

Class 15 RNASeq Analysis

Angelita Rivera (PID A15522236)

11/16/2021

Background

Our data for today come from Himes et al. RNASeq analysis of the drug dexamethasone, a synthetic glucocorticoid steroid with anti-inflammatory effects (Himes et al. 2014).

Read the 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
```

Q1. How many genes are in this dataset?

```
nrow(counts)
```

```
## [1] 38694
```

There are 38694 genes in this dataset.

Q2. How many ‘control’ cell lines do we have?

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

```
## [1] 4
```

#We need to look at the metadata, the dex column.

#Then, we use the "==" to see what values in the dataset are control. #Then, we use "sum()" around the w

We have 4 ‘control’ cell lines.

First I need to extract all the “control” columns. Then I will take the rowise mean to get the average count values for all genes in these four experiments.

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

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

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

You could simplify the code to take the mean in a more general way, as shown above. The problem with getting the mean in the two codes (from the lab workbook, where you divide by 4) is that if our dataset changes (i.e. we get more or less data values), the mean will not be correct. We can simplify, or make the code more robust, if we use a more straightforward, general approach when calculating the mean (so it is applicable to a lot of different scenarios).

Now do the same for the drug treated experiments (i.e. columns)

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.ind <- metadata$dex == "treated"  
treated.counts <- counts[ , treated.ind]  
head(treated.counts)
```

```

##          SRR1039509 SRR1039513 SRR1039517 SRR1039521
## ENSG000000000003      486       445      1097      604
## ENSG000000000005       0         0         0         0
## ENSG00000000419      523       371      781      509
## ENSG00000000457      258       237      447      324
## ENSG00000000460       81        66       94       74
## ENSG00000000938       0         0         0         0
treated.mean <- rowMeans(treated.counts)

```

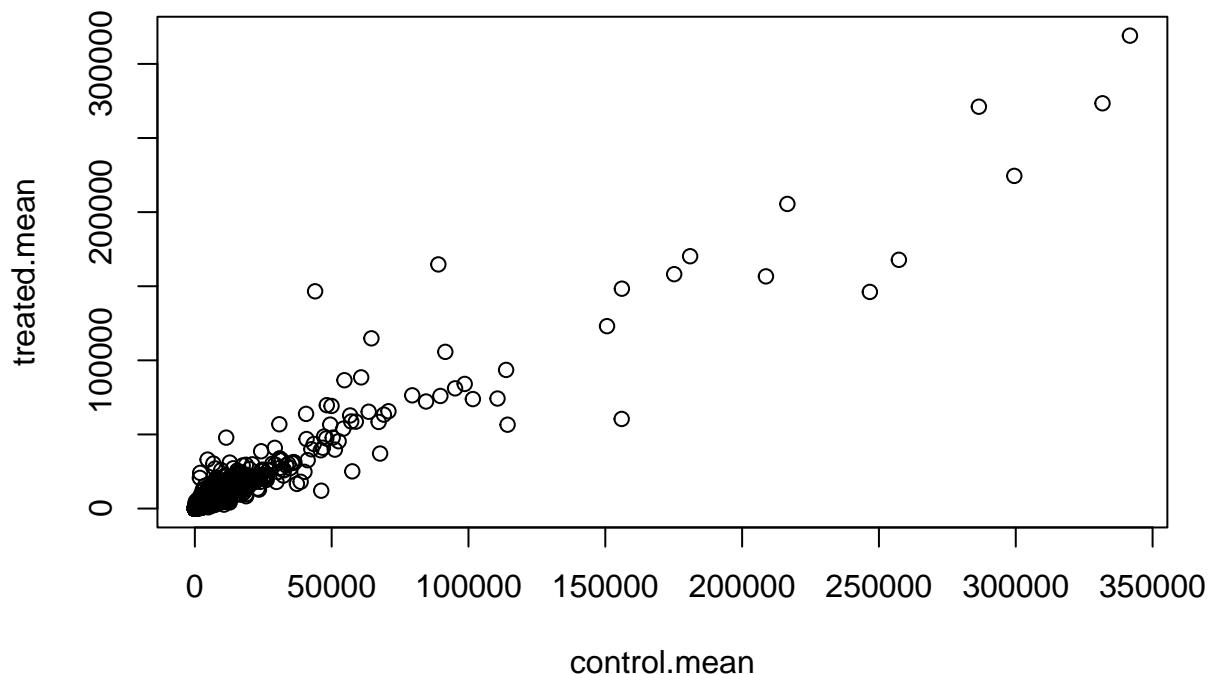
We will combine our meancount data for bookkeeping purposes.

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

Let's make a quick 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)
```



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?

You would use the `geom_point()` function.

We need a log transformation to see details of our data!

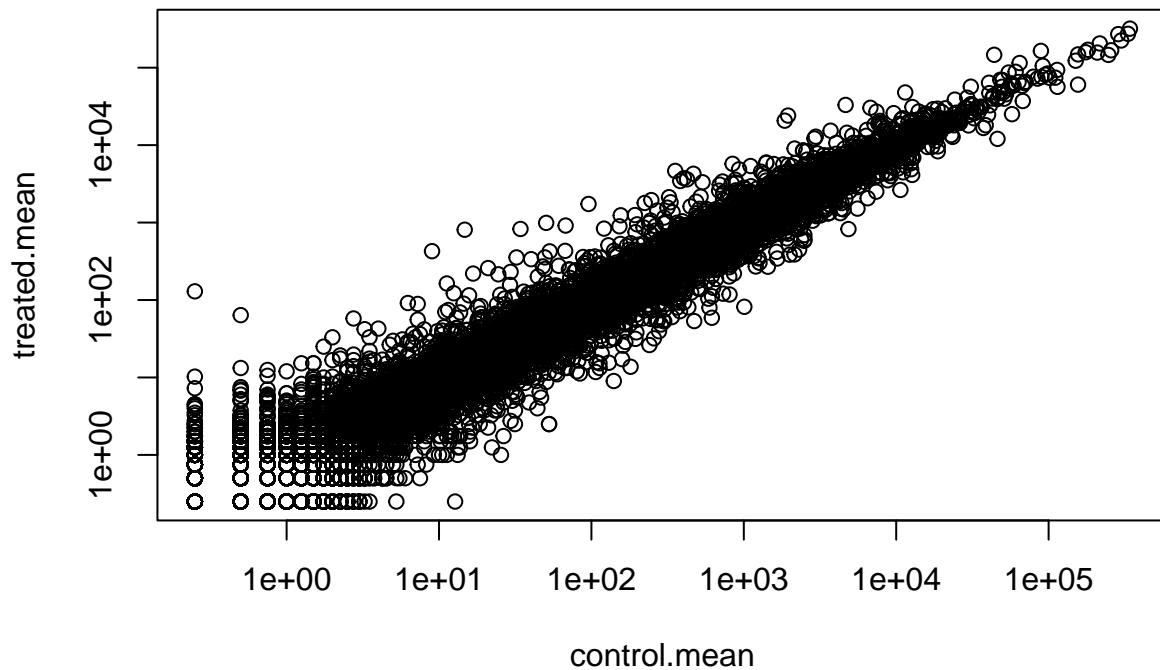
I am going to replot the plot; this time using a log scale!

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

```
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
```



The argument that allows us to put both axes on a log scale is `log = “xy”`.

We often use `log2` in this field because it has nice math properties that make interpretation easier.

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

```
## [1] 0
```

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

```
## [1] 2
```

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

```
## [1] -1
```

Cool, we see 0 values for no change and + values for increases and - values for decreases. This nice property leads us to work with **log2(fold-change)** all the time in the genomics and proteomics field.

Let's add the **log2(fold-change)** values to our 'meancounts' dataframe.

```
meancounts$log2fc <- log2(meancounts[,"treated.mean"]/meancounts[,"control.mean"])
head(meancounts)
```

	control.mean	treated.mean	log2fc
## ENSG000000000003	900.75	658.00	-0.45303916
## ENSG000000000005	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

I need to exclude the genes (i.e. rows) with zero counts because we can't say anything about these as we have no data for them!

```
head(meancounts[,1:2])
```

	control.mean	treated.mean
## ENSG000000000003	900.75	658.00
## ENSG000000000005	0.00	0.00
## ENSG00000000419	520.50	546.00
## ENSG00000000457	339.75	316.50
## ENSG00000000460	97.25	78.75
## ENSG00000000938	0.75	0.00

```
head(meancounts[,1:2] == 0)
```

	control.mean	treated.mean
## ENSG000000000003	FALSE	FALSE
## ENSG000000000005	TRUE	TRUE
## ENSG00000000419	FALSE	FALSE
## ENSG00000000457	FALSE	FALSE
## ENSG00000000460	FALSE	FALSE
## ENSG00000000938	FALSE	TRUE

```
which(c(F,F,T,T))
```

```
## [1] 3 4
```

I can use the `which()` function with the ‘arr.ind=TRUE’ argument to get the columns and rows where TRUE values are (i.e. the zero counts in our case).

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

```
##           row col
## ENSG00000000005    2   1
## ENSG00000004848   65   1
## ENSG00000004948   70   1
## ENSG00000005001   73   1
## ENSG00000006059  121   1
## ENSG00000006071  123   1
```

```
to.rm <- unique(zero.vals[, "row"])
head(sort(to.rm))
```

```
## [1] 2 6 65 70 73 81
```

Now remove these from our ‘meancounts’ dataframe.

```
mycounts <- meancounts[-to.rm,]
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
```

How many do we have left?

```
nrow(mycounts)
```

```
## [1] 21817
```

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?

It will tell you the row and column positions where the TRUE values (zeros in our case) are.

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

Upregulated:

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

```
## [1] 250
```

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

Downregulated:

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

```
## [1] 367
```

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

Probably not. This is because we have not analyzed whether or not any/all of the differences we have seen are significant (based on p-values).

DESeq2 analysis

Let's do this the right way. DESeq2 is an R package specifically for analyzing count-based NGS data like RNA-seq. It is available from Bioconductor.

```
library(DESeq2)
```

```
## Loading required package: S4Vectors
```

```
## Loading required package: stats4
```

```
## Loading required package: BiocGenerics
```

```
##
```

```
## Attaching package: 'BiocGenerics'
```

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

```
##
```

```
##      IQR, mad, sd, var, xtabs
```

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

```
##
```

```
##      anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##      dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##      grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##      order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##      rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##      union, unique, unsplit, which.max, which.min
```

```
##
```

```
## Attaching package: 'S4Vectors'
```

```

## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname

## Loading required package: IRanges

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: SummarizedExperiment

## Loading required package: MatrixGenerics

## Loading required package: matrixStats

##
## Attaching package: 'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars

## Loading required package: Biobase

## Welcome to Bioconductor
##
## Vignettes contain introductory material; view with
## 'browseVignettes()'. To cite Bioconductor, see
## 'citation("Biobase")', and for packages 'citation("pkgname")'.

##
## Attaching package: 'Biobase'

## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians

```

```

## The following objects are masked from 'package:matrixStats':
##
##     anyMissing, rowMedians

```

We need to first setup the input object for 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

```

```
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

```

Now we can run DESeq analysis:

```

dds <- DESeq(dds)

## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

```

To get at the results, here we use the deseq ‘results()’ function:

```

res <- results(dds)
head(res)

```

```

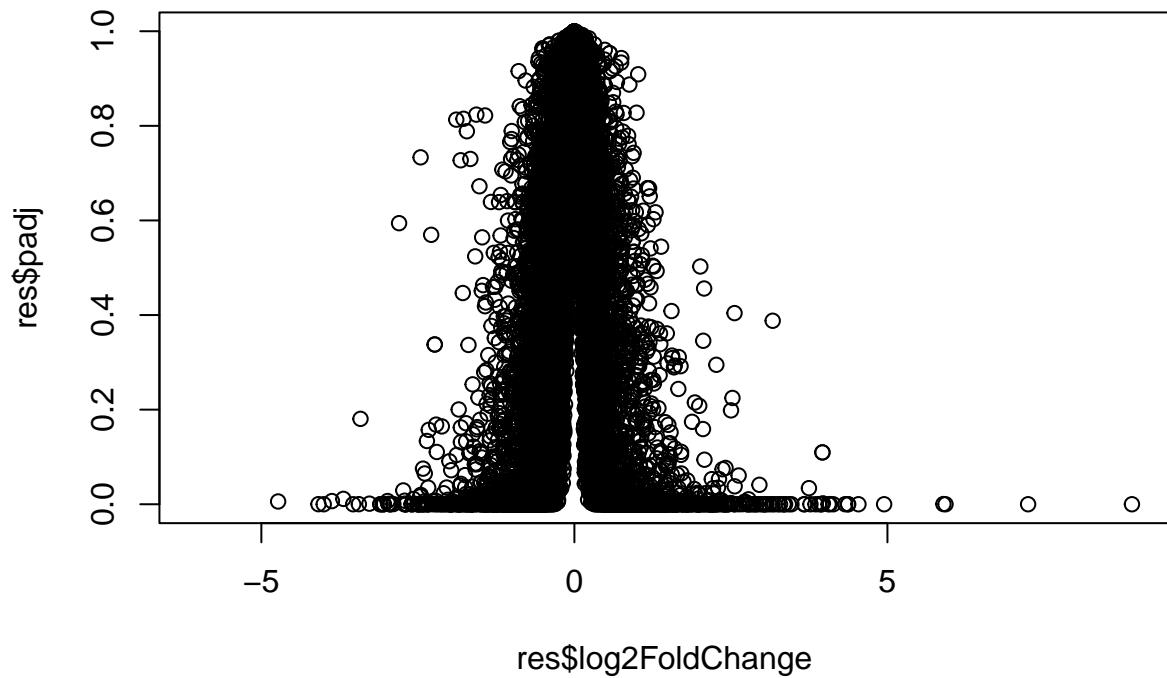
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 6 rows and 6 columns
##           baseMean log2FoldChange      lfcSE      stat     pvalue
##           <numeric>      <numeric> <numeric> <numeric> <numeric>
## ENSG00000000003  747.194195 -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

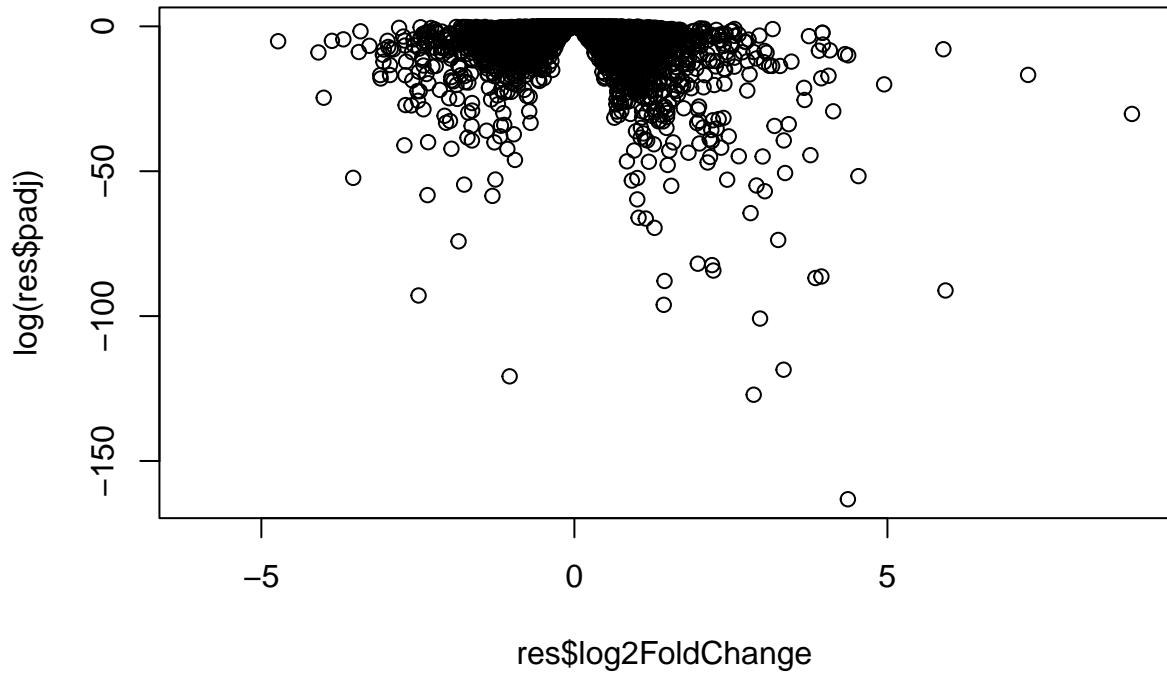
Let's make a commonly produced visualization from this data, namely a so-called Volcano plot. These summary figures are frequently used to highlight the proportion of genes that are both significantly regulated and display a high fold change.

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



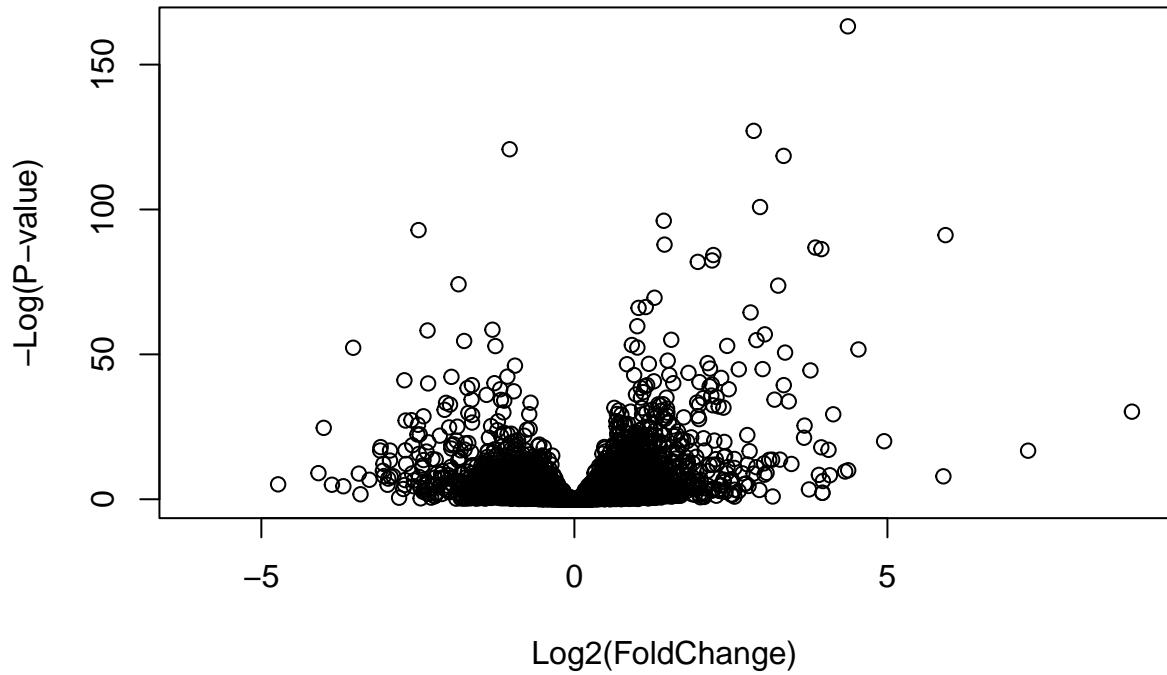
That is not a useful plot because all the small p-values are hidden at the bottom of the plot and we can't really see them. Log will help.

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



Getting better... We can flip this p-value axis by just putting a minus sign on it then we will have the classic volcano plot used by the rest of the world. We can also change the labels.

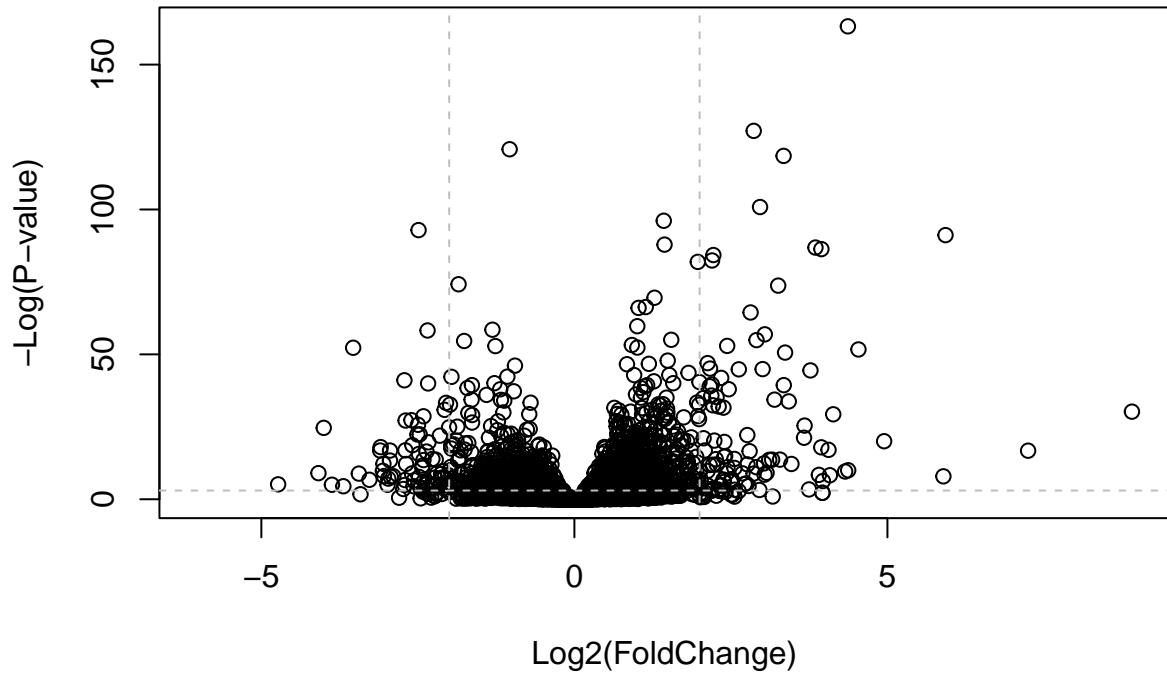
```
plot( res$log2FoldChange, -log(res$padj),  
      xlab="Log2(FoldChange)",  
      ylab="-Log(P-value)")
```



Finally, let's add some lines and color to this point to draw attention to the genes (i.e. points) we care about - that is those with large fold-change and low p-values (i.e. high -log(p-values)).

```
plot( res$log2FoldChange, -log(res$padj),
      ylab="-Log(P-value)", xlab="Log2(FoldChange)")

# Add some cut-off lines
abline(v=c(-2,2), col="gray", lty=2)
abline(h=-log(0.05), col="gray", lty=2)
```



```

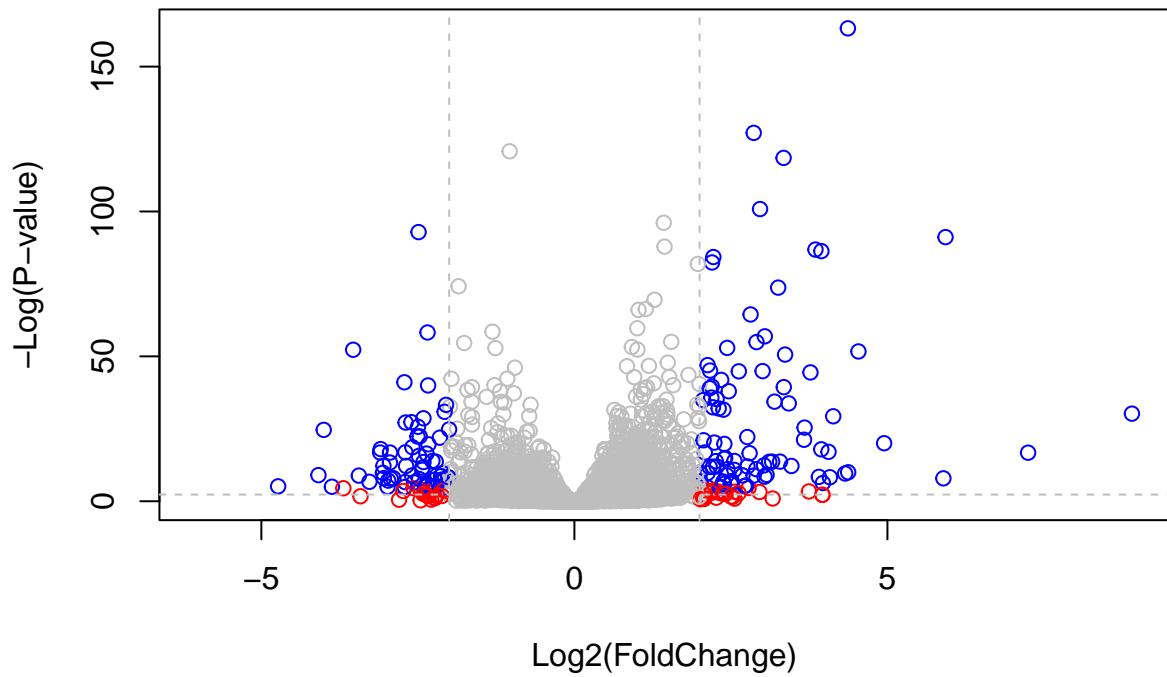
# Setup our custom point color vector
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2 ] <- "red"

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

```



Yay!