# Gregoire_Final_Project

Michelle

4/19/2021

## BIO539: Final Project: RNA Seq Analysis

## Finding and downloading a dataset

To start this analysis, a dataset was first found on the Gene Expression Omnibus (GEO) from NCBI. GEO is a database repository of high throughput gene expression data and hybridization arrays, chips, microarrays, and RNA seq data.

Dataset: Human Airway Smooth Muscle Transcriptome Changes in Response to Asthma Medications Available from: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE52778 Accession: PRJNA229998 ID: 229998 Description: "Asthma is a chronic inflammatory airway disease. The most common medications used for its treatment are Beta2-agonists and glucocorticosteroids, and one of the primary tissues that these drugs target in the treatment of asthma is the airway smooth muscle. We used RNA-Seq to characterize the human airway smooth muscle (HASM) transcriptome at baseline and under three asthma treatment conditions"

This file contains 16 Sequence Read Archives (SRA files).

To get these files, we have to click on the "SRA Run Selector" button towards the bottom of the page. Click on SRR file name and then on the "Data access" tab. Copy the link address from NCBI. On the terminal command line, use the command "wget" followed by the links to download the SRA files onto the HPC:

- wget https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039508/SRR1039508.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039509/SRR1039509.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039510/SRR1039510.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039511/SRR1039511.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039512/SRR1039512.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039513/SRR1039513.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039514/SRR1039514.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039515/SRR1039515.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039516/SRR1039516.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039517/SRR1039517.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039518/SRR1039518.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039519/SRR1039519.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039520/SRR1039520.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039521/SRR1039521.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039522/SRR1039522.1
  https://sra-downloadb.be-md.ncbi.nlm.nih.gov/sos1/sra-pub-run-5/SRR1039523/SRR1039523.1

## Preparing the files for quality check

To check the quality of the sequences, the SRA files first need to be converted into fastq files. This can be accomplished by using the SRA Toolkit from NCBI.

SRA Toolkit can be downloaded/install/set-up with the following set of commands:

- wget https://ftp-trace.ncbi.nlm.nih.gov/sra/sdk/2.11.0/sratoolkit.2.11.0-centos_linux64.tar.gz
- tar xzvf sratoolkit.2.11.0-centos_linux64.tar.gz #to unzip the file
- export PATH=$PATH:/data/fallinilab/sratoolkit.2.11.0-centos_linux64/bin #to install the toolkit
- vdb-config –interactive #to set-up the toolkit
- x #to exit the config

Now that the toolkit is ready, the files can be converted into fastq files. **Note this will take some time for each file! Use the following command:

- fasterq-dump –threads 3 –progress SRR*

This generates two files for each SRR. One will be _1.fastq and the other _2.fastq. These files are the paired ends.

## Checking the quality of the samples

Sequence quality is rated with quality scores (Q) which are derived from the formula Q = -10log10(e). "e" is the estimated probability of the base call being wrong. A Q of 30 indicates that there is a 1 in 1000 probability of an incorrect base call and that the Inferred Base Call Accuracy is 99.9%. Likewise a Q of 20 indicates there is a 1 in 100 probability of an incorrect base call and has an accuracy of 99%. Generally a Q of 20 or above indicates good quality sequence data. We can check the quality of the sequences with a program called fastqc. Fastqc is available from Fastqc. To load the fastqc module on the commandline and subsequently run fastqc use the following codes:

- module load FastQC/0.11.9-Java-11
- fastqc *.fastq

This makes a file of the same name but under _fastqc.html. To view these files, you must download them from the commandline to your computer. If you are on an HPC, you will need to use a file transfer program like CyberDuck. The fastqc report will give information on: basic statistics (such as the type of sequencing platform and %GC content), per base sequence quality (in the form of Q scores to represent how accurate each base was –again a Q of 30 would mean a 1 in 1000 chance that the base was wrongly incorporated), per sequence quality scores (the average Q score for the whole sequence), per base sequence content (this should ideally show parallel lines), perbase N content, per base GC content (can indicate sample contamination), sequence length distribution, sequence duplication levels, overrepresented sequences, and adaptor content. Since the reports are rather lengthy, I will not include each of the graphs here in this file. I will just summarize the results below.

For per base sequence quality (Q score) the samples were generally in the mid 30s, meaning that there is a 1 in 1000 chance of a wrong base call. This was also true for the per sequence quality scores. This means these are good quality reads. The per base sequence content was a little variable at beginning but this is okay because RNA seq data has a little variation from the random hexamers used to generate the cDNA and since there is the same pattern of bias across files it is okay. The per base n content was also low across the files. The per base GC for the reads also reflected the theortetical distribution meaning the samples are not likely to be contaminated. The sequence length was 63 bp. Again some sequence duplication was seen at beginning but that is okay because RNA seq data reflects the transcriptome where some transcripts may be duplicated in high numbers. Likewise, there were some overrepresented sequences but these are alright. The reads also had low adaptor contents (close to 0). Overall, the results from the fastqc demonstrated that these are good quality reads.

## Trimming reads?

The results from our fastqc for these files are good and indicate clean samples that do not need to be trimmed for quality or for adaptor content! This is good especially since our reads are short at only 63 base pairs.

However, if you needed to trim for quality or for adaptor content, any of the following programs could be used trimmomatic, cutadapt, or bbtools. Note that if you have loaded a different module in the command line, prior to using these new modules you might have to unload the previous module with the command: * module unload (name of module here)

If you needed to trim the reads due to poor quality or adaptors, run fastqc again to ensure that you now have good quality reads. This time specify an output with the flag "-o filname(something like trimmed or clean).fastq", so that you do not overwrite your original files, or alternatively cd into a new directory and run the fastqc command there to save the files there.

## Aligning to a Reference Genome

The next step in the RNA seq workflow is to align the reads to a reference genome. To do this we will use STAR . We will use the human reference genome GRCh38 (hg38) because it is a well cited and relatively recent reference genome. It was released in December 2013 and has the latest assembly of the human genome with greatly expanded alternate (ALT) contigs. These alternate haplotypes include highly variable HLA loci and represent common complex variations of the human genome. To align to this reference, we will first need to build an index of the reference genome. The hg38 reference genome (as a .fa FASTA file) and its annotation which provides information about the gene structure and splicing sites (in Gene Transfer Format (GTF)). Both of these are available for download to the command line from the UCSC genome browser with the following commands:

- wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_37/GRCh38.p13.genome.fa.gz
- gunzip GRCh38.p13.genome.fa.gz
- wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_37/gencode.v37.chr_patch_hapl_scaff.basic.annotation.gtf.gz
- gunzip gencode.v37.chr_patch_hapl_scaff.basic.annotation.gtf.gz

Next, load the STAR module on the command line with the command:

- module load STAR

Then we'll generate the genome indexes with the following command:

- STAR –runThreadN 6 –runMode genomeGenerate –genomeDir /data/fallinilab/STAR_index/ –genomeFastaFiles /data/fallinilab/STAR_index/GRCh38.p13.genome.fa –sjdbGTFfile /data/fallinilab/STAR_index/gen –sjdbOverhang 62

Then we'll align the reads with the following commands where we will specify STAR to make the output file to a BAM format that is sorted by genome coordinates (this allows us to skip the step of converting the default SAM output into BAM which is usually done with SAMtools:

- STAR –runThreadN 4 –genomeDir /data/fallinilab/STAR_index/ –readFilesIn /data/fallinilab/*_1.fastq /data/fallinilab/raw_data/*_2.fastq –outSAMtype BAM SortedByCoordinate –quantMode GeneCounts

## Preparing the BAM files for differential gene expression analysis

To analyze the files for differential gene expression, they need to be changed from coordinate sorted BAM files to gtf files. This can be done using the program Stringtie with the following commands:

- stringtie * .bam -o /data/fallinilab/*.gtf
- ls *.gtf > mergelist.txt
- stringtie mergelist.txt –merge -o data_merge.gtf

The next step is to convert the file from StringTie and convert it into a matrix of counts as a txt file that can then be read by Deseq2 in R. This can be done by running a following python script supplied by Stringtie. Make a modification of the script using the command nano prepDE.py and paste the following:

- "‘ #!/bin/bash F=/data/fallinilab/ array=($(ls data_merge.gtf)) for i in *array*[@];*doecho*"(echo ${i}|sed "s/..*//“) *F*/{i}" » counts_list.txt done

echo "STOP" $(date)

Run the python script using:

- #python prepDE.py -i counts_list.txt

Convert the txt file to a csv file with: * sed 's/ +/,/g' counts_list.txt > counts.csv

## Testing diffferential gene expression

Download the files to your computer using CyberDuck and upload the file to R-Studio.

Install and load the following packages in the R library:

```
BiocManager::install("DESeq2")
library("DESeq2")
library(ggplot2)
library(tidyverse)
```

Run the following commands in R to evaluate the differential gene expression for treatment with dexamethasone (dex):

```
#load the data
counts <- read.csv("~/Final_Project/counts.csv")
metadata <- read.csv("https://4va.github.io/biodatasci/data/airway_metadata.csv")
#the metadata is a csv that was created with the information for the detamethasone and control samples
```

```
#check that the column names of counts (except for the first column which lists the ensgene) are the sa
#names(counts)[-1]
#metadata$id
#names(counts)[-1]==metadata$id
all(names(counts)[-1]==metadata$id)
```

```
## [1] TRUE
```

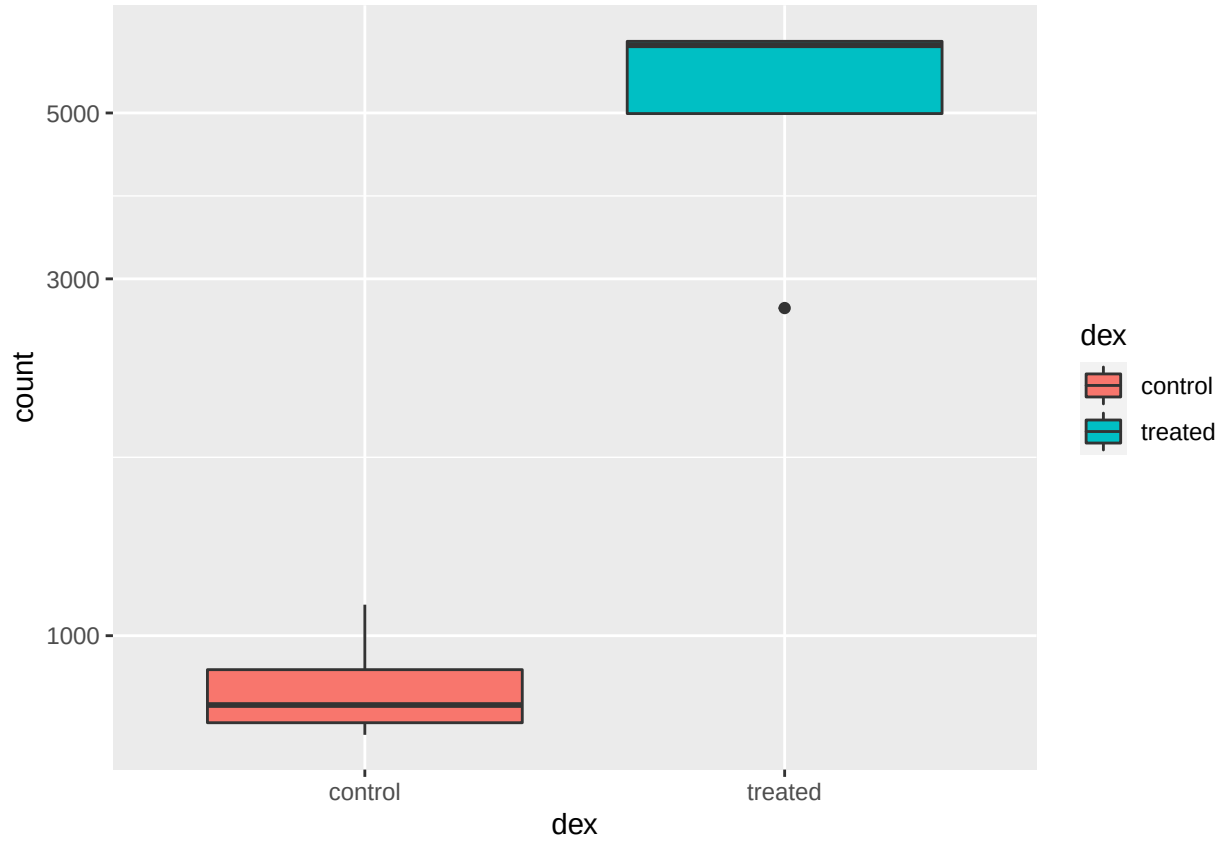
```
#make variable for deseq
dds <- DESeqDataSetFromMatrix(countData=counts,
                              colData=metadata,
                              design=~dex,
                              tidy=TRUE)
#look at the output
dds
```

```
#run deseq and look at results
dds <- DESeq(dds)
sizeFactors(dds)
dispersions(dds)
results(dds)
```

```
#summary of differential gene expression
res <- results(dds, tidy=TRUE)
summary(res)
```

```
##      row                baseMean          log2FoldChange       lfcSE
##  Length:38694       Min.    :     0.0   Min.   :-6.030   Min.   :0.057
##  Class :character   1st Qu.:     0.0   1st Qu.:-0.425   1st Qu.:0.174
##  Mode  :character   Median :     1.1   Median :-0.009   Median :0.445
##                     Mean   :   570.2   Mean   :-0.011   Mean   :1.136
##                     3rd Qu.:   201.8   3rd Qu.: 0.306   3rd Qu.:1.848
##                     Max.   :329280.4   Max.   : 8.906   Max.   :3.534
##                                        NA's   :13436    NA's   :13436
##       stat              pvalue           padj
##  Min.   :-15.894   Min.   :0.000   Min.   :0.000
##  1st Qu.: -0.643   1st Qu.:0.168   1st Qu.:0.203
##  Median : -0.027   Median :0.533   Median :0.606
##  Mean   :  0.045   Mean   :0.495   Mean   :0.539
##  3rd Qu.:  0.593   3rd Qu.:0.800   3rd Qu.:0.866
##  Max.   : 18.422   Max.   :1.000   Max.   :1.000
##  NA's   :13436     NA's   :13578   NA's   :23549
```

```
#plot one of the genes
plotCounts(dds, gene="ENSG00000103196", intgroup="dex", returnData=TRUE) %>%
  ggplot(aes(dex, count)) + geom_boxplot(aes(fill=dex)) + scale_y_log10()
```

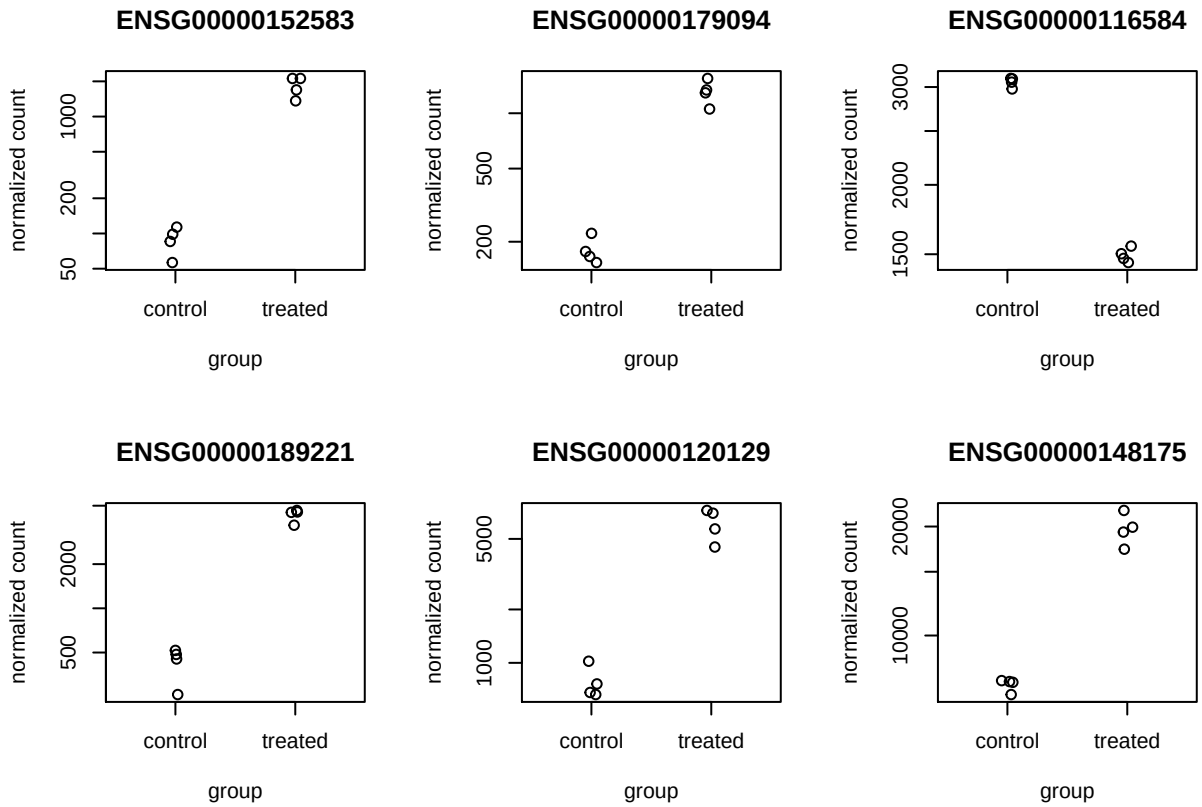



This is nice and we can see the difference between the dex treatment and controls for this gene, but we want to see what's really relevant in our dataset. So let's sort the data by p value (adjusted p value = padj) and plot the top 6.

```
#sort by p value
res <- res[order(res$padj),]
head(res)
```

```
##                   row   baseMean log2FoldChange      lfcSE      stat
## 9541  ENSG00000152583   954.7709       4.368359 0.23712679  18.42204
## 14524 ENSG00000179094   743.2527       2.863889 0.17556931  16.31201
## 4586  ENSG00000116584  2277.9135      -1.034701 0.06509844 -15.89440
## 16731 ENSG00000189221  2383.7537       3.341544 0.21240579  15.73189
## 5010  ENSG00000120129  3440.7038       2.965211 0.20369513  14.55710
## 9039  ENSG00000148175 13493.9204       1.427168 0.10038904  14.21638
##              pvalue         padj
## 9541  8.744898e-76 1.324415e-71
## 14524 8.107836e-60 6.139658e-56
## 4586  6.928546e-57 3.497761e-53
## 16731 9.144326e-56 3.462270e-52
## 5010  5.264243e-48 1.594539e-44
## 9039  7.251278e-46 1.830344e-42
```

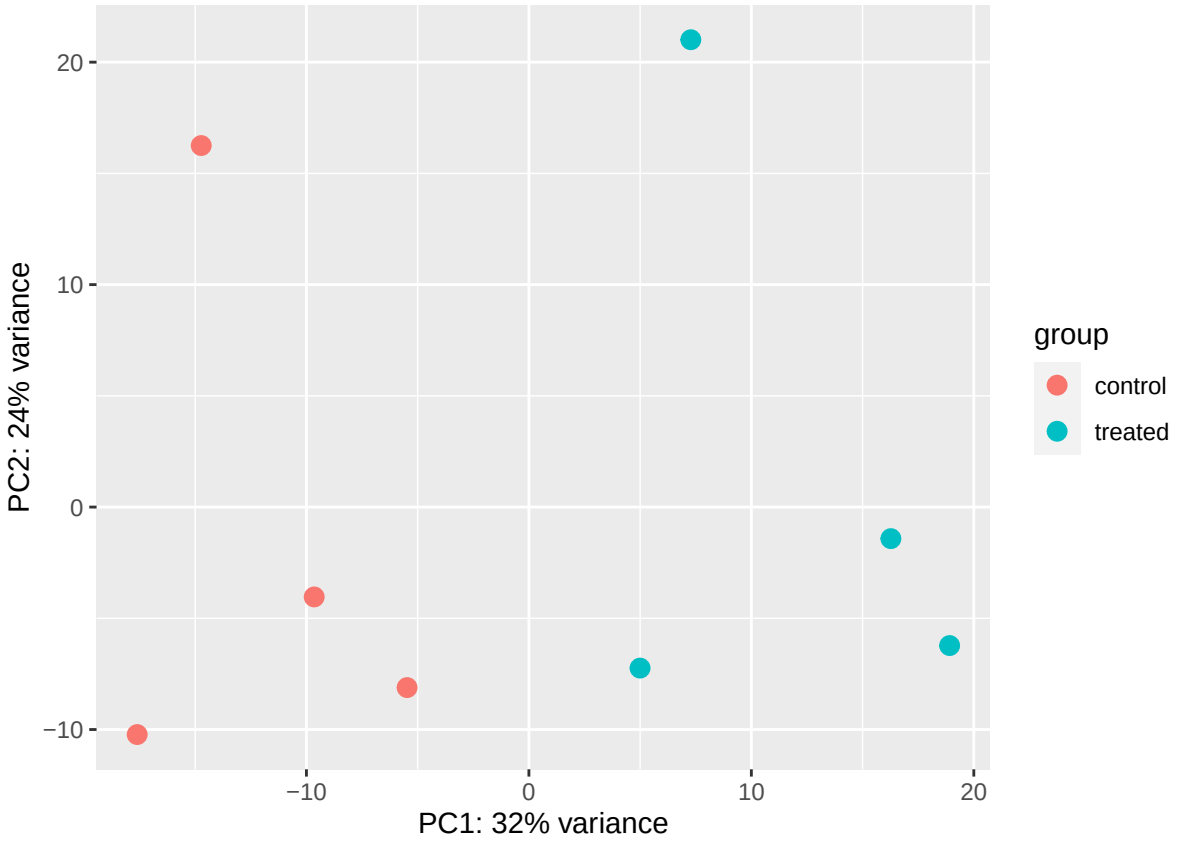
```
#plot the top 6 results, 3 per row, 2 columns
par(mfrow=c(2,3))

plotCounts(dds, gene="ENSG00000152583", intgroup="dex")
plotCounts(dds, gene="ENSG00000179094", intgroup="dex")
plotCounts(dds, gene="ENSG00000116584", intgroup="dex")
plotCounts(dds, gene="ENSG00000189221", intgroup="dex")
plotCounts(dds, gene="ENSG00000120129", intgroup="dex")
plotCounts(dds, gene="ENSG00000148175", intgroup="dex")
```




Next we'll plot the principal component analysis. The other graphs which looked at differential expression

use raw data, but for other downstream analysis of RNA seq, the raw count data needs to be transformed because for these analyses it is not clear how to best compute a distance metric. Here we'll transform with a variance stabilizing transformation which will remove the dependence of the variance on the mean.

```
# plot principal components analysis variance stabilization with vst
vsdata <- vst(dds, blind=FALSE)
plotPCA(vsdata, intgroup="dex")
```




That concludes how far we're going to analyze this data, but there are many more analyses that can be performed! This is just the tip of the iceburg!