Lab 12-143

Ethan

2/16/2023

New Installations: install.packages("BiocManager") BiocManager::install("DESeq2")

```
# Call in our new packages
library(BiocManager)
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, aperm, 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
```

```
# Read csv file into variables
counts <- read.csv("airway_scaledcounts.csv", row.names=1)
metadata <-  read.csv("airway_metadata.csv")

# See what they look like
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? 38694 genes

```
# nRow gives us gene count
nrow(counts)
```

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

Q2. How many 'control' cell lines do we have? 4 control cell lines

```
# table to count and categorize control vs treated cell dex
table(metadata$dex)
```

```
##
## control treated
##       4       4
```

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
```

```
colnames(counts)
```

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

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 cunts for control samples

```
# Separate the controls by using a t/f vector and then applying it to metadata$id
# Then access counts limited by metadata$id
control.tf <- metadata$dex=="control"
control.id <- metadata$id[control.tf]
control.counts <- counts[,control.id]
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.

```
# Get the average with apply (1 means rows, 2 means columns)
# 'rowMeans(control.counts)' also works
control.mean <- apply(control.counts,1,mean)
```

Q3. How would you make the above code in either approach more robust? The means in the example from the lab walkthrough uses "/4" which isn't applicable to all data sets. Our code above already takes into account differently sized data frames.

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)

```
# Same logic as above (instead of control it's now treated)
treated.tf <- metadata$dex=="treated"
treated.id <- metadata$id[treated.tf]
treated.counts <- counts[,treated.id]
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
```

```
# Find the average
treated.mean <- apply(treated.counts,1,mean)
head(treated.mean)
```

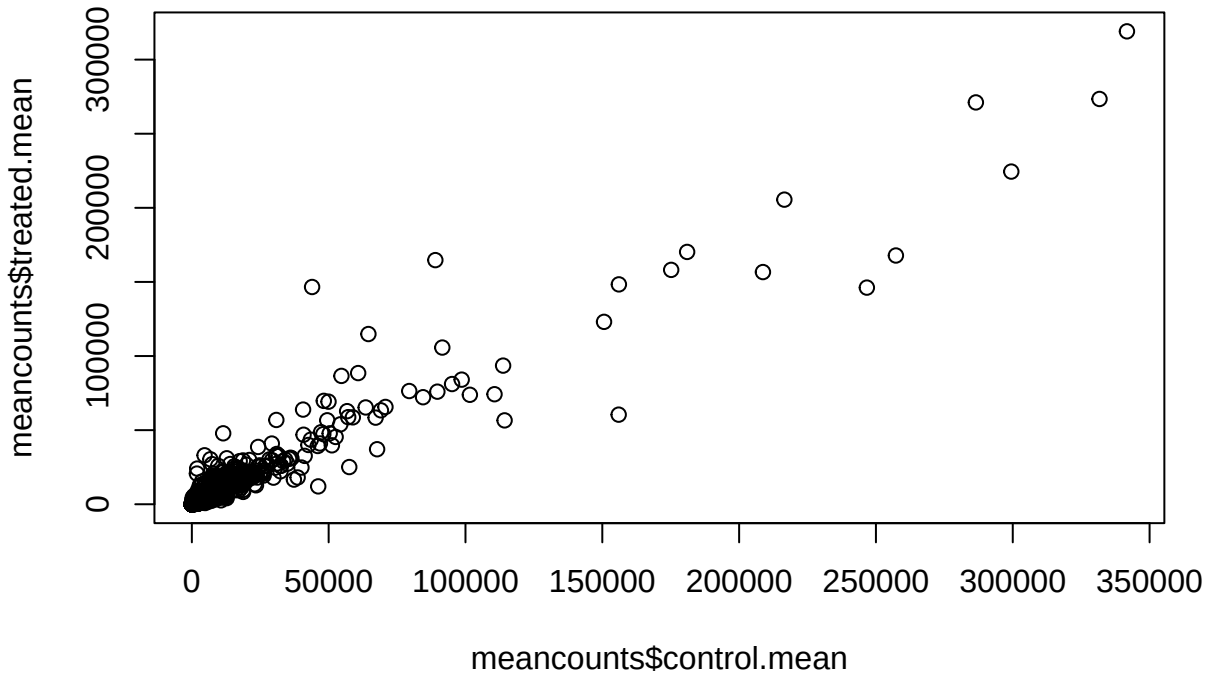
```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
##          658.00            0.00          546.00          316.50           78.75
## ENSG00000000938
##            0.00
```

make a plot to see progress

```
# Turn out means into a dataframe
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. Your plot should look something like the following.

```
# Normal base R function plot
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?

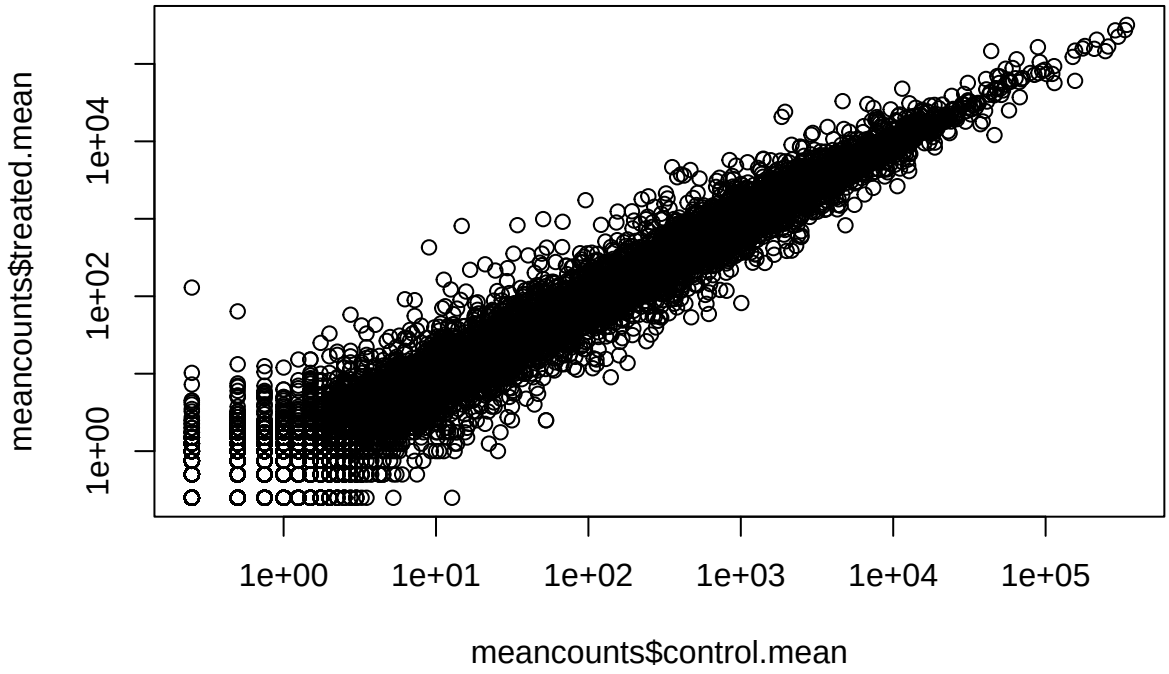
geom_point

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

```
# Adding a log argument on both x and y axis
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
```



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

Most usefule and straightforward to understand this is a log2 transformation. 0 means 0 deviation = drug didn't work

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

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

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

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

log2 = 1 then 1 means that it doubled the opposite is true. -1 means that it was halved

Add a "log2 fold-change"

```
# Make a new column in the df which is the log2 transform
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
```

Note the NaN in the log2fc column. We need to get ride 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
```

```
# Add the rows together and check if it equals 0
noZero<- rowSums(meancounts[,1:2]==0)==0
mycounts <- meancounts[noZero,]
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? We never used this function

For the next 2 questions. The answer is different from the file because we used >= instead of only >.

Q8. How many genes are upregulated at the log2fc level of +2? 314 genes are upregulated at a level greater than/equal to +2

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

```
## [1] 314
```

Q9. How many genes are downregulated at the log2fc level of -2? 485 genes are downregulated at a level less than/equal to -2

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

```
## [1] 485
```

Q10. Do you trust these results? No. We only calculated the differences between the averages. We haven't conducted a significance test.

DEseq2 Analysis

DEseq2 has already been accessed above (first code chunk)

```
library(DESeq2)
```

Like most bioconductor package, DEseq2 wants it's input and output in a very specific format.

```
# Using DEseq analysis. Requires 3 inputs shown below
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 <- DESeq(dds)
```

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

```
## fitting model and testing
```

```
dds
```

```
## class: DESeqDataSet
## dim: 38694 8
## metadata(1): version
## assays(4): counts mu H cooks
## rownames(38694): ENSG00000000003 ENSG00000000005 ... ENSG00000283120
##   ENSG00000283123
## rowData names(22): baseMean baseVar ... deviance maxCooks
## colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
## colData names(5): id dex celltype geo_id sizeFactor
```

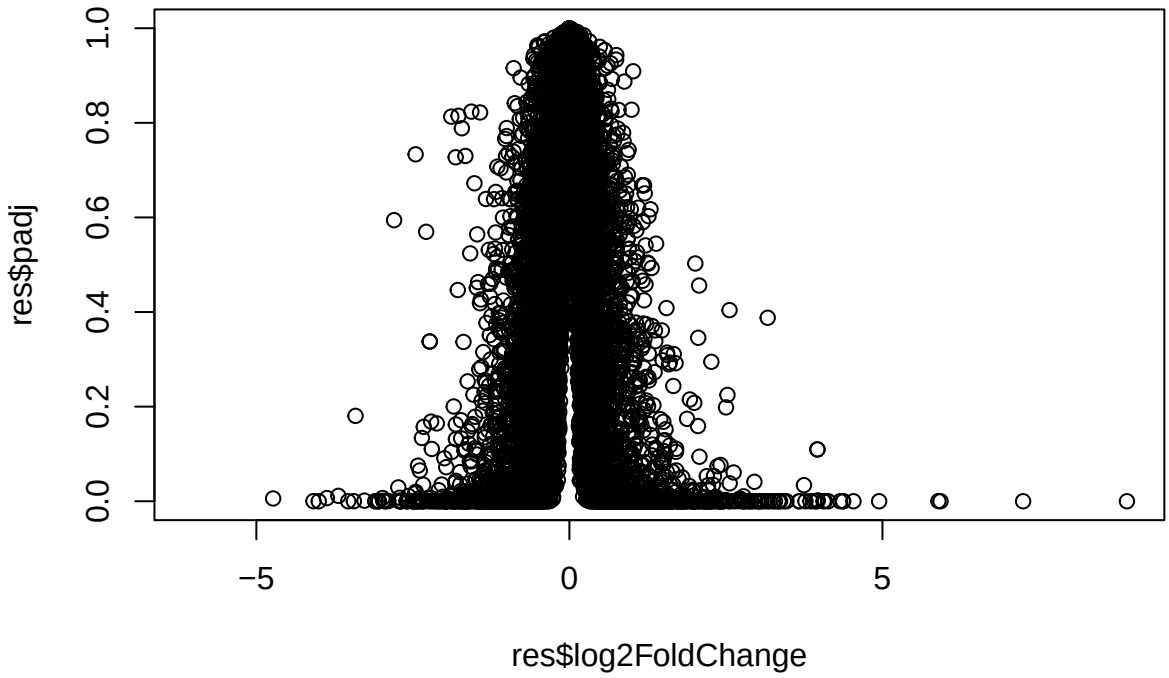
```
res <- results(dds)
res
```

```
## 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
```

Volcano Plots

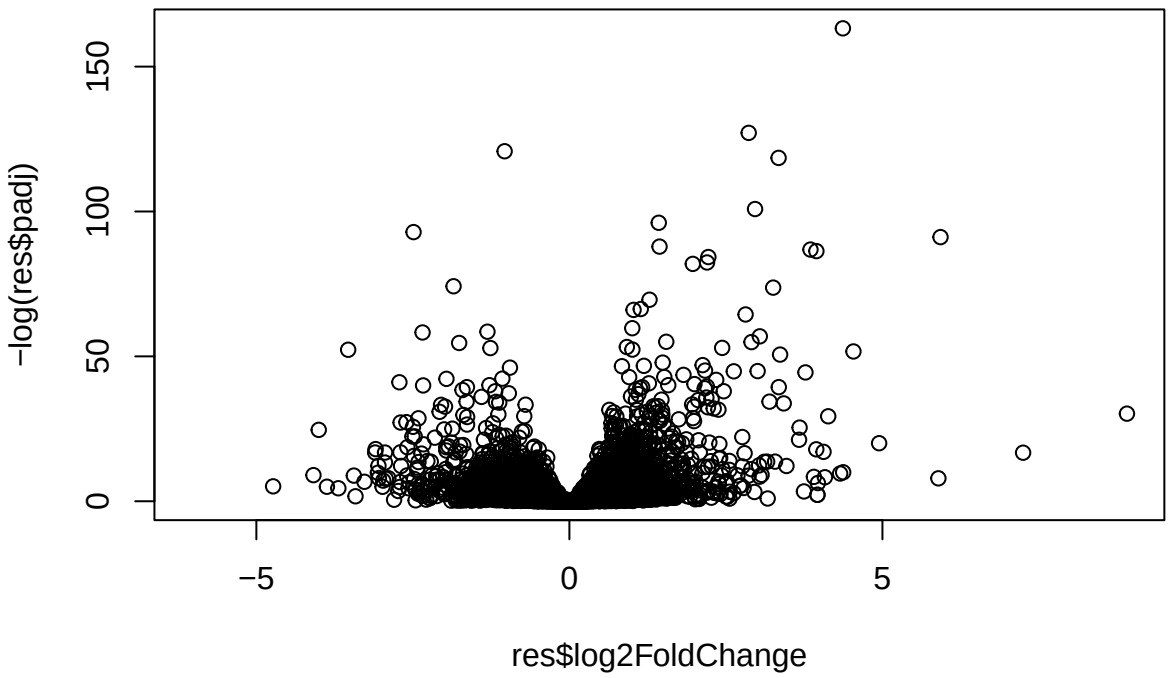
A major summary figure of this type of analysis is called a volcano plot.

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



Improve this plot by taking log of that p-value

```
# Added the -log() to make the graph upright
plot(res$log2FoldChange, -log(res$padj))
```



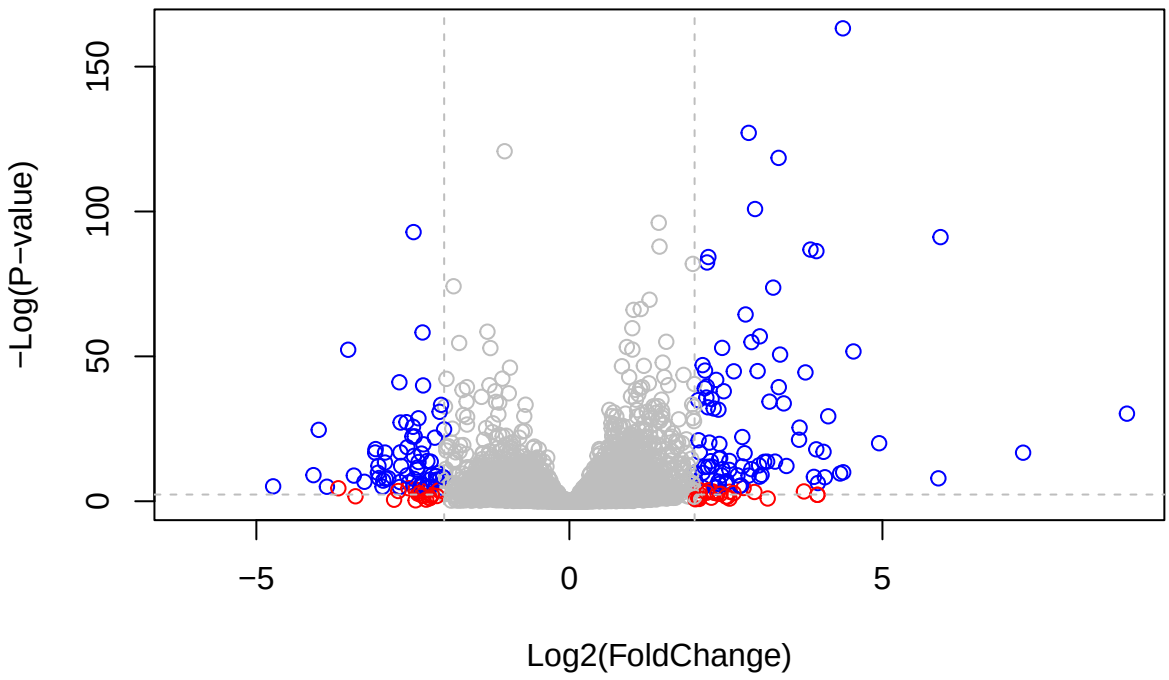
Now, we want to make a colored ver.

```
# Setup our custom point color vector
# Replicate (rep) "gray" once for every result we have
mycols <- rep("gray", nrow(res))
# Set limits above 2 (abs make sure its on both sides)
# Shows "good" log2fc level changes (which we set as 2)
mycols[ abs(res$log2FoldChange) > 2 ]  <- "red"

# Makes the points with both good p-value and "good" log2fc
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)
```


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