeric(as.dist(d.chisq)), as.numeric(as.dist(as.matrix(d.uuf))), as.numeric(as.dist(as.matrix(d.wuf)))) colnames(d.vector.matrix) <- c('Euc','BC','ChiSq','UUF','WUF') pairs(d.vector.matrix)
```

And display the pairwise pearson correlations

```
cor(d.vector.matrix)
```

Which distance metric best recovered physical sample distances based on END_DEPTH?

*** There was no endepth variable in the map file (PS there was, I initially pulled the global gut fules instead of the guerrero negro ones)

```
# y is the euclidean distance matrix based on ending depth of each layer
y <- as.vector(dist(map$END_DEPTH))
# Test the correlation of END_DEPTH distance and ecological distance
# for each metric
metrics <- list(d.euc, d.bray, d.chisq, d.uuf, d.wuf)
names(metrics) <- colnames(d.vector.matrix)
# reuse pairwise column names
for(i in 1:length(metrics)){     d.name <- names(metrics)[i]
# convert distance matrix to vector form
d <- as.vector(as.dist(metrics[[i]]))
cat('Correlation of ',d.name,':','\n',sep='')
print(cor.test(d, y, method='spear', exact=FALSE)) }
```

Here is the one-liner version if you just want to test one distance metric vs. your continuous variable of interest. There are two ways to do this:

2. ask whether the **overall** distance metric is correlated with your gradient:

```
# y is the euclidean distance matrix based on your variable of interest (here depth)
# note: euclidean diatnace makes sense in the Guerrero Negro sampling depth because
# it is measuring physical distance (in millimeters)
# Note: the "as.vector" stretches it out to a single numeric vector (no longer a matrix)
y <- as.vector(dist(map$END_DEPTH))
# Stretch out the d
d <- as.vector(as.dist(d.chisq)) cor.test(d, y, method='spear', exact=FALSE)
```

3. You could ask whether PC1, the **first principal axis of variation** (not overall distance), is significantly correlated with your variable of interest. This is an even stronger result if significant.

```
cor.test(map$END_DEPTH, pc.chisq[,1], method='spear', exact=FALSE)
```

Statistical Analysis (Global Gut)

All of the code in this page is meant to be run in R unless otherwise specified.

Loading a genus table and the metadata into R

Before loading data into R, this QIIME command must be run on the command line to collapse OTU counts into genus (L6) and phylum (L2) count tables:

```
cd ~/mice_tutorial

cd mice8992-2016-master

cd data

cd globalgut

# (run on command line)

summarize_taxa.py -i otu_table.biom -L 6

summarize_taxa.py -i otu_table.biom -L 2

# convert to JSON BIOM format to load into R using R biom package:

biom convert -i otu_table_L6.biom -o otu_table_L6_json.biom --to-json

biom convert -i otu_table_L2.biom -o otu_table_L2_json.biom --to-json
```

Inside R, Install and load the updated biom package and the vegan package

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("biomformat")
install.packages('vegan')
library('biomformat')
library('vegan')
```

```
# load biom file

genus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut/otu_table_L6_json.biom')

** Or on PC after MSI transfer

/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/

genus.biom <- read_biom('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut/otu_table_L6_json.biom')

# Extract data matrix (genus counts) from biom table

genus <- as.matrix(biom_data(genus.biom))
```

```
# transpose so that rows are samples and columns are genera

genus <- t(genus)

# load mapping file

map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut/map.txt', sep='\t', comment='', head=T, row.names=1)

** Or on PC after MSI transfer

/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/

map <- read.table('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut/map.txt', sep='\t', comment='', head=T, row.names=1)
```

It is extremely important to ensure that your genus table and metadata table sample IDs are lined up correctly.

```
# find the overlapping samples

common.ids <- intersect(rownames(map), rownames(genus))

# get just the overlapping samples

genus <- genus[common.ids,]

map <- map[common.ids,]
```

See dimensions of genus table. Then drop genera present in < 10% of samples.

```
dim(genus)

genus <- genus[,colMeans(genus > 0) >= .1]

dim(genus)
```

Show only the first ten genera in genus table

```
colnames(genus)[1:10]
```

Abbreviate the taxonomic names to make them easier to display.

```
colnames(genus) <- sapply(strsplit(colnames(genus),';'),function(xx) paste(paste(substr(xx[-c(1,length(xx))],4,7),collapse=';'),substring(xx[length(xx)],4),sep=';'))
```

Show the first 10 rows and first 2 columns of the genus table

```
genus[1:10,1:2]
```

See available columns in the metadata

```
colnames(map)
```

Show how many samples are from each Country

```
table(map$COUNTRY)
```

Basic association testing

Let’s run some tests on Prevotella. First extract the Prevotella column and save it to a variable prevotella for convenience.

```
# find out what column Prevotella is in # the "$" tells grep to find only column names that end with "Prevotella".
grep('Prevotella$',colnames(genus))
# save that column in a variable
prevotella <- genus[,grep('Prevotella$',colnames(genus))]
```

Visualize the distribution of Prevotella

```
# find out what column Prevotella is in
hist(prevotella, br=30)
```

Run a test of Pearson’s correlation of Prevotella and age. Note that the result is not quite significant (p=0.0531).

```
cor.test(prevotella, map$AGE)
```

Now run a linear regression of prevotella against age. Notice that statistically this is equivalent to running the Pearson’s correlation. The p-value in row 2 column 4 of the “Coefficients” table is the same as the p-value from the correlation test.

```
# fit a linear model. The "~" means "as a function of"
fit <- lm(prevotella ~ map$AGE)
# print a summary of the results
summary(fit)
# A nice way to get the exact p-value for the age regression coefficient using the anova function
pval <- anova(fit)['map$AGE','Pr(>F)']
```

```
pval
```

Testing for normally distributed data

We can test whether the residuals are normally distributed Kolmogorov-Smirnov test. If p < 0.05, we can reject the null hypothesis that the data came from a normal distribution, meaning that the linear test is not appropriate.

```
# Make a quantile-quantile plot of the (studentized) residuals vs. a normal distribution

qqnorm(rstudent(fit)); abline(0,1)

# Kolmogorov-Smirnov test

ks.test(rstudent(fit), pnorm, mean=mean(rstudent(fit)), sd=sd(rstudent(fit)))
```

Controlling for confounders

Perhaps country of origin is a confounder that is obscuring the association of Prevotella and Age. Using lm() we can add confounders to the regression. Now after removing the effects of country, there is a strong association of Prevotella and age.

```
# fit a linear model. The "~" means "as a function of"

fit <- lm(prevotella ~ map$AGE + map$COUNTRY)

# print a summary of the results

summary(fit)
```

Testing multiple hypotheses

We have so far only tested one genus. Let’s test them all using a loop.

```
# pvals is a vector initialized with zeroes # with enough slots for the different genera

pvals <- numeric(ncol(genus))

# "name" the pvalues after the genera

names(pvals) <- colnames(genus)

# Loop through the columns of the genus table, testing each one

for(i in 1:ncol(genus)) {     fit <- lm(genus[,i] ~ map$AGE + map$COUNTRY)     pvals[i] <-  an
ova(fit)['map$AGE','Pr(>F)'] }

# note, you could put this all on one line with: # for(i in 1:ncol(genus)) pvals[i] <-  anova(lm(ge
nus[,i] ~ map$AGE + map$COUNTRY))['map$AGE','Pr(>F)']

# print the 10 smallest p-values:
```

```
sort(pvals)[1:10]

# "apply" with genus, 2 means do something to every column of genus

# ("apply" with genus, 1 would mean do something every row)

# the last part defines a new function to do to each column, which

# will be passed in the temporary variable named "xx"

pvals <- apply(genus, 2, function(xx) anova(lm(xx ~ map$AGE + map$COUNTRY))['map$AGE','Pr(>F)'])

# print the 10 smallest p-values:

sort(pvals)[1:10]
```

Looks like there are some significant p-values. But did they happen just by chance? There are 97 columns in the genus table, so that means we did 97 tests, and about 5% of them should be p < 0.05 just by chance. To correct for this, we can adjust the p-values using the p.adjustfunction. Here we are correcting for multiple hypothesis testing use False Discovery Rate (FDR). The adjusted p-values are often called “q-values.”

```
qvals <- p.adjust(pvals,'fdr')

# print the lowest 10 q-values

sort(qvals)[1:10]
```

Testing with generalized linear regression.

Let’s test whether the residuals from linear regression were normally distributed for all of the taxa. To follow along, you can put the following code in a separate text file called stats.r (or whatever you want to call it, and then call it using source('stats.r').

```
ks.pvals <- numeric(ncol(genus))

# "name" the pvalues after the genera

names(ks.pvals) <- colnames(genus)

# turn annoying warnings off  options(warn=-1)

# Loop through the columns of the genus table, testing each one (Need to doublecheck the nomenclature on github)

for(i in 1:ncol(genus)) {     fit <- lm(genus[,i] ~ map$AGE + map$COUNTRY)     ks.pvals[i] <- ks.test(rstudent(fit), pnorm, mean=mean(rstudent(fit)), sd=sd(rstudent(fit)),exact=FALSE)$p.value }

# turn warnings back on options(warn=0)

# Now since we ran 97 tests we should correct for multiple hypothesis testing
```

```
ks.qvals <- p.adjust(ks.pvals,'fdr')
# print the lowest 10 q-values
sort(ks.qvals)[1:10]
```

Let’s use a negative binomial distribution instead. We will use the edgeR package. If you don’t have it, you can install it with:

```
source("https://bioconductor.org/biocLite.R")
# If the previous command doesn't work, try http://
biocLite("edgeR")
library(limma)
library(edgeR)
```

Note: the negative binomial uses raw counts of sequences (rarefied or not), not the relative abundances. Therefore we must re-run summarize_taxa.py with the -a flag to use absolute abundance. These commands are run on the command line (not in R)

```
# (run on command line)
summarize_taxa.py -i otu_table.biom -L 6 -a -o taxa-absolute
# convert to JSON BIOM format to load into R using R biom package:
biom convert -i taxa-absolute/otu_table_L6.biom -o taxa-absolute/otu_table_L6_json.biom --to-json
# load biom file
genus.biom <- read_biom('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut/taxa-absolute/otu_table_L6_json.biom')
** Or on PC after MSI transfer
/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/
genus.biom <- read_biom('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut/taxa-absolute/otu_table_L6_json.biom')
# load mapping file
map <- read.table('/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut/map.txt', sep='\t', comment='', head=T, row.names=1)
** Or on PC after MSI transfer
/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/
```

```
map <- read.table('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/global
gut/map.txt', sep='\t', comment='', head=T, row.names=1)
```

Now we need to process the genus table again. (Extract, transpose, cross-reference to mapping file, and remove OTUs with less than 10% presence in our samples)

```
genus.a <- as.matrix(biom_data(genus.biom))

genus.a <- t(genus.a)

genus.a <- genus.a[common.ids,]

genus.a <- genus.a[,colMeans(genus.a > 0) >= .1]

colnames(genus.a) <- sapply(strsplit(colnames(genus.a),';'),function(xx) paste(paste(substr(xx[-c
(1,length(xx))],4,7),collapse=';'),substring(xx[length(xx)],4),sep=';'))
```

First let us run the regression without covariates. The main function in this script is glm.edgeR(). This function will perform multiple hypothesis testing on all columns of a data matrix. It has the following main parameters:
- x: the independent variable. If discrete must be binary; OK to be continuous. - Y: A matrix with samples in rows and dependent variables in columns - covariates: a matrix of additional covariates you want to control for (default NULL)

The glm.edgeR function seems to be from the custom wrapper designed by Dr. Dan Knights. I did not find it in the functions from the edgeR packade on the biocoductor website

You have to source the following file in the SRC folder from the mice_tutorial

source('/Users/chigu142000/mice_tutorial/mice8992-2016-master/src/wrap.edgeR.r')

```
** Or on PC after MSI transfer

/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/
```

source('/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/src/wrap.edgeR.r')

```
result <- glm.edgeR(x=map$AGE, Y=genus.a)
```

We can print the top “tags” (genera) using the topTags function. Note that two genera are significant even after correction for multiple hypothesis testing.

```
topTags(result)
```

If we plot all p-values coming from edgeR in a quantile-quantile plot, we see that they mostly follow the null (uniform) distribution:

```
pvals <- topTags(result,n=Inf)$table[,'PValue'] plot(-log10(seq(0,1,length=98)[-1]), -log10(sort(
pvals))); abline(0,1)
```

However, we have not controlled for COUNTRY, which may be a confounder. We can do this with edgeR.

```
result <- glm.edgeR(x=map$AGE, Y=genus.a, covariates=map$COUNTRY)

topTags(result)
```

Note that no genera are significantly associated with age after controlling for country, and using the negative binomial as the assumed distribution for the residuals.

We can also pass in a matrix of covariates like this, although note that SEX is not a good variable here because there are only 6 males and there are 6 with unknown gender:

```
result <- glm.edgeR(x=map$AGE, Y=genus.a, covariates=map[ , c('COUNTRY','SEX')])
```

Let us test whether any genera are significantly associated with being from the USA, while controling for age:

```
# make a vector that is TRUE if sample is from USA, FALSE otherwise

# the "==" means "test if equal"

is.USA <- map$COUNTRY == "GAZ:United States of America"

# run with is.USA as the independent variable result <- glm.edgeR(x=is.USA, Y=genus.a, covari
ates=map$AGE)

topTags(result)
```

Write the results to a tab-delimited file (to open in Excel) with:

```
write.table(topTags(result, n=Inf)$table, file='edgeR_results.txt',sep='\t',quote=FALSE, col.name
s=NA)
```

How many genera were significantly associated with the USA?

```
sum(topTags(result,n=Inf)$table$FDR <= 0.05)
```

Non-parametric tests

Whenever we do not need to control for confounding variables, we can use non-parametric tests. These are the safest because they don’t rely on any null distribution (e.g. normal). They typically consider only the **ranks** of values (order of values), not the actual values themselves.

For testing differences between two categories, we can use the Mann-Whitney U test, sometimes called the Wilcoxon Signed Rank test or Wilcoxon Rank Sum test. There are slight differences between these tests. Here we will test the difference in **Prevotella** abundance between USA and non-USA. Make sure to use the relative abundances, not the absolute abundances.

```
wilcox.test(prevotella ~ is.USA, exact=FALSE)

# get the exact p-value

wilcox.test(prevotella ~ is.USA, exact=FALSE)$p.value
```

We can do a test for differentiation across multiple categories, analogous to ANOVA, using the Kruskal-Wallis test.

```
kruskal.test(prevotella ~ map$COUNTRY)
```

For continuous variables, we can use Spearman correlation instead of Pearson correlation.

```
cor.test(prevotella, map$AGE, method='spearman', exact=FALSE)
```

SPECIAL NOTE: GLM.EDGER Custom Function

You have to source the following file in the SRC folder from the mice_tutorial

source('/Users/chigu142000/mice_tutorial/mice8992-2016-master/src/wrap.edgeR.r')

which has the following R code

```
**************************************************************************************
**************************************************************************************
**************************************************************************************

require('edgeR')

# x is the independent variable, a 2-group factor
# Y is a matrix of samples x dependent variables
# returns p-values
"exact.test.edgeR" <- function(x, Y, use.fdr=TRUE,
        norm.factor.method=c('none','RLE')[1],
        include.foldchange=FALSE){
 require('edgeR')

 x <- as.factor(x)
 if(length(levels(x)) != 2) stop('x must be a 2-level factor')

 d <- DGEList(count=t(Y), group=x)
 d <- calcNormFactors(d, method=norm.factor.method)
 d <- estimateCommonDisp(d)
 d <- estimateTagwiseDisp(d)
 et <- exactTest(d)

 return(et)
}

# x is the independent variable
# Y is a matrix of samples x dependent variables
# returns p-values
"glm.edgeR" <- function(x, Y, covariates=NULL,
        use.fdr=TRUE, estimate.trended.disp=TRUE,
        verbose=TRUE){

 require('edgeR')

 # drop NA's
 ix <- !is.na(x)
 Y <- Y[ix,]
 x <- x[ix]
 if(!is.null(covariates)){
        if(is.null(dim(covariates))){
                covariates <- as.data.frame(covariates)
        }
        covariates <- covariates[ix,,drop=F]

        # drop constant covariates
        covariates <- covariates[,apply(covariates,2,function(xx) length(unique(xx)) > 1),drop=F]
 }

 if(verbose) cat('Making DGEList...\n')
 d <- DGEList(count=t(Y), group=x)
 if(verbose) cat('calcNormFactors...\n')
 d <- calcNormFactors(d,)
 if(!is.null(covariates)){
        covariates <- as.data.frame(covariates)
        covariates <- cbind(x, covariates)
        covariates <- droplevels(covariates)
        design <- model.matrix(~ ., data=covariates)
 } else {
        design <- model.matrix(~x)
 }

 if(verbose) cat('estimate common dispersion...\n')
 d <- estimateGLMCommonDisp(d, design)
 if(estimate.trended.disp){
        if(verbose) cat('estimate trended dispersion...\n')
        d <- estimateGLMTrendedDisp(d, design)
 }
 if(verbose) cat('estimate tagwise dispersion...\n')
 d <- estimateGLMTagwiseDisp(d,design)

 if(verbose) cat('fit glm...\n')
 fit <- glmFit(d,design)
 if(verbose) cat('likelihood ratio test...\n')
 lrt <- glmLRT(fit,coef=2)

 return(lrt)
}


# runs set of differential "expression" tests
# x is a sample x obs count matrix, e.g. taxa, otus
#
# test.list is a named list of tests, each of the form:
#    list(ix=<logical indices of samples to test>,
#        group=<grouping vector, factor with 2 levels>,
#        covariate.names=<list of map column headers, can be omitted>)
# map must be included if any tests have covariate.names listed
#
"run.DGE.tests" <- function(x, test.list, map=NULL, verbose=FALSE){
 res <- list()

 for(j in seq_along(test.list)){
        ix <- test.list[[j]]$ix
        y <- test.list[[j]]$group
        covariate.names <- test.list[[j]]$covariate.names
        if(is.null(covariate.names)){
                res[[names(test.list)[j]]] <- exact.test.edgeR(x[ix,], y[ix])
        } else {
                res[[names(test.list)[j]]] <-
                                exact.test.edgeR.covariates(x[ix,],y[ix],
                                        covariates = map[ix,covariate.names])
        }
        if(verbose){
                cat('\n\n',names(test.list)[j],'\n')
                print(topTags(res[[names(test.list)[j]]]))
        }
 }
 return(res)
}
```

****

****

****

Step 5: Ordination

- Advanced association testing (Y= Disease Outcome) vs X(bacteria OUT Count)
- Ordination allows us to visualize the distance matrix between samples so that we can estimate if the degree of variation in the communities correspond to principal axes of variation (be it treatment groups, or covariate gradients)
- You first plot the ordination based on the pairwise distance between each species in a sample, then you check for clustering based on treatment groups or covariates
- Gradient = driver of variation in microbiome along a dimensional axis

OTU Table (transposed for analysis)			
	Genera 1	Genera 2	Genera 3
Sample 1			
Sample 2			
Sample 3			

Beta Diversity Matrix (Distance between species)			
	Sample 1	Sample 2	Sample 3
Sample 1	0		
Sample 2		0	
Sample 3			0

Principal components / coordinates Matrix			
	PC 1	PC 2	PC 3
Sample 1			
Sample 2			
Sample 3			

Ordination (Guerrero Negro)

All of the code in this page is meant to be run in R unless otherwise specified.

```
library('biom')

library('vegan')

# load biom file

otus.biom <- read_biom('otu_table_json.biom')

# Extract data matrix (OTU counts) from biom table

otus <- as.matrix(biom_data(otus.biom))

# transpose so that rows are samples and columns are OTUs

otus <- t(otus)

# convert OTU counts to relative abundances

otus <- sweep(otus, 1, rowSums(otus),'/')

# load mapping file

map <- read.table('map.txt', sep='\t', comment='', head=T, row.names=1)

# find the overlapping samples

common.ids <- intersect(rownames(map), rownames(otus))

# get just the overlapping samples
```

```
otus <- otus[common.ids,]

map <- map[common.ids,]

# get Euclidean distance

d.euc <- dist(otus)

# get Bray-Curtis distances (default for Vegan)

d.bray <- vegdist(otus)

# get Chi-square distances using vegan command

# we will extract chi-square distances from correspondence analysis

my.ca <- cca(otus)

d.chisq <- as.matrix(dist(my.ca$CA$u[,1:2]))
```

Ordination: PCA

Outside the field of ecology people often use principal components analysis (PCA). PCA is equivalent to calculating Euclidean distances on the data table, and then doing principal coordinates analysis (PCoA). The benefit of PCoA is that it allows us to use any distance metric, and not just Euclidean distances.

We will calculate PCA just for demonstration purposes. In practice, Euclidean distance is generally not a good distance metric to use on community ecology data. PCA will only work if there are more samples than features, so we will calculate it using just the top 10 OTUs.

```
# get the indices of the 10 dominant OTUs

otu.ix <- order(colMeans(otus),decreasing=TRUE)[1:10]

# PCA on the subset including 10 OTUs

pca.otus <- princomp(otus[,otu.ix])$scores

# For comparison, do PCoA on Euclidean distances

pc.euc <- cmdscale(dist(otus[,otu.ix]))

# plot the PC1 scores for PCA and PCoA

# note: we might have to flip one axis because the directionality is arbitrary

# these are perfectly correlated

plot(pc.euc[,1], pca.otus[,1])
```

Ordination: NMDS

Another common approach is non-metric multidimensional scaling (MDS or NMDS). We can do this using the Vegan package. We will apply this to Bray-Curtis distances and plot the ordination colored by physical sample depth in the microbial mat.

```
# NMDS using Vegan package

mds.bray <- metaMDS(otus)$points

# makes a gradient from red to blue

my.colors <- colorRampPalette(c('red','blue'))(10)

layer <- map[,'LAYER']

plot(mds.bray[,1], mds.bray[,2], col=my.colors[layer], cex=3, pch=16)
```

Ordination: PCoA

NMDS seems to work fine at recovering a smooth gradient along PC1. Let us compare to PCoA on the same Bray-Curtis distances.

```
# Run PCoA (not PCA)

pc.bray <- cmdscale(d.bray,k=2)

plot(pc.bray[,1], pc.bray[,2], col=my.colors[layer], cex=3, pch=16)
```

In general, PCoA and NMDS tend to give similar results. PCoA is more common in microbiome analysis.

Biplots

One benefit of PCA over PCoA is that it automatically provides “loadings” for the features (OTUs/taxa/genes) along each axis, which can help visualize which features are driving the positions of samples along the principal axes of variation. This kind of plot is called a “biplot”. Fortunately there are ways to produce biplots using PCoA. The way that we make biplots with PCoA is to plot each species as a weighted average of the positions of the samples in which it is present. The weights are the relative abundances of that species in the samples.

```
# First get a normalized version of the OTU table

# where each OTU's relative abundances sum to 1

otus.norm <- sweep(otus,2,colSums(otus),'/')

# use matrix multiplication to calculated weighted average of each taxon along each axis

wa <- t(otus.norm) %*% pc.bray

# plot the PCoA sample scores
```

```
plot(pc.bray[,1], pc.bray[,2], col=my.colors[layer], cex=3, pch=16)
# add points and labels for the weighted average positions of the top 10 dominant OTUs
points(wa[otu.ix,1],wa[otu.ix,2],cex=2,pch=1)
text(wa[otu.ix,1],wa[otu.ix,2],colnames(otus)[otu.ix],pos=1, cex=.5)
```

Step 6: Constrained Ordination

- Correspondence analysis is one using the vegan package in R
- It determines the proportion of the variance in genetic distance explained along different axes
- Estimated Variance = eigen value / total variance (fraction of variance explained by each axis from the unconstrained analysis)
- In direct gradient analysis, we directly relate species responses to an a priori determined covariates (i.e. age, sex, environmental factors)

Constrained ordination (Guerrero Negro)

All of the code in this page is meant to be run in R unless otherwise specified.

```
library('biom',quietly=TRUE, warn=FALSE)
library('vegan',quietly=TRUE, warn=FALSE)
# load biom file
otus.biom <- read_biom('otu_table_json.biom')
# Extract data matrix (OTU counts) from biom table
otus <- as.matrix(biom_data(otus.biom))
# transpose so that rows are samples and columns are OTUs
otus <- t(otus)
# convert OTU counts to relative abundances
otus <- sweep(otus, 1, rowSums(otus),'/')
# load mapping file
map <- read.table('map.txt', sep='\t', comment='', head=T, row.names=1)
# find the overlapping samples
common.ids <- intersect(rownames(map), rownames(otus))
# get just the overlapping samples
otus <- otus[common.ids,]
map <- map[common.ids,]
```

```
# Keep only OTUs present in at least 50% of samples

# This is fairly aggressive but will reduce the clutter in biplots

otus <- otus[,colMeans(otus>0)>.5]
```

Regular Correspondence Analysis

We have seen ordination using “Chi-square” distances and PCoA. But there is another interpretation of this approach. It is essentially equivalent to doing “Correspondence analysis,” which tries to put the samples in order along the x-axis so that all species have a unimodal response to the primary gradient. In other words, each species should peak in abundance only one time somewhere in the middle of the gradient, or at one of the ends of the gradient, and should not have additional peaks anywhere along the gradient. If there is truly an ordering that makes this possible, then correspondence analysis will find it. We will use the vegan package to run correspondence analysis. We can also plot a biplot using vegan by calling plot() on the resulting CA object.

```
# run CA using vegan command

my.ca <- cca(otus)

plot(my.ca)
```

What fraction of total inertia is explained by each axis?

```
my.ca$CA$eig/my.ca$tot.chi
```

Constrained correspondence analysis.

Now we can perform “Direct Gradient Analysis,” in which we relate species directly to environmental variable. According to Mike Palmer, “Canonical Correspondence Analysis is the marriage between CA and multiple regression.” Like CCA, CA maximizes the correlation between species scores and sample scores. However, in CCA the sample scores are constrained to be linear combinations of environmental variables. Therefore CCA must explain less variation than pure CA.

```
# run CA using vegan command

my.cca <- cca(otus ~ END_DEPTH + CHEMOTAXIS + FLAGELLA, data=map)

plot(my.cca)
```

What fraction of total inertia is explained by each axis in CCA? Compare this to the fraction of total inertia explained by CA.

```
my.cca$CCA$eig/my.cca$tot.chi
```

Assessing significance

We can compare the variance explained by the constrained and unconstrained correspondence analyses in the first axis. We want to see that constrained CA explains a good fraction of the explainable variation.

```
a <- my.ca$CA$eig/my.ca$tot.chi

b <- my.cca$CCA$eig/my.cca$tot.chi

# Test what fraction of CA1 variance is explained in CCA1

b[1]/a[1]
```

```
##CCA1

## 0.7678969
```

We can also simulate random data by shuffling or permuting the metadata values. We will shuffle them together to preserve correlations between metadata variables. If we shuffle them 10,000 times and calculate the variance explained in CCA axis 1 each time, we can compare this to the observed variation explained to get a p-value.

```
# store the observed value

obs.val <- my.cca$CCA$eig[1]/my.cca$tot.chi

# Perform 999 randomized CCAs

mc.vals <- replicate(999,{my.cca <- cca(otus ~ END_DEPTH + CHEMOTAXIS + FLAGELLA,
 data=map[sample(1:nrow(map)),]); my.cca$CCA$eig[1]/my.cca$tot.chi})

# include the observed value as one of the "null" values to be conservative

mc.vals <- c(mc.vals, obs.val)

# What fraction of the randomized values was greater than the observed value? (p-value)

mean(c(obs.val, mc.vals) >= obs.val)
```

```
## [1] 0.001998002
```

Note that a randomized CCA does not look very good.

```
my.cca <- cca(otus ~ END_DEPTH + CHEMOTAXIS + FLAGELLA, data=map[sample(1:nro
w(map)),])

plot(my.cca)
```

Part 14_Microbiome Differential Analyses Summary_LeFse_PiCRUST_KEGG

Statistical Analysis Summary Review
Phyloseq Tutorial
Phyloseq to EdgeR differential abundance analysis tutorial
Phyloseq to DESeq2 differential abundance analsysis tutorial
Permanova Tutorial
Picrust Tutorial
LEFse & Galaxy Network tutorial
Complete Phyloseq + DESeq2 + Picrust + LEFse Tutorial

Statistical analysis Summary Review (McMurdie & Holmes, 2014)

In recent generation DNA sequencing the total reads per sample (library size; sometimes referred to as depths of coverage) can vary by orders of magnitude within a single sequencing run. Comparison across samples with different library sizes requires more than a simple linear or logarithmic scaling adjustment because it also implies different levels of uncertainty, as measured by the sampling variance of the proportion estimate for each feature (a feature is a gene in the RNA-Seq context, and is a species or Operational Taxonomic Unit, OTU, in the context of microbiome sequencing). In this article we are primarily concerned with optimal methods for addressing differences in library sizes from microbiome sequencing data.
Variation in the read counts of features between technical replicates have been adequately modeled by Poisson random variables. However, we are usually interested in understanding the variation of features among biological replicates in order to make inferences that are relevant to the corresponding population; in which case a mixture model is necessary to account for the added uncertainty. Taking a hierarchical model approach with the Gamma-Poisson which gives the negative binomial (NB) distribution has provided a satisfactory fit to RNA-Seq data, as well as a valid regression framework that leverages the power of generalized linear models. However, the variance for the negative binomial distribution becomes poisson when $\theta = 0$. Recognizing that $\theta > 0$ and estimating its value is necessary in gene-level tests in order to control the rate of false positive genes. Many false positive genes appear significantly differentially expressed between experimental conditions under the assumption of a Poisson distribution, but are nevertheless not-significant in tests that account for the larger variance that results from nonzero dispersion.

The uncertainty in estimating θ for every gene when there is a small number of samples — or a small number of biological replicates — can be mitigated by sharing information across the thousands of genes in an experiment, leveraging a systematic trend in the mean-dispersion relationship. This approach substantially increases the power to detect differences in proportions (differential expression) while still adequately controlling for false positives.
Although DNA sequencing-based microbiome investigations use the same sequencing machines and represent the processed sequence data in the same manner — a feature-by-sample contingency table where the features are OTUs instead of genes — to our knowledge the modeling and normalization methods currently used in RNA-Seq analysis have not been transferred to microbiome research. Instead, microbiome analysis workflows often begin with an ad hoc library size normalization by random subsampling without replacement, or so-called rarefying.

Rarefying is most often defined by the following steps.
1. Select a minimum library size, NL,min. This has also been called the rarefaction level
2. Discard libraries (microbiome samples) that have fewer reads than NL,min.
3. Subsample the remaining libraries without replacement such that they all have size NL,min.
Often NL,min is chosen to be equal to the size of the smallest library that is not considered defective.

To our knowledge, rarefying was first recommended for microbiome counts in order to moderate the sensitivity of the UniFrac distance to library size, especially differences in the presence of rare OTUs. We demonstrate the applicability of a variance stabilization technique based on a mixture model of microbiome count data. This approach simultaneously addresses both problems of (1) DNA sequencing libraries of widely different sizes, and (2) OTU (feature) count proportions that vary more than expected under a Poisson model. We utilize the most popular implementations of this approach currently used in RNA-Seq analysis, namely edgeR and DESeq, adapted here for microbiome data. This approach allows valid comparison across OTUs while substantially improving both power and accuracy in the detection of differential abundance.

Suppose we want to compare two different samples, called A and B, comprised of 100 and 1000 DNA sequencing reads, respectively. In statistical terms, these library sizes are also equivalent to the number of trials in a sampling experiment. In practice, the library size associated with each biological sample is a random number generated by the technology, often varying from hundreds to millions.

Formally comparing the two proportions according to a standard test could technically be done either using a x2 test (equivalent to a two sample proportion test here) or a Fisher exact test. By first rarefying (Figure 1, Rarefied Abundance section) so that both samples have the same library size before doing the tests, we are no longer able to differentiate the samples (Figure 1, tests). This loss of power is completely attributable to reducing the size of B by a factor of 10, which also increases the width of the confidence intervals corresponding to each proportion such that they are no longer distinguishable from those in A even though they are distinguishable in the original data. This is because the variance of the proportion's estimate is multiplied by 10 when the total count is divided by 10. This is a common occurrence and one that is traditionally dealt with in statistics by applying variance-stabilizing transformations.

Statistical approach 1 (distinguishing patterns of relationships between whole microbiome samples)

The main goal is to distinguish patterns of relationships between whole microbiome samples through normalization followed by the calculation of sample-wise distances. Many early microbiome investigations are variants of this example, and also used rarefying prior to calculating UniFrac distances.

Microbiome studies often graphically represent the results of their pairwise sample distances using multidimensional scaling (also called Principal Coordinate Analysis, PCoA), which is useful if the desired effects are clearly evident among the first two or three ordination axes. In some cases, formal testing of sample covariates is also done using a permutation MANOVA (e.g. vegan::adonis in R) with the (squared) distances and covariates as response and linear predictors, respectively.

Normalizations Methods for distinguishing patterns of relationships between whole microbiome samples

1. DESeqVS. Variance Stabilization implemented in the DESeq package.
2. None. Counts not transformed. Differences in total library size could affect the values of some distance metrics.
3. Proportion. Counts are divided by total library size.
4. Rarefy. Rarefying is performed as defined in the introduction, using rarefy_even_depth implemented in the phyloseq package, with NL,min set to the 15th-percentile of library sizes within each simulated experiment.
5. UQ-logFC. The Upper-Quartile Log-Fold Change normalization implemented in the edgeR package, coupled with the topMSD distance (see below).

Distance Metrics Methods. for distinguishing patterns of relationships between whole microbiome samples. For each of the previous normalizations we calculated sample-wise distance/dissimilarity matrices using the following methods, if applicable.

1. Bray-Curtis. The Bray-Curtis dissimilarity first defined in 1957 for forest ecology.
2. Euclidean. The euclidean distance treating each OTU as a dimension.
3. PoissonDist. Our abbreviation of PoissonDistance,a sample-wise distance implemented in the PoiClaClu package.
4. top-MSD. The mean squared difference of top OTUs, as implemented in edgeR.
5. UniFrac-u. The Unweighted UniFrac distance.
6. UniFrac-w. The Weighted UniFrac distance.

Statistical Approach 2 (differential abundance analyses)

The goal is to detect microbes that are differentially abundant between two pre-determined classes of samples. This experimental design appears in many clinical settings (health/ disease, target/control, etc.), and other settings for which there is sufficient a priori knowledge about the microbiological conditions, and we want to enumerate the OTUs that are different between these microbiomes, along with a measure of confidence that the proportions differ.

Normalization methods for differential abundance analyses.

For each simulated experiment, we used the following normalization/ modeling methods prior to testing for differential abundance.

1. Model/None. A parametric model was applied to the data, or, in the case of the t-test, no normalization was applied (note: the t-test without normalization can only work with a high degree of balance between classes, and is provided here for comparison but is not recommended in general).
2. Rarefied. Rarefying is performed as defined in the introduction, using rarefy_even_depth implemented in the phyloseq package, with NL,min set to the 15th-percentile of library sizes within each simulated experiment.
3. Proportion. Counts are divided by total library size.

Testing methods for differential analyses. For each OTU of each simulated experiment we used the following to test for differential abundance.

1. two sided Welch t-test. A two-sided t-test with unequal variances, using the mt wrapper in phyloseq of the mt.maxT method in the multtest package.
2. edgeR - exactTest. An exact binomial test (see base R's stats::binom.test) generalized for overdispersed counts and implemented in the exactTest method of the edgeR package.
3. DESeq - nbinomTest. A Negative Binomial conditioned test similar to the edgeR test above, implemented in the nbinomTest method of the DESeq package.
4. DESeq2 - nbinomWaldTest. A Negative Binomial Wald Test using standard maximum likelihood estimates for GLM coefficients assuming a zero-mean normal prior distribution, implemented in the nbinom WaldTest method of the DESeq2 package.

All tests were corrected for multiple inferences using the Benjamini-Hochberg method to control the False Discovery Rate. Please note that in the context of these simulations library size is altogether different from effect size; the former being equivalent to both the column sums and the number of reads per sample.

Statistical Testing Considerations in Molecular Biology and Bioinformatics

In the field of genomics (and more generally in bioinformatics), the modern usage is to define fold change in terms of ratios and not by the alternative definition.

However, log-ratios are often used for analysis and visualization of fold changes. The logarithm to base 2 is most commonly used as it is easy to interpret, e.g. a doubling in the original scaling is equal to a $\log_2$ fold change of 1, a quadrupling is equal to a $\log_2$ fold change of 2 and so on. Conversely, the measure is symmetric when the change decreases by an equivalent amount e.g. a halving is equal to a $\log_2$ fold change of -1, a quartering is equal to a $\log_2$fold change of -2 and so on. This leads to more aesthetically pleasing plots as exponential changes are displayed as linear and so the dynamic range is increased. For example, on a plot axis showing $\log_2$ fold changes, an 8-fold increase will be displayed at an axis value of 3 (since 2^3 = 8). However, there is no mathematical reason to only use logarithm to base 2, and due to many discrepancies in describing the $\log_2$ fold changes in gene/protein expression, a new term "loget" has been proposed.

Fold change is a measure describing how much a quantity changes between an original and a subsequent measurement. It is defined as the ratio between the two quantities; for quantities A and B, then the fold change of B with respect to A is B/A. Fold change is often used when analysing multiple measurements of a biological system taken at different times as the change described by the ratio between the time points is easier to interpret than the difference.

Fold change is so-called as it is common to describe an increase of multiple X as an "X-fold increase". As such, several dictionaries, including the Oxford English Dictionary and Merriam-Webster Dictionary, as well as Collins's Dictionary of Mathematics, define "-fold" to mean "times," as in "2-fold" = "2 times" = "double." Likely because of this definition, many scientists use not only "fold" but also "fold change" to be synonymous with "times", as in "3-fold larger" = "3 times larger.". More ambiguous is fold decrease, where for instance a decrease of 50% between two measurements would generally be referred to a "half-fold change" rather than a "2-fold decrease".

Fold change is often used in analysis of gene expression data from microarray and RNA-Seq experiments for measuring change in the expression level of a gene. A disadvantage and serious risk of using fold change in this setting is that it is biased and may misclassify differentially expressed genes with large differences (B-A) but small ratios (B/A), leading to poor identification of changes at high expression levels. Furthermore, when the denominator is close to zero, the ratio is not stable and the fold change value can be disproportionately affected by measurement noise.

With these microarrays you have measured the expression of the miRNAs in two conditions. The ratio of theses expression values ("treatment" condition vs. "reference" condition) if called the "fold-change" (FC). It's logarithm is called the log fold-change, abbreviated logFC. The logFC is the more attractive measure for differential expression than the FC, because

- its error distribution is symmetric (not skewed)
- "zero" (0) means "no change", positive values indicate up- and negative value indicate down-regulation
- similar absolute values (e.g. -2 and +2) can be seen as effects of similar biological relevance (only in different directions)
- similar differences between values can be seen as differential effects of similar biological relevance

The logarithm in the logFC is typically calculated for the base 2. That means one unit of the logFCs translates to a two-fold change in expression. The FCs can be calculated from the logFCs as FC = 2^logFC.

Sometimes, not only the regulation (differential expression) of the miRNA is interesting but also the general level at which it is expressed. This is giben by the geometic average of the expressions under both conditions, what is the square-root of the product of the two expression values. On the logarithmic scale this translates simply to the average. The result is shown in the colum AveExp (for Average (log-)Expression). The higher the value, the higher is the general expression (abudance, concentration) of the miRNA.

The logFC was determined as the mean of several samples. The result scatters around an (unknown) expected value and is used as an estimate for this unknown value. The precision (or variability) of this estimate is usually indicated either by the standard error (SE) or (better) by a confidence interval. These values are missing in the table you got.
The estimated logFC can be compared to a hypothesized value using a t-test. The null hypothesis is typically that the gene is not regulated, so the expected value is zero. For this hypothesis a test statistic with known distribution can be calculated. This is the t-value which is calculated as t = logFC/SE. So if desired you can retrieve the missing SE simply by dividing the logFC by t. For the t-value a correspoding p-values can be calculated. The p-value is the probability, under the null hypothesis, to get more extreme t-values than the one calculated. P-values close to zero indicate that the obtained t-value is unlikely if the miRNA was not regulated. By some strange inverse (and wrong) logic most people conclude that this would demonstrate that the miRNA is regulated (differentially expressed). The story behind the interpretation of P-values is long and complicated and should not be discussed here. You can find several enlightning threads about this topic in ReseachGate.

The B-value is another statistic about the regulation. It is sometimes called "log odds ratio" and gives the logarithm of the ratio of the odds for regulation to the odds against regulation. Similar to the logFC, B-values of 0 indicate 50:50 odds (maximally undecided), positive values indicate that the data favours up regulation, negative values indicates that the data favours down regulation. The B-value is based on Bayesian theorem (therefore possibly the abbreviation "B") and the prior expectation that some particular fraction of the miRNAs is regulated. This proportion should also be given by these bioinformatics people (actually you should have told them which value to use, because this depends on your expert judgement and not on statistics).
No, there is no general objective justification for any particular log-fold change threshold. Mathematically speaking, it is possible to reject the null hypothesis at any non-zero log-fold change if the variability is low enough. One could argue that small log-fold changes are not biologically relevant, but the exact definition of "small" is open to interpretation. Larger log-fold changes are also more robustly detected across technologies (e.g., RNA-seq and qPCR), though selecting a threshold on this basis would depend on the sensitivities of the technologies involved. Somewhere between 1.1 to 1.5 is a common choice for a "sensible" threshold.

But all this is getting away from the main point, which is the detection of DE genes. If you want to do this in a statistically rigorous manner, use the BH-adjusted p-values to control the false discovery rate. This ensures that the expected proportion of false positives in your set of significant DE genes is below a certain threshold (usually 5%). Now, you might say that this approach also involves the selection of an arbitrary threshold. However, with this approach, at least the choice of threshold is directly related to the probability of whether the genes are truly DE or not. A log-fold change threshold doesn't tell you much about the error rate, as it doesn't account for the variability of the expression values.

Finally, if you do need a log-fold change threshold, the `treat` function should be used, and DE genes selected on the basis of the adjusted p-values. This ensures that the FDR is controlled while only considering genes with log-fold changes above a minimum value.

Controlling for multiple testing
For 1 test, we typically set the alpha value at 0.05
This means that if we run 100 test, at least 5 of them will be statistically significant
A p-value is an area in the tail of a distribution that tells you to odds of a result happening by chance.

Bonferroni test
Divide the alpha value by the number of test being done: more stringent, and it controls the probability of having one or more false positive

False discovery adjustment
Guarantees the expected number of false positive, independent of the number of test
So if your FDR is at 0.2, then 20% of all the q-values you call as hits will be false positives
A Q-value is a p-value that has been adjusted for the False Discovery Rate(FDR). The False Discovery Rate is the proportion of false positives you can expect to get from a test. A p-value gives you the probability of a false positive on a single test; If you're running hundreds or thousands of tests from small samples (which are common in fields like genomics), you should use q-values.

Why are Q-Values Necessary?
Usually, you decide ahead of time the level of false positives you're willing to accept: under 5% is the norm. This means that you run the risk of getting a false statistically significant result 5% of the time. You can't escape this fact when you're running tests: false positives (p-values) are a fact of life and are unavoidable. While 5% might be an acceptable false positive rate for running one test, it becomes completely unacceptable if you run thousands of tests on the same small data set. Here's why:
Imagine you're planning scratch off lottery, and you have a 5% chance of getting a winning ticket. One ticket gives you a 5% chance, but if you buy enough tickets, probability tells us that you'll eventually get a winner (buying 1,000 lottery tickets should do the trick and will in fact give you, on average, 50 winning tickets). The same is true for lab tests.

- The first test on your data, you have a 5% chance of a false positive.
- The second test on your data, you have another 5% chance of a false positive.
- The thousandth test on your data, you have had a 5% chance of a false positive a thousand times.

Essentially, you'll get a false positive — a false "significant" result — if you run enough tests. In fact, at a 5% FDR, you'll get 5 false results for every 100 tests you run, or 50 for every thousand. That's pretty high. This is called the multiple testing problem.
The False Discovery Rate approach to p-values assigns an adjusted p-value for each test. This is the "q-value." A p-value of 5% means that 5% of all tests will result in false positives. A q-value

of 5% means that 5% of significant results will result in false positives. Q-values usually result in much smaller numbers of false positives, although this isn't always the case.
To put this another way, p-values tell you the percentage of false positives to expect and take into account the number of tests being run. For example, if you run 1600 tests, you would expect to see about 80 false positives. The q-value doesn't take into account all the tests; they only take into account the tests that are below a threshold that you choose (i.e. tests reporting a q-value of 5% or less).

Note: The Q-value is not the same as the "Q" you sometimes see in statistics. Q on its own (as opposed to a Q-value) refers to elements in a set that don't have a particular attribute. For example, let's say you had 100 people and 57 of them like pizza. The proportion of people who like pizza is P=0.57. Therefore, Q = 0.43 (which is just 1 – P).

What are p-values?
The object of differential 2D expression analysis is to find those spots which show expression difference between groups, thereby signifying that they may be involved in some biological process of interest to the researcher. Due to chance, there will always be some difference in expression between groups. However, it is the size of this difference in comparison to the variance (i.e. the range over which expression values fall) that will tell us if this expression difference is significant or not. Thus, if the difference is large but the variance is also large, then the difference may not be significant. On the other hand, a small difference coupled with a very small variance could be significant. We use the one way Anova test (equivalent t-test for two groups) to formalise this calculation. The tests return a p-value that takes into account the mean difference and the variance and also the sample size. The p-value is a measure of how likely you are to get this spot data if no real difference existed. Therefore, a small p-value indicates that there is a small chance of getting this data if no real difference existed and therefore you decide that the difference in group expression data is significant. By small we usually mean 0.05.

What are q-values, and why are they important?

False positives
A positive is a significant result, i.e. the p-value is less than your cut off value, normally 0.05. A false positive is when you get a significant difference when, in reality, none exists. As I mentioned above, the p-value is the chance that this data could occur given no difference actually exists. So, choosing a cut off of 0.05 means there is a 5% chance that we make the wrong decision.
The multiple testing problem
When we set a p-value threshold of, for example, 0.05, we are saying that there is a 5% chance that the result is a false positive. In other words, although we have found a statistically significant result, in reality, there is no difference in the group means. While 5% is acceptable for one test, if we do lots of tests on the data, then this 5% can result in a large number of false positives. For example, if there are 200 spots on a gel and we apply an ANOVA or t-test to each, then we would expect to get 10 false positives by chance alone. This is known as the multiple testing problem.

Multiple testing and the False Discovery Rate

While there are a number of approaches to overcoming the problems due to multiple testing, they all attempt to assign an adjusted p-value to each test, or similarly, reduce the p-value threshold. Many traditional techniques such as the Bonferroni correction are too conservative in the sense that while they reduce the number of false positives, they also reduce the number of true discoveries. The False Discovery Rate approach is a more recent development. This approach also determines adjusted p-values for each test.

However, it controls the number of false discoveries in those tests that result in a discovery (i.e. a significant result). Because of this, it is less conservative that the Bonferroni approach and has greater ability (i.e. power) to find truly significant results.
Another way to look at the difference is that a p-value of 0.05 implies that 5% of all tests will result in false positives. An FDR adjusted p-value (or q-value) of 0.05 implies that 5% of significant tests will result in false positives. The latter is clearly a far smaller quantity.

q-values

q-values are the name given to the adjusted p-values found using an optimized FDR approach. The FDR approach is optimized by using characteristics of the p-value distribution to produce a list of q-values. In what follows I will tie up some ideas and hopefully this will help clarify some of the ideas about p and q values. It is usual to test many hundreds or thousands of spot variables in a proteomics experiment. Each of these tests will produce a p-value. The p-values take on a value between 0 and 1 and we can create a histogram to get an idea of how the p-values are distributed between 0 and 1. Some typical p-value distributions are shown below. On the x-axis we have histogram bars representing p-values. Each has a width of 0.05 and so in the first bar (red or green) we have those p-values that are between 0 and 0.05. Similarly, the last bar represents those p-values between 0.95 and 1.0, and so on. The height of each bar gives an indication of how many values are in the bar. This is called a density distribution because the area of all the bars always adds up to 1. Although the two distributions appear quite different, you will notice that they flatten off towards the right of the histogram. The red (or green) bar represents the significant values, if you set a p-value threshold of 0.05.

If there are no significant changes in the experiment, you will expect to see a distribution more like that on the left above while an experiment with significant changes will look more like that on the right. So, even if there are no significant changes in the experiment, we still expect, by chance, to get p-values < 0.05. These are false positives, and shown in red. Even in an experiment with significant changes (in green), we are still unsure if a p-value < 0.05 represents a true discovery or a false positive. Now, the q-value approach tries to find the height where the p-value distribution flattens out and incorporates this height value into the calculation of FDR adjusted p-values. We can see this in the histogram below. This approach helps to establish just how many of the significant values are actually false positives (the red portion of the green bar). Now, the q-values are simply a set of values that will lie between 0 and 1. Also, if you order the p-values used to calculate the q-values, then the q-values will also be ordered.

To interpret the q-values, you need to look at the ordered list of q-values. There are 839 spots in this experiment. If we take spot 52 as an example, we see that it has a p-value of 0.01 and a q-value of 0.0141. Recall that a p-value of 0.01 implies a 1% chance of false positives, and so with 839 spots, we expect between 8 or 9 false positives, on average, i.e. 839*0.01 = 8.39. In this experiment, there are 52 spots with a value of 0.01 or less, and so 8 or 9 of these will be false positives. On the other hand, the q-value is a little greater at 0.0141, which means we should expect 1.41% of all the spots with q-value less than this to be false positives. This is a much better situation. We know that 52 spots have a q-value less than 0.0141 and so we should expect 52*0.0141 = 0.7332 false positives, i.e. less than one false positive. Just to reiterate, false positives according to p-values take all 839 values into account when determining how many false positives we should expect to see while q-values take into account only those tests with q-values less the threshold we choose. Of course, it is not always the case that q-values will result in less false positives, but what we can say is that they give a far more accurate indication of the level of false positives for a given cutoff value. When doing lots of tests, as in a proteomics experiment, it is more intuitive to interpret p and q values by looking at the entire list of values in this way rather that looking at each one independently. In this way, a threshold of 0.05 has meaning across the entire experiment. When deciding on a cut-off or threshold value, you should do this from the point of view of how many false positives will this result in, rather than just randomly picking a p- or q-value of 0.05 and saying that everything with a value less than this is significant.

Phyloseq, EdgeR and DESeq 2 Quick Tutorial

Phyloseq

http://bioconductor.org/packages/release/bioc/vignettes/phyloseq/inst/doc/phyloseq-basics.html
http://bioconductor.org/packages/release/bioc/vignettes/phyloseq/inst/doc/phyloseq-analysis.html

EdgeR

https://ucdavis-bioinformatics-training.github.io/2017-September-Microbial-Community-Analysis-Workshop/friday/MCA_Workshop_R/phyloseq.html
http://joey711.github.io/phyloseq-extensions/edgeR.html

DESeq2

https://bioconductor.org/packages/devel/bioc/vignettes/phyloseq/inst/doc/phyloseq-mixture-models.html#import-data-with-phyloseq-convert-to-deseq2
https://bioconductor.org/packages/release/bioc/vignettes/DESeq2/inst/doc/DESeq2.html

PERMANOVA

https://rdrr.io/rforge/vegan/man/vegan-package.html
https://rdrr.io/rforge/vegan/man/adonis.html
https://microbiome.github.io/microbiome/PERMANOVA.html

Microbiome analysis review

1_QIIME / MACQIIME (Clean sequences > Phylogeny tree > OTU Table)
2_Alpha / Beta diversity / PCA Differences in Microbiome community composition
3_Differential abundance on pre-specified taxa (SAS / GEE)
4_Ranked differential abundance on rarefied OTU table (R / edgeR)
5_Ranked differential abundance on non-rarefied OTU table (Phyloseq / DESeq2)
6_Ranked differential abundance on functional pathways (Picrust / LEFse & LDA)

Important terms for Differential abundance in CRC cases as compared to controls (QIIME-CR)

OTU: Operational Taxonomic Unit,
LogFC: Log2Fold Change,
lfcse: Log2Fold Change standard error,
stat: Wald test statistic,
pval: p-value associated with Wald test,
padj: FDR adjusted p-value,
unc: unclassified.
Base Mean: average of the normalized count values, dividing by size factors.

***Positive Log2Fold Change indicates enriched in intervention group, compared to the reference group.

R - ***Installing R on your system and adding the following libraries has become much more important in the latest QIIME versions 1.9+.*** There are now many neat features in QIIME that require R. If you don't have R installed, you can get it from here. *Please note* that even if you installed R and these libraries previously for MacQIIME 1.8.0, you should still upgrade to/install the latest version of R, at least version **3.1.2**, and re-install all these R packages to get everything working. Make sure the R executable is in your PATH. You will have to open R and install additional packages: in the R command line, do the following, in the following order, one line at a time. **(*** Figure out how to pull out the script window in R for MAC: Open R > Open a document editor, type your code there, then copy and paste into the R code window as needed)**

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("DESeq2")
biocLite("biomformat")
biocLite("metagenomeSeq")

install.packages('randomForest')
install.packages('ape')
install.packages('vegan')
install.packages('optparse')
install.packages('gtools')
install.packages('klaR')
install.packages('RColorBrewer')
```

Basic storage, access, and manipulation of phylogenetic sequencing data with phyloseq

Accessing Datasets through QIITA

Set up an account at https://qiita.ucsd.edu/
Username: onyea005@umn.edu
Password: Fall2018!
Go to studies > View studies > download the dataset of interest
http://bioconductor.org/packages/release/bioc/vignettes/phyloseq/inst/doc/phyloseq-basics.html

Introduction

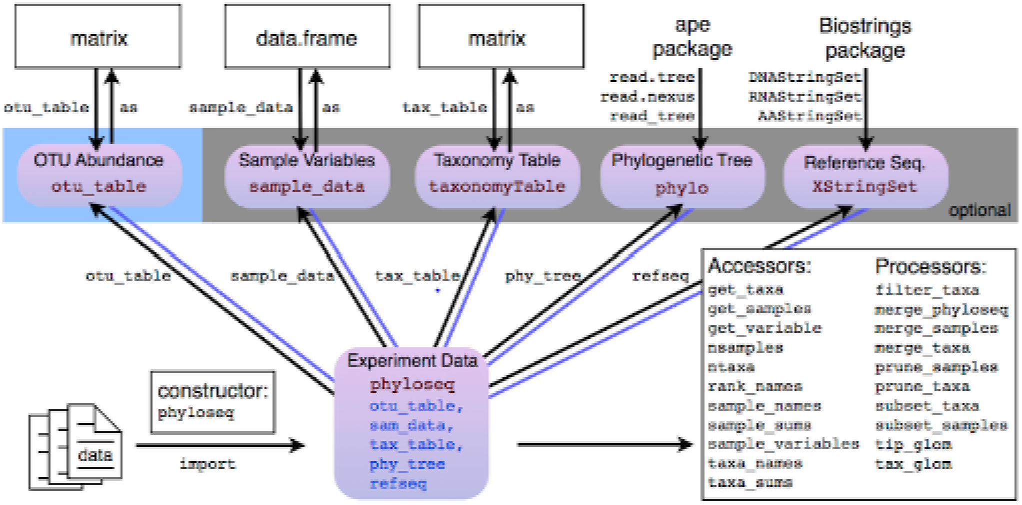
The analysis of microbiological communities brings many challenges: the integration of many different types of data with methods from ecology, genetics, phylogenetics, network analysis, visualization and testing. The data itself may originate from widely different sources, such as the microbiomes of humans, soils, surface and ocean waters, wastewater treatment plants, industrial facilities, and so on; and as a result, these varied sample types may have very different forms and scales of related data that is extremely dependent upon the experiment and its question(s). The phyloseq package is a tool to import, store, analyze, and graphically display complex phylogenetic sequencing data that has already been clustered into Operational Taxonomic Units (OTUs), especially when there is associated sample data, phylogenetic tree, and/or taxonomic assignment of the OTUs. This package leverages many of the tools available in R for ecology and phylogenetic analysis (vegan, ade4, ape, picante), while also using advanced/flexible graphic systems (ggplot2) to easily produce publication-quality graphics of complex phylogenetic data. phyloseq uses a specialized system of S4 classes to store all related phylogenetic sequencing data as single experiment-level object, making it easier to share data and reproduce analyses. In general, phyloseq seeks to facilitate the use of R for efficient interactive and reproducible analysis of OTU-clustered high-throughput phylogenetic sequencing data.

phyloseq classes

The class structure in the phyloseq package follows the inheritance diagram shown in the figure below. Currently, phyloseq uses 4 core data classes. They are (1) the OTU abundance table **(otu_table)**, (2) a table of sample data **(sample_data)**; (3) a table of taxonomic descriptors **(taxonomyTable)**; and (4) a phylogenetic tree **("phylo"-class, ape package)**.

- The otu_table class can be considered the central data type, as it directly represents the number and type of sequences observed in each sample. otu_table extends the numeric matrix class in the R base, and has a few additonal feature slots. The most important of these feature slots is the taxa_are_rows slot, which holds a single logical that indicates whether the table is oriented with taxa as rows (as in the genefilter package in Bioconductor or with taxa as columns (as in vegan and picante packages). In phyloseq methods, as well as its extensions of methods in other packages, the taxa_are_rows value is checked to ensure proper orientation of the otu_table. A phyloseq user is only required to specify the otu_table orientation during initialization, following which all handling is internal.
- The sample_data class directly inherits R's data.frame class, and thus effectively stores both categorical and numerical data about each sample. The orientation of a data.frame in this context requires that samples/trials are rows, and variables are columns (consistent with vegan and other packages).
- The taxonomyTable class directly inherits the matrix class, and is oriented such that rows are taxa/OTUs and columns are taxonomic levels (e.g. Phylum).
- The phyloseq-class can be considered an "experiment-level class" and should contain two or more of the previously-described core data classes. We assume that phyloseq users will be interested in analyses that utilize their abundance counts derived from the phylogenetic sequencing data, and so the phyloseq() constructor will stop with an error if

the arguments do not include an otu_table. There are a number of common methods that require either an otu_table and sample_data combination, or an otu_table and phylogenetic tree combination. These methods can operate on instances of the phyloseq-class, and will stop with an error if the required component data is missing.



*****microbio_me_qiime (DEFUNCT) > http://joey711.github.io/phyloseq/import-data**

microbio_me_qiime is a function in phyloseq that USED TO interface with QIIME_DB. QIIME-DB IS DOWN INDEFINITELY. The function is listed here for reference only. The following details in this section are the most recent useful tutorial details when the server was still up.

Accessing Datasets through QIITA

Set up an account at https://qiita.ucsd.edu/
Username: onyea005@umn.edu
Password: Fall2018!

Go to studies > View studies > download the dataset of interest
You will need to setup an account to browse the available data sets and their IDs. If you know a datasets ID already, or its assigned number, you can provide that as the sole argument to this function and it will download, unpack, and import the data into R, all in one command. Alternatively, if you have already downloaded the data from the QIIME server, and now have it locally on your hard drive, you can provide the local path to this tar-gz or zip file, and it will perform the unpacking and importing step for you. I'm finding this increasingly useful for creating demonstrations of methods and graphics, and can be a very effective way for you to provide fully reproducible analysis if your own data is already hosted on the microbio.me server.
****
The original tutorial called for using a dataset accessible through the microbio_me-qiime function. However, since we know that the phyloseq function can be used for individual subset files (sample data, otu tree and ref database, I will locate those from the QIIME and the MICE tutorials that I have and use those instead to create this tutorial
****

Load phyloseq and import data

Load phyloseq

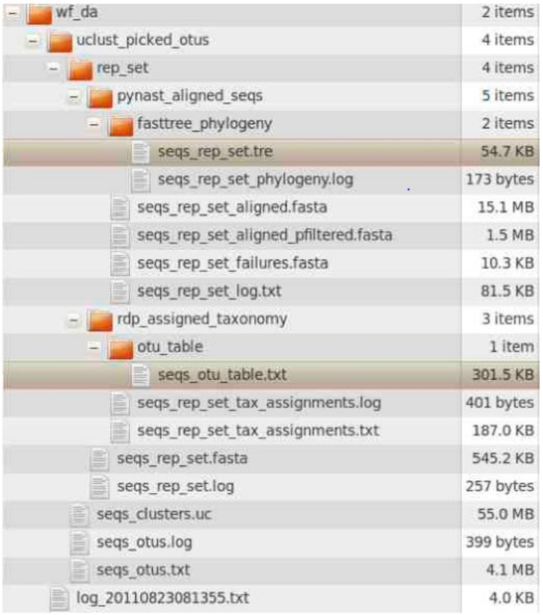
To use phyloseq in a new R session, it will have to be loaded. This can be done in your package manager, or at the command line using the library() command:

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("phyloseq")
biocLite("ggplot2")
library("phyloseq")
library("ggplot2")
```

Import from QIIME Legacy

QIIME is a free, open-source OTU clustering and analysis pipeline written for Unix (mostly Linux). It is distributed in a number of different forms (including a pre-installed virtual machine). See the QIIME home page for details.

- **Input**
  - One QIIME input file (sample map), and two QIIME output files (otu_table.txt, .tre) are recognized by the import_qiime() function. Only one of the three input files is required to run, although an "otu_table.txt" file is required if import_qiime() is to return a complete experiment object.
  - In practice, you will have to find the relevant QIIME files among a number of other files created by the QIIME pipeline. A screenshot of the directory structure created during a typical QIIME run is shown in the QIIME Directory Figure.



```
– wf_da                                              2 items
  – uclust_picked_otus                               4 items
    – rep_set                                        4 items
      – pynast_aligned_seqs                          5 items
        – fasttree_phylogeny                         2 items
            seqs_rep_set.tre                         54.7 KB
            seqs_rep_set_phylogeny.log               173 bytes
          seqs_rep_set_aligned.fasta                 15.1 MB
          seqs_rep_set_aligned_pfiltered.fasta       1.5 MB
          seqs_rep_set_failures.fasta                10.3 KB
          seqs_rep_set_log.txt                       81.5 KB
      – rdp_assigned_taxonomy                        3 items
        – otu_table                                  1 item
            seqs_otu_table.txt                       301.5 KB
          seqs_rep_set_tax_assignments.log           401 bytes
          seqs_rep_set_tax_assignments.txt           187.0 KB
        seqs_rep_set.fasta                           545.2 KB
        seqs_rep_set.log                             257 bytes
      seqs_clusters.uc                               55.0 MB
      seqs_otus.log                                  399 bytes
      seqs_otus.txt                                  4.1 MB
    log_20110823081355.txt                           4.0 KB
```

A typical QIIME output directory. The two output files suitable for import by *phyloseq* are highlighted. A third file describing the samples, their barcodes and covariates, is created by the user and required as *input* to QIIME. It is a good idea to import this file, as it can be converted directly to a sample_data object and can be extremely useful for certain analyses.

***Note
I saved the microbiome analysis files under the following paths (You may need to switch the "\" to "/" in R for the code to work. Here are the files you need to manually move and the functions associated with them (I think you may need the RDP reference link too)
## phyloseq-class experiment-level object

- ## otu_table()   OTU Table:
- ## sample_data() Sample Data:
- ## tax_table()   Taxonomy Table
- ## phy_tree()   Phylogenetic Tree:

For the mice tutorial
C:\Users\onyea005\Documents\mice_tutorial

Biom OTU table (otu_data)
/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_json2.biom

Text OTU table (otu_data)
/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_L2.txt
Mapping file (sample_data)
C:\Users\onyea005\Documents\mice_tutorial\mice8992-2016-master\data\globalgut-66-adults/map.txt

Taxonomy table (Tax_table)
C:\Users\onyea005\Documents\mice_tutorial\mice8992-2016-master\data\globalgut-66-adults\denovo\uclust_assigned_taxonomy\seqs_rep_set_tax_assignments

Phylogenetic tree for closed reference clusters(phy_tree)
C:\Users\onyea005\Documents\mice_tutorial\mice8992-2016-master\data\globalgut-66-adults\closedref\97_otus.tre

RDP database link (might already be part of phyloseq)
C:\Users\onyea005\Documents\mice_tutorial\mice8992-2016-master\data\ref\greengenes

For the qiime tutorial
C:\Users\onyea005\Documents\qiime_tutorial

Text OTU table (otu_data)
C:\Users\onyea005\Documents\qiime_tutorial\ otu_table_tabseparated.txt

Mapping file (sample_data)

C:\Users\onyea005\Documents\qiime_tutorial\ Fasting_map.txt

Taxonomy table (Tax_table)
C:\Users\onyea005\Documents\qiime_tutorial\taxonomy_results\ rep_set_tax_assignments.txt

Phylogenetic tree for closed reference clusters(phy_tree)
C:\Users\onyea005\Documents\qiime_tutorial\rep_set_tree.tre

RDP database link (might already be part of phyloseq)
C:\Users\onyea005\Documents\mice_tutorial\mice8992-2016-master\data\ref\greengenes

Import from biom-format

New versions of QIIME (see below) produce a file in version 2 of the biom file format, which is a specialized definition of the HDF5 format.
The phyloseq package provides the import_biom() function, which can import both Version 1 (JSON) and Version 2 (HDF5) of the BIOM file format.
The phyloseq package fully supports both taxa and sample observations of the biom format standard, and works with the BIOM files output from QIIME, RDP, MG-RAST, etc.

Import from QIIME (Modern)

The default output from modern versions of QIIME is a BIOM-format file (among others). This is suppored in phyloseq.

- **Sample data from QIIME**
  - Sometimes inaccurately referred to as metadata, additional observations on samples provided as mapping file to QIIME have not typically been output in the BIOM files, even though BIOM format supports it. This failure to support the full capability of the BIOM format means that you'll have to provide sample observations as a separate file. There are many ways to do this, but the QIIME sample map is supported.
- **Input**
  - Two QIIME output files (.biom, .tre) are recognized by the import_biom() function. One QIIME input file (sample map, tab-delimited), is recognized by the import_qiime_sample_data() function.
  - The objects created by each of the import functions above should be merged using merge_phyloseq to create one coordinated, self-consistent object.
- **Output**
  - Before Merging - Before merging with merge_phyloseq, the output from these import activities is the three separate objects listed in the previous table.
  - After Merging - After merging you have a single self-consistent phyloseq object that contains an OTU table, taxonomy table, sample-data, and a phylogenetic tree.
  - For example: QIIME Example Tutorial > QIIME's "Moving Pictures" example tutorial output is a little too large to include within the phyloseq package (and thus

is not directly included in this vignette). However, the phyloseq home page includes a full reproducible example of the import procedure described above:

- For reference, or if you want to try yourself, the following is the relative paths within the QIIME tutorial directory for each of the files you will need.
    - BIOM file, originally at: moving_pictures_tutorial-1.9.0/illumina/precomputed-output/otus/otu_table_mc2_w_tax_no_pynast_failures.biom
    - Tree file, originally at: moving_pictures_tutorial-1.9.0/illumina/precomputed-output/otus/rep_set.tre
    - Map File, originally at: moving_pictures_tutorial-1.9.0/illumina/map.tsv

*****Note: I saved the microbiome analysis files under the following paths (You may need to switch the "\" to "/" in R for the code to work. Here are the files you need to manually move and the functions associated with them (I think you may need the RDP reference link too)**

## phyloseq-class experiment-level object

- OTU Table: import.biom
- Phylogenetic tree: import.biom
- Sample Map: import_qiime_sample_data

For the mice tutorial
C:\Users\onyea005\Documents\mice_tutorial
Biom OTU table (otu_data)
/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/otu_table_json2.biom.
Phylogenetic tree for closed reference clusters(phy_tree)
C:\Users\onyea005\Documents\mice_tutorial\mice8992-2016-master\data\globalgut-66-adults\closedref\97_otus.tre
Mapping file (sample_data)
C:\Users\onyea005\Documents\mice_tutorial\mice8992-2016-master\data\globalgut-66-adults/map.txt

For the qiime tutorial
C:\Users\onyea005\Documents\qiime_tutorial
Biom OTU table (otu_data)
C:\Users\onyea005\Documents\qiime_tutorial\ otu_table.biom
Phylogenetic tree for closed reference clusters(phy_tree)
C:\Users\onyea005\Documents\qiime_tutorial\rep_set_tree.tre
Mapping file (sample_data)
C:\Users\onyea005\Documents\qiime_tutorial\ Fasting_map.txt

I tried importing the modern qiime files from the qiime tutorial, but there was an irregularity with the mapping file so I tried with the mice global gut 66 tutorial and I was successful. In addition, it seems that import_biom does not play nice with the tre file like it said in the tutorial, so I am using the older read_tree function instead (check https://github.com/joey711/phyloseq/issues/167)

```
modernotu <- import_biom ("/Users/onyea005/Documents/qiime_tutorial/otu_table.biom")
moderntree <- import_biom ("/Users/onyea005/Documents/qiime_tutorial/rep_set_tree.tre")
moderntree2 <- read_tree("/Users/onyea005/Documents/qiime_tutorial/rep_set_tree.tre")
modernmap <- import_qiime_sample_data("/Users/onyea005/Documents/qiime_tutorial/Fasting_Map.txt")
modernmap2 <- import_qiime_sample_data("/Users/onyea005/Documents/qiime_tutorial/mapping_output/Fasting_Map_corrected.txt")
```

PC version

```
modernotumice <- import_biom ("/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/closedref/otu_table.biom")
moderntreemice <- read_tree("/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/closedref/97_otus.tree")
modernmapmice <- import_qiime_sample_data("/Users/onyea005/Documents/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/map.txt")
phyloseqmice <- merge_phyloseq (modernotumice,modernmapmice,moderntreemice)
```

Mac version

```
modernotumice <- import_biom ("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/closedref/otu_table.biom")
moderntreemice <- read_tree("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/closedref/97_otus.tree")
modernmapmice <- import_qiime_sample_data("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/map.txt")
phyloseqmice <- merge_phyloseq (modernotumice,modernmapmice,moderntreemice)
```

Example Phyloseq Data (These are already converted to phyloseq format)**

There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets.

To load example data into the working environment, use the data() command:

```
data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
```

Similarly, entering ?enterotype will reveal the documentation for the so-called “enterotype” dataset.

phyloseq Object Summaries

In small font, the following is the summary of the GlobalPatterns dataset that prints to the terminal. These summaries are consistent among all phyloseq-class objects. Although the components of GlobalPatterns have many thousands of elements, the command-line returns only a short summary of each component. This encourages you to check that an object is still what you expect, without needing to let thousands of elements scroll across the terminal. In the cases in which you do want to see more of a particular component, use an accessor function (see table below). *** R entries are case sensitive

```
data(GlobalPatterns)
GlobalPatterns
```

```
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 19216 tips and 19215 internal nodes ]
```

Convert raw data to phyloseq components

Suppose you have already imported raw data from an experiment into R, and their indices are labeled correctly. How do you get phyloseq to recognize these tables as the appropriate class of data? And further combine them together? Table Table of Component Constructor Functions lists key functions for converting these core data formats into specific component data objects recognized by phyloseq. These will also

- Table of component constructor functions for building component data objects
- phyloseq constructors: functions for building/merging phyloseq objects.

The following example illustrates using the constructor methods for component data tables.

```
otu1 <- otu_table(raw_abundance_matrix, taxa_are_rows=FALSE)
sam1 <- sample_data(raw_sample_data.frame)
tax1 <- tax_table(raw_taxonomy_matrix)
tre1 <- read_tree(my_tree_file)
```

****
For importing raw data files, see Dr. Demmer's example in
E:\Dissertation Proposal Methods Review

Vignette for phyloseq: Analysis of high-throughput microbiome census data
http://bioconductor.org/packages/release/bioc/vignettes/phyloseq/inst/doc/phyloseq-analysis.html

Summary

The analysis of microbiological communities brings many challenges: the integration of many different types of data with methods from ecology, genetics, phylogenetics, network analysis, visualization and testing. The data itself may originate from widely different sources, such as the microbiomes of humans, soils, surface and ocean waters, wastewater treatment plants, industrial facilities, and so on; and as a result, these varied sample types may have very different forms and scales of related data that is extremely dependent upon the experiment and its question(s). The phyloseq package is a tool to import, store, analyze, and graphically display complex phylogenetic sequencing data that has already been clustered into Operational Taxonomic Units (OTUs), especially when there is associated sample data, phylogenetic tree, and/or taxonomic assignment of the OTUs. This package leverages many of the tools available in R for ecology and phylogenetic analysis (vegan, ade4, ape, picante), while also using advanced/flexible graphic systems (ggplot2) to easily produce publication-quality graphics of complex phylogenetic data. phyloseq uses a specialized system of S4 classes to store all related phylogenetic sequencing data as single experiment-level object, making it easier to share data and reproduce analyses. In general, phyloseq seeks to facilitate the use of R for efficient interactive and reproducible analysis of OTU-clustered high-throughput phylogenetic sequencing data.
By contrast, this vignette is intended to provide functional examples of the analysis tools and wrappers included in phyloseq. All necessary code for performing the analysis and producing graphics will be included with its description, and the focus will be on the use of example data that is included and documented within the phyloseq-package.

Let's start by loading the `phyloseq-package:

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("phyloseq")
library("phyloseq")
library("ggplot2")
```

Included Data

To facilitate testing and exploration of tools in phyloseq, this package includes example data from published studies. Many of the examples in this vignette use either the Global Patterns or enterotype datasets as source data. The Global Patterns data was described in a 2011 article in PNAS(Caporaso 2011), and compares the microbial communities of 25 environmental samples and three known "mock communities" — a total of 9 sample types — at a depth averaging 3.1 million reads per sample. The human enterotype dataset was described in a 2011 article in Nature(Arumugam 2011), which compares the faecal microbial communities from 22 subjects using complete shotgun DNA sequencing. The authors further compare these microbial communities with the faecal communities of subjects from other studies, for a total of 280 faecal samples / subjects, and 553 genera. Sourcing data from different studies invariable leads to gaps in the data for certain variables, and this is easily handled by R's coreNA features.

Because this data is included in the package, the examples can easily be run on your own computer using the code shown in this vignette. The data is loaded into memory using the datacommand. Let’s start by loading the Global Patterns data.

```
data(GlobalPatterns)
```

Later on we will use an additional categorical designation — human versus non-human associated samples — that was not in the original dataset. Now is a good time to add it as an explicit variable of the sample_data, and because we don’t want to type long words over and over, we’ll choose a shorter name for this modified version of Global Patterns, call it GP, and also remove a handful of taxa that are not present in any of the samples included in this dataset (probably an artifact):

```
# prune OTUs that are not present in at least one sample
GP <- prune_taxa(taxa_sums(GlobalPatterns) > 0, GlobalPatterns)
# Define a human-associated versus non-human categorical variable:
human <- get_variable(GP, "SampleType") %in% c("Feces", "Mock", "Skin", "Tongue")
# Add new human variable to sample data:
sample_data(GP)$human <- factor(human)
```

Simple exploratory graphics

Easy Richness Estimates

We can easily create a complex graphic that compares the richness estimates of samples from different environment types in the Global Patterns dataset, using the plot_richness function. Note that it is important to use raw (untrimmed) OTU-clustered data when performing richness estimates, as they can be highly dependent on the number of singletons in a sample.

```
alpha_meas = c("Observed", "Chao1", "ACE", "Shannon", "Simpson", "InvSimpson")
(p <- plot_richness(GP, "human", "SampleType", measures=alpha_meas))
```

Add a ggplot2 box plot layer to the previous plot

```
p + geom_boxplot(data=p$data, aes(x=human, y=value, color=NULL), al
pha=0.1)
```

Alpha diversity estimators for samples in the *Global Patterns* dataset. Each panel shows a different type of estimator. Individual color-shaded points and brackets represent the richness estimate and the theoretical standard error range associated with that estimate, respectively. The colors group the sample-sources into “types”. Within each panel, the samples are further organized into human-associated (TRUE) or not (FALSE), and a boxplot is overlayed on top of this for the two groups, illustrating that these human-associated samples are less rich than the environmental samples (although the type of environment appears to matter a great deal as well).

Exploratory tree plots

For further details, see the plot_tree tutorial (http://joey711.github.io/phyloseq/plot_tree-examples.html)

phyloseq also contains a method for easily plotting an annotated phylogenetic tree with information regarding the sample in which a particular taxa was observed, and optionally the number of individuals that were observed.

For the sake of creating a readable tree, let's subset the data to just the Chlamydiae phylum, which consists of obligate intracellular pathogens and is present in only a subset of environments in this dataset.

```
GP.chl <- subset_taxa(GP, Phylum=="Chlamydiae")
```

And now we will create the tree graphic form this subset of Global Patterns, shading by the "`SampleType" variable, which indicates the environment category from which the microbiome samples originated. The following command also takes the option of labeling the number of individuals observed in each sample (if at all) of each taxa. The symbols are slightly enlarged as the number of individuals increases.

```
plot_tree(GP.chl, color="SampleType", shape="Family", label.tips="Genus", size="Abundance")
```

Exploratory bar plots

For further details, see the plot_bar tutorial (http://joey711.github.io/phyloseq/plot_bar-examples.html)

In the following example we use the included "enterotype" dataset (Arumugam 2011).

```
data(enterotype)
```

We start with a simple rank-abundance barplot, using the cumulative fractional abundance of each OTU in the dataset. In the enterotype dataset, the available published data are simplified as sample-wise fractional occurrences, rather than counts of individuals\footnote{Unfortunate, as this means we lose information about the total number of reads and associated confidences, ability to do more sophisticated richness estimates, etc. For example, knowing that we observed 1 sequence read of a species out of 100 total reads means something very different from observing 10,000 reads out of 1,000,000 total., and OTUs are clustered/labeled at the genus level, but no other taxonomic assignment is available. In this barplot we further normalized by the total number of samples (280).

```
par(mar = c(10, 4, 4, 2) + 0.1) # make more room on bottom margin

N <- 30
barplot(sort(taxa_sums(enterotype), TRUE)[1:N]/nsamples(enterotype), las=2)
```

An example exploratory barplot using base R graphics and thetaxa_sums and `nsamples functions. Note that this first barplot is clipped at the 30th OTU. This was chosen because ntaxa(enterotype) =553 OTUs would not be legible on the plot. As you can see, the relative abundances have decreased dramatically by the 10th-ranked OTU.

So what are these OTUs? In the enterotype dataset, only a single taxonomic rank type is present:

```
rank_names(enterotype)
## [1] "Genus"
```

This means the OTUs in this dataset have been grouped at the level of genera, and no other taxonomic grouping/transformation is possible without additional information (like might be present in a phylogenetic tree, or with further taxonomic classification analysis).

We need to know which taxonomic rank classifiers, if any, we have available to specify in the second barplot function in this example, plot_bar().We have already observed how quickly the abundance decreases with rank, so wo we will subset the enterotype dataset to the most abundantN taxa in order to make the barplot legible on this page.

```
TopNOTUs <- names(sort(taxa_sums(enterotype), TRUE)[1:10])
ent10 <- prune_taxa(TopNOTUs, enterotype)
print(ent10)
```

Note also that there are 280 samples in this dataset, and so a remaining challenge is to consolidate these samples into meaningful groups. A good place to look is the available sample variables, which in most cases will carry more “meaning” than the sample names alone.

```
sample_variables(ent10)
```

The parameters to plot_bar in the following code-chunk were chosen after various trials. We suggest that you also try different parameter settings while you’re exploring different features of the data. In addition to the variables names of sample_data, the plot_bar() function recognizes the names of taxonomic ranks (if present). See the help documentation and further details in the examples and on the wiki page. In this example we have also elected to organize data by “facets” (separate, adjacent sub-plots) according to the genus of each OTU. Within each genus facet, the data is further separated by sequencing technology, and the enterotype label for the sample from which each OTU originated is indicated by fill color. Abundance values from different samples and OTUs but having the same variables mapped to the horizontal (x) axis are sorted and stacked, with thin horizontal lines designating the boundaries. With this display it is very clear that the choice of sequencing technology had a large effect on which genera were detected, as well as the fraction of OTUs that were assigned to a Genus.

```
plot_bar(ent10, "SeqTech", fill="Enterotype", facet_grid=~Genus)
```

An example exploratory bar plot using the plot_bar function. In this case we have faceted the data (abundance values) according to the genera of each OTU. The subset of OTUs that have not been assigned to a specific genus are in the NA panel. Within each facet, the data is further separated by sequencing technology, and each OTU is shaded according to the enterotype of the sample it form which it came. Abundance values from different samples and OTUs but having the same variables mapped to the horizontal (x) axis are sorted and stacked, with thin horizontal lines designating the boundaries.

The figure summarizes quantitatively the increased abundances of Bacteroides and Prevotella in the Enterotypes 1 and 2, respectively. Interestingly, a large relative abundance of Blautia was observed for Enterotype 3, but only from 454-pyrosequencing data sets, not the Illumina or Sanger datasets. This suggests the increased Blautia might actually be an artifact. Similarly, Prevotella appears to be one of the most abundant genera in the Illumina-sequenced samples among Enterotype 3, but this is not reproduced in the 454-pyrosequencing or Sanger sequencing data.

Using the Phyloseq package

https://ucdavis-bioinformatics-training.github.io/2017-September-Microbial-Community-Analysis-Workshop/friday/MCA_Workshop_R/phyloseq.html

The phyloseq package is fast becoming a good way a managing micobial community data, filtering and visualizing that data and performing analysis such as ordination. Along with the standard R environment and packages vegan and vegetarian you can perform virually any analysis. Today we will

1. Install R packages 2 Load data straight from dbcAmplicons (biom file)
2. Filter out Phylum
3. Filter out additional Taxa
4. Filter out samples
5. Graphical Summaries
6. Ordination
7. Differential Abundances

We first need to make sure we have the necessary packages: phyloseq, ggplot2, gridExtra, gridR, ape, and edgeR.

```
source("http://bioconductor.org/biocLite.R")
biocLite("phyloseq")
biocLite("ggplot2")
biocLite("gridExtra")
biocLite("edgeR")
biocLite("vegan")
library(phyloseq)
library(ggplot2)
library(gridExtra)
library(vegan)
```

Read in the dataset, biom file generated from dbcAmplicons pipeline (*Since we do not have access to that particular dataset, we will use datasets loaded from the phyloseq data as well as manually importing the OTU table, phylogenetic tree and sample map from the mice example in Dr. Dan Knight's lecture)**

First read in the dataset, see what the objects look like. Our Biom file, produces 3 tables: otu_table, taxa_table, sample_data. Look at the head of each. Get the sample names and tax ranks, finally view the phyloseq object. Lets draw a first bar plot.

```
slashpile_16sV1V3 <- "16sV1V3.biom"
s16sV1V3 = import_biom(BIOMfilename = slashpile_16sV1V3, parseFunction =
parse_taxonomy_default) colnames(tax_table(s16sV1V3)) <- c("Kingdom", "Phylum", "Class",
"Order", "Family", "Genus")

head(otu_table(s16sV1V3))

head(sample_data(s16sV1V3))

head(tax_table(s16sV1V3))

rank_names(s16sV1V3)

sample_variables(s16sV1V3)

s16sV1V3

plot_bar(s16sV1V3, fill = "Phylum") + theme(legend.position="bottom")
```

(*Since we do not have access to that particular dataset, we will use datasets loaded from the phyloseq data as well as manually importing the OTU table, phylogenetic tree and sample map from the mice example in Dr. Dan Knight's lecture)**

```
modernotumice <- import_biom ("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/closedref/otu_table.biom")
modernotumice
head(otu_table(modernotumice))
head(tax_table(modernotumice))
```

(*** Since we are importing individual pieces of the dataset using a phyloseq function, we have to tell R where the sample data would be since we had not yet merged it into a phyloseq object

```
modernmapmice <- import_qiime_sample_data("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/map.txt")
head(sample_data(modernmapmice))
```

** This is the more straightforward alternative

```
modernotumice <- import_biom ("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/closedref/otu_table.biom")
moderntreemice <- read_tree("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/closedref/97_otus.tree")
modernmapmice <- import_qiime_sample_data("/Users/chigu142000/mice_tutorial/mice8992-2016-master/data/globalgut-66-adults/map.txt")
phyloseqmice <- merge_phyloseq (modernotumice,modernmapmice,moderntreemice)
head(sample_data(phyloseqmice))
```

*For rank name, which refers to the phylogenetic rank of each taxa, you can manually rename each column as shown below, but make sure that the dimension for your new column names matches the number of columns in the dataset

```
rank_names(phyloseqmice)
colnames(tax_table(phyloseqmice)) <- c("Kingdom", "Phylum", "Class", "Order", "Family",
"Genus", "Species")
rank_names(phyloseqmice)
sample_variables(phyloseqmice)
phyloseqmice
plot_bar(phyloseqmice, fill = "Phylum") + theme(legend.position="bottom")
```

Filtering our dataset

Lets generate a prevelance table (number of samples each taxa occurs in) for each taxa.

```
prevelancedf = apply(X = otu_table(s16sV1V3),
MARGIN = 1,
FUN = function(x){sum(x > 0)})
# Add taxonomy and total read counts to this data.frame
prevelancedf = data.frame(Prevalence = prevelancedf,                    TotalAbundance =
taxa_sums(s16sV1V3),
tax_table(s16sV1V3))
prevelancedf[1:10,]
```

*** Remember to use open parentases or operators such as "," or "+" at the end of each line for long functions

```
prevelancedf = apply(X = otu_table(phyloseqmice),
+ MARGIN = 1,
+ FUN = function(x){sum(x > 0)})
# Add taxonomy and total read counts to this data.frame
prevelancedf = data.frame(Prevalence = prevelancedf, TotalAbundance =
taxa_sums(phyloseqmice),
tax_table(phyloseqmice))
prevelancedf[1:10,]
```

Whole phylum filtering

First lets remove of the feature with ambiguous phylum annotation

```
s16sV1V3.1
s16sV1V3.1 <- subset_taxa(s16sV1V3, !is.na(Phylum) & !Phylum %in% c("",
"uncharacterized"))
s16sV1V3.1
```

phyloseqmice
phyloseqmice.1 <- subset_taxa(phyloseqmice, !is.na(Phylum) & !Phylum %in% c("",
"uncharacterized"))
phyloseqmice.1

Now lets investigate low prevelance/abundance phylum and subset them out.

```
plyr::ddply(prevelancedf, "Phylum", function(df1){
data.frame(mean_prevalence=mean(df1$Prevalence),total_abundance=sum(df1$TotalAbundanc
e,na.rm = T),stringsAsFactors = F)   })
```

Using the table above, determine the phyla to filter (From the tutorial)

```
phyla2Filter = c("p__Aquificae", "p__candidate division ZB3", "p__Crenarchaeota",
"p__Deinococcus-Thermus","p__Omnitrophica","p__Tenericutes",
"p__Thermodesulfobacteria")
```

*** I had issues with the style of quotation marks on my mac keyboard not being recognized in R, so I had to manually copy each phylum into the original code above

```
phyla2Filter = c("p__Actinobacteria ", " p__Armatimonadetes", " p__Bacteroidetes ",
"p__Deinococcus-Thermus","p__Omnitrophica","p__Tenericutes",
"p__Thermodesulfobacteria")
```

```
# Filter entries with unidentified Phylum.
s16sV1V3.1
s16sV1V3.1 = subset_taxa(s16sV1V3.1, !Phylum %in% phyla2Filter)
s16sV1V3.1
```

```
# Filter entries with unidentified Phylum.
Phyloseqmice.1
Phyloseqmice.1 = subset_taxa(s16sV1V3.1, !Phylum %in% phyla2Filter)
Phyloseqmice.1
```

Individual Taxa Filtering

Subset to the remaining phyla by prevelance.

```
prevelancedf1 = subset(prevelancedf, Phylum %in% get_taxa_unique(s16sV1V3.1,
taxonomic.rank = "Phylum"))
```

```
ggplot(prevelancedf1, aes(TotalAbundance, Prevalence /
nsamples(s16sV1V3.1),color=Phylum)) +
# Include a guess for parameter
geom_hline(yintercept = 0.05, alpha = 0.5, linetype = 2) + geom_point(size = 2, alpha = 0.7) +
scale_x_log10() +  xlab("Total Abundance") + ylab("Prevalence [Frac. Samples]") +
facet_wrap(~Phylum) + theme(legend.position="none")

prevelancedf1 = subset(prevelancedf, Phylum %in% get_taxa_unique(phyloseqmice.1,
taxonomic.rank = "Phylum"))

ggplot(prevelancedf1, aes(TotalAbundance, Prevalence /
nsamples(phyloseqmice.1),color=Phylum)) +
# Include a guess for parameter
geom_hline(yintercept = 0.05, alpha = 0.5, linetype = 2) + geom_point(size = 2, alpha = 0.7) +
scale_x_log10() +  xlab("Total Abundance") + ylab("Prevalence [Frac. Samples]") +
facet_wrap(~Phylum) + theme(legend.position="none")
```

Sometimes you see a clear break, however we aren’t seeing one here. In this case I’m moslty interested in those organisms consistantly present in the dataset, so I’m removing all taxa present in less than 50% of samples.

```
#  Define prevalence threshold as 10% of total samples
prevalenceThreshold = 0.50 * nsamples(s16sV1V3.1)
prevalenceThreshold
```

```
#  Define prevalence threshold as 10% of total samples
prevalenceThreshold = 0.50 * nsamples(phyloseqmice.1)
prevalenceThreshold
```

```
# Execute prevalence filter, using `prune_taxa()` function
keepTaxa = rownames(prevelancedf1)[(prevelancedf1$Prevalence >= prevalenceThreshold)]
length(keepTaxa)
s16sV1V3.2 = prune_taxa(keepTaxa, s16sV1V3.1)
s16sV1V3.2
```

```
keepTaxa = rownames(prevelancedf1)[(prevelancedf1$Prevalence >= prevalenceThreshold)]
length(keepTaxa)
phyloseqmice.2 = prune_taxa(keepTaxa, phyloseqmice.1)
phyloseqmice.2
```

Agglomerate taxa at the Genus level (combine all with the same name) and remove all taxa without genus level assignment

```
length(get_taxa_unique(s16sV1V3.2, taxonomic.rank = "Genus"))
s16sV1V3.3 = tax_glom(s16sV1V3.2, "Genus", NArm = TRUE)
s16sV1V3.3
```

```
length(get_taxa_unique(phyloseqmice.2, taxonomic.rank = "Genus"))
phyloseqmice.3 = tax_glom(phyloseqmice.2, "Genus", NArm = TRUE)
phyloseqmice.3
```

```
## out of curiosity how many "reads" does this leave us at???
sum(colSums(otu_table(s16sV1V3.3)))
```

```
sum(colSums(otu_table(phyloseqmice.3)))
```

Now lets filter out samples (outliers and low performing samples)

Do some simple ordination looking for outlier samples, first we variance stabilize the data with a log transform, the perform PCoA using bray's distances

```
logt  = transform_sample_counts(s16sV1V3.3, function(x) log(1 + x) )
out.pcoa.logt <- ordinate(logt, method = "PCoA", distance = "bray")
evals <- out.pcoa.logt$values$Eigenvalues plot_ordination(logt, out.pcoa.logt, type = "samples",
color = "Slash_pile_number", shape = "Depth_cm") + labs(col = "Slash pile number") +
coord_fixed(sqrt(evals[2] / evals[1]))
```

*** The slash pile number is the variable name seen in head(otu_table(s16sV1V3)) so we need to modify our code accordingly: The color is for the individual datapoint categories while the shape is for the other category we want to evaluate them by.

```
head(otu_table(phyloseqmice))
sample_variables(phyloseqmice)
head(sample_data(phyloseqmice))
```

```
logt  = transform_sample_counts(phyloseqmice.3, function(x) log(1 + x) )
out.pcoa.logt <- ordinate(logt, method = "PCoA", distance = "bray")
evals <- out.pcoa.logt$values$Eigenvalues plot_ordination(logt, out.pcoa.logt, type = "samples",
color = " X.SampleID", shape = "SEX") + labs(col = " X.SampleID") +
coord_fixed(sqrt(evals[2] / evals[1]))
plot_ordination(logt, out.pcoa.logt, type = "species", color = "Phylum")
```

***I got the following error (Error in .C("veg_distance", x = as.double(x), nr = N, nc = ncol(x), d = double(N * : "veg_distance" not available for .C() for package "vegan")
I tried calculating it the same way that it was done in Dr. Dan Knight's class, but it did not work likely due to the dataset being a phyloseq class object

```
d.bray <- vegdist(otus)
```

The solution for ordination without all of the bells and whistles was found at the phyloseq quick tutorial (http://bioconductor.org/packages/release/bioc/vignettes/phyloseq/inst/doc/phyloseq-analysis.html#ordination-methods)

```
my.ord <- ordinate(my.physeq)
plot_ordination(my.physeq, my.ord, color="myFavoriteVarible")
```

```
out.pcoa.logt <- ordinate(logt)
plot_ordination(phyloseqmice.3, out.pcoa.logt)
plot_ordination(phyloseqmice.3, out.pcoa.logt, type = "species", color = "Phylum")
```

*** You could also troubleshoot by using help(ordinate) to find out that in the current version, PCoA has been chanced to DPCoA so that option works

```
ordinate(physeq, method = "DCA", distance = "bray", formula = NULL, ...)
```

Currently supported method options are: c("DCA", "CCA", "RDA", "CAP", "DPCoA", "NMDS", "MDS", "PCoA")

You could also use the MDS method of ordination here, edit the code to do so. Can also edit the distance method used to jaccard, jsd, euclidean. Play with changing those parameters

```
#Can view the distance method options with  ?distanceMethodList  # can veiw the oridinate methods with ?ordinate
```

Show taxa proportions per sample

```
grid.arrange(nrow = 2, qplot(as(otu_table(logt),"matrix")[, "Slashpile18"], geom = "histogram", bins=30) +   xlab("Relative abundance"),  qplot(as(otu_table(logt),"matrix")[, "Slashpile10"], geom = "histogram", bins=30) +   xlab("Relative abundance") )
```

*Need to replace the slashpile sample numbers with sample ids from our own dataset (note that nrow=2 indicates that we are only comparing 2 rows in the code above)

```
head(otu_table(phyloseqmice))
sample_variables(phyloseqmice)
head(sample_data(phyloseqmice))
grid.arrange(nrow = 2, qplot(as(otu_table(logt),"matrix")[, "h186M.1.418788"], geom = "histogram", bins=30) +   xlab("Relative abundance"),  qplot(as(otu_table(logt),"matrix")[, "h273M.1.418507"], geom = "histogram", bins=30) +   xlab("Relative abundance") )
```

```
# if you needed to remove candidate outliers, can use the below to remove sample Slashpile18
#s16sV1V3.4 <- prune_samples(sample_names(s16sV1V3.4) != "Slashpile18", s16sV1V3.4)
```

Look for low performing samples

```
qplot(colSums(otu_table(s16sV1V3.3)),bins=30) +   xlab("Logged counts-per-sample")
```

```
qplot(colSums(otu_table(phyloseqmice.3)),bins=30) +   xlab("Logged counts-per-sample")
```

```
s16sV1V3.4 <- prune_samples(sample_sums(s16sV1V3.3)>=10000, s16sV1V3.3)
s16sV1V3.4
```

** You need to look at the x axis of the previous gplot to determine what your read cutpoint will be!! I am going to set it at 500

```
phyloseqmice.4 <- prune_samples(sample_sums(phyloseqmice.3)>=500, phyloseqmice.3)
phyloseqmice.4
```

Investigate transformations. We transform microbiome count data to account for differences in library size, variance, scale, etc.

```
## for Firmictures
plot_abundance = function(physeq, meta, title = "",
Facet = "Order", Color = "Order"){
# Arbitrary subset, based on Phylum, for plotting
p1f = subset_taxa(physeq, Phylum %in% c("p__Firmicutes"))
mphyseq = psmelt(p1f)
mphyseq <- subset(mphyseq, Abundance > 0)
ggplot(data = mphyseq, mapping = aes_string(x = meta,y = "Abundance",
color = Color, fill = Color)) +
geom_violin(fill = NA) +
geom_point(size = 1, alpha = 0.3,
position = position_jitter(width = 0.3)) +
facet_wrap(facets = Facet) + scale_y_log10()+
theme(legend.position="none") }

# transform counts into "abundances"
s16sV1V3.4ra = transform_sample_counts(s16sV1V3.4, function(x){x / sum(x)})
s16sV1V3.4hell <- s16sV1V3.4
otu_table(s16sV1V3.4hell) <-otu_table(decostand(otu_table(s16sV1V3.4hell),
method = "hellinger"), taxa_are_rows=TRUE)
s16sV1V3.4log <- transform_sample_counts(s16sV1V3.4, function(x) log(1 + x))  plotOriginal
= plot_abundance(s16sV1V3.4, "Slash_pile_number", title="original") plotRelative =
plot_abundance(s16sV1V3.4ra, "Slash_pile_number", title="relative") plotHellinger =
plot_abundance(s16sV1V3.4hell, "Slash_pile_number", title="Hellinger")
plotLog = plot_abundance(s16sV1V3.4log, "Slash_pile_number", title="Log")
# Combine each plot into one graphic.
grid.arrange(nrow = 4, plotOriginal, plotRelative, plotHellinger, plotLog)
```

** Use head(tax_table(modernotumice)) or head(tax_table(phyloseqmice)) to look at which phylum you have in your dataset > When I tried changing the function to Euryarchaeota instead of Firnicures in the custom made function, the custom plot_abundance function would not read any of my datasets. I had to keel phylum as firnicutes

**More details on the custom plot_abundance function can be found at (https://www.bioconductor.org/help/course-materials/2017/BioC2017/Day1/Workshops/Microbiome/doc/MicrobiomeWorkshopII.html) Looking back, my mistake was using phyloseq mice instead of phyloseqmice.4 so the Euryarchaeota kingdom was probably pruned out

** Use head(sample_data(phyloseqmice)) to find the sample variable in your dataset

head(tax_table(phyloseqmice.4))
head(sample_data(phyloseqmice.4))

```
## for Firnicutes
plot_abundance = function(physeq, meta, title = "",
Facet = "Order", Color = "Order"){
# Arbitrary subset, based on Phylum, for plotting
p1f = subset_taxa(physeq, Phylum %in% c("p__Firmicutes"))
mphyseq = psmelt(p1f)
mphyseq <- subset(mphyseq, Abundance > 0)
ggplot(data = mphyseq, mapping = aes_string(x = meta,y = "Abundance",
color = Color, fill = Color)) +
geom_violin(fill = NA) +
geom_point(size = 1, alpha = 0.3,
position = position_jitter(width = 0.3)) +
facet_wrap(facets = Facet) + scale_y_log10()+
theme(legend.position="none") }

# transform counts into "abundances"
phyloseqmice.4ra = transform_sample_counts(phyloseqmice.4, function(x){x / sum(x)})
phyloseqmice.4hell <- phyloseqmice.4
otu_table(phyloseqmice.4hell) <-otu_table(decostand(otu_table(phyloseqmice.4hell),
method = "hellinger"), taxa_are_rows=TRUE)
phyloseqmice.4log <- transform_sample_counts(phyloseqmice.4, function(x) log(1 + x))
plotOriginal = plot_abundance(phyloseqmice.4, "X.SampleID", title="original") plotRelative =
plot_abundance(phyloseqmice.4ra, "X.SampleID", title="relative")
plotHellinger = plot_abundance(phyloseqmice.4hell, "X.SampleID", title="Hellinger")
plotLog = plot_abundance(phyloseqmice.4log, "X.SampleID", title="Log")
# Combine each plot into one graphic.
grid.arrange(nrow = 4, plotOriginal, plotRelative, plotHellinger, plotLog)
grid.arrange(nrow = 1, plotOriginal)
```

Graphical Summaries
** Plot Alpha diversity

```
plot_richness(s16sV1V3.4, measures=c("Observed","Chao1"))
```

```
plot_richness(phyloseqmice.4, measures=c("Observed","Chao1"))
plot_richness(s16sV1V3.4, x = "Slash_pile_number", color="Depth_cm", measures=c("Chao1",
"Shannon"))
```

```
plot_richness(phyloseqmice.4, x = "X.SampleID", color="SEX", measures=c("Chao1",
"Shannon"))
```

```
# Subset dataset by phylum
s16sV1V3.4hell_acidob = subset_taxa(s16sV1V3.4hell, Phylum=="p__Acidobacteria")
title = "plot_bar; Acidobacteria-only"
plot_bar(s16sV1V3.4hell_acidob, "Slash_pile_number", "Abundance", "Family", title=title)
head(tax_table(phyloseqmice.4))
phyloseqmice.4hell_acidob = subset_taxa(phyloseqmice.4hell, Phylum=="p__Proteobacteria")
title = "plot_bar; Proteobacteria -only"
plot_bar(phyloseqmice.4hell_acidob, "X.SampleID", "Abundance", "Family", title=title)
```

*** Plot Heat Maps

```
prop  = transform_sample_counts(s16sV1V3.4, function(x) x / sum(x) )
keepTaxa <- ((apply(otu_table(prop) >= 0.005,1,sum,na.rm=TRUE) > 2) |
(apply(otu_table(prop) >= 0.05, 1, sum,na.rm=TRUE) > 0))
table(keepTaxa)
```

```
prop  = transform_sample_counts(phyloseqmice.4, function(x) x / sum(x) )
keepTaxa <- ((apply(otu_table(prop) >= 0.005,1,sum,na.rm=TRUE) > 2) |
(apply(otu_table(prop) >= 0.05, 1, sum,na.rm=TRUE) > 0))
table(keepTaxa)
```

```
s16sV1V3.4hell_trim <- prune_taxa(keepTaxa,s16sV1V3.4hell)
plot_heatmap(s16sV1V3.4hell_trim, "PCoA", distance="bray",
sample.label="Slash_pile_number", taxa.label="Genus", low="#FFFFCC", high="#000033",
na.value="white")
```

*** I needed to change from the PCoA method to the DCA method in my current version of vegan (You could always use the default settings)

```
phyloseqmice.4hell_trim <- prune_taxa(keepTaxa,phyloseqmice.4hell)
plot_heatmap(phyloseqmice.4hell_trim, "DCA", distance="bray", sample.label="X.SampleID",
taxa.label="Genus", low="#FFFFCC", high="#000033", na.value="white")
```

*Plot community network

```
plot_net(s16sV1V3.4hell_trim, maxdist=0.4, color="Slash_pile_number", shape="Depth_cm")
```

```
plot_net(phyloseqmice.4hell_trim, maxdist=0.4, color="X.SampleID", shape="SEX")
```

****I tried using the help(plot_net) functions to use the default setting for plotting networks as shown above but it does not work in the current version of phyloseq I am using

```
plot_net(physeq, distance = "bray", type = "samples", maxdist = 0.7,
  laymeth = "fruchterman.reingold", color = NULL, shape = NULL,
  rescale = FALSE, point_size = 5, point_alpha = 1, point_label = NULL,
  hjust = 1.35, title = NULL)
data(enterotype)
plot_net(enterotype, color="SeqTech", maxdist = 0.3)
```

*** Ordination

Ordination

```
v4.hell.ord <- ordinate(s16sV1V3.4hell, "NMDS", "bray")
v4.hell.ord <- ordinate(phyloseqmice.4hell, "NMDS", "bray")

p1 = plot_ordination(s16sV1V3.4hell, v4.hell.ord, type="taxa", color="Phylum", title="taxa")
print(p1)
p1 + facet_wrap(~Phylum, 5)


p1 = plot_ordination(phyloseqmice.4hell, v4.hell.ord, type="taxa", color="Phylum", title="taxa")
print(p1)
p1 + facet_wrap(~Phylum, 5)
```

You can also try doing oridination with other transformations, such as relative abundance, log. Also looks and see if you can find any trends in an independent variable in your dataset, such as SEX or Contry

Phyloseq to EdgeR Example 1

Differential Abundances

For differential abundances we use RNAseq pipeline EdgeR and limma voom.

```
library("edgeR")
```

```
m = as(otu_table(s16sV1V3.4), "matrix")
# Add one to protect against overflow, log(0) issues.
m = m + 1
# Define gene annotations (`genes`) as tax_table
taxonomy = tax_table(s16sV1V3.4, errorIfNULL=FALSE)
if( !is.null(taxonomy) ){    taxonomy = data.frame(as(taxonomy, "matrix")) }
# Now turn into a DGEList
d = DGEList(counts=m, genes=taxonomy, remove.zeros = TRUE)
# Calculate the normalization factors
z = calcNormFactors(d, method="RLE")
# Check for division by zero inside `calcNormFactors`
if( !all(is.finite(z$samples$norm.factors)) ){    stop("Something wrong with
edgeR::calcNormFactors on this data,        non-finite $norm.factors, consider changing `method`
argument") }
plotMDS(z, col = as.numeric(factor(sample_data(s16sV1V3.4)$Slash_pile_number)), labels =
sample_names(s16sV1V3.4))
```

*** Since it is a phyloseq object, you can just pull the out table and the sample data information from the one dataset

```
m = as(otu_table(phyloseqmice.4), "matrix")
# Add one to protect against overflow, log(0) issues.
m = m + 1
# Define gene annotations (`genes`) as tax_table
taxonomy = tax_table(phyloseqmice.4, errorIfNULL=FALSE)
if( !is.null(taxonomy) ){   taxonomy = data.frame(as(taxonomy, "matrix")) }
# Now turn into a DGEList
d = DGEList(counts=m, genes=taxonomy, remove.zeros = TRUE)
# Calculate the normalization factors
z = calcNormFactors(d, method="RLE")
# Check for division by zero inside `calcNormFactors`
if( !all(is.finite(z$samples$norm.factors)) ){   stop("Something wrong with
edgeR::calcNormFactors on this data,        non-finite $norm.factors, consider changing `method`
argument") }
plotMDS(z, col = as.numeric(factor(sample_data(phyloseqmice.4)$X.SampleID)), labels =
sample_names(phyloseqmice.4))
```

```
# Creat a model based on Slash_pile_number and depth
mm <- model.matrix(~ Slash_pile_number + Depth_cm,
data=data.frame(as(sample_data(s16sV1V3.4),"matrix")
```

```
# Creat a model based on Slash_pile_number and depth
mm <- model.matrix(~ X.SampleID + SEX,
data=data.frame(as(sample_data(phyloseqmice.4),"matrix")))
# specify model with no intercept for easier contrasts

*Check the mean variance trend
y <- voom(d, mm, plot = T)

fit <- lmFit(y, mm) head(coef(fit))
```

Phyloseq to EdgeR Example 2

(http://joey711.github.io/phyloseq-extensions/edgeR.html)
For this example, instead of using the the publicly available data from a study on colorectal cancer (kosticDB) from the website, we will use datasets that are included in phyloseq

Example Phyloseq Data (These are already converted to phyloseq format)**

There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets.

To load example data into the working environment, use the data() command:

```
data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
```

Similarly, entering ?enterotype will reveal the documentation for the so-called “enterotype” dataset.

Included Data

To facilitate testing and exploration of tools in phyloseq, this package includes example data from published studies. Many of the examples in this vignette use either the Global Patterns or enterotype datasets as source data. The Global Patterns data was described in a 2011 article in PNAS(Caporaso 2011), and compares the microbial communities of 25 environmental samples and three known “mock communities” — a total of 9 sample types — at a depth averaging 3.1 million reads per sample. The human enterotype dataset was described in a 2011 article in Nature(Arumugam 2011), which compares the faecal microbial communities from 22 subjects using complete shotgun DNA sequencing. The authors further compare these microbial communities with the faecal communities of subjects from other studies, for a total of 280 faecal samples / subjects, and 553 genera. Sourcing data from different studies invariable leads to gaps in the data for certain variables, and this is easily handled by R's coreNA features.

Because this data is included in the package, the examples can easily be run on your own computer using the code shown in this vignette. The data is loaded into memory using the datacommand. Let’s start by loading the `phyloseq-package and the GlobalPatterns data

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("phyloseq")
library("phyloseq")
library("ggplot2")
data(GlobalPatterns)
```

Independent filtering

Independent filtering is done automatically in DESeq2, but not in edgeR. Here we will filter OTUs for which the variance across all samples is very low, and we’ll do this before ever passing the data to EdgeR.

```
kosticp = transform_sample_counts(kosticB, function(x){x/sum(x)})

hist(log10(apply(otu_table(kosticp), 1, var)),      xlab="log10(variance)", breaks=50,      main="A large fraction of OTUs have very low variance")

GlobalPatternsp = transform_sample_counts(GlobalPatterns, function(x){x/sum(x)})

hist(log10(apply(otu_table(GlobalPatternsp), 1, var)),      xlab="log10(variance)", breaks=50, main="A large fraction of OTUs have very low variance")
```

You will notice that a large fraction of OTUs have very low variance and we will need to filter them

```
varianceThreshold = 1e-5
```

```
keepOTUs = names(which(apply(otu_table(GlobalPatternsp), 1, var) > varianceThreshold)) Glob
alPatternsB = prune_taxa(keepOTUs, GlobalPatterns)

GlobalPatternsB
```

Here we've used an arbitrary but not-unreasonable variance threshold of 10^{-5}. It is important to keep in mind that this filtering is independent of our downstream test. The sample classifications were not used.

```
phyloseq_to_edgeR = function(physeq, group, method="RLE", ...){

require("edgeR")

require("phyloseq")

# Enforce orientation.

if( !taxa_are_rows(physeq) ){ physeq <- t(physeq) }

x = as(otu_table(physeq), "matrix")

# Add one to protect against overflow, log(0) issues.

x = x + 1

# Check `group` argument

if( identical(all.equal(length(group), 1), TRUE) & nsamples(physeq) > 1 ){

# Assume that group was a sample variable name (must be categorical)

group = get_variable(physeq, group)   }

# Define gene annotations (`genes`) as tax_table

taxonomy = tax_table(physeq, errorIfNULL=FALSE)

if( !is.null(taxonomy) ){     taxonomy = data.frame(as(taxonomy, "matrix"))   }

# Now turn into a DGEList

y = DGEList(counts=x, group=group, genes=taxonomy, remove.zeros = TRUE, ...)

# Calculate the normalization factors

z = calcNormFactors(y, method=method)

# Check for division by zero inside `calcNormFactors`

if( !all(is.finite(z$samples$norm.factors)) ){     stop("Something wrong with edgeR::calcNormFa
ctors on this data,          non-finite $norm.factors, consider changing `method` argument")   }

# Estimate dispersions

return(estimateTagwiseDisp(estimateCommonDisp(z))) }
```

Now let's use a function to convert the phyloseq data object into an edgeR "DGE" data object. In order to do that, we will need to run the custom function above. However, as we showed in example #1 you do not need a wrapper R function as you can walk through each of those automated steps manually

GlobalPatternsB

```
library("edgeR")
```

head(sample_data(GlobalPatternsB))

We will use SampleType as the categorical variable for our differential abundance analyses

```
dge = phyloseq_to_edgeR(kosticB, group="DIAGNOSIS")

dge = phyloseq_to_edgeR(GlobalPatternsB, group="SampleType")

Now we can run our tests to display our differentially abundant groups

# Perform binary test

et = exactTest(dge)

# Extract values from test results

tt = topTags(et, n=nrow(dge$table), adjust.method="BH", sort.by="PValue")

res = tt@.Data[[1]]

alpha = 0.001

sigtab = res[(res$FDR < alpha), ]

sigtab = cbind(as(sigtab, "data.frame"), as(tax_table(GlobalPatternsB)[rownames(sigtab), ], "mat
rix"))

dim(sigtab)

head(sigtab)
```

Here is a bar plot showing the log2-fold-change, showing Genus and Phylum. Uses some ggplot2 commands.

```
library("ggplot2")

theme_set(theme_bw())

scale_fill_discrete <- function(palname = "Set1", ...) {     scale_fill_brewer(palette = palname, ...
) }

sigtabgen = subset(sigtab, !is.na(Genus))

# Phylum order

x = tapply(sigtabgen$logFC, sigtabgen$Phylum, function(x) max(x))

x = sort(x, TRUE)

sigtabgen$Phylum = factor(as.character(sigtabgen$Phylum), levels = names(x))
```

```
# Genus order
x = tapply(sigtabgen$logFC, sigtabgen$Genus, function(x) max(x))
x = sort(x, TRUE)
sigtabgen$Genus = factor(as.character(sigtabgen$Genus), levels = names(x))
ggplot(sigtabgen, aes(x = Genus, y = logFC, color = Phylum)) + geom_point(size=6) +    theme(
axis.text.x = element_text(angle = -90, hjust = 0, vjust = 0.5))
```

Phyloseq to DESeq2 Example

Example using Negative Binomial in Microbiome Differential Abundance Testing
https://bioconductor.org/packages/devel/bioc/vignettes/phyloseq/inst/doc/phyloseq-mixture-models.html#import-data-with-phyloseq-convert-to-deseq2

****
The original tutorial called for using a dataset accessible through the microbio_me-qiime function. However, since we know that the phyloseq function can be used for individual subset files (sample data, otu tree and ref database, I will locate those from the QIIME and the MICE tutorials that I have and use those instead to create this tutorial. Actually for this example, I will use one of the datasets that is pre-loaded into Phyloseq for the demo
****

Load phyloseq & DESeq2 and import data

Load phyloseq

To use phyloseq in a new R session, it will have to be loaded. This can be done in your package manager, or at the command line using the library() command:

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("DESeq2")
biocLite("phyloseq")
library("phyloseq")
library("DESeq2")
library("ggplot2")
```

****
Actually for this example, I will use one of the datasets that is pre-loaded into Phyloseq for the demo
****

Example Phyloseq Data (**These are already converted to phyloseq format)

There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets. To load example data into the working environment, use the data() command:

```
data(GlobalPatterns)
```

phyloseq Object Summaries

In small font, the following is the summary of the GlobalPatterns dataset that prints to the terminal. These summaries are consistent among all phyloseq-class objects. Although the components of GlobalPatterns have many thousands of elements, the command-line returns only a short summary of each component. This encourages you to check that an object is still what you expect, without needing to let thousands of elements scroll across the terminal. In the cases in which you do want to see more of a particular component, use an accessor function (see table below).

```
data(GlobalPatterns)
```

```
GlobalPatterns
## phyloseq-class experiment-level object
## otu_table()   OTU Table:         [ 19216 taxa and 26 samples ]
## sample_data() Sample Data:       [ 26 samples by 7 sample variables ]
## tax_table()   Taxonomy Table:    [ 19216 taxa by 7 taxonomic ranks ]
## phy_tree()    Phylogenetic Tree: [ 19216 tips and 19215 internal nodes ]
```

Convert to DESeq2’s DESeqDataSet class

First, we need to define the study design factor. For example if the goal was to compare the microbiomes of pairs of healthy and cancerous tissues, you would use the diagnosis variable as your study design factor. Your study could have a more complex or nested design, and you should think carefully about the study design formula, because this is critical to the test results and their meaning. You might even need to define a new factor if none of the variables in your current table appropriately represent your study’s design. See the DESeq2 home page for more details.

Here is the summary of the phyloseq object that we are about to use, GlobalPatterns

```
head(sample_data(GlobalPatterns))
```

In this example, the study factor variable would be SampleType

DESeq2 conversion and call

First remove any samples that had no study factor attribute assigned. These introduce a spurious third design class that is actually a rare artifact in the dataset.

Second remove samples with less than 500 reads (counts). Note that this kind of data cleanup is useful, necessary, and should be well-documented because it can also be dangerous to alter or omit data without clear documentation.

```
GlobalPatterns <- subset_samples(GlobalPatterns, SampleType != "NA")
GlobalPatterns <- prune_samples(sample_sums(GlobalPatterns) > 500, GlobalPatterns)
GlobalPatterns
```

The following two lines actually do all the complicated DESeq2 work. The function phyloseq_to_deseq2 converts your phyloseq-format microbiome data into a DESeqDataSetwith dispersions estimated, using the experimental design formula, also shown (the ~SampleType). The DESeq function does the rest of the testing, in this case with default testing framework, but you can actually use alternatives.

```
#Convert file from Phyloseq to DESeq2
diagdds = phyloseq_to_deseq2(GlobalPatterns, ~ SampleType)

# calculate geometric means prior to estimate size factors
gm_mean = function(x, na.rm=TRUE){ exp(sum(log(x[x > 0]), na.rm=na.rm) / length(x)) }
geoMeans = apply(counts(diagdds), 1, gm_mean)
diagdds = estimateSizeFactors(diagdds, geoMeans = geoMeans)
diagdds = DESeq(diagdds, fitType="local")
```

Note: The default multiple-inference correction is Benjamini-Hochberg, and occurs within the DESeq function.

Investigate test results table

The following results function call creates a table of the results of the tests. Very fast. The hard work was already stored with the rest of the DESeq2-related data in our latest version of the diagdds object (see above). I then order by the adjusted p-value, removing the entries with an NA value. The rest of this example is just formatting the results table with taxonomic information for nice(ish) display in the HTML output.

```
res = results(diagdds)
res = res[order(res$padj, na.last=NA), ]
alpha = 0.01
sigtab = res[(res$padj < alpha), ]
sigtab = cbind(as(sigtab, "data.frame"), as(tax_table(GlobalPatterns)[rownames(sigtab), ],
"matrix"))
head(sigtab)
```

Let's look at just the OTUs that were significantly enriched in the carcinoma tissue. First, cleaning up the table a little for legibility.

```
posigtab = sigtab[sigtab[, "log2FoldChange"] > 0, ]
posigtab = posigtab[, c("baseMean", "log2FoldChange", "lfcSE", "padj", "Phylum", "Class",
"Family", "Genus")]
```

Plot Results

Here is a bar plot showing the log2-fold-change, showing Genus and Phylum. Uses some ggplot2 commands.

```
library("ggplot2")
theme_set(theme_bw())
sigtabgen = subset(sigtab, !is.na(Genus))

# Phylum order
x = tapply(sigtabgen$log2FoldChange, sigtabgen$Phylum, function(x) max(x))
x = sort(x, TRUE)
sigtabgen$Phylum = factor(as.character(sigtabgen$Phylum), levels=names(x))

# Genus order
x = tapply(sigtabgen$log2FoldChange, sigtabgen$Genus, function(x) max(x))
x = sort(x, TRUE)
sigtabgen$Genus = factor(as.character(sigtabgen$Genus), levels=names(x)) ggplot(sigtabgen,
aes(y=Genus, x=log2FoldChange, color=Phylum)) + geom_vline(xintercept = 0.0, color =
"gray", size = 0.5) + geom_point(size=6) + theme(axis.text.x = element_text(angle = -90, hjust =
0, vjust=0.5)
```

The phyloseq Extensions Index
Make sure to visit the original extension for edgeR and DESeq2 extensions for phyloseq
http://joey711.github.io/phyloseq-extensions/extensions-index.html

PERMANOVA Example

Description

Analysis of variance using distance matrices — for partitioning distance matrices among sources of variation and fitting linear models (e.g., factors, polynomial regression) to distance matrices; uses a permutation test with pseudo-F ratios.

https://rdrr.io/rforge/vegan/man/vegan-package.html
https://rdrr.io/rforge/vegan/man/adonis.html
https://microbiome.github.io/microbiome/PERMANOVA.html

Permutational Multivariate Analysis of Variance Using Distance Matrices

adonis is a function for the analysis and partitioning sums of squares using semimetric and metric distance matrices. Insofar as it partitions sums of squares of a multivariate data set, it is directly analogous to MANOVA (multivariate analysis of variance). M.J. Anderson (McArdle and Anderson 2001, Anderson 2001) refers to the method as "permutational manova" (formerly "nonparametric manova"). Further, as its inputs are linear predictors, and a response matrix of an arbitrary number of columns (2 to millions), it is a robust alternative to both parametric MANOVA and to ordination methods for describing how variation is attributed to different experimental treatments or uncontrolled covariates. It is also analogous to redundancy analysis (Legendre and Anderson 1999).

Typical uses of adonis include analysis of ecological community data (samples X species matrices) or genetic data where we might have a limited number of samples of individuals and thousands or millions of columns of gene expression data (e.g. Zapala and Schork 2006).

adonis is an alternative to AMOVA (nested analysis of molecular variance, Excoffier, Smouse, and Quattro, 1992; amova in the ade4 package) for both crossed and nested factors.

If the experimental design has nestedness, then use strata to test hypotheses. For instance, imagine we are testing the whether a plant community is influenced by nitrate amendments, and we have two replicate plots at each of two levels of nitrate (0, 10 ppm). We have replicated the experiment in three fields with (perhaps) different average productivity. In this design, we would need to specify strata = field so that randomizations occur only *within each field* and not across all fields . See example below.

Like AMOVA (Excoffier et al. 1992), adonis relies on a long-understood phenomenon that allows one to partition sums of squared deviations from a centroid in two different ways (McArdle and Anderson 2001). The most widely recognized method, used, e.g., for ANOVA and MANOVA, is to first identify the relevant centroids and then to calculated the squared deviations from these points. For a centered *n x p* response matrix *Y*, this method uses the *p x p* inner product matrix *Y'Y*. The less appreciated method is to use the *n x n* outer product matrix *YY'*. Both AMOVA and adonis use this latter method. This allows the use of any semimetric (e.g. Bray-Curtis, aka Steinhaus, Czekanowski, and Sørensen) or metric (e.g. Euclidean) distance matrix (McArdle and Anderson 2001). Using Euclidean distances with the second method results in the same analysis as the first method.

Significance tests are done using *F*-tests based on sequential sums of squares from permutations of the raw data, and not permutations of residuals. Permutations of the raw data may have better small sample characteristics. Further, the precise meaning of hypothesis tests will depend upon precisely what is permuted. The strata argument keeps groups intact for a particular hypothesis test where one does not want to permute the data among particular groups. For instance, strata = B causes permutations among levels of A but retains data within levels of B (no permutation among levels of B).

The default contrasts are different than in **R** in general. Specifically, they use "sum" contrasts, sometimes known as "ANOVA" contrasts. See a useful text (e.g. Crawley, 2002) for a transparent introduction to linear model contrasts. This choice of contrasts is simply a personal pedagogical preference. The particular contrasts can be set to any contrasts specified in **R**, including Helmert and treatment contrasts.

Value

This function returns typical, but limited, output for analysis of variance (general linear models).

aov.tab Typical AOV table showing sources of variation, degrees of freedom, sequential sums of squares, mean squares, *F* statistics, partial R-squared and *P* values, based on *N* permutations.

coefficients matrix of coefficients of the linear model, with rows representing sources of variation and columns representing species; each column represents a fit of a species abundance to the linear model. These are what you get when you fit one species to your predictors. These are NOT available if you supply the distance matrix in the formula, rather than the site x species matrix

coef.sites matrix of coefficients of the linear model, with rows representing sources of variation and columns representing sites; each column represents a fit of a sites distances (from all other sites) to the linear model.These are what you get when you fit distances of one site to your predictors.

f.perms an *N* by *m* matrix of the null *F* statistics for each source of variation based on *N* permutations of the data.

model.matrix The model.matrix for the right hand side of the formula.

terms The terms component of the model.

In order to test the PERMANOVA technique, you can use the phyloseq package as vegan is part of phyloseq.

Example Phyloseq Data (These are already converted to phyloseq format)**

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("phyloseq")
```

```
library("phyloseq")
data(GlobalPatterns)
sample_variables(GlobalPatterns)
head(sample_data(GlobalPatterns))
```

```
library("vegan")
```

```
data(dune)
data(dune.env)
head(dune)
names(dune)
sample_variables(dune.env)
adonis(dune ~ Management*A1, data=dune.env, permutations=99)
```

*** Using the ading the microbiome R package to use the dune dataset for the PERMANOVA Adonis function example in the vegan pacjage

https://microbiome.github.io/microbiome/

We will be using a new microbiome package so that we can use the subset function to separate the otu table and map item data from a phyloseq class object to use in the Adonis function

Installing microbiome R package

https://microbiome.github.io/microbiome/Installation.html

Open R and install the package. If the installation fails, ensure from the RStudio tools panel that you have access to the Bioconductor repository.

```
library(BiocInstaller)

source("http://www.bioconductor.org/biocLite.R")

useDevel()

biocLite("microbiome")

library(microbiome)
```

Loading and subsetting the phyloseq item

https://microbiome.github.io/microbiome/Comparisons.html
https://microbiome.github.io/microbiome/PERMANOVA.html

Multivariate comparisons

For community-level multivariate comparisons

Example Phyloseq Data (**These are already converted to phyloseq format)

There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets.

To load example data into the working environment, use the data() command:

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("phyloseq")
library("phyloseq")
data(GlobalPatterns)
sample_variables(GlobalPatterns)
head(sample_data(GlobalPatterns))
```

```
data(GlobalPatterns)

GlobalPatterns.rel <- microbiome::transform(GlobalPatterns, "compositional")


otu <- abundances(GlobalPatterns.rel)

meta <- meta(GlobalPatterns.rel)
```

PERMANOVA significance test for group-level differences

Now let us evaluate whether the group (probiotics vs. placebo) has a significant effect on overall gut microbiota composition. Perform PERMANOVA:

```
# samples x species as input

library(vegan)

permanova <- adonis(t(otu) ~ SampleType,

data = meta, permutations=99, method = "bray")

# P-value

print(as.data.frame(permanova$aov.tab)["group", "Pr(>F)"])
```

Checking the homogeneity condition

Check that variance homogeneity assumptions hold (to ensure the reliability of the results):

```
# Note the assumption of similar multivariate spread among the groups # ie. analogous to
variance homogeneity # Here the groups have signif. different spreads and # permanova result
may be potentially explained by that.

dist <- vegdist(t(otu))

anova(betadisper(dist, meta$SampleType))
```

Investigate the top factors

Show coefficients for the top taxa separating the groups

```
coef <- coefficients(permanova)["group1",]

top.coef <- coef[rev(order(abs(coef)))[1:20]]

par(mar = c(3, 14, 2, 1))

barplot(sort(top.coef), horiz = T, las = 1, main = "Top taxa")
```

Introduction to the microbiome R package

https://microbiome.github.io/microbiome/

Tools for microbiome analysis; with multiple example data sets from published studies; extending the phyloseq class. The package is in Bioconductor and aims to provide a comprehensive collection of tools and tutorials, with a particular focus on amplicon sequencing data.

Installing microbiome R package

https://microbiome.github.io/microbiome/Installation.html

Open R and install the package. If the installation fails, ensure from the RStudio tools panel that you have access to the Bioconductor repository.

```
library(BiocInstaller)

source("http://www.bioconductor.org/biocLite.R")

useDevel()

biocLite("microbiome")

library(microbiome)
```

Microbiome data processing

https://microbiome.github.io/microbiome/Preprocessing.html

Processing phyloseq objects

Instructions to manipulate microbiome data sets using tools from the phyloseq package and some extensions from the microbiome package, including subsetting, aggregating and filtering.

***Load example data: I will be using example data from the phyloseq package rather than the ones in the original tutorials

Example Phyloseq Data (These are already converted to phyloseq format)**

There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets.

```
data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
```

To load example data into the working environment, use the data() command:

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("phyloseq")
library("phyloseq")
```

```
library(phyloseq)

library(microbiome)

library(knitr)

data(atlas1006)
```

*** Using Global patterns instead)

```
data(GlobalPatterns)
sample_variables(GlobalPatterns)
head(sample_data(GlobalPatterns))
```

```
# Rename the example data (which is a phyloseq object)

pseq <- GlobalPatterns
```

Summarizing the contents of a phyloseq object

```
summarize_phyloseq(pseq)
```

Retrieving data elements from phyloseq object

A phyloseq object contains OTU table (taxa abundances), sample metadata, taxonomy table (mapping between OTUs and higher-level taxonomic classifications), and phylogenetic tree (relations between the taxa). Some of these are optional.

Pick metadata as data.frame:

```
meta <- meta(pseq)
```

Taxonomy table:

```
taxonomy <- tax_table(pseq)
```

Abundances for taxonomic groups (‘OTU table’) as a TaxaxSamples matrix:

```
# Absolute abundances

otu.absolute <- abundances(pseq)

# Relative abundances

otu.relative <- abundances(pseq, "compositional")
```

Melting phyloseq data for easier plotting:

```
df <- psmelt(pseq)

kable(head(df))
```

Sample operations

Sample names and variables

```
head(sample_names(pseq))
```

Total OTU abundance in each sample

```
s <- sample_sums(pseq)
```

Abundance of a given species in each sample

```
head(tax_table(pseq))

head(abundances(pseq)["Akkermansia",])
```

***** Okay for this particular case, your taxonomy column under head(tax_table(pseq)) needs to be properly labeled. In the tutorial, that was the case for the atlas1006 example dataset, whereas in Globalpatterns you have to use the taxonomy code

```
pseq2 <-atlas1006
head(abundances(pseq2)["Akkermansia",])
head(tax_table(pseq2))
head(tax_table(pseq))
```

head(abundances(pseq)["549322",])

Select a subset by metadata fields:

```
pseq.subset <- subset_samples(pseq, nationality == "AFR")
```

head(sample_data(pseq))

```
pseq.subset <- subset_samples(pseq, SampleType == "Soil")
```

Select a subset by providing sample names:

```
# Check sample names for African Females in this phyloseq object

s <- rownames(subset(meta(pseq), nationality == "AFR" & sex == "Female"))

# Pick the phyloseq subset with these sample names

pseq.subset2 <- prune_samples(s, pseq)
```

head(sample_data(pseq))

```
s <- rownames(subset(meta(pseq), SampleType == "Soil"))

pseq.subset2 <- prune_samples(s, pseq)
```

Pick samples at the baseline time points only:

```
pseq0 <- baseline(pseq)
```

*** This would not work in the Globalpatterns dataset because there is no time, sample, subject in our phyloseq sample_data object

Data transformations

The microbiome package provides a wrapper for standard sample/OTU transforms. For arbitrary transforms, use the transform_sample_counts function in the phyloseq package. Log10 transform is log(1+x) if the data contains zeroes. Also “Z”, “clr”, “hellinger”, and “shift” are available as common transformations. Relative abundances (note that the input data needs to be in absolute scale, not logarithmic!):

```
pseq.compositional <- microbiome::transform(pseq, "compositional")
data(dietswap)
x <- dietswap
# Compositional data
x2 <- transform(x, "compositional")
```

Variable operations

Sample variable names

```
sample_variables(pseq)
```

Pick values for a given variable

```
head(get_variable(pseq, sample_variables(pseq)[1]))
```

Assign new fields to metadata

```
# Calculate diversity for samples
div <- alpha(pseq, index = "shannon")
# Assign the estimated diversity to sample metadata
sample_data(pseq)$diversity <- div
```

Taxa operations

Number of taxa

```
n <- ntaxa(pseq)
```

Most abundant taxa

```
topx <- top_taxa(pseq, n = 10)
```

Names

```
ranks <- rank_names(pseq)
# Taxonomic levels
taxa  <- taxa(pseq)
# Taxa names at the analysed level
```

Subset taxa

```
pseq.bac <- subset_taxa(pseq, Phylum == "Bacteroidetes")
head(tax_table(pseq))
pseq.cre <- subset_taxa(pseq, Phylum == " Crenarchaeota")
```

Prune (select) taxa:

```
# List of Genera in the Bacteroideted Phylum
summary(tax_table(pseq))
taxa <- map_levels(NULL, "Phylum", "Genus", pseq)$Bacteroidetes
# With given taxon names
ex2 <- prune_taxa(taxa, pseq)
# Taxa with positive sum across samples
ex3 <- prune_taxa(taxa_sums(pseq) > 0, pseq)
```

Filter by user-specified function values (here variance):

```
f <- filter_taxa(pseq, function(x) var(x) > 1e-05, TRUE)
```

List unique phylum-level groups:

```
head(get_taxa_unique(pseq, "Phylum"))
```

Pick the taxa abundances for a given sample:

```
samplename <- sample_names(pseq)[[1]]  # Pick abundances for a particular taxon
tax.abundances <- abundances(pseq)[, samplename]
```

Merging operations

Aggregate taxa to higher taxonomic levels. This is particularly useful if the phylogenetic tree is missing. When it is available, see merge_samples, merge_taxa and tax_glom).

Put the less abundant taxa together in the “Other” category:

```
pseq2 <- aggregate_taxa(pseq, "Phylum", top = 5)
```

Merge the desired taxa into “Other” category. Here, we merge all Bacteroides groups into a single group named Bacteroides.

```
pseq2 <- merge_taxa2(pseq, pattern = "^Bacteroides", name = "Bacteroides")
```

Merging phyloseq objects with the phyloseq package:

```
merge_phyloseq(pseqA, pseqB)
```

Rarification

```
pseq.rarified <- rarefy_even_depth(pseq)
```

Taxonomy

Convert between taxonomic levels (here from Genus (Akkermansia) to Phylum (Verrucomicrobia):

```
summary(tax_table(pseq))

m <- map_levels("Akkermansia", "Genus", "Phylum", tax_table(pseq))
```

```
print(m)
```

Metadata

Visualize frequencies of given factor (sex) levels within the indicated groups (group):

```
sample_variables(pseq)

p <- plot_frequencies(sample_data(pseq), "bmi_group", "sex")

print(p)

sample_variables(pseq)

p <- plot_frequencies(sample_data(pseq), "SampleType")
```

*** This is a custom function where we need both a Group variable (like BMI) and a factor variable (like sex) but in the GlobalPatterns Data that I chose to use there was no other variable I could use besides Sampletype

Add metadata to a phyloseq object. For reproducibility, we just use the existing metadata in this example, but this can be replaced by another data.frame (samples x fields).

```
# Example data

data(dietswap)

pseq <- dietswap

# Pick the existing metadata from a phyloseq object

# (or retrieve this from another source)

df <- meta(pseq)

# Merge the metadata back in the phyloseq object

pseq2 <- merge_phyloseq(pseq, sample_data(df))
```

Alpha Diversity

Global Ecosystem State Variables

https://microbiome.github.io/microbiome/Diversity.html

***Load example data: I will be using example data from the phyloseq package rather than the ones in the original tutorials

Example Phyloseq Data (These are already converted to phyloseq format)**

There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets.

```
data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
```

Load example data

```
source("http://www.bioconductor.org/biocLite.R")

biocLite("microbiome")

library(microbiome)
```

```
biocLite("phyloseq")
library("phyloseq")
data(GlobalPatterns)
```

```
pseq <- GlobalPatterns
```

Richness

This returns observed richness with given detection threshold(s).

```
tab <- richness(pseq)

kable(head(tab))
```

Dominance

The dominance index refers to the abundance of the most abundant species. Various dominance indices are available (see the function help for a list of options).

```
# Absolute abundances for the single most abundant taxa in each sample

tab <- dominance(pseq, index = "all")

kable(head(tab))
```

We also have a function to list the dominating (most abundant) taxa in each sample.

```
dominant(pseq)
```

Rarity and low abundance

The rarity indices quantify the concentration of rare or low abundance taxa. Various rarity indices are available (see the function help for a list of options).

```
tab <- rarity(pseq, index = "all")

kable(head(tab))
```

Coverage

The coverage index gives the number of groups needed to have a given proportion of the ecosystem occupied (by default 0.5 ie 50%).

```
tab <- coverage(pseq, threshold = 0.5)

kable(head(tab))
```

Core abundance

The core_abundance function refers to the relative proportion of the core species. Non-core abundance provides the complement (1-x; see noncore_abundance).

```
tab <- core_abundance(pseq, detection = .1/100, prevalence = 50/100)
```

Gini index

Gini index is a common measure for inequality in economical income. The inverse gini index (1/x) can also be used as a community diversity measure.

```
tab <- inequality(pseq)
```

Evenness

Various evenness measures are also available.

```
tab <- evenness(pseq, "all")

kable(head(tab))
```

Community comparisons

Group-wise comparisons

https://microbiome.github.io/microbiome/Comparisons.html

Univariate comparisons

For individual taxa, diversity indicators etc.

For individual taxa, diversity indicators etc.

- Linear mixed effect models
- Negative binomial test

Mixed models for univariate comparisons

https://microbiome.github.io/microbiome/Mixedmodels.html

Load example data

***Load example data: I will be using example data from the phyloseq package rather than the ones in the original tutorials

Example Phyloseq Data (These are already converted to phyloseq format)**
There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets.

```
data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
```

Load example data

```
source("http://www.bioconductor.org/biocLite.R")

biocLite("microbiome")
```

```
library(microbiome)
```

```
biocLite("phyloseq")
library("phyloseq")
data(GlobalPatterns)
```

```
pseq <- GlobalPatterns
```

Linear model comparison with random effect subject term

Test individual taxonomic group

```
# Get sample metadata

dfs <- meta(pseq)

# Add abundance as the signal to model

dfs$signal <- abundances(pseq)["Akkermansia", rownames(dfs)]

sample_variables(pseq)

head(tax_table(pseq))

head(sample_data(pseq))

dfs$signal <- abundances(pseq)["951", rownames(dfs)]
```

***** Okay for this particular case, your taxonomy column under head(tax_table(pseq)) needs to be properly labeled. In the tutorial, that was the case for the atlas1006 example dataset, whereas in Globalpatterns you have to use the taxonomy code

```
# Paired comparison # with fixed group effect and random subject effect library(lme4)

out <- lmer(signal ~ group + (1|subject), data = dfs)

out0 <- lmer(signal ~ (1|subject), data = dfs)

comp <- anova(out0, out)

pv <- comp[["Pr(>Chisq)"]][[2]]

print(pv)

# Paired comparison # with fixed group effect and random subject effect
```

```
install.packages("lme4",    repos=c("http://lme4.r-forge.r-project.org/repos",
getOption("repos")[["CRAN"]]))
```

```
library(lme4)

head(sample_data(pseq))

out <- lmer(signal ~ SampleType + (1|X.SampleID), data = dfs)

out0 <- lmer(signal ~ (1|subject), data = dfs)

comp <- anova(out0, out)

pv <- comp[["Pr(>Chisq)"]][[2]]

print(pv)

*** Error: number of levels of each grouping factor must be < number of observations (Since
you are running a mixed effect model you need to have repeating observation so the
Globalpatterns datases is not suitable for this)

we will use the atlas1006 dataset instead

data(atlas1006)

pseq.2 <-atlas1006

head(sample_data(pseq.2))

# Get sample metadata

dfs <- meta(pseq.2)

# Add abundance as the signal to model

dfs$signal <- abundances(pseq)["951", rownames(dfs)]

sample_variables(pseq.2)

head(tax_table(pseq.2))

head(sample_data(pseq.2))

**Remember that the taxonomy comes from the taxonomy column in tax_table, not the actual
genus name you see under genus so it may be a code like in Globalpatterns!

dfs$signal <- abundances(pseq.2)["Akkermansia", rownames(dfs)]

# Paired comparison # with fixed group effect and random subject effect
```

```
install.packages("lme4",    repos=c("http://lme4.r-forge.r-project.org/repos",
getOption("repos")[["CRAN"]]))
```

```
library(lme4)
```

```
head(sample_data(pseq.2))

out <- lmer(signal ~ gender + (1|subject), data = dfs)

out0 <- lmer(signal ~ (1|subject), data = dfs)

*** The gender variable had some missing observation so we needed to subset it
```

Select a subset by providing sample names:

```
s <- rownames(subset(meta(pseq), nationality == "AFR" & sex == "Female"))

# Pick the phyloseq subset with these sample names

pseq.subset2 <- prune_samples(s, pseq)

comp <- anova(out0, out)

pv <- comp[["Pr(>Chisq)"]][[2]]

print(pv)
```

Group-wise comparisons with negative binomial

https://microbiome.github.io/microbiome/Negativebinomial.html
***Load example data: I will be using example data from the phyloseq package rather than the ones in the original tutorials

Example Phyloseq Data (These are already converted to phyloseq format)**
There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with "data-" to see the documentation of currently available example datasets.
data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
Load example data

```
source("http://www.bioconductor.org/biocLite.R")

biocLite("microbiome")

library(microbiome)
```

```
biocLite("phyloseq")
library("phyloseq")
data(GlobalPatterns)
```

```
pseq <- GlobalPatterns
```

Test statistical significance with negative binomial:

```
library(MASS)

sample_variables(pseq)

head(tax_table(pseq))

head(sample_data(pseq))

# Analyse specific taxa

tax <- "Akkermansia"

**We need to pick a code under the taxonomy column that corresponds to the taxa name under the genus we want

head(tax_table(pseq))

summary(tax_table(pseq))

tax <- "951"

# Pick the signal (abundance) for this tax

sample_data(pseq)$signal <- get_sample(pseq, tax)

# Negative binomial test with group and gender included

res <- glm.nb(signal ~ group + gender, data = meta(pseq))

# Negative binomial test with group = Sampletype

head(sample_data(pseq))

res <- glm.nb(signal ~ SampleType, data = meta(pseq))

# Show the results

print(coef(summary(res)))
```

Community comparisons

Group-wise comparisons

https://microbiome.github.io/microbiome/Comparisons.html

Multivariate comparisons

For community-level multivariate comparisons

- PERMANOVA

https://microbiome.github.io/microbiome/PERMANOVA.html

We will be using a new microbiome package so that we can use the subset function to separate the otu table and map item data from a phyloseq class object to use in the Adonis function

Installing microbiome R package

https://microbiome.github.io/microbiome/Installation.html

Open R and install the package. If the installation fails, ensure from the RStudio tools panel that you have access to the Bioconductor repository.

```
library(BiocInstaller)

source("http://www.bioconductor.org/biocLite.R")

useDevel()

biocLite("microbiome")

library(microbiome)
```

Loading and subsetting the phyloseq item

https://microbiome.github.io/microbiome/Comparisons.html
https://microbiome.github.io/microbiome/PERMANOVA.html

Multivariate comparisons

For community-level multivariate comparisons

Example Phyloseq Data (These are already converted to phyloseq format)**
There are multiple example data sets included in phyloseq. Many are from published investigations and include documentation with a summary and references, as well as some

example code representing some aspect of analysis available in phyloseq. In the package index, go to the names beginning with “data-” to see the documentation of currently available example datasets.

```
data(GlobalPatterns)
data(esophagus)
data(enterotype)
data(soilrep)
```

To load example data into the working environment, use the data() command:

```
source("http://bioconductor.org/biocLite.R")
biocLite ()
biocLite("phyloseq")
library("phyloseq")
data(GlobalPatterns)
sample_variables(GlobalPatterns)
head(sample_data(GlobalPatterns))
```

```
data(GlobalPatterns)

GlobalPatterns.rel <- microbiome::transform(GlobalPatterns, "compositional")

otu <- abundances(GlobalPatterns.rel)

meta <- meta(GlobalPatterns.rel)
```

PERMANOVA significance test for group-level differences

Now let us evaluate whether the group (probiotics vs. placebo) has a significant effect on overall gut microbiota composition. Perform PERMANOVA:

```
# samples x species as input

library(vegan)

permanova <- adonis(t(otu) ~ SampleType,

data = meta, permutations=99, method = "bray")

# P-value

print(as.data.frame(permanova$aov.tab)["group", "Pr(>F)"])
```

Checking the homogeneity condition

Check that variance homogeneity assumptions hold (to ensure the reliability of the results):

```
# Note the assumption of similar multivariate spread among the groups # ie. analogous to
variance homogeneity # Here the groups have signif. different spreads and # permanova result
may be potentially explained by that.

dist <- vegdist(t(otu))

anova(betadisper(dist, meta$SampleType))
```

Investigate the top factors

Show coefficients for the top taxa separating the groups

```
coef <- coefficients(permanova)["group1",]

top.coef <- coef[rev(order(abs(coef)))[1:20]]

par(mar = c(3, 14, 2, 1))

barplot(sort(top.coef), horiz = T, las = 1, main = "Top taxa")
```

Curating Raw 16S Sequencing Data

- α diversity (Chao1, Shannon Index): A greater value indicates more community diversity
- β diversity (Bray Curtis, Jaccard Index for abundance and carriage measures, respectively): Bray-Curtis is a dissimilarity index, so we use that to determine the distance between each pair of samples. UniFrac is the other popular alternative. ANOSIM is the statistical test on the Bray-Curtis distances; it determines if there is a rank difference in similarity among samples grouped by some categorical variable.
- DESeq2: Differential Abundance analysis between two groups using raw sequence counts rather than losing sequencing data by first clustering sequences into OTUs, then rarefying the read number.

Analyzing curated 16S data

- Random effects model: Degree to which each microbiome marker is consistant across studies or models (using study as a random effect effect model).
- Random Forest model: Identify a composite marker of disease using nested regression models by first doing a ranked test for differential abundance, and then determining the best models tuned by area under the receiver operator characteristic curve (AUROC)
- Regression model: Look for treatment effects, or treatment by visit interaction. Alternatively, you can use it to address the variation between samples when assessing the treatment effect on microbes, by using participants as a Random Effect

Special analyses

PiCrust (https://twbattaglia.gitbooks.io/introduction-to-qiime/content/picrust.html)

PICRUSt is a bioinformatic tool developed to gain insight into the metagenomic function of the microbiome based on 16S rRNA amplicon data. By using know genomes and their genomic composition, PICRUSt inferres abundance of genes based on the abundance of OTU's. Using this method, you can attempt to infer microbial genomic potential without the need for costly Whole Metagenomic Shotgun Sequencing (WMS).

LefSe (https://twbattaglia.gitbooks.io/introduction-to-qiime/content/lefse.html)

LEfSe uses a table of relative abundances which also includes sample identifiers and group meta data. QIIME typically keeps abundance data and meta data in separate files, so to generate a file compatible with LEfSe, we need to prepare the data in a few steps.
The input table that is required for analysis must have:

- Sample Identifiers, which is normally #SampleID within the mapping file
- Class variable. This is the most important variable and will be used to compare the two or more groups in your sample meta data.
- Subclass variable. This variable is optional and is only needed if your question and data are setup for use with it. See the paper/link for more information.