Class 12

Mady Welch

Import countData and colData

```
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
```

Lets check the correspondence of the metadata and count data.

```
metadata$id
```

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

```
colnames(counts)
```

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

To check that they are in the same order we can use `==` test of equality and `all()`, which tells us if all of the outcomes are TRUE.

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

```
[1] TRUE
```

Q1. How many genes are in this dataset?

```
nrow(counts)
```

```
[1] 38694
```

- 38,694 genes are in the dataset.

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

```
table(metadata$dex)
```

```
control treated
      4       4
```

- We have 4 control cell lines.

Analysis via comparison of CONTROL vs TREATED

The "treated" have the dex drug and the "control" do not. First I need to be able to extract just the control columns in the counts data set.

```
control.inds <- metadata$dex == "control"
metadata[control.inds,]
```

```
         id     dex celltype     geo_id
1 SRR1039508 control   N61311 GSM1275862
3 SRR1039512 control  N052611 GSM1275866
5 SRR1039516 control  N080611 GSM1275870
7 SRR1039520 control  N061011 GSM1275874
```

This code chunk will find the sample ids for those labeled "control" and then calculate the mean counts per gene across these samples.

```
control <- metadata[metadata[,"dex"]=="control",]
control.counts <- counts[ ,control$id]
control.mean <- rowSums( control.counts )/4
head(control.mean)
```

```
ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
         900.75            0.00          520.50          339.75           97.25
ENSG00000000938
           0.75
```

There is another way we could do this: using the dplyr package from tidyverse.

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
control <- metadata %>% filter(dex=="control")
control.counts <- counts %>% select(control$id)
control.mean <- rowSums(control.counts)/4
head(control.mean)
```

```
ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
         900.75            0.00          520.50          339.75           97.25
ENSG00000000938
           0.75
```

Q3. How would you make the above code in either approach more robust?

- The exact code above would not give the correct mean values becuase it include `/4`, which only works in this case because there are 4 control cell lines. If we added more data, we would need to change the code to divide by the new number of control cell lines.

Q4. Follow the same procedure for the treated samples (i.e. calculate the mean per gene across drug treated samples and assign to a labeled vector called treated.mean)

```
treated <- metadata[metadata[,"dex"]=="treated",]
treated.counts <- counts[ ,treated$id]
treated.mean <- rowSums( treated.counts )/4
head(treated.mean)
```

```
ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
         658.00            0.00          546.00          316.50           78.75
ENSG00000000938
           0.00
```

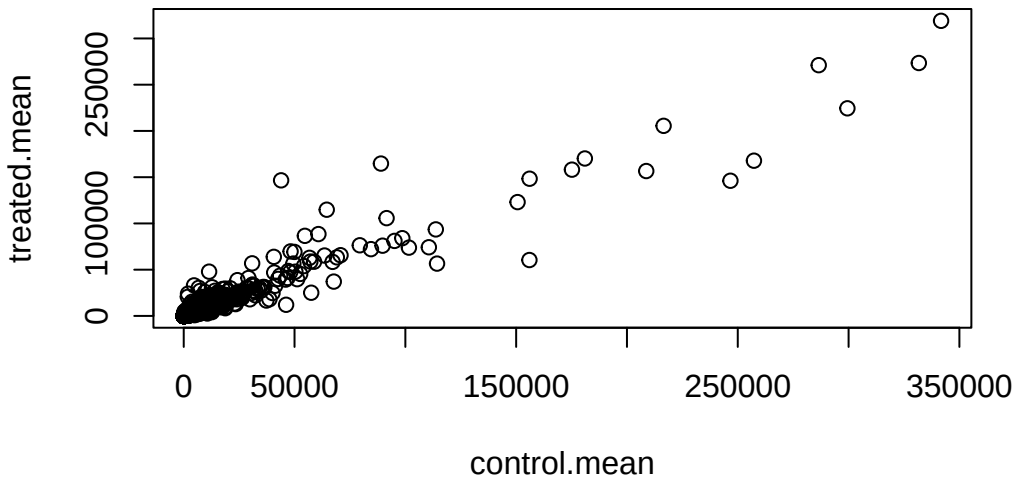
We can combine the meancount data:

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

```
control.mean treated.mean
    23005324     22196524
```

Q5. a) Create a scatter plot showing the mean of the treated samples against the mean of the control samples.

```
plot(meancounts)
```



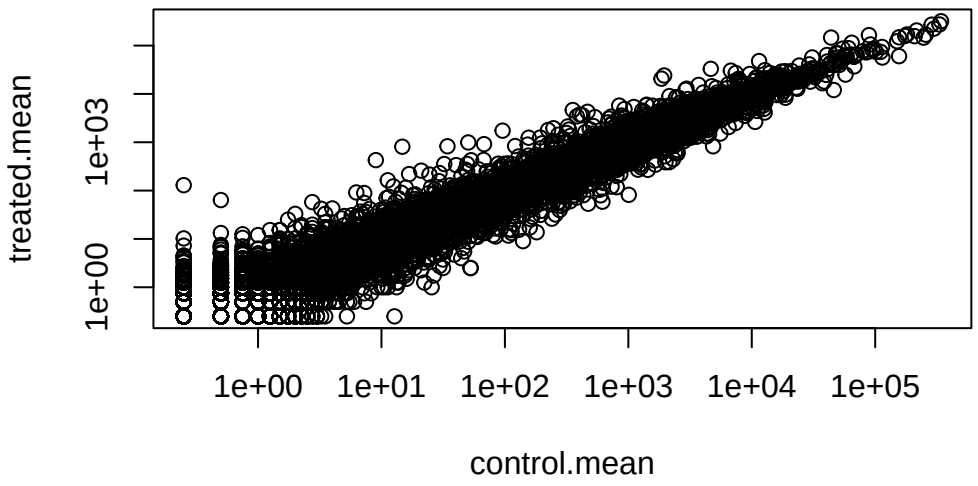
Q6. Try plotting both axes on a log sclae.

This is very heavily skewed and over a wide range… this calls for a log transformation

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

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

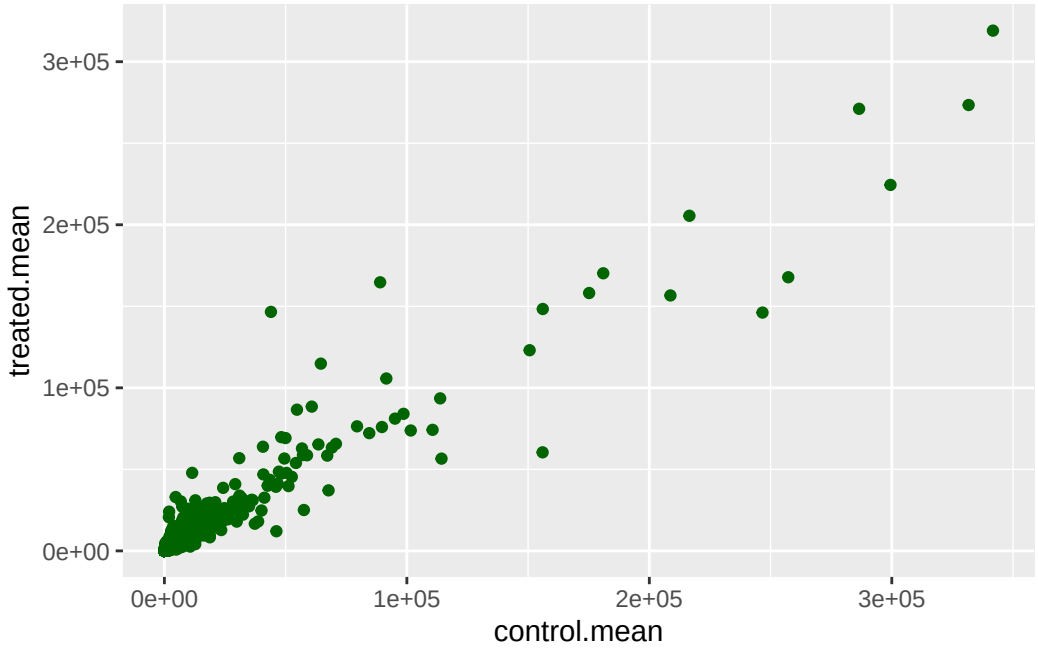
```
Warning in xy.coords(x, y, xlabel, ylabel, log): 15281 y values <= 0 omitted
from logarithmic plot
```



Q5. b) .You could also use the ggplot2 package to make this figure producing the plot below. What geom_?() function would you use for this plot?

- geom_point.

```
library(ggplot2)
ggplot(meancounts) +
  aes(x=control.mean, y=treated.mean) +
  geom_point(col = "dark green")
```



We like working with log transformed data as it can help make things more straightforward to interpret.

If we have no change:

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

```
[1] 0
```

What if we had a doubling?

```
log2(40/20)
```

```
[1] 1
```

Half?

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

```
[1] -1
```

We like working with log2 fold-change values. Let's calculate them for our data.

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

- There are some weird results such as NaN (not a number) and -Inf (negative infinity).

Let's filter our data to remove these genes (rows):

```
to.keep.inds <- rowSums(meancounts[, 1:2] == 0) == 0
head(to.keep.inds)
```

```
ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
           TRUE           FALSE            TRUE            TRUE            TRUE
ENSG00000000938
          FALSE
```

```
mycounts <- meancounts[to.keep.inds, ]
nrow(mycounts)
```

```
[1] 21817
```

A common threshold for calling genes as differentially expressed is a log2 fold-change of +2 or -2.

```
sum(mycounts$log2fc >= +2)
```

```
[1] 314
```

What % is this?

```
sum(mycounts$log2fc >= +2)/nrow(mycounts) *100
```

```
[1] 1.439245
```

Q8. How many up regulated genes we have at the greater than 2 fc level?

```
sum(mycounts$log2fc > +2)
```

```
[1] 250
```

What proportion is downregulated?

```
sum(mycounts$log2fc <= -2)/nrow(mycounts) *100
```

```
[1] 2.223037
```

Q9. How many down regulated genes we have at the greater than 2 fc level?

```
sum(mycounts$log2fc > -2)
```

```
[1] 21332
```

Q7. What is the purpose of the arr.ind argument in the which() function call above? Why would we then take the first column of the output and need to call the unique() function?

- `arr.ind` calls for values that are TRUE, where the genes and sample have 0 counts. Calling `unique()` makes sure we don't double count any rows that have 0 counts in both samples.

We need some stats to check if the drug induced difference is significant.

Turn to DESeq2

Load DESeq2:

```
library(DESeq2)
```

```
Warning: package 'matrixStats' was built under R version 4.2.2
```

The main function in the DESeq2 package is called `deseq()`. It wants our count data and our colData (metadata) as input in a specific way.

```
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
```

Now I can run the DESeq analysis

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

```
estimating size factors
```

```
estimating dispersions
```

```
gene-wise dispersion estimates
```

```
mean-dispersion relationship
```

```
final dispersion estimates
```

```
fitting model and testing
```

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

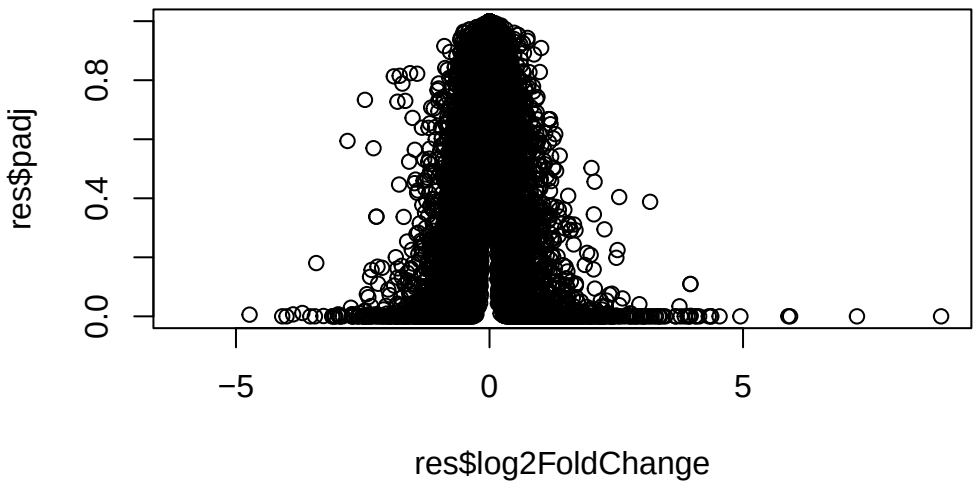
```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 38694 rows and 6 columns
                 baseMean log2FoldChange     lfcSE      stat    pvalue
                <numeric>      <numeric> <numeric> <numeric> <numeric>
ENSG00000000003  747.1942     -0.3507030  0.168246 -2.084470 0.0371175
ENSG00000000005    0.0000             NA        NA        NA        NA
ENSG00000000419  520.1342      0.2061078  0.101059  2.039475 0.0414026
ENSG00000000457  322.6648      0.0245269  0.145145  0.168982 0.8658106
ENSG00000000460   87.6826     -0.1471420  0.257007 -0.572521 0.5669691
...                    ...            ...       ...       ...       ...
ENSG00000283115  0.000000             NA        NA        NA        NA
ENSG00000283116  0.000000             NA        NA        NA        NA
ENSG00000283119  0.000000             NA        NA        NA        NA
ENSG00000283120  0.974916      -0.668258   1.69456 -0.394354  0.693319
ENSG00000283123  0.000000             NA        NA        NA        NA
                     padj
                <numeric>
ENSG00000000003  0.163035
ENSG00000000005        NA
ENSG00000000419  0.176032
ENSG00000000457  0.961694
ENSG00000000460  0.815849
...                   ...
ENSG00000283115        NA
ENSG00000283116        NA
ENSG00000283119        NA
ENSG00000283120        NA
ENSG00000283123        NA
```

What we have so far is the log2 fold-change and the adjusted p-value for significance.

Data Visualization

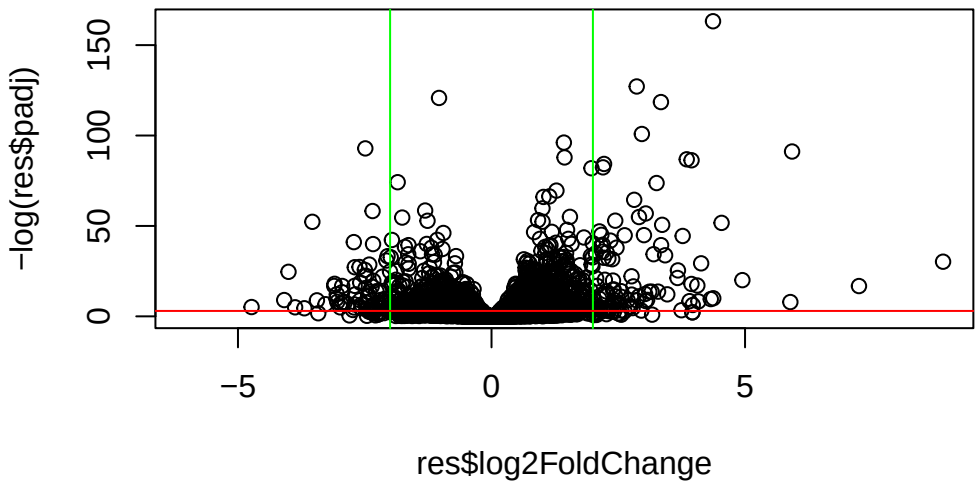
A first plot:

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



All the interesting p-values are down below zero... I'm going to take the log of the p-value Note: Take the -log to flip the plot.

```
plot(res$log2FoldChange, -log(res$padj))
abline(v = c(-2, 2), col = "green")
abline(h = -log(0.05), col = "red")
```



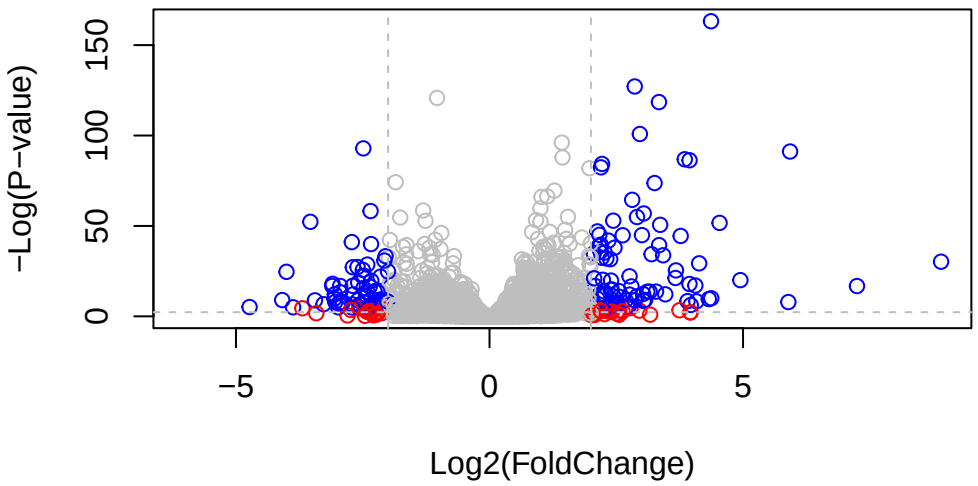
Setup our custom print color vector:

```
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 and cut-off lines:

```
plot( res$log2FoldChange,  -log(res$padj),
 col=mycols, ylab="-Log(P-value)", xlab="Log2(FoldChange)" )
abline(v=c(-2,2), col="gray", lty=2)
abline(h=-log(0.1), col="gray", lty=2)
```


−Log(P−value)
0
50
100
150
−5
0
5
Log2(FoldChange)