Class15: RNASeq Analysis

Monica Lin (PID: A15524235)

11/16/2021

Background

Our data for today comes 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")
```

Let's have a look at these

```
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
```

```
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
```

Q1. How many genes are in this dataset?

```
nrow(counts)
```

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

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

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

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

4 control cell lines

Differential gene expression

The control samples are SRR1039508, SRR1039512, SRR1039516, and SRR1039520. This bit of code will first find the sample `id` for those labeled control. Then calculate the mean counts per gene across these samples. This is the walkthrough method from the hands-on lab worksheet.

```
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
```

Alternatively, extract all the "control" columns, then take the rowwise mean to get the average count values for all genes in these four experiments. This is the walkthrough method from in-person lab class.

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

```
##                 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.count)
```

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

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

```
##                 SRR1039509 SRR1039513 SRR1039517 SRR1039521
## ENSG00000000003        486        445       1097        604
## ENSG00000000005          0          0          0          0
## ENSG00000000419        523        371        781        509
## ENSG00000000457        258        237        447        324
## ENSG00000000460         81         66         94         74
## ENSG00000000938          0          0          0          0
```

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

In the lab worksheet example, there are only 4 control samples, which is why `control.mean` is calculated via `rowSums( control.counts )/4`. However, this is not robust, so in the lab class example, we altered this to `rowMeans(control.count)`.

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.mean <- rowSums (counts[ , treated$id]) /4
names(treated.mean) <- counts$ensgene
```

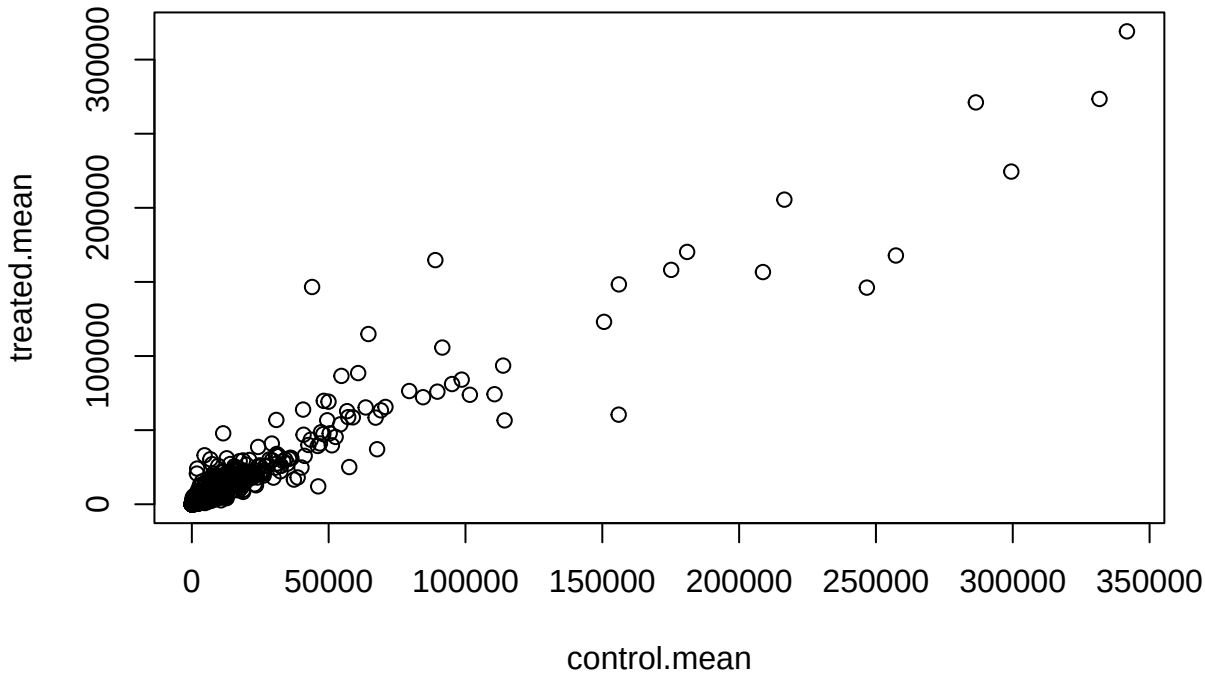
Combine the meancount data for bookkeeping purposes.

```
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)
```



Q5(b). You could also use the **ggplot2** package to make this figure.

Use the geom_point() function to make this plot.

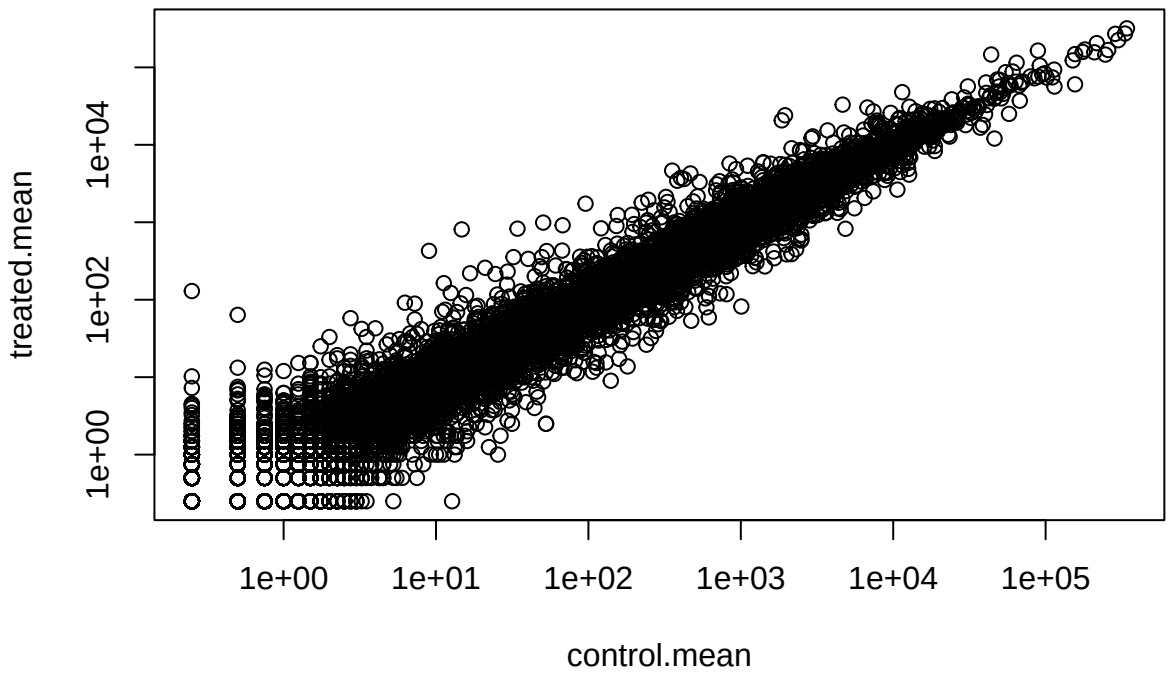
Q6. Try plotting both axes on a log scale. What is the argumen to **plot()** that allows you to do this?

This plot indicates that we need a log transformation to see details of our data. Replot on a log-log scale

```
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
```



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

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

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

This means there was no change; no deviation from 0. Control and treatment have the same effects.

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

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

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

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

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

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

Doubling gives us a value of 1. Quadrupling gives a value of 2. Halving gives a value of -1. This gives us log2 -fold changes, such as 2-fold changes when quadrupling.

We see 0 values for no change, + values for increases, and - values for decreases. This property lets us 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
## 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
```

Exclude the genes (i.e. rows) with zero values and -infinity.Need to remove these from the data because we can't say anything about these; there is no data for them.

```
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
```

I can use the **which()** function with the `arr.ind=TRUE` argument to get the columns and rows where the 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
```

```
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 genes 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?

The `arr.ind=TRUE` argument causes **which()** to return both the row and column indices (i.e. positions) where there are TRUE values. In this case, this will tel us which genes (rows) and samples (columns) have zero counts. We are going to ignore any genes that have zero counts in any sample so we just focus on the `row` answer. Calling **unique()** will ensure we don't count any row twice if it has zero entries in both samples.

A common threshold used for calling something differentially expressed is a log2(FoldChange) of greater than 2 or less than -2. Filter the dataset both ways to see how many genes are up- or down-regulated.

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

```
## [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?

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

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

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

All our analysis has been done based on fold change. However, fold change can be large (e.g. »two-fold up- or down-regulation) without being statistically significant (e.g. based on p-values). We have not done anything yet to determine whether the differences we are seeing are significant. These results in their current form are likely to be very misleading. Thus, the next section helps determine statistical significance of our results via the **DESeq2** package.

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

##
## Attaching package: 'IRanges'

## The following object is masked from 'package:grDevices':
##
##     windows

## 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
```

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
```

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

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

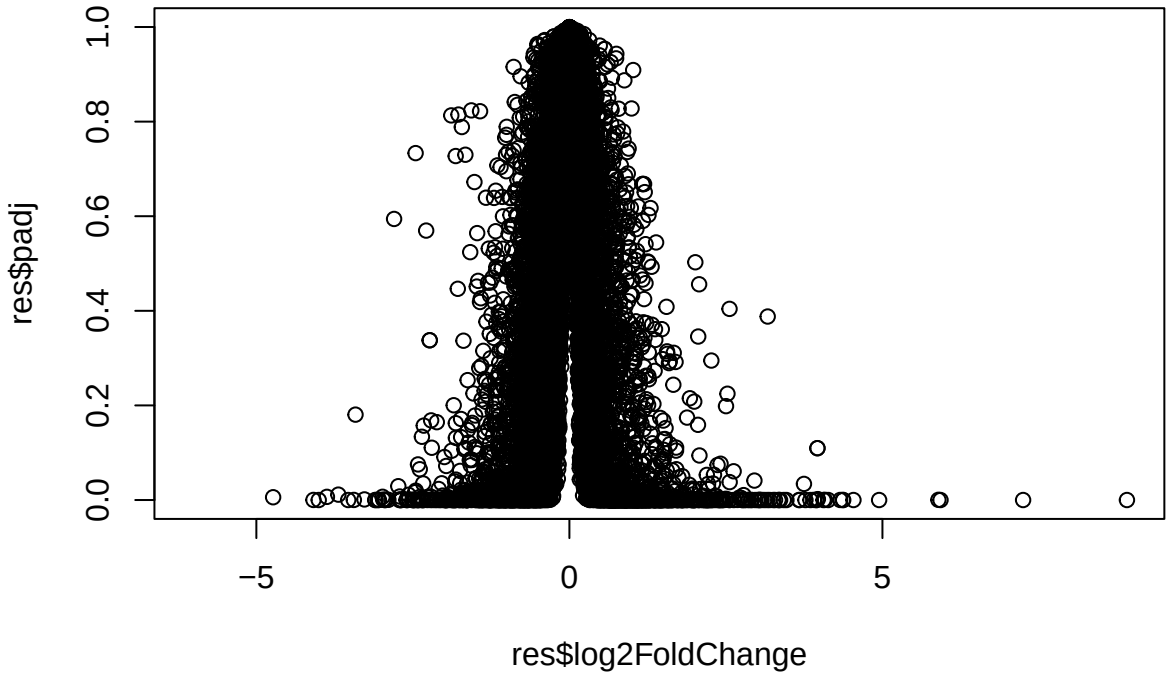
```
head(res)
```

```
## 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
```

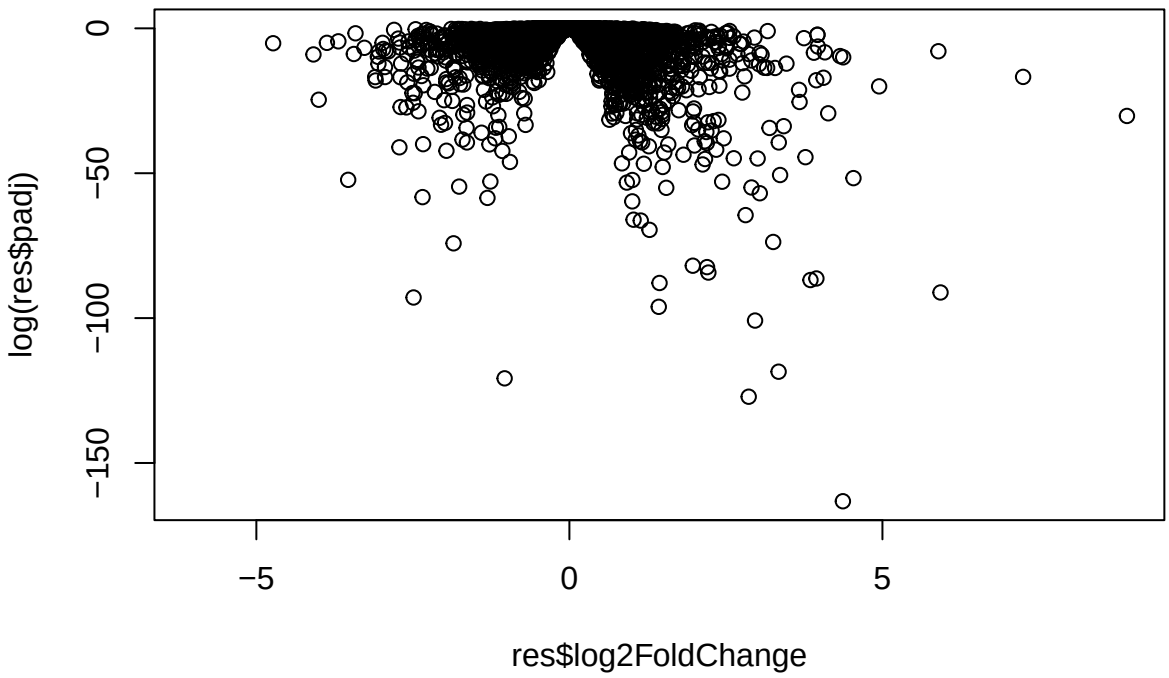
Volcano plots

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)
```

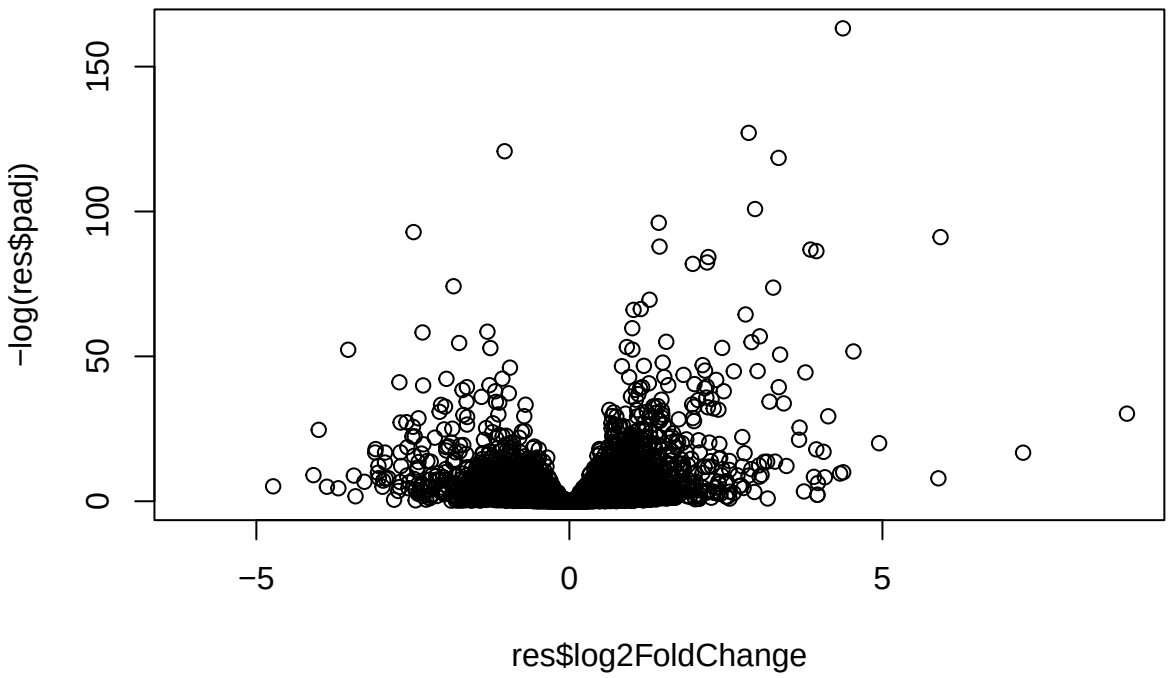



```
plot(res$log2FoldChange, log(res$padj))
```



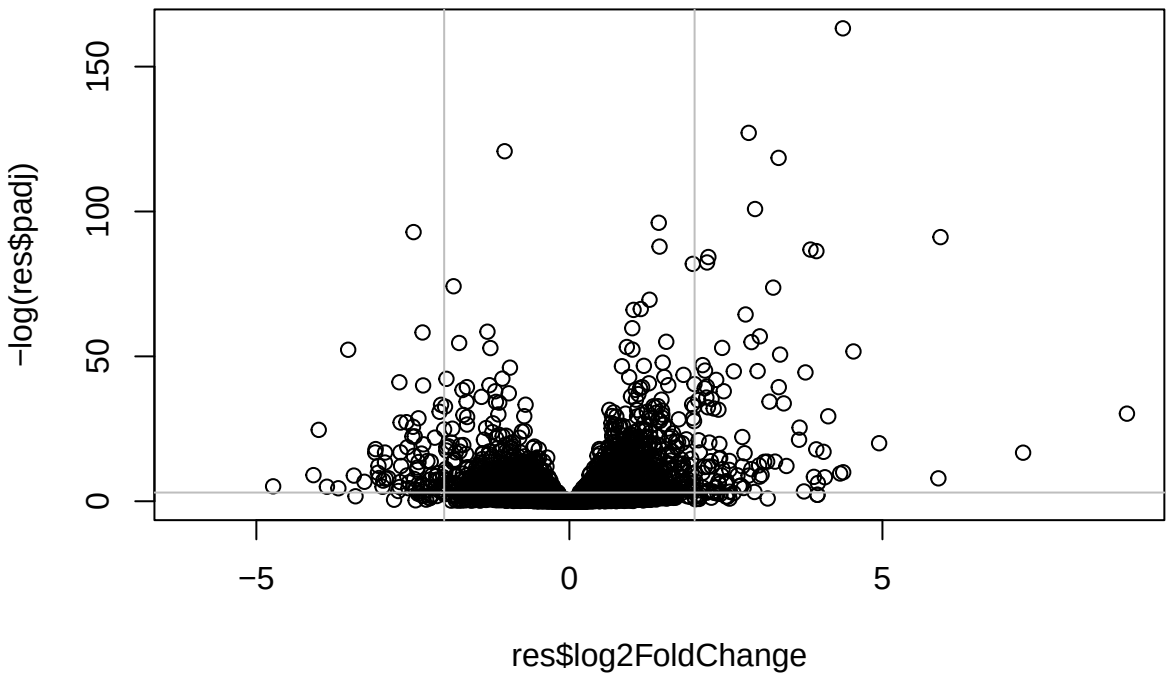
We can flip this pvalue axis by just putting a minus sign on it then we will have the classic volcano plot that the rest of the world uses.

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



Finally let's add some color to this plot to draw attention to the genes (i.e. points) we care about – that is, those with large fold-changes and low p-values (i.e. high -log(pvalues)). We are focusing on the points in the left and rightmost quadrants.

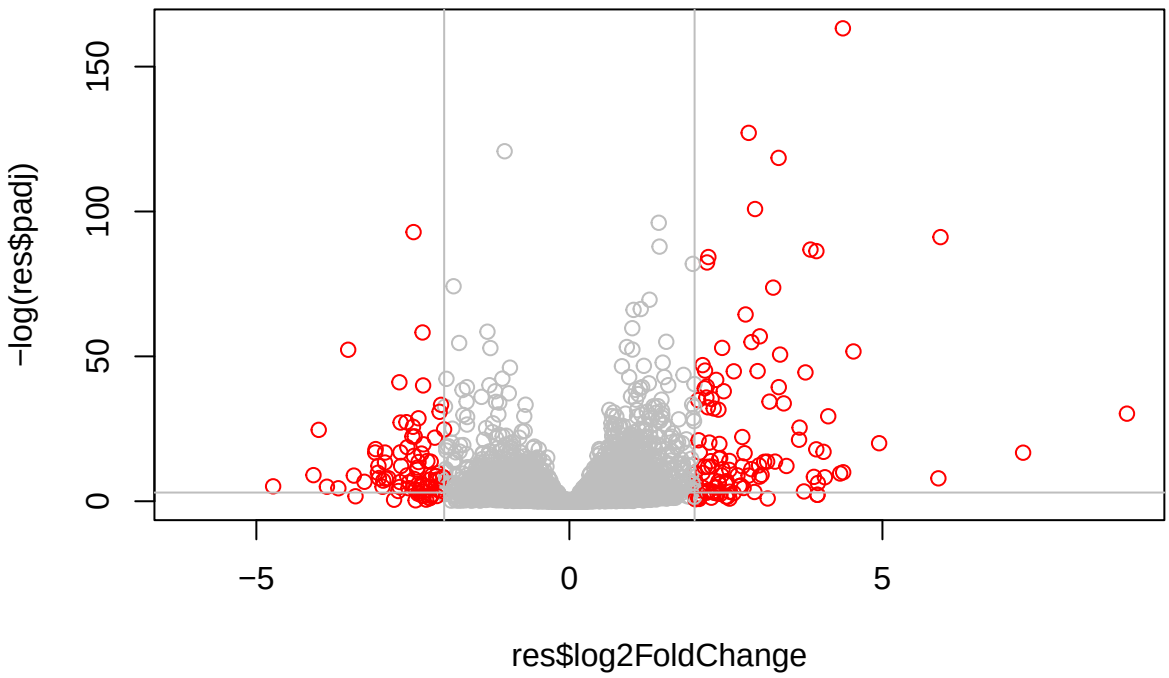
```
plot(res$log2FoldChange, -log(res$padj))
abline(v= c(-2, +2), col="gray")
abline(h=-log(0.05), col="gray")
```



Now add some color to the points of importance:

```
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2] <- "red"
```

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


−log(res$padj)
0
50
100
150
−5
0
5
res$log2FoldChange