Week 9 DESeq analysis

Yash Garodia

22/02/2022

This week we are looking at differential expression analysis.

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/Read the data from Himes et al.

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

Lets have a wee peak at the metadata file.

```
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 do a little sanity check on the correspondence of counts on metadata

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

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

Q1. How many genes are in this dataset?

There are 38694 genes in this dataset

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

We have 4 control cell lines in this dataset.

Toy differential gene expression

Lets perform some exploratory differential gene expression analysis. Note: this analysis is for demonstration only. NEVER do differential expression analysis this way!

Note that 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:

```
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
```

An alternative method is using the dplyr package:

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.2
```

```
##
## 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?

Extract and summarize the control samples

To find out where the control samples are we need the metadata

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

> 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)

Lets repeat the procedure for the treated columns.

```
n.treated <- sum(metadata$dex == "treated")
```

We have 4 treated cell lines in this dataset

Extract and summarize the treated (i.e drug) samples

```
treated <- metadata[metadata$dex == "treated",]
treated.counts <- counts[,treated$id]
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
```

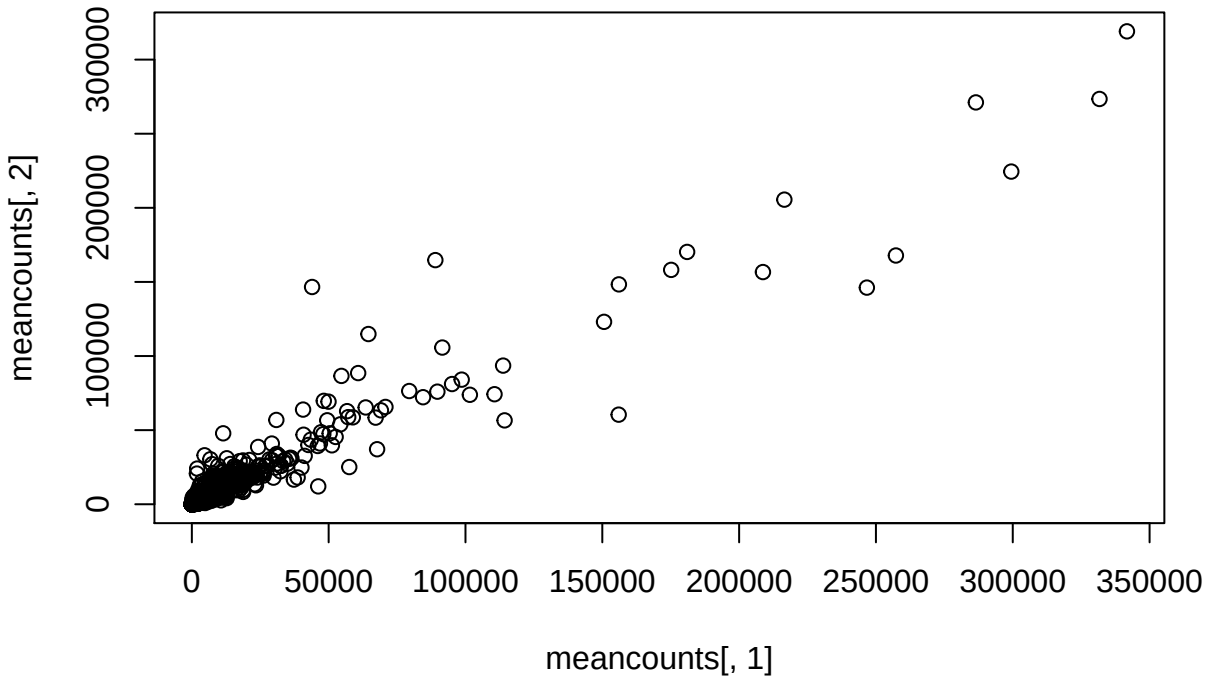
We will combine our meancount data for bookkeeping purposes. We will store these results together in a dataframe called `meancounts`

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

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

Lets make a plot to explore the results a little

```
plot(meancounts[,1], meancounts[,2])
```



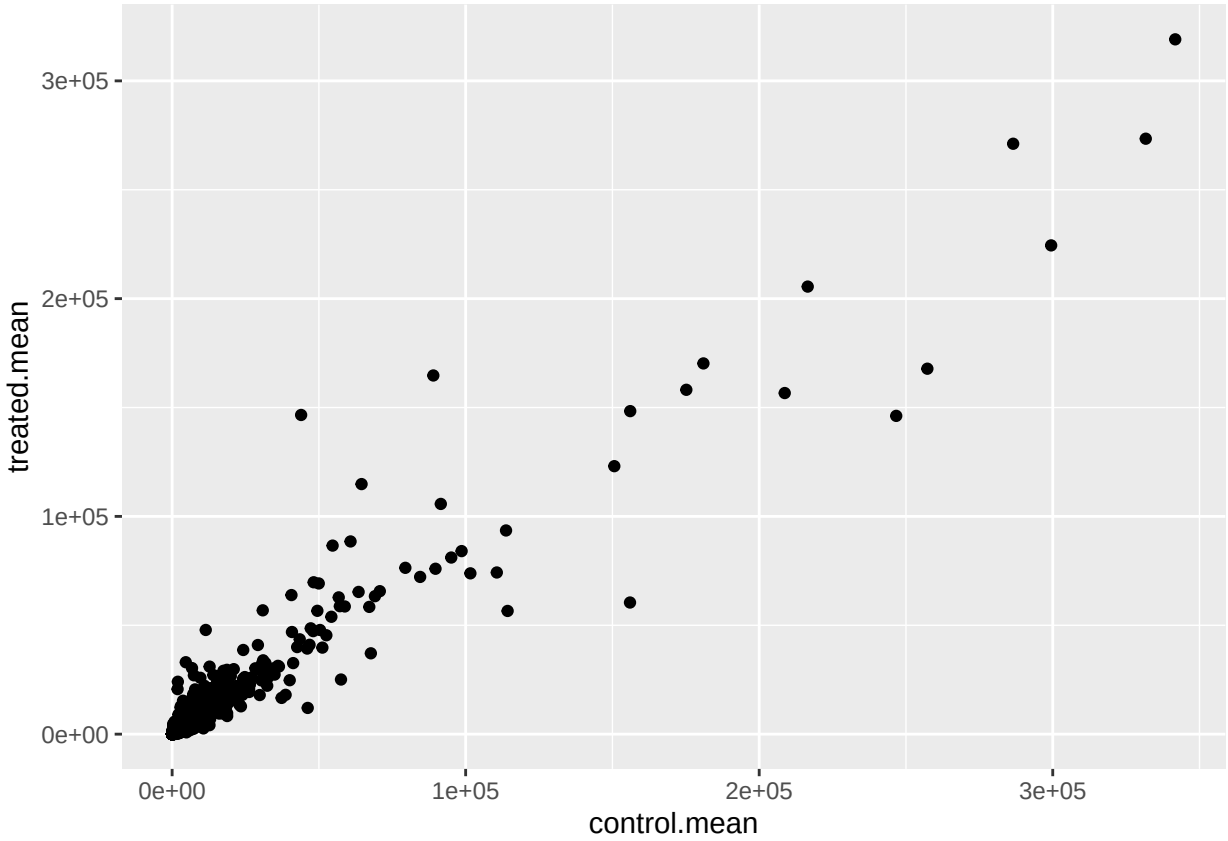
Q5 (b).You could also use the ggplot2 package to make this figure producing the plot below. What geom_?() function would you use for this plot?

We would use a geom_point() function for this plot

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

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



Wait a sec. There are 60,000-some rows in this data, but I'm only seeing a few dozen dots at most outside of the big clump around the origin.

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

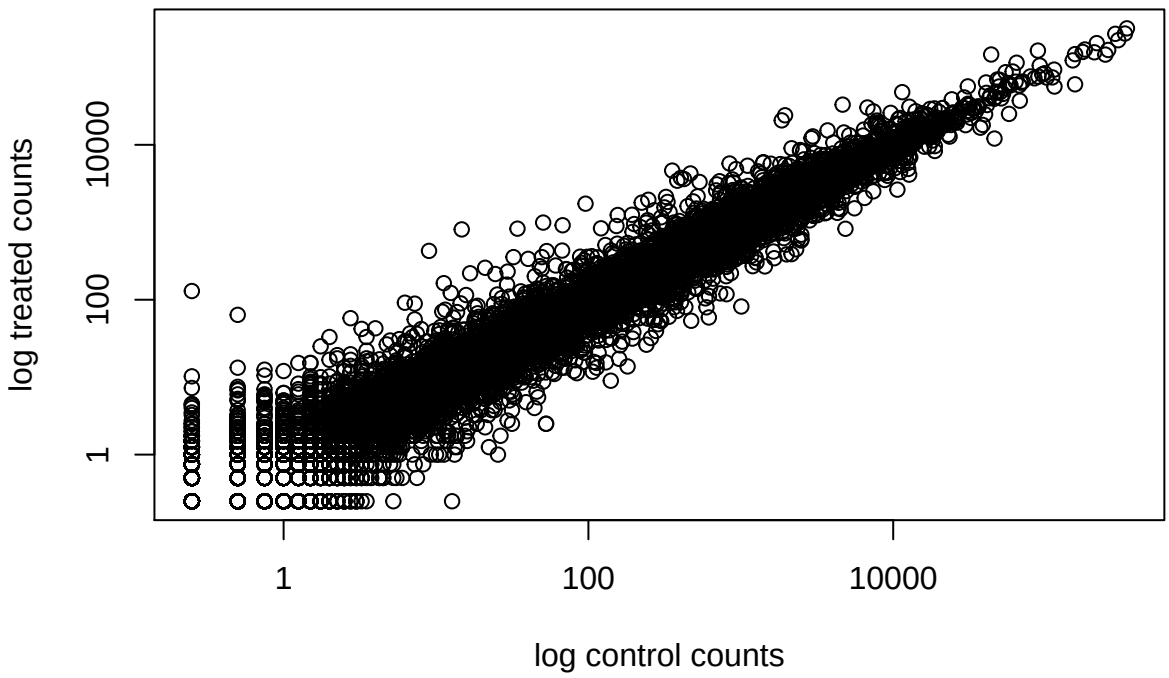
We will make a log-log plot to draw out this skewed data and see whats going on.

We can add the log = "xy" argument to plot() that will allow us to do this.

```
plot(meancounts[,1], meancounts[,2], log = "xy", xlab = "log control counts", ylab = "log treated counts
```

```
## 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
```

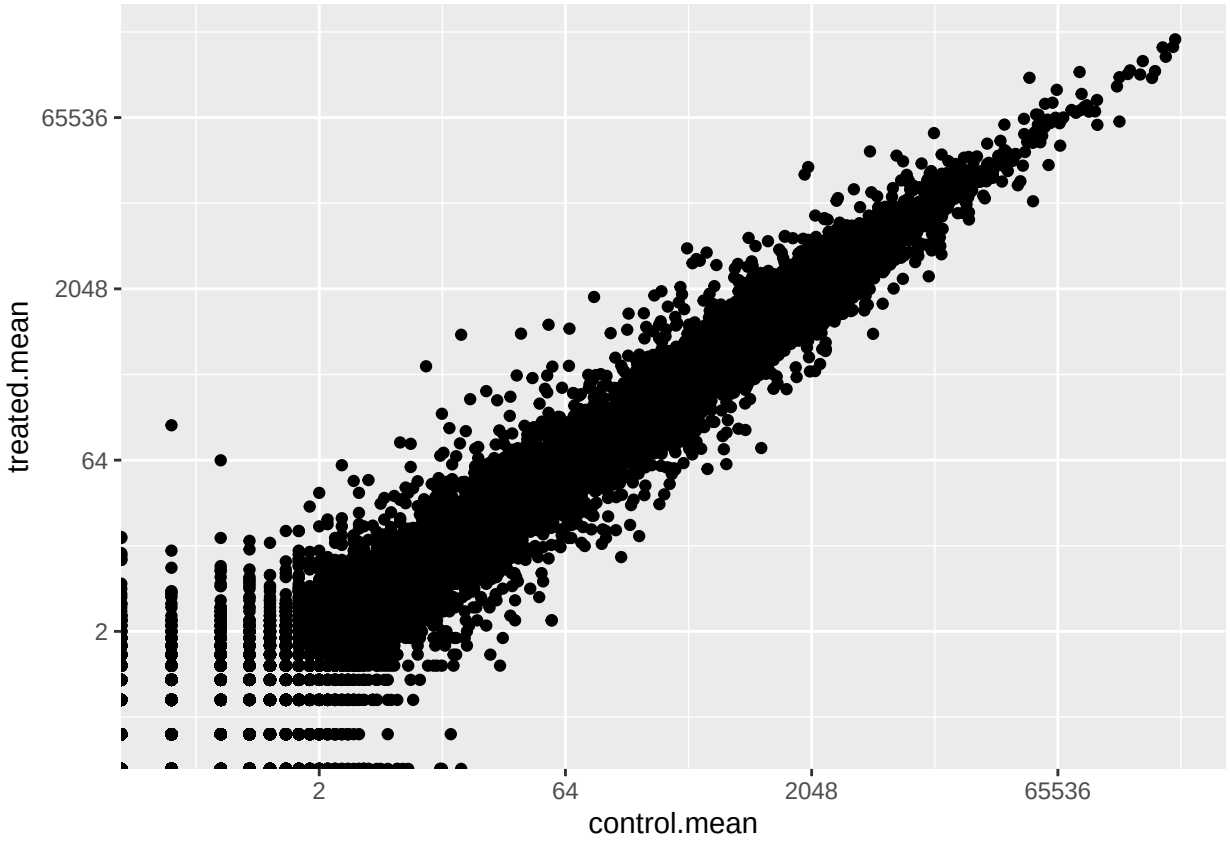


We can repeat the same for ggplot using the scale_x_continuous and scale_y_continuous function:

```
library(ggplot2)
ggplot(meancounts) + aes(control.mean, treated.mean) + geom_point() + scale_x_continuous(trans="log2") +
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```



We often use log2 transformations when dealing with this sort of data.

```
# No effect of drug, so effect remains the same
log2(20/20)
```

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

```
# Suppose drug enhances things:
log2(40/20)
```

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

```
# Suppose drug inhibits things:
log2(20/40)
```

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

```
# What if the scale is different?
log2(80/20)
```

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

This log2 transformation has this nice property where if there is no change then the log2 value will be zero and if it doubles we will have a log2 value of 1 and if halved it will be -1.

So lets add log2 fold change column to our results so far.

```
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
```

We need to get rid of zero count genes that we cannot say anything about

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

> 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 purpose of the arr.ind argument is to return only the location of those genes in the dataframe that have either the control.mean = 0 or treated.mean = 0. We then need to call the unique function to get rid of any duplicate genes so each gene location is only mentioned once.

How many genes are remaining?

```
nrow(mycounts)
```

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

A common threshold used for calling something differentially expressed is a log2(FoldChange) of greater than 2 or less than -2. Let's 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
```

There are 250 upregulated genes

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

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

There are 367 downregulated genes

There are clearly more number of downregulated genes than upregulated genes

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

We cannot trust these results as we are entirely basing our results on a threshold we created. Fold change is very important, but we haven't looked at the statistic aspect and whether the fold change is significant. We haven't looked at the p-values, so we can't tell that the change is significant or not.

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. Bioconductor is a project to provide tools for analyzing high-throughput genomic data including RNA-seq, ChIP-seq and arrays.

```
# Load up DESeq2

library(DESeq2)
```

```
## Loading required package: S4Vectors

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union

## 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, which.max, which.min

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:dplyr':
##
##     first, rename

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: SummarizedExperiment

## Warning: package 'SummarizedExperiment' was built under R version 4.0.2

## 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")'.
```

```
## Loading required package: DelayedArray

## Warning: package 'DelayedArray' was built under R version 4.0.2

## Loading required package: matrixStats

## Warning: package 'matrixStats' was built under R version 4.0.2

##
## Attaching package: 'matrixStats'

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

## The following object is masked from 'package:dplyr':
##
##     count

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum
```

```
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
```

```
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
```

We can visualize this in a table using this function:

```
resview <- as.data.frame(res)
head(resview)
```

```
##                    baseMean log2FoldChange     lfcSE       stat     pvalue
## ENSG00000000003 747.1941954    -0.35070302 0.1682457 -2.0844697 0.03711747
## ENSG00000000005   0.0000000             NA        NA         NA         NA
## ENSG00000000419 520.1341601     0.20610777 0.1010592  2.0394752 0.04140263
## ENSG00000000457 322.6648439     0.02452695 0.1451451  0.1689823 0.86581056
## ENSG00000000460  87.6826252    -0.14714205 0.2570073 -0.5725210 0.56696907
## ENSG00000000938   0.3191666    -1.73228897 3.4936010 -0.4958463 0.62000288
##                       padj
## ENSG00000000003 0.1630348
## ENSG00000000005        NA
## ENSG00000000419 0.1760317
## ENSG00000000457 0.9616942
## ENSG00000000460 0.8158486
## ENSG00000000938        NA
```

We can get some basic summary tallies using the `summary` function.

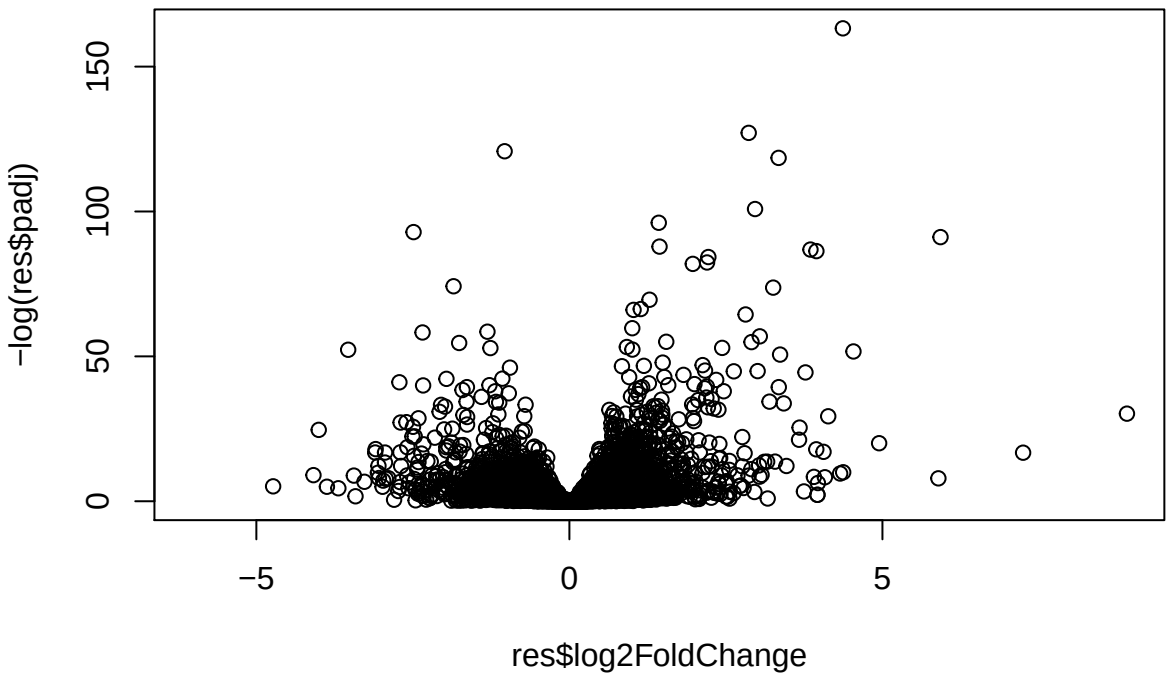
```
summary(res, alpha = 0.05)
```

```
##
## out of 25258 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)       : 1242, 4.9%
## LFC < 0 (down)     : 939, 3.7%
## outliers [1]       : 142, 0.56%
## low counts [2]     : 9971, 39%
## (mean count < 10)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Volcano Plot

Make a summary plot of our results:

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



The genes that change a lot have a higher level of change

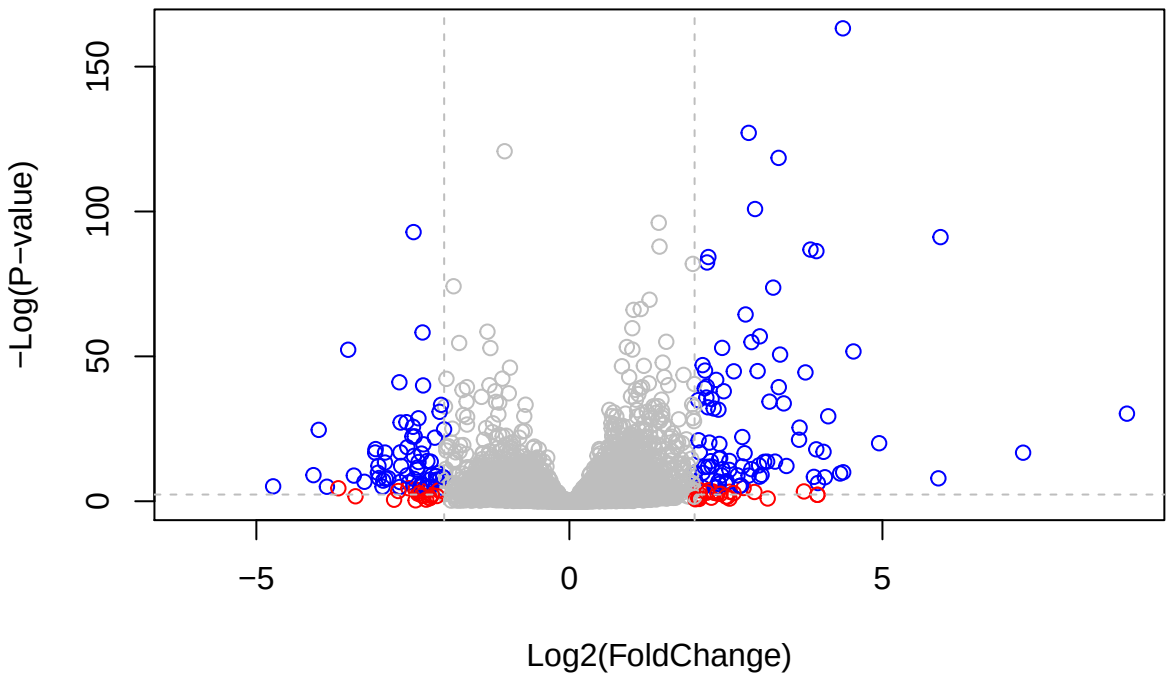
To color the points we will setup a custom color vector indicating transcripts with large fold change and significant differences between conditions:

```
# 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)
```



For even more customization you might find the EnhancedVolcano bioconductor package useful (Note. It uses ggplot under the hood):

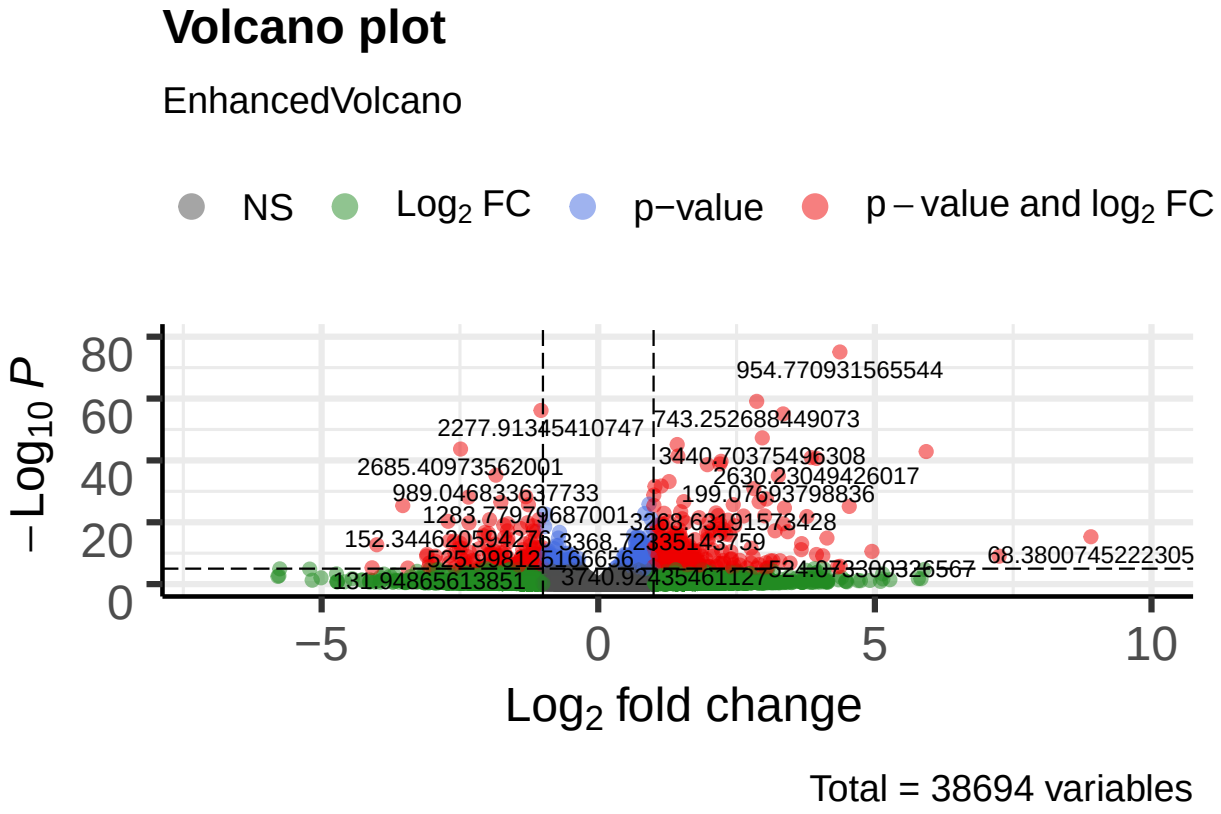
```
#BiocManager::install("EnhancedVolcano")
library(EnhancedVolcano)
```

```
## Loading required package: ggrepel
```

```
## Warning: package 'ggrepel' was built under R version 4.0.2
```

```
x <- as.data.frame(res,)

EnhancedVolcano(x,
    lab = x$baseMean,
    x = 'log2FoldChange',
    y = 'pvalue')
```



Finish for today by saving our results.

```
write.csv(res, file = "DESeq2_results.csv")
```