Class 13: Transcriptomics and the analysis of RNA-Seq data

Kevyn Aguilar Ramirez (PID: A16321291)

In today's class we will explore and analyze

##Data Import

We have two input files,

```
counts <- read.csv("airway_scaledcounts.csv", row.names=1)
metadata <-  read.csv("airway_metadata.csv")
```

```
head(counts)
```

```
                SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516
ENSG00000000003        723        486        904        445       1170
ENSG00000000005          0          0          0          0          0
ENSG00000000419        467        523        616        371        582
ENSG00000000457        347        258        364        237        318
ENSG00000000460         96         81         73         66        118
ENSG00000000938          0          0          1          0          2
                SRR1039517 SRR1039520 SRR1039521
ENSG00000000003       1097        806        604
ENSG00000000005          0          0          0
ENSG00000000419        781        417        509
ENSG00000000457        447        330        324
ENSG00000000460         94        102         74
ENSG00000000938          0          0          0
```

```
head(metadata)
```

```
          id     dex celltype     geo_id
1 SRR1039508 control   N61311 GSM1275862
2 SRR1039509 treated   N61311 GSM1275863
3 SRR1039512 control  N052611 GSM1275866
4 SRR1039513 treated  N052611 GSM1275867
5 SRR1039516 control  N080611 GSM1275870
6 SRR1039517 treated  N080611 GSM1275871
```

Q1. How many genes are in this dataset?

```
dim(counts)
```

```
[1] 38694     8
```

This dataaet has 38694 genes

Q2. How many 'control' cell lines do we have

```
sum(metadata$dex == "control")
```

```
[1] 4
```

There are 4 'control' cell lines

Data Explore

Time to do some analysis.

We have 4 control and 4 treated samples/experiments/columns

Make sure the metadata id column matches the columns in our count data.

```
colnames(counts)
```

```
[1] "SRR1039508" "SRR1039509" "SRR1039512" "SRR1039513" "SRR1039516"
[6] "SRR1039517" "SRR1039520" "SRR1039521"
```

```
colnames(counts) == metadata$id
```

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

To check that all elements of a vector are TRUE, we can use the `all()` function.

```
all(c(T,T,T,F))
```

```
[1] FALSE
```

```
all(colnames(counts) == metadata$id)
```

```
[1] TRUE
```

To start I will calculate the `control.mean` and `treated.mean` values and compare them.

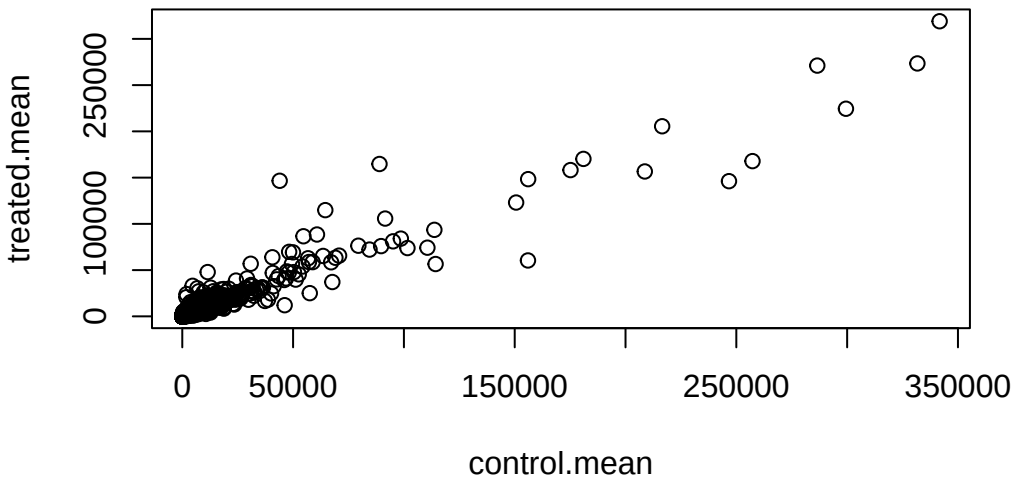
- Identify and extract the `control` only columns
- Determine the mean valus for each genes (i.e. row)
- Do the same for `treated`

```
# Where does it tell me which columns are control?
control.inds <- metadata$dex == "control"
control.counts <- counts[, control.inds]
control.mean <- apply(control.counts, 1, mean)
```

```
treated.inds <- metadata$dex == "treated"
treated.counts <- counts[, treated.inds]
treated.mean <- apply(treated.counts, 1, mean)
```

```
meancounts <- data.frame(control.mean, treated.mean)
```

```
plot(meancounts)
```

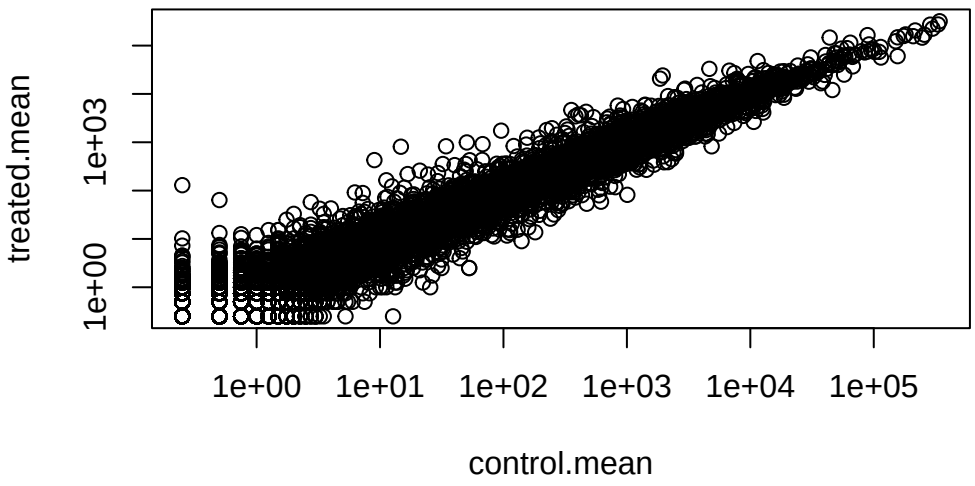


This data is screaming at us to log transform

```
plot(meancounts, log = "xy")
```

```
Warning in xy.coords(x, y, xlabel, ylabel, log): 15032 x values <= 0 omitted
from logarithmic plot
```

```
Warning in xy.coords(x, y, xlabel, ylabel, log): 15281 y values <= 0 omitted
from logarithmic plot
```



I want to compare the treated and the control values here and we will use fold change in log2 units to do this. log2(Treated/Control)

```
log2fc <- log2(meancounts$treated.mean/meancounts$control.mean)
meancounts$log2fc <- log2fc
```

No difference

```
log2(20/20)
```

```
[1] 0
```

A doubling in the treated:

```
log2(20/10)
```

```
[1] 1
```

Downregulated

```
log2(5/10)
```

```
[1] -1
```

Quadrulpe

```
log2(40/10)
```

```
[1] 2
```

A common rule of thumb cut-off for calling a gene "differentially expressed" is a log2 fold change value of either >+2 or <-2 for "upregulated" and "downregulated" respectively.

```
head(meancounts)
```

```
                control.mean treated.mean      log2fc
ENSG00000000003       900.75       658.00 -0.45303916
ENSG00000000005         0.00         0.00         NaN
ENSG00000000419       520.50       546.00  0.06900279
ENSG00000000457       339.75       316.50 -0.10226805
ENSG00000000460        97.25        78.75 -0.30441833
ENSG00000000938         0.75         0.00        -Inf
```

```
#Not proper way b/c we need to remove NaN
sum(meancounts$log2fc > +2, na.rm = T)
```

```
[1] 1846
```

We first need to remove zero count genes - as we can't say anything about these genes anyway and their division of log values are messing things up (divide by zero) or the -infinity log problem.

```
#"TRUE"s are zero values
#meancounts[,1:2]==0
```

```
to.rm.ind <- rowSums(meancounts[,1:2]==0) > 0
mycounts <- meancounts[!to.rm.ind, ]
```

> How many genes do we have left that we can say something about (i.e. they don't have any zero counts)?

```
nrow(mycounts)
```

```
[1] 21817
```

```
up.ind <- mycounts$log2fc > 2
down.ind <- mycounts$log2fc < (-2)
```

```
sum(up.ind)
```

```
[1] 250
```

```
sum (down.ind)
```

```
[1] 367
```

Q10. No, we are missing stats!! Are these differences significant?

DESeq analysis

Let's do this properly with the help of the DESeq2 package

```
library(DESeq2)
```

We have to use a specific data onject for working with DESeq

```
dds <- DESeqDataSetFromMatrix(countData = counts,
                        colData = metadata,
                        design = ~dex)
```

```
converting counts to integer mode
```

```
Warning in DESeqDataSet(se, design = design, ignoreRank): some variables in
design formula are characters, converting to factors
```

Run our main analysis with the `DESeq()` function

```
dds <- DESeq(dds)
```

```
estimating size factors

estimating dispersions

gene-wise dispersion estimates

mean-dispersion relationship

final dispersion estimates

fitting model and testing
```

To get the results out of our `dds` object we can use the DESeq function called `results()`:

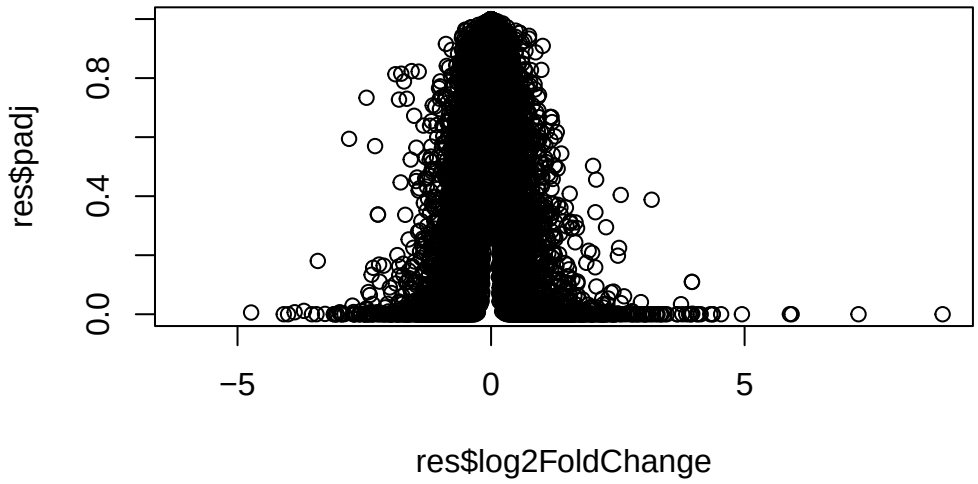
```
res <- results(dds)
head(res)
```

```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 6 columns
                  baseMean log2FoldChange     lfcSE      stat    pvalue
                 <numeric>      <numeric> <numeric> <numeric> <numeric>
ENSG00000000003 747.194195     -0.3507030  0.168246 -2.084470 0.0371175
ENSG00000000005   0.000000             NA        NA        NA        NA
ENSG00000000419 520.134160      0.2061078  0.101059  2.039475 0.0414026
ENSG00000000457 322.664844      0.0245269  0.145145  0.168982 0.8658106
ENSG00000000460  87.682625     -0.1471420  0.257007 -0.572521 0.5669691
ENSG00000000938   0.319167     -1.7322890  3.493601 -0.495846 0.6200029
                     padj
                <numeric>
ENSG00000000003  0.163035
ENSG00000000005        NA
ENSG00000000419  0.176032
ENSG00000000457  0.961694
ENSG00000000460  0.815849
ENSG00000000938        NA
```

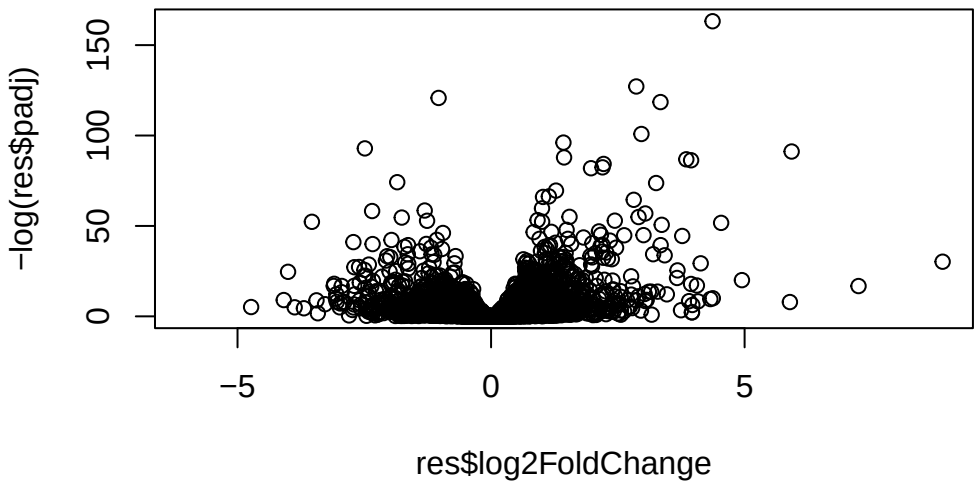
Volcano Plot

A very common and useful summary figure from this type of analysis is called a volcano plot - a plot of log2FC vs P-value. We use the `padj` the adjusted P-value for multiple testing

```
plot(res$log2FoldChange, res$padj)
```




```
plot(res$log2FoldChange, -log(res$padj))
```




```
# smaller p-value results in more extreme negative value
log(0.00005)
```

```
[1] -9.903488
```

```
log(0.5)
```

```
[1] -0.6931472
```

```
# Setup our custom point color vector
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2 ]  <- "red"

inds <- (res$padj < 0.01) & (abs(res$log2FoldChange) > 2 )
mycols[ inds ] <- "blue"

# Volcano plot with custom colors
plot( res$log2FoldChange,  -log(res$padj),
 col=mycols, ylab="-Log(P-value)", xlab="Log2(FoldChange)" )

# Cut-off lines
```

```r
abline(v=c(-2,2), col="gray", lty=2)
abline(h=-log(0.1), col="gray", lty=2)
```