**Gene-Level Analysis of Mouse Tissues for Transcriptomics**

**Kaylee Helfrich**

Smith Lab

8/19/19

**Introduction**: This protocol details how to start with FASTQ files from UNC-CH’s High Throughput Sequencing Facility (HTSF) and turn them into usable data, such as PCA plots, pathway analysis, GO analysis, and raw data for graphing purposes (as well as other analyses as needed). This document includes step-by-step instructions for moving through the process of quality control, aligning the sequences to a genome, sorting and visualizing files, counting alignments to genes, normalizing gene counts, producing p-values and log2fold changes, and beginning analysis of the results. This document includes lists of common UNIX commands as well as some example Bash scripts, but for the original Bash scripts, you should use the .txt or .sh files on the server, as the scripts in this document will have different formatting. This document does not include common R codes, as this is beyond the scope of the protocol. You can view many online tutorials for help learning R. This document includes many helpful hints that I have discovered during this process, but is by no means comprehensive in possibilities. You can also refer to the document “Gene Level Analysis for RNA-Seq Expression Data” by Abrar Al-Shaer for more help on some of these steps as well as a slightly different perspective.

**Recommended reference**: I recommend reading this paper (<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4728800/>) before beginning. It provides a nice description of some of the steps of RNAseq as well as providing best practices for analyzing the data. Although it is not comprehensive or exceedingly technical, it is a helpful read.

**Expected result**: The expected result is a file of raw counts for all genes (~24,000) for all inputted samples as well as a table of differentially expressed genes between 2 treatment groups. This document will also guide creation of graphical representations of the data.

**Github**: All of the code used in this analysis, and this manual, can be found on my GitHub page under the RNAseq folder: <https://github.com/kayleehelfrich/RNAseqData>

**Methodology and tools**:

1. **Longleaf and SLURM**

Turning fastq sequencing files into an end product requires multiple steps that are computationally heavy and will take a long time on a local machine. To shorten run times, I use UNC’s Longleaf server (see here for more information on Longleaf: <https://help.unc.edu/help/getting-started-on-longleaf/>). You will need an account on Longleaf to proceed. Once you have an account and want to connect to longleaf, open a terminal (ex. Bash or MobaXterm) and type “ssh onyen@longleaf.unc.edu” Then, navigate the system as you would a normal file system. There are a few things to note on Longleaf. The “/pine/scr/o/n/onyen” space is given to all Longleaf users (where “o” and “n” are the first two letters of your onyen). This is where you should run your scripts. However, these files are deleted every 21 days, so when you are done running your programs, you should move your files out of those files. (Note: depending on space, the files are sometimes left there for longer than 21 days. However, to be safe, make sure to move any important files to a more permanent location.) You can move them to the ms space on Longleaf (with limited storage) or to local storage in the Smith lab folder. As a lab, we also have a proj space (under ssmithlab). This is where HTSF puts our completed files. This has a limited amount of storage space and files are usually not deleted from here until space runs out. Therefore, you can store temporary files here, but not anything permanent.

When you log into Longleaf, you will be on one of two master access nodes on Longleaf (Longleaf 1 and Longleaf 2 are the head nodes). These two are where you run simple scripts, move files, etc. However, these nodes mostly function to transfer jobs to the rest of the cluster. You will use the scheduling system SLURM to submit your large jobs to other computers.

Longleaf uses the SLURM scheduling system to submit jobs. This prevents the computing cluster from getting bogged down and also makes sure that no one person hogs the system. You can submit jobs to SLURM one of two ways:

1. submit jobs with “sbatch” at the beginning, such as “sbatch –N 1 –t 1-0 --wrap “fastqc file”

2. submit jobs with #SBATCH commands in the script and then call the script with “sbatch”

You will see examples of both types of commands later in this manual. You can log out of Longleaf once you’ve submitted a job and the job will continue running. You can set up an option for SLURM to send you an e-mail when the job is finished running so that you don’t have to keep logging in and checking on your script. (You also can and should have it send you an e-mail if the job fails). You can submit multiple jobs at one time and they will all run simultaneously.

All scripts in the rest of this manual are submitted via SLURM (except for steps after featureCounts). For more information on SLURM, see <https://help.unc.edu/help/longleaf-slurm-examples/> and <https://help.unc.edu/help/longleaf-frequently-asked-questions-faqs/> and <https://its.unc.edu/rc-services/longleaf-cluster/>

1. **Longleaf modules**

As on your computer where you can install programs that will then run there (ex. Word, Javascript, or Ingenuity Pathway Analysis), there are also many programs that are installed on Longleaf. Unlike your computer where you can double-click a program to run it, you must first load the program (called “module”) onto your current window before you utilize that program. To see what modules are available, use the command “module avail.” To load a module, use “module load <program>”. To see your currently loaded modules, use “module list”. To remove a module, use “module unload <program>”. To remove all modules, use “module purge”. Other options are listed below under common commands.

For almost every module, there are multiple versions (i.e. releases). One of the versions of each program module will be designated with a “D”, indicating that this is the default. This is the module that loads when you type “module load <program>” if you don’t specify which version. This may not necessarily be the newest version. If you want to specify a certain version (for example, you want to keep the version the same across time to keep your analyses consistent), then make sure to explicitly choose the version that you want using “module load <program 1.3.2>”, replacing program with the program of interest and 1.3.2 with the version you want. See here for more information on modules: <https://help.unc.edu/help/modules-approach-to-software-management/>

1. **Common UNIX/Longleaf/SLURM commands**
   * space bar – to go forward
   * b – to go back
   * q – to quit
   * tab – to autocomplete the current command or file or directory name
   * Ctrl C – to cancel a command executed on the normal terminal
   * \* - stands for any character or no character (ex. \*.fastq indicates all files with that file extension)
   * ; - UNIX reads it as an enter, and it can be used to run multiple commands at once
   * nano - a text editor. Call it by typing “nano” to open it. If you want to open a file, use “nano <file>” (Ctrl 0 to save the new file)
   * man <program> gives information about or a manual for the program
   * info <program> gives information about the program
   * df <file> describe the file and tell you what type of file it is
   * <command> -h gives help and arguments for the command
   * echo <word> repeats that word or variable; prints it to the screen
   * ls lists the files within the current directory
   * ls /pine/scr/ lists the files within the listed directory
   * ll lists the files within the listed directory in a single line w/ more info
   * cd /pine/scr/ change directory to /pine/scr/
   * cd ../scr/ change directory by going up one file and back down into the scr directory
   * mkdir RNAseq make directory called “RNAseq”
   * cp RNAseq /pine/scr/ copy the file RNAseq into the /pine/scr/ directory
   * file nano.fq to figure out what type of file it is
   * cat nano.fq to see what is in the file
   * zcat nano.fq to see what is in a zipped file
   * more nano.fq to see the file in portions, using the space bar to go through the file
   * less nano.fq to see the file and be able to scroll through it; can be used on zipped files
   * tail -25 nano.fq show the last 25 lines of the file
   * head -25 nano.fq show the first 25 lines of the file
   * sbatch run a program or script on SLURM
   * module avail provides a list of all the modules available on Longleaf
   * module list lists currently loaded modules
   * module load <program> loads the default version of the program
   * module load fastqc/0.11.5 loads version 0.11.5 of FASTQC
   * module purge unloads all loaded modules
   * module unload fastqc unloads fastqc only
   * module swap bbmap/38/22 bbmap switches out the version of bbmap in use
   * cp –r copy a directory
   * sbatch send a command to SLURM
     + -p tells SLURM the lane for the program to run in (ex. general)
     + --mem=10g sets how much memory the job should use
     + -o shows where the standard out should be written
     + -e shows where the standard error should be written
     + --wrap shows where the command actually starts
     + squeue –u onyen shows whether the job is working
     + %j tells sbatch to write in the job ID
     + sinfo shows what is currently happening in the SLURM system
     + scancel <jobID> cancels a currently running job
   * exit quit the Longleaf server
   * | pass the results from one program directly to the next program
   * mv \*fastq backup/ moves files ending in “fastq” into the folder “backup”
   * du <file> to see the file size
   * date prints out current time and date
   * rm <file> to delete a file. Be careful, as there is no undo in UNIX
   * FASTQ=SRR1.fastq assigns the file “SRR1.fastq” to the variable FASTQ
   * grep search through a file
   * for file in $(ls -1 \*trim\*gz); do echo $file; done a basic for-loop
   * diff <file1> <file2> tells whether 2 files are different. If they are the same, nothing happens.
   * mv fileold.txt filenew.txt Renames a file
   * rm –r mydir to remove the directory “mydir”
   * dos2unix myscript.sh to convert a script written on a Windows computer to one that can run on Linux
   * chmod u+x myscript.sh to create an executable script
2. **Other necessary programs** 
   1. R and R Studio- <https://www.r-project.org/> <https://www.rstudio.com/products/rstudio/download/>
   2. MobaXTerm- <https://mobaxterm.mobatek.net/>
   3. FileZilla- <https://filezilla-project.org/>

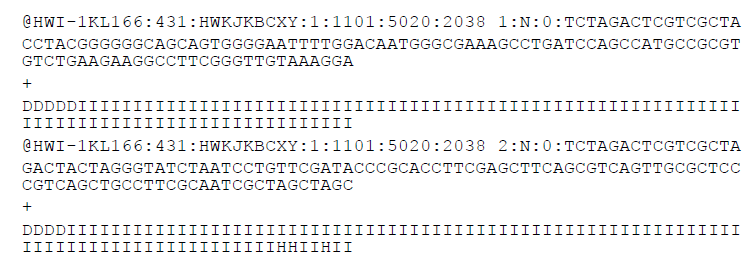
**Protocol from raw files to gene counts**:

\*\*For every step, make sure that all files needed for the script are in the same directory as the script. Otherwise, the script will not be able to find the files.

\*\*To move any files onto or off of the Longleaf cluster, it is easiest to use FileZilla. To do this, open FileZilla, type “longleaf.unc.edu” into the “Host” box, enter your onyen into the “Username” box, put your onyen password into the “Password” box, and type “22” into the “Port” box. Then, you can easily move files back and forth between Longleaf (on the right) and the Smith lab folders (on the left). Once the files are off of Longleaf, they can be easily viewed, opened, or manipulated in the same way that you can work with normal files.

1. **Raw fastq.gz files**

FASTA or FASTQ data is the most basic data that a researcher uses (researchers do not usually work with the raw Ilumina data). FASTQ data is FASTA data plus the quality scores for each base read. You can see an example of a single read in FASTQ format below. Most FASTQ files will have millions of reads, with the example shown below repeated millions of times, with a single set for each read. If you are working with paired-end data, then there will be two sets of reads for every DNA piece that was sequenced, and these 2 sets of reads will be in separate files (R1 and R2; see more information below).



Raw FASTQ files look similar to the picture directly above. The components of this file are as follows:

* The first line has information about the run. “@HWI-1KL166” is the serial number of the instrument, “431” is the run number, “HWKJKBCXY” is the serial number of the cell, “1” is the flow lane, “1101:5020:2038” are the xyz coordinates of the sequence on the flow cell, “1” indicates that it is the first read of a paired-end read, and “TCTAGACTCGTCGCTA” is the adapter sequence so that the DNA can be multiplexed and still separated out by sample.
* The second two lines are the sequence as read by Ilumina.
* The 2 lines after the “+” are the quality scores. See <https://en.wikipedia.org/wiki/FASTQ_format> for more information on how quality scores work.

Most runs performed in this lab are paired-end. This means that the data is read from two ends, which improves the quality of the data obtained. Thus, for every sample, there will be two files. HTSF names these files as Part1.R1.001.fastq.gz and Part1.R2.001.fastq.gz. These files will need to be processed together for all steps (except for FASTQC, where they are handled separately).

1. **FASTQC**

Before doing anything with FASTQ files, it is wise to look at the quality of the data provided by the sequencing facility. FASTQC is a great tool for summarizing data quality in an easy-to-digest manner. For more information about FASTQC, see here: <https://www.bioinformatics.babraham.ac.uk/projects/fastqc/> under “Documentation”. This will also show you examples of good and bad data.

##Copy an entire folder of FASTQ files into my scratch space

cp /proj/ssmithlab/HTSF/180906\_UNC31-K00269\_0151\_AHWNCYBBXX/ /pine/scr/k/h/khelfri//fastqc\_190213/

##Copy FASTQ files from the “E” group of files into my scratch space

cp /proj/ssmithlab/HTSF/180919\_UNC32-K00270\_0120\_AHWNC3BBXX/PREGB6-E\*.fastq.gz /pine/scr/k/h/khelfri/fastqc\_190213/

\*\*During this step, do not click anything else or exit out of the terminal or the copying will stop and you will have to start over.\*\*

#Load the fastqc program

module load fastqc

#Check which version of the fastqc program was used and record it for later reference

module list

#Get help and more info on the fastqc command

fastqc –h

#Run program on the local machine or on the original server (not recommended, especially for large jobs)

fastqc filename.gz

#Run program on the server for a single file

sbatch -p general --mem=10g --time=8:0:0 --wrap="fastqc PREGB6-B1\_ATTACTCG-CCTATCCT\_S33\_L005\_R1\_001.fastq.gz -o ../fastqc\_output/"

#Run multiple FASTQ files at the same time on the server. If I wanted to put the output into a folder that is at the same level as the files that are running, I could use "-o fastqc\_output/" instead of “../fastqc\_output/”

sbatch -p general --mem=10g --time=8:0:0 --wrap="fastqc \*.fastq.gz -o ../fastqc\_output/"

***Approx. run time***: If run on Longleaf exactly as the script is written, 1 file will require approximately 15min.

The output from this command will be 2 things: an HTML file and a folder with many files of the FASTQC raw data. To view the results, download the HTML files using FileZilla and then double click the HTML file to view it in an internet browser. All of the actual data is in the file, and this includes the FASTQC raw data which can be used to generate your own graphs if desired. When viewing the FASTQC output, use your own judgement, as warnings given by FASTQC are not correct depending on your specific data. The FASTQC file will include information such as the following (bolded ones are important to look at): **Basic statistics** (file name, file type, encoding, total number of sequences, sequences flagged as poor quality, sequence length, and GC content), **per base sequence quality**, per tile sequence quality, **per sequence quality scores,** per base sequence content, per base GC content, **per base N content**, sequence length distribution, sequence duplication levels, overrepresented sequences, and **adapter content**. When looking at the per base sequence quality, note that optimal quality scores will be between Q30 and Q40. Anything less than this says there may be something wrong with the data. The higher the Q score is, the more accurate the base calling.

If you are running a lot of files and want a summary of the overall quality of all your files, you can use MultiQC. (see more about this here: <https://multiqc.info/> There are some good examples of what the output will look like and what you can do with the information). MultiQC is already installed on Longleaf.

#Load multiqc

module load multiqc

#Check with version of multiqc you’re using and record this

module list

#Run multiqc via a SLURM script on FASTQC output in the current directory and put output into the current directory

sbatch -p general --time=8:0:0 --mem=10g --wrap="multiqc ."

***Approx. run time***: If run on Longleaf exactly as the script is written, less than 1 minute. You can probably run this directly on the node without submitting the job to SLURM if you want.

1. **Trimming (Trimmomatic)**

The goal of trimming is to rescue suboptimal data, not to make the data better. Essentially, it can turn un-useful reads into useful reads. Trimming is mainly used if there is adapter contamination in the sequences, which you will know from the FASTQC step. When trimming, you must include both the R1 and R2 files in the run. If the files are run separately and one read is deleted from R1 due to poor quality, then the read from R2 will not be deleted, which will lead to issues with data not being paired. After trimming, you can re-run FASTQC to see whether the trimming improved your data quality.

The data I worked with did not require trimming as it was good quality, so I did not write a guide for this section. If you need to trim the data, then see my notes from the NGS data analysis workshop as well as Abrar’s manual for guidance on this process.

1. **Alignment (Bowtie2)**

Once you know that the data is of sufficient quality, then the next step is to actually align the reads to a genome so that you know what your reads are. The alignment step will pair your reads to the genome that you indicate and output SAM files (Sequence Alignment Map) where the reads are sorted and matched to genes but are not in any type of order.

A few items of note:

* It is very important to use the exact same genome (ex. MM10 vs. MM9) for all steps, including alignment, samtools, etc. This is because different genome builds have placed genes in slightly different positions, and using different genome builds for the same dataset will result in useless results.
* Running a file twice may not produce the exact same results, especially when there are low reads or poor quality reads. This is because there is a degree of random assignment in these reads. Given this, be careful which version of the alignment is used, and try not to run this step multiple times.
* In Longleaf, many of the most common genomes are already built into indices, and can be viewed under /proj/seq/data/ If you want a new genome to be placed into this folder, contact HTSF to add the new one.
* The actual folder to be used for the alignment will be under a file path such as /proj/seq/data/MM10\_UCSC/Sequence/Bowtie2Index. Then, all of the files are labeled with “genome” and the only thing different is the file extension. To indicate the folder where all these files are, just use /proj/seq/data/MM10\_UCSC/Sequence/Bowtie2Index/genome, and Bowtie2 will know to use all of the files starting with the word “genome”.
* Unfortunately, alignment will never be perfect. According to HTSF, do not worry too much about the alignment percentage unless it is below 40%, which could indicate contamination in the sample.

**BOWTIE2 SCRIPT**: (if running this script, use the one in Notepad, as Word changes formatting)

#!/bin/bash

# Check for inputs

if [[ $# -eq 0 ]] ; then

echo 'Please enter the reference path'

exit 2

fi

# Print date of script start

date

echo "Running bowtie2"

# Assign Variables

refpath=$1

module load bowtie2

sleep 10

# Run Bowtie2

for file in $(ls -1 \*R1\*.fastq.gz); do

file\_input1=$file

file\_input2=$(echo $file\_input1 | sed -r 's/\_R1\_/\_R2\_/g')

file\_output=$(echo $file\_input1 | sed -r 's/\_.\*/.sam/g')

slurm\_output=$(echo $file\_input1 | sed -r 's/\_.\*/\_slurm/g')

# Run Bowtie 2

sbatch -p general -N 1 -n 8 --mem 32768 -t 4-0 -o $slurm\_output.out -e $slurm\_output.err --mail-type=END,FAIL --mail-user=khelfri@live.unc.edu --wrap="bowtie2 --non-deterministic --met-stderr $errorpath --sensitive -x $refpath -1 $file\_input1 -2 $file\_input2 -S $file\_output"

sleep 10

done

# Show date and that that its finished

date

echo "ending bowtie2"

Decoding the script:

* #!/bin/bash
  + makes it a bash script
* if [[ $# -eq 0 ]] ; then

echo 'Please enter the reference path'

exit 2

fi

* + an if-then statement to make sure that the reference path variable is entered for the script
* date

echo "Running bowtie2"

* + prints the date and time when script begins running and prints that it is now running the script
* refpath=$1
  + This creates the variable “refpath”. The variable is entered when the script is kicked off (ex. sbatch script.sh **/proj/seq/data/MM10\_UCSC/Sequence/Bowtie2Index/genome**) This indicates the genome to which the reads will be aligned.
* module load bowtie2

sleep 10

* + This loads the bowtie2 module so that the program can run. The sleep command makes sure that the module is loaded before the next step.
* for file in $(ls -1 \*R1\*.fastq.gz); do

file\_input1=$file

file\_input2=$(echo $file\_input1 | sed -r 's/\_R1\_/\_R2\_/g')

file\_output=$(echo $file\_input1 | sed -r 's/\_.\*/.sam/g')

slurm\_output=$(echo $file\_input1 | sed -r 's/\_.\*/\_slurm/g')

* + This for-loop searches for files in the current directory that end in R1 and fastq.gz, and it sets these equal to the variable “file\_input1”. It then sets the R2 file equal to “file\_input2”. It also defines what the output files should be called based on the input file.
* sbatch #submits the job to SLURM
  + -p general #Submits the job to the general partition of Longleaf
  + -N 1 #Requests 1 node
  + -n 8 #Requests 8 threads on the node
  + --mem 32768 #requests 32gigs of memory
  + -t 4-0 #tells it to kill the job if it is still running after 4 days
  + -o $slurm\_output.out #sets the name of the slurm output file
  + -e $slurm\_output.err #sets the name of the slurm error file
  + --mail-type=END,FAIL #sends the user and e-mail when the job finishes or fails
  + --mail-user=khelfri@live.unc.edu #sets the email for the user
  + --wrap= #sets actual command
* bowtie2 #calls bowtie2 program to run. Variables should be written in the following order:
  + --non-deterministic #sets it so any multi-mapping reads are assigned randomly
  + --met-stderr $errorpath #prints out the error from the program
  + --sensitive #sets how sensitive bowtie should be (see bowtie2 documentation)
  + -x $refpath #sets the refpath for the genome to which the reads should align
  + -1 $file\_input1 #sets the R1 file
  + -2 $file\_input2 #sets the R2 file
  + -S $file\_output #sets the output file name and says it should be in SAM format
* done #closes the loop
* date

echo "ending bowtie2"

* + Shows that the program is finished

\*\*Sometimes when writing scripts on a Windows computer, the script encoding gets altered. To fix this so that the files will run, use “**dos2unix alignment\_script\_2.sh**”.

\*\*To make the file executable, use “**chmod u+x alignment\_script\_2.sh**”.

**RUN THE SCRIPT**:

sbatch alignment\_script\_2.sh /proj/seq/data/MM10\_UCSC/Sequence/Bowtie2Index/genome

***Approx. run time***: If run exactly as the script is written, requires approximately 4-8hrs for 3-16 files.

The output of this step will be a SAM file, an “out” file and an “error” file for every single paired sample. (ex. There will be an output of 3 files for the R1 and R2 of sample E4). The SAM file contains the actual alignment (for more information on the contents of SAM files, see here: <https://samtools.github.io/hts-specs/SAMv1.pdf>). The out file in this case contains nothing relevant. The error file contains a readout of the alignment process. It is not terribly helpful, but the end of this file contains important information about the quality of the alignment. To see this information, type “tail -20 PREGB6-E1\_slurm.err” or something similar to see just the very end of the alignment file. This contains information such as what is seen below. For an explanation of what it means to pair concordantly or discordantly, see the Bowtie2 documentation: <http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml>

46088156 reads; of these: **# This is the total number of read fragments as generated by Ilumina**

46088156 (100.00%) were paired; of these: **#This should match the total #, and is how many fragments should have a pair.**

21428898 (46.50%) aligned concordantly 0 times **#How many fragments did not align concordantly**

17817760 (38.66%) aligned concordantly exactly 1 time **#How many fragments aligned concordantly once.**

6841498 (14.84%) aligned concordantly >1 times **#How many fragments aligned concordantly multiple times.**

----

21428898 pairs aligned concordantly 0 times; of these: **#Matches the number above of fragments not aligning concordantly.**

9055457 (42.26%) aligned discordantly 1 time **#How many of the pairs that didn’t align concordantly actually aligned discordantly.**

----

12373441 pairs aligned 0 times concordantly or discordantly; of these:

24746882 mates make up the pairs; of these:

12199866 (49.30%) aligned 0 times

9354721 (37.80%) aligned exactly 1 time

3192295 (12.90%) aligned >1 times

86.76% overall alignment rate **#How many overall reads were aligned to the genome (of the total 46088156). This does not consider concordance or discordance.**

The output from Bowtie2 should not be discouraging. There is a lot of interpretation, and even if the alignment is not high, it is okay to go onto the next step. However, if you have low read counts at featureCounts, then something might have gone wrong at this step.

1. **Conversion of SAM to BAM and IGV visualization**

The reads in the SAM file output from alignment are in random order and are also not mapped to the chromosomes in any particular order. So that the reads are in the correct order, especially for visualization using IGV, we will sort the files as well as convert them to BAM format which can be read in IGV. See the samtools documentation here: <http://www.htslib.org/doc/samtools.html> Documentation can be very confusing, so stick with simple commands.

The annotation file used to sort the reads in a SAM file is called the annotation file. This file is in the GTF style (gene transfer format). The file contains the following components for every feature: seqname, source, feature (gene name, for example), start (start position of the feature), end (end position of the feature), score, strand (forward or reverse), frame, attribute. Within this file, genes that have different numbers but the same name are actually exons from the same gene.

**SAMTOOLS SCRIPT**: (if running this script, use the one in Notepad, as Word changes formatting)

#!/bin/bash

#

#SBATCH -p general

#SBATCH -N 1

#SBATCH -n 8

#SBATCH --mem 32768

#SBATCH -t 2-0 # time (D-HH:MM)

#SBATCH -o bam\_sort.%N.%j.out

#SBATCH -e bam\_sort.%N.%j.err

#SBATCH --mail-type=END,FAIL

#SBATCH --mail-user=khelfri@live.unc.edu

# Print date of script start

date

echo "Running samtools"

module load samtools

sleep 10

for file in \*.sam

do

echo $file

describer=$(echo ${file} | sed 's/.sam//')

echo $describer

# Run samtools

samtools view -b $file > ${describer}.uns.bam

sleep 2

samtools sort ${describer}.uns.bam -o ${describer}.bam

sleep 2

samtools index ${describer}.bam

sleep 2

done

# Show date and that that its finished

date

echo "samtools finished"

\*\*Sometimes when writing scripts on a Windows computer, the script encoding gets altered. To fix this so that the files will run, use “**dos2unix alignment\_script\_2.sh**”.

\*\*To make the file executable, use “**chmod u+x alignment\_script\_2.sh**”.

Decoding the script:

* For any of the commands used in the alignment script, see above rather than me describing them again.
* #SBATCH -o bam\_sort.%N.%j.out #defines the out file for the entire script
* #SBATCH -e bam\_sort.%N.%j.err #defines the err file for the entire script
* for file in \*.sam

do

echo $file

describer=$(echo ${file} | sed 's/.sam//')

echo $describer

* + This for-loop sets of the names of the files to be used
* samtools view -b $file > ${describer}.uns.bam #This step converts SAM to BAM files and produces the file.uns.bam file, which is BAM but is not sorted or indexed.
* samtools sort ${describer}.uns.bam -o ${describer}.bam #This step sorts the BAM file and this is the file used for future steps.
* samtools index ${describer}.bam #This step indexes the BAM file and outputs the file.bai.

**RUN THE SCRIPT**:

sbatch SortSAM\_Script.sh

***Approx. run time***: If run exactly as the script is written, requires approximately 30min-1hour per file

The output of this step is 3 files: file.uns.bam, file.bam.bai, and file.bam. Although you should save all 3 files, only the file.bam will be used in future steps. The file.uns.bam is the SAM file converted to BAM before it has been sorted into any order. The file.bam.bai is the BAM file before indexing, and the file.bam is the final BAM file output.

To view your new BAM files with IGV, load the IGV module (module load igv) and then run “igv PREGB6-B1.bam PREGB6-B2.bam”, listing as many BAM files as you want to view at one time. This allows you to see visually where your reads are aligning and provides information about individual reads as they match up to the genome. I don’t recommend using this for anything other than a quick check, as it could easily sap a lot of time and is not quantitative.

1. **Alignment counts (featureCounts)**

This program works by counting the number of reads that fall within a specific exon and then summarizing the counts on a per-gene basis. The documentation for featureCounts is very helpful, and you should read the entire featureCounts chapter of the subread program (see here: <http://subread.sourceforge.net/> and here: <http://bioinf.wehi.edu.au/subread-package/SubreadUsersGuide.pdf>). Looking through this documentation will ensure that there are no more arguments you need to add to your code to get the output you wish. However, once you choose a parameter, make sure that you include it for all files run that way so that there is no question about them matching up.

This program requires an annotation file. You can find the annotation file on Longleaf. These files are at /proj/seq/data/. Then, you enter the genome build that you need and go to Annotation/Genes/genes.gtf where you will find the annotation file. Make sure that this build matches the one used for the original alignment or you will get very strange results.

**FEATURECOUNTS SCRIPT**: (use the script in Notepad, as Word changes formatting)

#!/bin/bash

#

#SBATCH -p general

#SBATCH -N 1

#SBATCH -n 8

#SBATCH --mem 32768

#SBATCH -t 1-0 # time (D-HH:MM)

#SBATCH -o featureCounts.%N.%j.out

#SBATCH -e featureCounts.%N.%j.err

#SBATCH --mail-type=END,FAIL

#SBATCH --mail-user=khelfri@live.unc.edu

module load subread

sleep 8

Bam1=PREGB6-A1.bam

Bam2=PREGB6-A2.bam

Bam3=PREGB6-A3.bam

Bam4=PREGB6-A4.bam

Bam5=PREGB6-A5.bam

Bam6=PREGB6-A6.bam

Bam7=PREGB6-A7.bam

Bam8=PREGB6-A8.bam

Bam9=PREGB6-B1.bam

Bam10=PREGB6-B2.bam

Bam11=PREGB6-B4.bam

Bam12=PREGB6-B5.bam

Bam13=PREGB6-B6.bam

Bam14=PREGB6-B7.bam

Bam15=PREGB6-B8.bam

Bam16=PREGB6-B9.bam

gtf=/proj/seq/data/MM10\_UCSC/Annotation/Genes/genes.gtf

# Print date of script start

date

echo "Running featureCounts"

featureCounts -p -B -a $gtf -o MatMLiver\_MdvsEtOH.counts -T 8 -g gene\_id $Bam1 $Bam2 $Bam3 $Bam4 $Bam5 $Bam6 $Bam7 $Bam8 $Bam9 $Bam10 $Bam11 $Bam12 $Bam13 $Bam14 $Bam15 $Bam16

# Make sure that the files are in the order you will want the final columns in the table

# Show date and that that its finished

date

echo "featureCounts finished"

\*\*Sometimes when writing scripts on a Windows computer, the script encoding gets altered. To fix this so that the files will run, use “**dos2unix alignment\_script\_2.sh**”.

\*\*To make the file executable, use “**chmod u+x alignment\_script\_2.sh**”.

Decoding the script:

In this script, everything is similar to the previous script for the sbatch commands. The “Bam1=PREGB6-A1.bam” is setting the file equal to the variable so that it is easier to call the files later on during the script. Setting “gtf” equal to the path to get to the indexing file shortens the command later on.

featureCounts:

* -p indicates that the reads are paired end
* -B makes the program more stringent by saying that only fragments that have both ends successfully aligned are considered for summarization. Although this lowers the overall counts, it is a more conservative setting.
* -a sets the name of the annotation file
* -o specifies the name of the output file
* -T sets the number of threads that the program should use; this number should match the number of threads that you set for SBATCH.
* -g sets that the output should be indexed by the gene ID
* The list of variables is the input into featureCounts and defines the files that will be counted. These files should be provided in the order that you want them listed in the final matrix.

**RUN THE SCRIPT**:

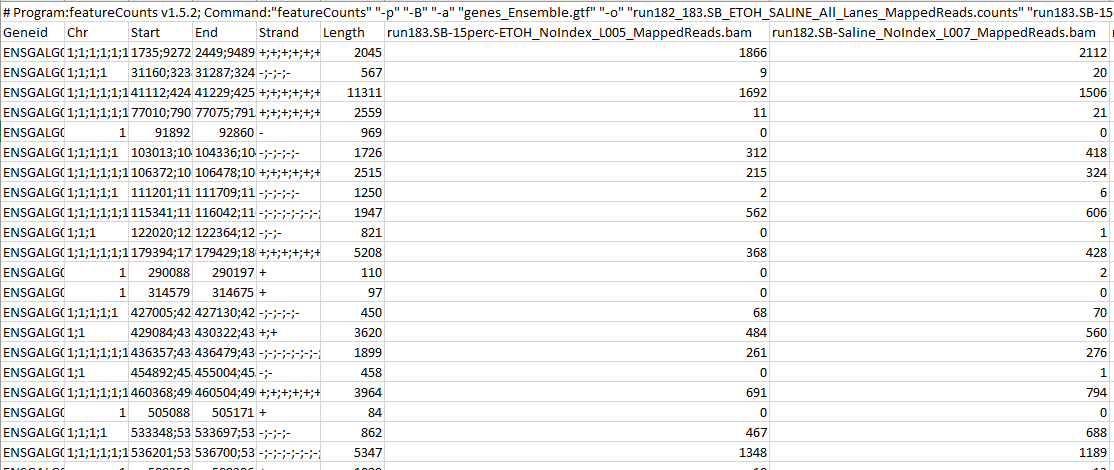
sbatch FeatureCounts\_Script.sh

***Approx. Run time***: If run exactly as the script is written, running 16 files took approximately 20 minutes.

The output of this program is a matrix where the genes are in rows and the samples are in the columns. At this point, the files are small enough that we will switch to running them on a local machine. To get the output matrix into a usable format, open the “counts” file with a text editor. Then, select all and paste into a CSV. Save as a CSV, and use as a spreadsheet from here on.

The other output of this program is a “counts.summary” file. This is important as it tells you how many reads were counted, and why certain reads were not counted (due to parameters you entered to make the counting more stringent or due to the program itself knowing that those reads were not optimal). This file should be kept as reference, but it will not be processed moving forward.

FeatureCounts will output a .counts file that looks like this:



At this point, you may be tempted to look at your raw data and start performing analyses for any genes that interest you. However, this will not yield valid results, because the genes have not yet been normalized to the entire gene list (must be done to account for gene size). This step will be performed next in DESeq2. The overall pattern of change seen between control and treatment groups will stay the same, but the degree of change may differ.

1. **Differential expression, quality control, normalization, and log2 fold change adjustment (DESeq2)**

*Getting started*

DESeq2 is a statistical package in R that is used to analyze RNA-Seq data. Along with a few other packages, it executes a few functions on the transcriptome data to normalize and scale the data, thus making it comparable across all of the genes. DESeq2 uses the negative binomial distribution to fit the statistical models in the transcriptome data since the data has variance larger than the mean. Other statistical tests, such as those based on a Poisson distribution, assume that the mean is greater than the variance, which is an incorrect assumption for RNA-Seq data. To avoid a large amount of significantly differentially expressed genes (false positives) DESeq2 uses the negative binomial distribution, which applies stricter statistical measures on the data and reduces the number of false positives.

To run DESeq2, you will have to install DESeq2 via Bioconductor:

* Install Bioconductor: <https://www.bioconductor.org/install/>
* Install DESeq2: <https://bioconductor.org/packages/release/bioc/html/DESeq2.html>

I highly recommend reading thoroughly and in-depth through the DESeq2 manual, as well as a few guides before you start with the data analysis. There are a lot of judgement calls in this portion of the data analysis, and it is really important that you understand which decisions you are making. I’ve included the links to some of my favorite references below:

* DESeq2 manual: <http://www.bioconductor.org/packages/release/bioc/html/DESeq2.html>
  + Go to “documentation” then “PDF”
* DESeq2 vignette: <http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html>
  + This is a more detailed version of the manual written by the creators
* Some of the DESeq2 commands are explained here in a digestible manner: <https://hbctraining.github.io/DGE_workshop/lessons/05_DGE_DESeq2_analysis2.html>
* DESeq2 workflow: <https://f1000research.com/articles/4-1070>
* Abrar’s class workflow: <http://genomeintelligence.org/?p=1234>

This manual does not detail every piece of code, which were mostly chosen due to information in these manuals, and are not included here due to time constraints. The code and workflow is detailed nicely in these manuals, especially in the DESeq2 workflow resource.

DESeq2 was partially written by Michael Love, who will respond very quickly to your questions if they are posted on forums, such as Stack Overflow or Biostars. However, make sure to check and see if your question has already been answered before you ask a question.

*Viewing your counts data*

If you want to view the raw counts data that were produced in featureCounts, you may be tempted to open the file in Excel. However, opening a file in Excel without performing certain functions (discussed later in this manual) will cause Excel to misinterpret gene names as dates and other things (example, turn the gene “March9” into “03-09-19” or “Sept7” into “09-07-19”), **and it will alter the document permanently**. If you really want to view this list, then copy all the data from the original text file and create a new Excel file and paste it in. DO NOT overwrite the original file. You will have the view the information in Excel and know that gene names might be altered. To import files for use in DESeq2, import the original txt file generated by featureCounts. You could also perform the step that will be described later here to avoid this problem, but I prefer to use the completely untouched data.

When I viewed the gene names on the list, I was confused by a large number of genes named things like “Gm00099977” and “87876Rik”, and I wondered what these were. According to people on BioStars, gene names that start with “Gm” or end with “Rik” are genes that are annotated but have not been assigned a canonical name. In my dataset, there are nearly 1500 genes that end with “Rik”, so there might be quite a few of these in your dataset. According to Ensembl biotype classifications for mice, genes that start or end with either of these 2 phrases include protein-coding RNAs, long non-coding RNAs, and antisense transcripts. Genes with the “Gm” prefix are apparently often enriched for pseudogenes. Be careful relying too heavily on the results from these types of genes, because it is possible that these could also be mis-mapped reads. The “Rik” genes are cDNA sequences from the RIKEN project that weren’t previously identified as transcripts.

*DESeq2 code* (remember to run the actual R scripts and not copy and paste from this document)

This code is written as a series of scripts, and should not be run all at once (or even all of the lines). Some of the lines that are included are optional, and whether you run them or not is a judgement call. Sometimes, you may have to run part of the data, see what happens, make a decision, and then clear out variables before running parts of the code again to get the correct result. How you do this depends largely on what you need from the code and how well you understand the process.

rm(list=ls())

setwd("~/KayleeStuff/Smith\_Lab/Data/RNA\_Seq/Fet\_Liv")

library("DESeq2")

library("ggplot2")

library("ggpubr")

library("DEP")

library("hexbin")

library("devtools")

library("apeglm")

library("pheatmap")

library("RColorBrewer")

library("SummarizedExperiment")

library("genefilter")

library("BiocManager")

####This step imports the file (read.table) and it is very important to include the “row.names = “Geneid” argument or you will have issues later. This step also cuts the table down to just the columns that are needed, then renames the column names. Then we convert the table into a matrix, which is the data form that DESeq2 requires. We create the treatment variable to define our groups, and then store this in the data frame as well.####

#Input file, select out correct file columns, rename file columns, turn table into matrix, create treatment table for colData

genes <- read.table("FetMLiver\_MdvsEtOH.txt", header = TRUE, sep = "", row.names = "Geneid")

genes.reduced <- genes[,c(6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21)]

names(genes.reduced) <- c("MD-E1", "MD-E2", "MD-E3", "MD-E4", "MD-E5", "MD-E6", "MD-E7", "MD-E8",

"Alc-F1", "Alc-F2", "Alc-F4", "Alc-F5", "Alc-F6", "Alc-F7", "Alc-F8", "Alc-F9")

countdata <- as.matrix(genes.reduced) #Table must be converted to a matrix for DeSeq to perform calculations

treatment <- factor(c(rep("Control",8), rep("Alcohol",8)))

coldata <- data.frame(row.names = colnames(countdata), treatment)

#Ensure that the rownames of the matrix are in the same order as in the other file. DESeq will not check.

#If correct, running this code will output "TRUE".

all (rownames(colData) == colnames(genes.reduced))

#Make a DESeq data set from the count matrix and treatment information

dds <- DESeqDataSetFromMatrix(countData = countdata, colData = coldata, design = ~treatment) #This step gives an error, but it is corrected in the next line of code

dds$treatment <- factor(dds$treatment, levels = c("Control","Alcohol")) #sets the comparison of alcohol vs. control for DeSeq2 step

#If you want to filter out the reads that have low copy number, then you can run this code. Only do this step if needed (might be needed to create the plotMA plot).

keep <- rowSums(counts(dds) >= 10) >=5

table(keep)

dds <- dds[keep,]

##Perform both the rlog and the vst transformation and then compare the meanSdPlot and PCA for them.

#Make sure to export the resulting datasets from these steps for further PCA and other analyses.

#For decreasing gene-wide dispersion use parametric if it isn't influenced by the data as much as local or mean. Find the trend that is least affected by the dispersion. Test all formats, but then erase the data and only use the one that worked the best. You can tell the best by the visualization of the meanSdPlot and the plotPCA.

#rlogTransformation

#rlog\_parametric <- rlogTransformation(dds, blind = FALSE, fitType = "parametric")

#rlog\_local <- rlogTransformation(dds, blind = FALSE, fitType = "local")

#rlog\_mean <- rlogTransformation(dds, blind = FALSE, fitType = "mean")

#head(assay(rlog\_mean), 10)

#colData(rlog\_mean)

#meanSdPlot <- meanSdPlot(rlog\_mean)

#plotPCA(rlog\_mean, intgroup = c("treatment"))

#vstTransformation

vst\_parametric <- varianceStabilizingTransformation(dds, blind = FALSE, fitType = "parametric")

#vst\_local <- varianceStabilizingTransformation(dds, blind = FALSE, fitType = "local")

#vst\_mean <- varianceStabilizingTransformation(dds, blind = FALSE, fitType = "mean")

head(assay(vst\_parametric), 10)

colData(vst\_parametric)

meanSdPlot2 <- meanSdPlot(vst\_parametric)

plotPCA(vst\_parametric, intgroup = c("treatment"))

####This step actually runs the DESeq2 statistical models. It will print the instructions of what it is doing to the screen as it runs. ####

#P-value calculation and DeSeq2 step

dds <- DESeq(dds)

#Plot gene dispersions

plotDispEsts(dds)

#Results

#To get the correct coefficient, run "resultsNames(dds)" to figure out what to put after "coef="".

#In "contrast" it should be: the name of the factor in the design formula, then the control, then the treated

results\_unshrunken <- results(dds)

results\_lfc <- lfcShrink(dds, coef="treatment\_Alcohol\_vs\_Control",type="apeglm")

plotMA(results\_unshrunken, ylim=c(-2,2))

plotMA(results\_lfc, ylim=c(-2,2))

#order results (res) by pvalue

results\_Ordered <- results\_lfc[order(results\_lfc$padj), ]

#summarize the results

summary(results\_Ordered)

sum(results\_unshrunken$padj < 0.1, na.rm=TRUE)

#merge results data frame with counts data frame to create a table that contains both counts and statistical results

results\_lfc\_combined <- merge(as.data.frame(results\_lfc), as.data.frame(counts(dds, normalized=TRUE)), by="row.names", sort=FALSE)

names(results\_lfc\_combined)[1] <- "GeneID"

results\_lfc\_combined\_rmNA <- na.omit(results\_lfc\_combined) #remove genes with NA values

results\_unshrunken\_combined <- merge(as.data.frame(results\_unshrunken), as.data.frame(counts(dds, normalized=TRUE)), by="row.names", sort=FALSE)

names(results\_unshrunken\_combined)[1] <- "GeneID"

results\_unshrunken\_combined\_rmNA <- na.omit(results\_unshrunken\_combined) #remove genes with NA values

#calculate the number of DE (true) and non-DE (false) genes

deCount <- table(results\_lfc$padj<0.05)

#write to CSV

write.table(results\_lfc\_combined, file="DEG\_Results\_MDvAlc\_FetLiv\_LFCShrink\_VSTParam.txt", sep = "\t")

write.table(results\_lfc\_combined\_rmNA, file="DEG\_Results\_MDvAlc\_FetLiv\_LFCShrink\_VSTParam\_RemovedNA.txt", sep = "\t")

write.table(results\_unshrunken\_combined, file="DEG\_Results\_MDvAlc\_FetLiv\_NoShrink\_VSTParam.txt", sep = "\t")

write.table(results\_unshrunken\_combined\_rmNA, file="DEG\_Results\_MDvAlc\_FetLiv\_NoShrink\_VSTParam\_RemovedNA.txt", sep = "\t")

#Plot the counts for a single normalized gene

plotCounts(dds, "Cebpa", intgroup = "treatment", normalized = TRUE, main = "Cebpa")

#Histogram of p-values

hist(results\_lfc$pvalue, breaks=50, col="grey")

#Pval adjustment using BH and Bonferonni methods as a check of methods

adjusted\_pvals\_BH <- p.adjust(results\_lfc$pvalue, method="BH")

adjusted\_pvals\_bon <- p.adjust(results\_lfc$pvalue, method="bonferroni")

hist(adjusted\_pvals\_BH)

hist(adjusted\_pvals\_bon)

BHtotal <- sum(adjusted\_pvals\_BH < 0.05, na.rm=TRUE) #removes NA values

Bontotal <- sum(adjusted\_pvals\_bon < 0.1, na.rm=TRUE)

#Heatmap for sample distance

sampleDists <- dist(t(assay(vst\_parametric)))

sampleDistMatrix <- as.matrix(sampleDists)

colors <- colorRampPalette((rev(brewer.pal(9, "Blues"))))(255)

#Heatmap with the rows and columns in order

pheatmap(sampleDistMatrix,

clustering\_distance\_rows = sampleDists,

cluster\_distance\_cols = sampleDists,

col = colors,

cluster\_rows = FALSE,

cluster\_cols = FALSE)

#Heatmap with the rows and columns clustered

pheatmap(sampleDistMatrix,

clustering\_distance\_rows = sampleDists,

cluster\_distance\_cols = sampleDists,

col = colors)

#Session info including packages used and their information

devtools::session\_info()

#Heatmap of genes with most variance

topVarGenes <- head(order(rowVars(assay(vst\_parametric)),decreasing=TRUE),20)

mat <- assay(vst\_parametric)[ topVarGenes, ]

mat <- mat - rowMeans(mat)

df <- as.data.frame(colData(vst\_parametric))

pheatmap(mat, annotation\_col=df, labels\_col = c("MD-E1", "MD-E2", "MD-E3", "MD-E4", "MD-E5", "MD-E6", "MD-E7", "MD-E8",

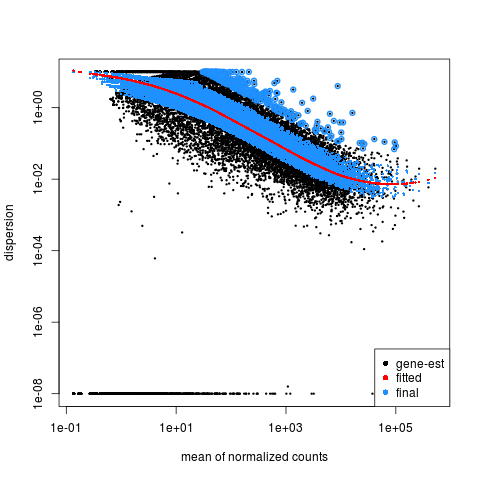
"Alc-F1", "Alc-F2", "Alc-F4", "Alc-F5", "Alc-F6", "Alc-F7", "Alc-F8", "Alc-F9"))

*From Abrar’s manual for reference*:

**Choosing DeSeq2 Parameters for rlog (or VST) Transformation**

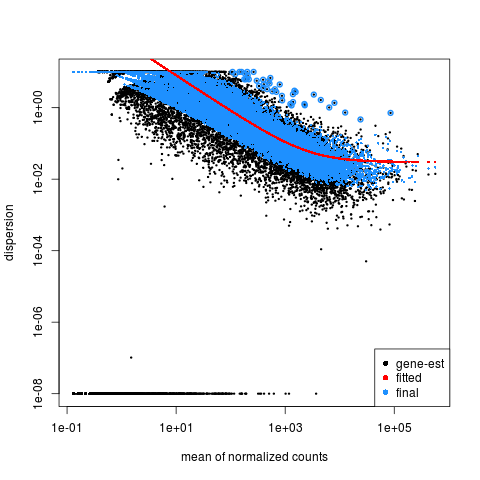
In this piece of code [rlogTransformation(dds, fitType="mean")] it is very important to choose the correct rlogTransformation parameter. There are 3 parameters: mean, parametric, and local.

Below is an example of a local fit:



This is an example of two fitted graphs, local and parametric. In this example, the DeSeq2 representatives recommended the use of a parametric fit because it was less affected by the gene-wise dispersion values; whereas, the local fit is flattening on the left side (<https://support.bioconductor.org/p/95436/>). So as you can see, the parametric graph’s fitted mean (red line) is not skewed by the gene dispersion data points.

Below is an example of a parametric fit:



* **Explanation (Bioconductor forum):**
  + For DESeq2, we have 3 options for the type of trend line: fitType = "parametric", "local" or "mean". These are described in ?estimateDispersions, but roughly the recommendation is: for decreasing gene-wise dispersion estimates over mean (using plotDispEsts) one should use parametric, unless the parametric fitting procedure does not work, in which case use "local" (local regression is actually automatically substituted with a message in the case that the parametric fitting procedure does not converge.) The "mean" option is useful when there is no apparent dependence of dispersion estimates over mean (using plotDispEsts). This choice does not depend on sample size, but on the apparent dependence of the gene-wise estimates (the MLE for each gene) on the mean of counts. (<https://support.bioconductor.org/p/63244/>)

**Documentation Information about DeSeq2’s Dispersion Parameters:**

### Description

This function obtains dispersion estimates for Negative Binomial distributed data.

### Usage

## S4 method for signature 'DESeqDataSet'

estimateDispersions(object, fitType = c("parametric",

"local", "mean"), maxit = 100, quiet = FALSE, modelMatrix = NULL)

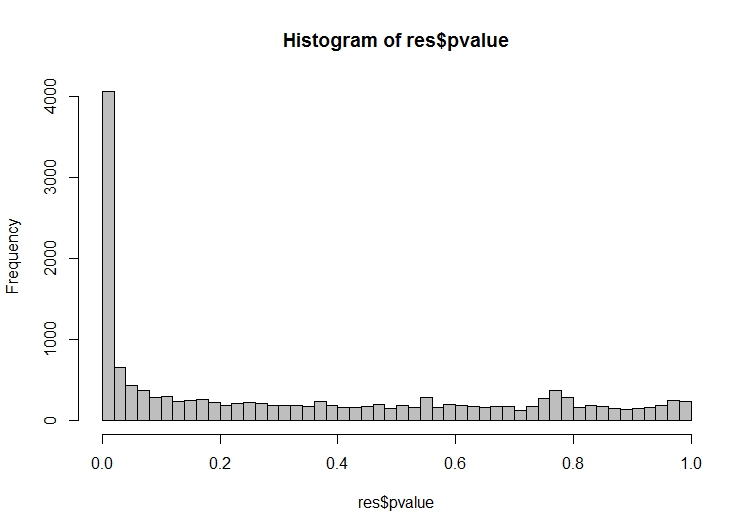
### Arguments

|  |  |
| --- | --- |
| object | a DESeqDataSet |
| **fitType** | **either "parametric", "local", or "mean" for the type of fitting of dispersions to the mean intensity.**   * **parametric - fit a dispersion-mean relation of the form:**   ***dispersion = asymptDisp + extraPois / mean***  **via a robust gamma-family GLM (generalized linear model). The coefficients asymptDisp and extraPois are given in the attribute coefficients of the**[**dispersionFunction**](http://127.0.0.1:24158/help/library/DESeq2/help/dispersionFunction)**of the object.**   * **local - use the locfit package to fit a local regression of log dispersions over log base mean (normal scale means and dispersions are input and output for**[**dispersionFunction**](http://127.0.0.1:24158/help/library/DESeq2/help/dispersionFunction)**). The points are weighted by normalized mean count in the local regression.** * **mean - use the mean of gene-wise dispersion estimates.** |

*Notes on some of the selected commands:*

* VST and rlog are two different ways to normalize the data. See Michael Love’s vignette (<http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html>) for more information on what these do and reasons to select one or the other for your data.
* DESeq2 requires 2 data frames as an input. These were both created in this code at the beginning. See manual for more information on what DESeq2 does with this data.
  + From Abrar:
    - Be careful when analyzing the upregulated and downregulated genes based off the positive and negative log2FC values. The fold changes are relative to the null hypothesis that there is no significant difference in the gene expression based on the control, assuming that you specified your control as your very first condition in the start of your DESeq code.
    - [samp <- data.frame(row.names = c("saline\_1","saline\_2","ETOH\_1","ETOH\_2"), condition = rep(c("SALINE","ETOH"),each=2))]. 🡪 **If you reverse the order of Saline & ETOH in this code above you will get the right FCs with positive = up and negative = down.**
    - Therefore, the positive FC values will be **increased in the control** (ex: Saline), and downregulated in the case (ex: ETOH). The negative FC values will be upregulated genes in the case condition (ex: ETOH) and downregulated in the control (ex: Saline). This is because the changes are based off the assumption that we are generating the FC values from the alternative hypothesis that there is a significant difference between the control and the case – based off the control as the null or baseline measure, so the FC’s are relative to the control. **Positive FC = upregulated in control, Negative FC = downregulated in control.**
* It can also be helpful to have qPCR data to which you can compare your RNAseq results. This way, you can make absolutely sure that you compared the control to the alcohol samples.
* At various points throughout the code, I implement some QC figures (ex. p-value histogram, heatmaps, PCA plots, MA plots, mean/Sd plots, and dispersion plots), which should be compared to figures online to see if they look okay for your data.
* It is important to record which packages and which versions of the packages you used for the analysis, for 2 reasons: 1) you want to know for publications, and 2) in case something goes wrong, you need to know what you used previously.

**P-values Histogram:**

A good distribution for the p-values after finishing the DeSeq2 analysis looks like this:

You want the p-value distribution to look like this and NOT like a uniform distribution. If the p-values were all uniform, as in the case of a uniform distribution that is problematic because it indicates that your significant p-values are not actually that significant, because relative to all the other p-values the same amount of significant p-values equals the number of non-significant p-values.

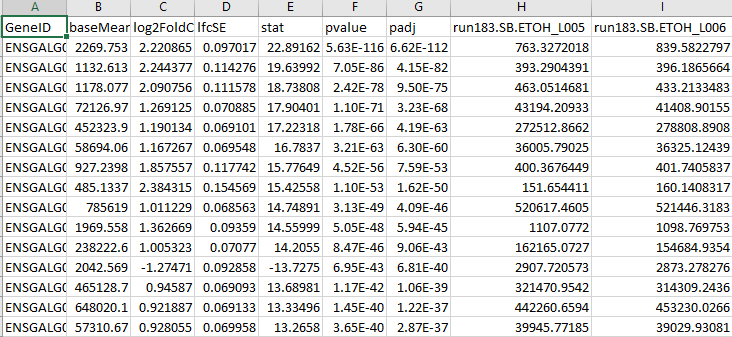
*Exporting files*:

Excel has a permanent setting that means it will change names around in the spreadsheets loaded into it. And unfortunately, the way that Excel does this, means that the name is permanently changed in the file as well. The best way I’ve figured out to fix this is to make sure you have exported a .txt file from DESeq2, and DO NOT OPEN IT, especially not in Excel. As soon as you open a file in Excel, the names of the genes will be changed, and you’ll have to recreate the file in R from scratch. Instead, import the original txt file into this website: (<http://apostl.moffitt.org/>) using the “Escape Excel” module, and follow the directions on the website. Just use the default parameters. After the program is finished, download the file. This file can now be imported into Excel and viewed. This program alters the gene names slightly so that Excel cannot change them but so that they will still be legible and able to be imported into other programs and recognized.

Also, make sure to back up (in multiple places), the DESeq2 output files. This step is annoying, and although you can redo it, it is easiest to have multiple versions saved in case something happens to one of them.

*Results*:

The main results provided from the DeSeq2 analysis that are of interest will look like this:



You will also have other results that were outputted from the DESeq2 analysis, including the graphs (described above in the code), as well as important quality metrics such as those shown here:

* Total number of genes with nonzero total read count, number and percentage of DE genes with log2foldchange greater and less than zero, number of outliers, number and percentage of genes with low counts, and total DE genes.
  + These outputs can be controlled for a p-value of interest, and even compared to results from other tissues.
* These metrics must be generated when running DESeq2 the first time, or else the code will have to be rewritten to ask questions of a different dataset.

1. **Quality control and overall data results (PCA and HCPC)**

At this point in the analysis, you can now use the results from the DESeq2 output files directly to graph results for individual genes and use the associated p-values from DESeq2 to assess significance. One way of how to do this is seen in the DESeq2 code above, but you can also use normal methods such as graphing it in ggpubr, ggplot2, etc. Even in Excel or other graphing systems. You can also pull out individual findings to ask other scientific questions about certain functions or results.

One important analysis to run before moving on is PCA (and potentially hierarchical clustering on the principal components (HCPC)) to look at the overall data, see if your treatment and control groups cluster (or don’t) together, and identify any overall outlier samples. I will not go into much information about the reasons for doing this or the code for how to do this, as I pulled most of my information from the following book, which I highly recommend:

“Practical Guide to Principal Component Methods in R” by Alboukadel Kassambara. Ed. 1. ISBN: 9781975721138.

This information from this book can also be found on his website: <http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/>

Here is the code I used to generate PCA and HCPC graphs on my data:

rm(list=ls())

setwd("~/KayleeStuff/Smith\_Lab/Data/RNA\_Seq/Fet\_Liv")

library("FactoMineR")

library("ggplot2")

library("factoextra")

library("tibble")

library("ggsci")

#Import table of differentially expressed genes and remove extraneous columns

DEG <- read.table("DEG\_Results\_MDvAlc\_FetLiv\_LFCShrink\_VSTParam\_RemovedNA\_ExcelProof.txt", header = TRUE, sep = "", row.names = "GeneID")

DEG\_reduced <- DEG[,7:22]

DEG\_reduced\_flip <- t(DEG\_reduced) #reverse rows and columns

group <- factor(c(rep("Control",8), rep("Alcohol",8))) #create group name list

group <- data.frame(group) #turn group name list into dataset

DEG\_reduced\_flip\_labeled <- cbind.data.frame(DEG\_reduced\_flip, group) #add in group name column

DEG\_reduced\_flip\_labeled <- DEG\_reduced\_flip\_labeled[,c(14818, 1:14817)] #rearrange the columns to put group names 1st

#Create PCA but do not graph it.

results.pca <- PCA(DEG\_reduced\_flip\_labeled[,-1], scale.unit = FALSE, ncp = 5, graph = FALSE) #ncp = number of principal components

print(results.pca)

#Create scree plot

png("FetLiv\_DEG\_ScreePlot\_highRes.png", units="in", width=7, height=7, res=600)

fviz\_eig(results.pca, addlabels = TRUE, ylim = c(0,80))

dev.off()

#See which variables contribute the most to each of the principal components

variables <- get\_pca\_var(results.pca)

variables$coord

contributions <- head(variables$contrib, 20)

#Look at individual contributions and graph PCA and export the graph

png("FetLiv\_DEG\_PCA\_basic\_highRes.png", units="in", width=9, height=7, res=600)

ind.pca <- fviz\_pca\_ind(results.pca, col.ind = DEG\_reduced\_flip\_labeled$group, palette = "aaas", addEllipses = TRUE, legend.title = "Group", mean.point = FALSE, repel = TRUE)

ggpubr::ggpar(ind.pca, title = "Principal Component Analysis", subtitle = "Fetal Liver", xlab = "PC1 (61.3%)", ylab = "PC2 (24.8%)")

dev.off()

#HCPC analysis

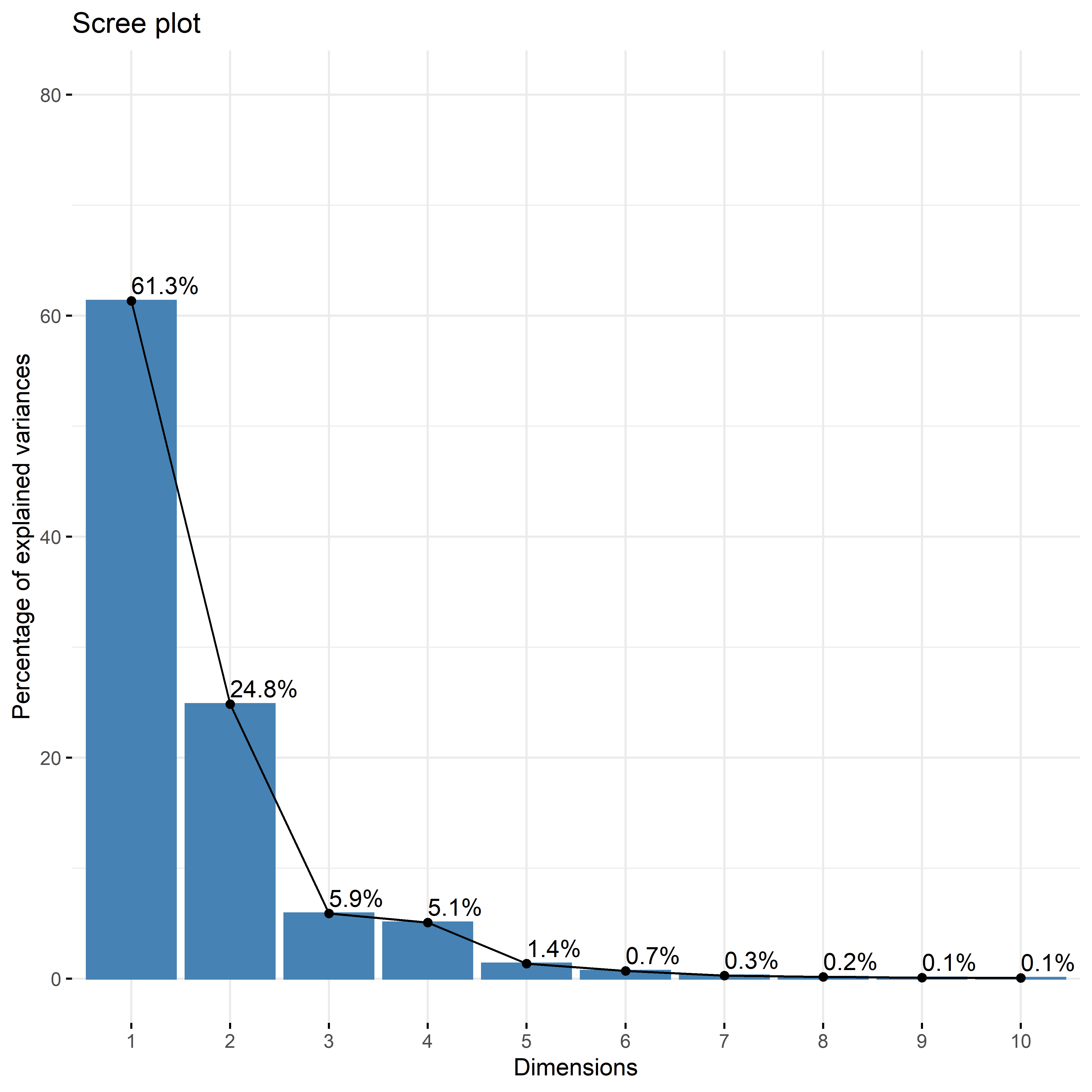
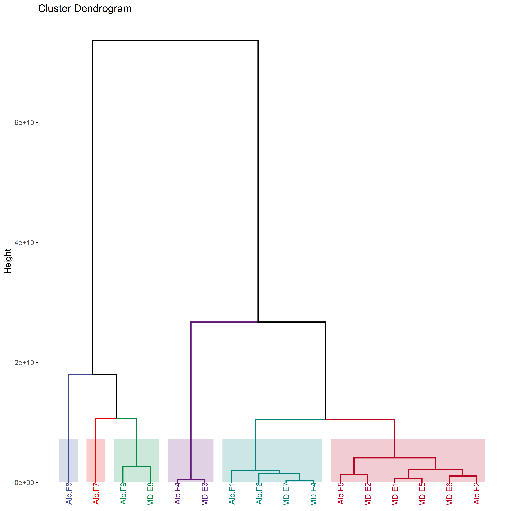
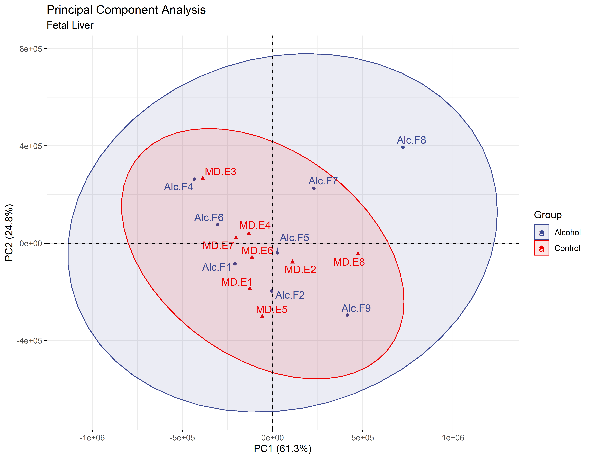
res.hcpc <- HCPC(results.pca, graph = FALSE)

png("FetLiv\_DEG\_HCPC\_basic\_highRes.png", units="in", width=9, height=9, res=600)

fviz\_dend(res.hcpc, cex = 0.7, palette = "aaas", rect = TRUE, rect\_fill = TRUE, rect\_border = "aaas", labels\_track\_height = 0.8)

dev.off()

Expected output: This analysis should generate 3 graphs: a PCA plot, a scree plot (showing the contributions of the individual principal components), and a clustering on the principal components. These graphs will look something like this:



For more description on what the graphs mean or how to interpret them, see the book or the website mentioned above.

1. **Further Analysis (gene set enrichment analysis, pathway analysis, overexpression analysis, annotation to KEGG pathways, comparison with other data sets, integrating transcriptome with metabolome data, etc.)**

At this point in the analysis, you have a lot of choices in how to analyze your data, and how you choose to do so depends on what you want to get out of the data. Because of the individuality of each user’s analysis, I will not go into detail on what analyses I conducted. However, all of my analyses, results, and reasoning can be found on the lab server under “Z:\Kaylee Helfrich\Experiments and Data\2-Metabolon&RNAseq”.

To mention a few pieces of information:

* I did not use this until this point of the analysis, but I wish I had, and I highly recommend that a new user writes their code using RMarkdown (https://rmarkdown.rstudio.com/). This is a fantastic tool that is easy to learn (I figured it out in ~2-3 hours) and a great way to record-keep and remind yourself of why you conducted certain analyses, how you conducted them, and why did you things a certain way. There is also a cheatsheet (https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf) and a set of tutorials (<https://rmarkdown.rstudio.com/lesson-1.html>), that are very helpful to learn RMarkdown.
  + The output files from this analysis can be found in the same folders with the code and the results. These should be referenced for details on the analysis, and can often be easier to understand than the code itself.
* I found this website (<https://hbctraining.github.io/DGE_workshop/lessons/09_functional_analysis.html>) very helpful when I didn’t know where to start for my analysis of the data. It walks you through a few of the options for analysis as well as a description and code for some of the most common methods. I used a lot of this code to write my own analyses. The same people wrote some other workflows as well which I also used, but this was the most helpful.

**Total time of data analysis**:

* This will entirely depend upon the amount of data and the experience of the user.
* However, as a reference, completing this entire analysis (excluding step 9 due to its subjectivity) with 2 experimental groups and 8 samples per group took me:
  + **~4 weeks** full time as an inexperienced user
  + **~3 days** full time as an experienced user
* I estimate an inexperienced user with the already written code could finish this analysis in approximately 3 weeks.