Class 12: Differential Expression Analysis

Patrick Tran

Analysis of RNA-Seq data

The data for this hands-on session comes from a published RNA-seq experiment where airway smooth muscle cells were treated with dexamethasone, a synthetic glucocorticoid steroid with anti-inflammatory effects (Himes et al. 2014).

Import countData and colData

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

Take a look at each.

```
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
```

Q1. How many genes are in this dataset?

```
nrow(counts)
```

```
[1] 38694
```

38694 genes are in this dataset.

```
ncol(counts)
```

```
[1] 8
```

and the metadata a.k.a "colData"

```
(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
7 SRR1039520 control  N061011 GSM1275874
8 SRR1039521 treated  N061011 GSM1275875
```

Let's make sure that the id column of the metadata match the order of the columns in countData.

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

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

We can use the `all()` function to check that all it's inputs are true.

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

```
[1] FALSE
```

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

```
[1] TRUE
```

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

There are 4 control cell lines.

Analysis by hand

```
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
7 SRR1039520 control  N061011 GSM1275874
8 SRR1039521 treated  N061011 GSM1275875
```

Let's first extract our counts for control samples as I want to compare this to the counts for treated (i.e with drug) samples.

```
control.inds <- metadata$dex == "control"
control.ids <- metadata$id[control.inds]
control.counts <- counts[, control.ids]
head(control.counts)
```

```
                SRR1039508 SRR1039512 SRR1039516 SRR1039520
ENSG00000000003        723        904       1170        806
ENSG00000000005          0          0          0          0
ENSG00000000419        467        616        582        417
ENSG00000000457        347        364        318        330
ENSG00000000460         96         73        118        102
ENSG00000000938          0          1          2          0
```

I want a single summary counts value for each gene in the control experiments. I will start by taking the average.

```
control.mean <- rowMeans(control.counts)
head(control.mean)
```

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

```
#apply(control.counts, 1, mean)
```

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

Instead of dividing the sum by the amount of samples, you can use the `rowMeans()` call or use `apply(x,1,mean)`. This will give you the average without needing to find the n samples.

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`)

Let's get the average for all of our treated samples.

```
treated.inds <- metadata$dex == "treated"
treated.ids <- metadata$id[treated.inds]
treated.counts <- counts[, treated.ids]
treated.mean <- rowMeans(treated.counts)
head(treated.mean)
```

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

To help us stay organized let's make a new data.frame to store these results together.

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

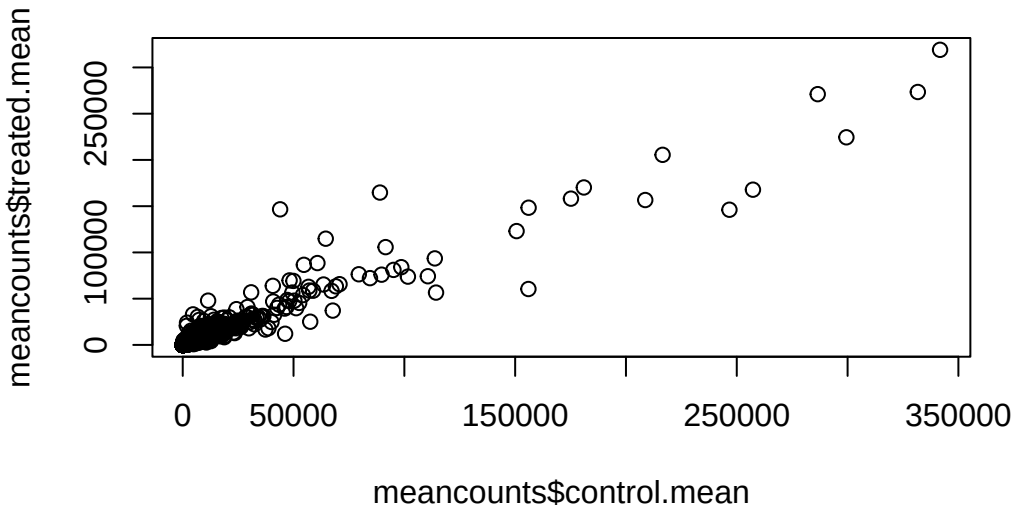
```
                control.mean treated.mean
ENSG00000000003       900.75       658.00
ENSG00000000005         0.00         0.00
```

```
ENSG00000000419       520.50       546.00
ENSG00000000457       339.75       316.50
ENSG00000000460        97.25        78.75
ENSG00000000938         0.75         0.00
```

And make a wee plot

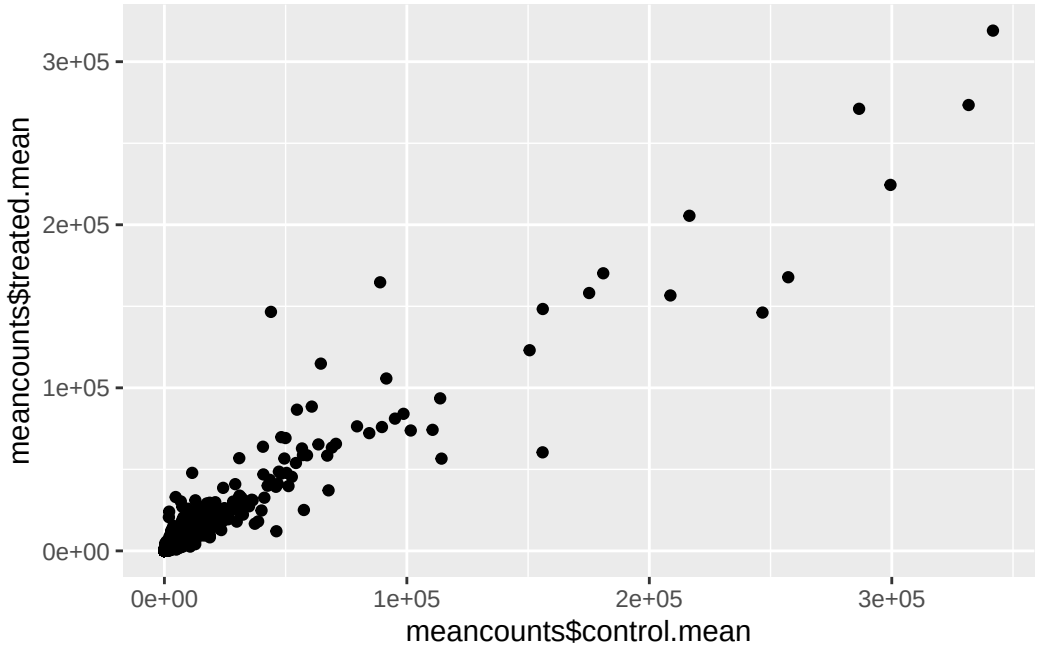
Q5 (a). Create a scatter plot showing the mean of the treated samples against the mean of the control samples. Your plot should look something like the following.

```
plot(meancounts$control.mean, meancounts$treated.mean)
```



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?

```
library(ggplot2)
ggplot(meancounts, aes(meancounts$control.mean, meancounts$treated.mean)) +
  geom_point()
```



You would use `geom_point()`.

This screams for a log transformation so we can see our data.

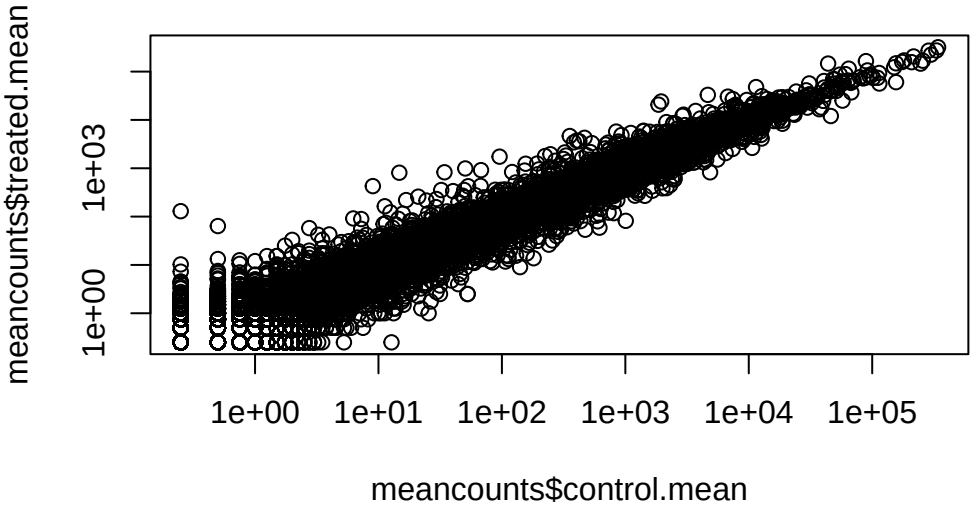
> Q6. Try plotting both axes on a log scale. What is the argument to plot() that allows you to do this?

The `log="xy"` argument for plot() allows for plotting both axes on a log scale.

```
plot(meancounts$control.mean, meancounts$treated.mean, 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
```



The most useful and most straight forward to understand is log2 transform.

```
20/20
```

```
[1] 1
```

Doubling

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

```
[1] 1
```

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

```
[1] -1
```

```
log2(80/20)
```

```
[1] 2
```

add a "log2 fold-change"

```
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
```

Hmmm... we need to get rid of the genes where we have no count data as taking the log2 of these 0 counts does not tell us anything.

```
head( meancounts == 0)
```

```
                control.mean treated.mean log2fc
ENSG00000000003        FALSE        FALSE  FALSE
ENSG00000000005         TRUE         TRUE     NA
ENSG00000000419        FALSE        FALSE  FALSE
ENSG00000000457        FALSE        FALSE  FALSE
ENSG00000000460        FALSE        FALSE  FALSE
ENSG00000000938        FALSE         TRUE  FALSE
```

```
#zero.vals <- which(meancounts[,1:2]==0, arr.ind=TRUE)
#to.rm <- unique(zero.vals[,1])
#mycounts <- meancounts[-to.rm,]
#head(mycounts)

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

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

```
                control.mean treated.mean      log2fc
ENSG00000000003       900.75       658.00 -0.45303916
ENSG00000000419       520.50       546.00  0.06900279
ENSG00000000457       339.75       316.50 -0.10226805
ENSG00000000460        97.25        78.75 -0.30441833
ENSG00000000971      5219.00      6687.50  0.35769358
ENSG00000001036      2327.00      1785.75 -0.38194109
```

> 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?

The `arr.ind=TRUE` argument in the which() function call returns the row and column indicies where TRUE values exist. It tells which genes and samples have zero counts so we can ignore them. The `unique()` function makes sure that there are no repetitive counting of the rows.

Let's see how many are up-regulated at the log2fc level of +2

> Q8. Using the up.ind vector above can you determine how many up regulated genes we have at the greater than 2 fc level?

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

```
[1] 314
```

and down regulated...

> Q9. Using the down.ind vector above can you determine how many down regulated genes we have at the greater than 2 fc level?

```
sum(mycounts$log2fc <= -2)
```

```
[1] 485
```

> Q10. Do you trust these results? Why or why not?

No. The analysis is based on fold change. Fold change measures the magnitude of quantity change between an original and subsequent measurement. The value can be large without being statistically significant. We need to analyze statistical significants and the results as of now can be misleading.

We are missing the stats. Are these big changes significant?

DESeq2 analysis

```
library(DESeq2)
```

Like most bioconductor packages DESeq wants it's input and output in a very specific format.

```
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
```

```
dds
```

```
class: DESeqDataSet
dim: 38694 8
metadata(1): version
assays(1): counts
rownames(38694): ENSG00000000003 ENSG00000000005 ... ENSG00000283120
  ENSG00000283123
rowData names(0):
colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
colData names(4): id dex celltype geo_id
```

The main DESeq function is called DESeq

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

```
estimating size factors
```

```
estimating dispersions
```

```
gene-wise dispersion estimates
```

```
mean-dispersion relationship

final dispersion estimates

fitting model and testing
```

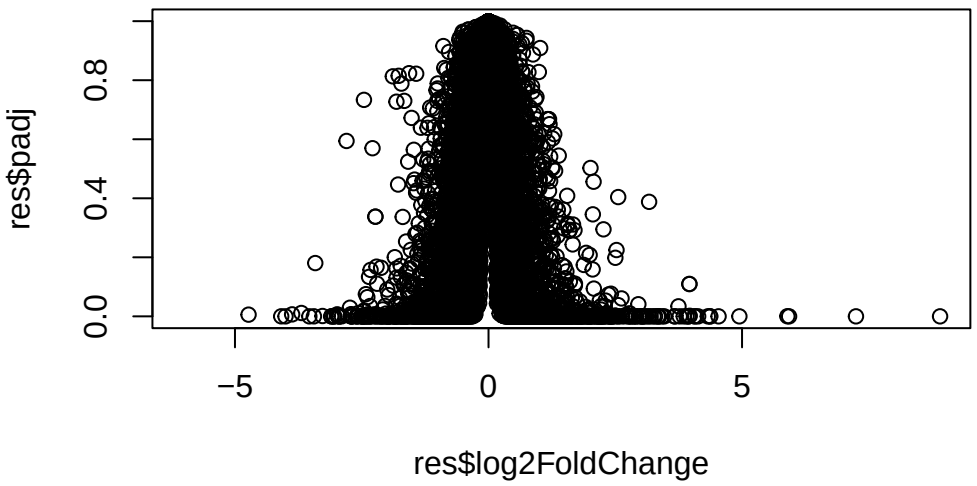
```
res <- results(dds)
head(res)
```

```
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 plots

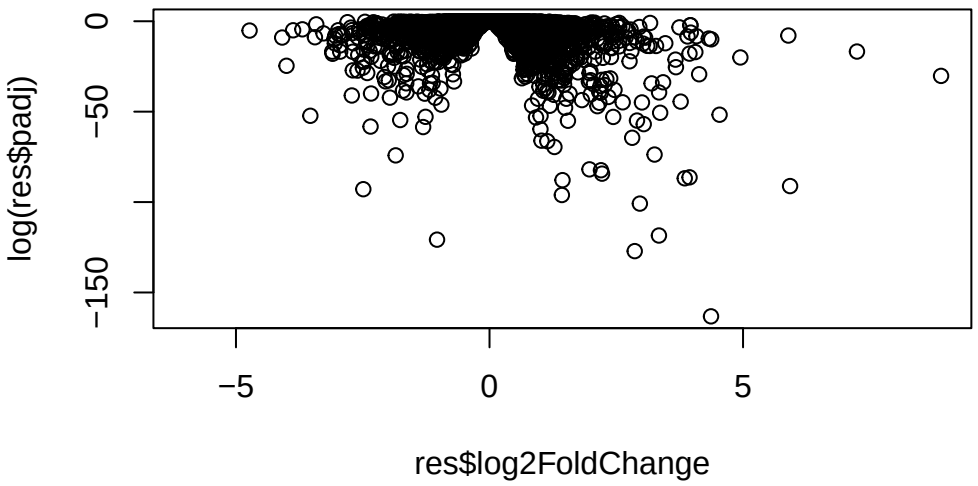
A major summary figure of this type of analysis is called a volcano plot - the idea here is to keep our inner biologist and inner stats person happy with one cool plot!

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



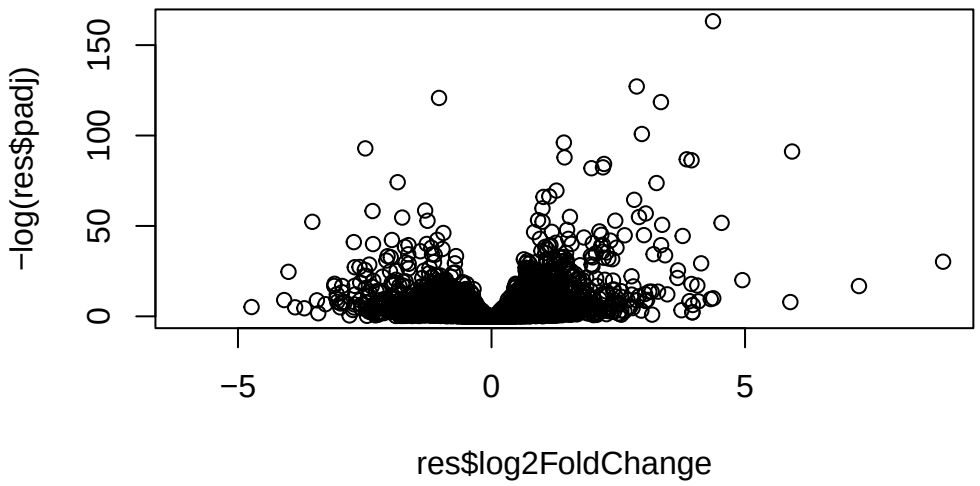
Improve this plot by taking the log of that p-value axis

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



I want to flip this y-axis so the values I care about (i.e. the low p-value or high log(p-values)) are at the top of the axis.

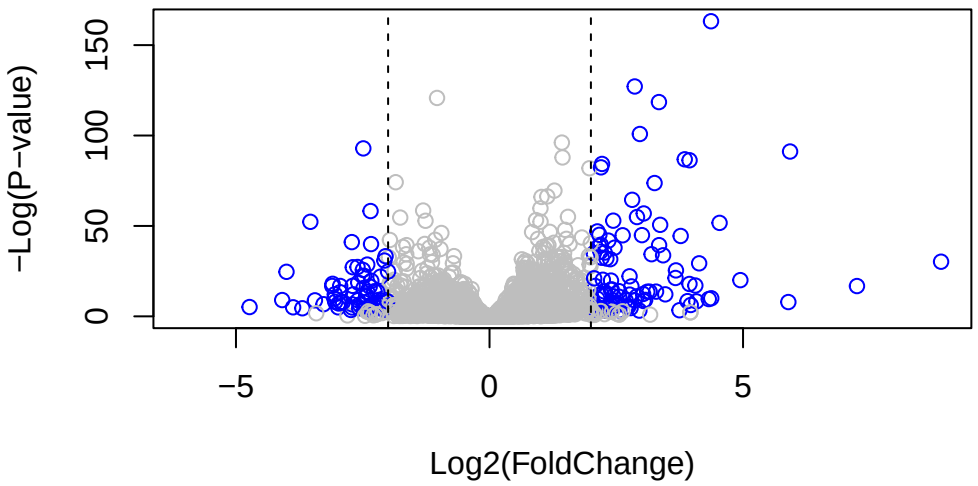
```
plot(res$log2FoldChange, -log(res$padj))
```



Add color to the volcano plot by setting up a custom color vector.

```
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) >= 2] <- "blue"
#mycols[ abs(res$log2FoldChange) > 2 ]  <- "red"
mycols[res$padj > 0.05] <- "gray"
#inds <- (res$padj < 0.01) & (abs(res$log2FoldChange) > 2 )
#mycols[ inds ] <- "blue"

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


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