Class12 Lab

Dennis Kim

RNA Seq Exploration Continued, Transcriptomics and the analysis of RNA-Seq data

Import countData and colData

Load the library 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


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
```

Read the data into R

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

Take a look at each

```r
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
```

```r
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
```

Let's make sure that the id column of the metadata match the order of the columns in Count-Data

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

```
[1] TRUE
```

```
#you can use `all()` to check that all the inputs are true,
#`==` will compare the two frames and check of they are identical
```

> Q1. How many genes are in this dataset?

```
dim(counts)
```

```
[1] 38694     8
```

38694 genes

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

```
View(metadata[,2])
```

There are 4 control lines.

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
```

Side-note: An alternative way to do this same thing using the dplyr package from the tidyverse is shown below. Which do you prefer and why?

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following object is masked from 'package:Biobase':

    combine

The following object is masked from 'package:matrixStats':

    count

The following objects are masked from 'package:GenomicRanges':

    intersect, setdiff, union

The following object is masked from 'package:GenomeInfoDb':

    intersect

The following objects are masked from 'package:IRanges':

    collapse, desc, intersect, setdiff, slice, union

The following objects are masked from 'package:S4Vectors':

    first, intersect, rename, setdiff, setequal, union

The following objects are masked from 'package:BiocGenerics':

    combine, intersect, setdiff, union

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?

Use the `apply()` function

Let's extract first extract our counts for control samples as I want to compare this to the counts 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
```

Let's summarize and get one value per gene in the control experiments. I'll start by taking the average

```
#apply(control.counts, 1, mean)
#can also use `rowMeans()`
control.mean <- rowMeans(control.counts)
```

> 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)

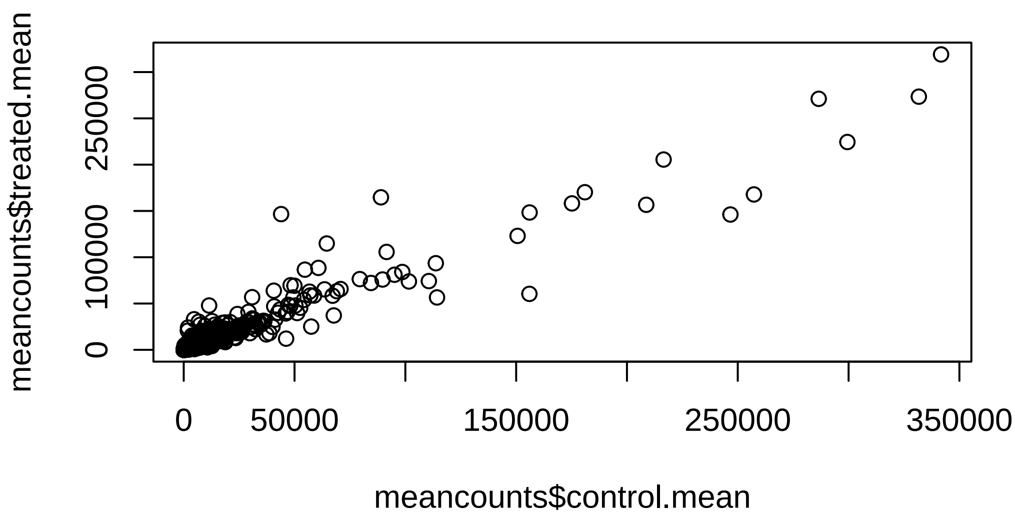
```
treated.inds <- metadata$dex == "treated"
treated.ids <- metadata$id[treated.inds]
treated.counts <- counts[, treated.ids]
treated.mean <- rowMeans(treated.counts)
```

```
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
```

Time to make a 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.

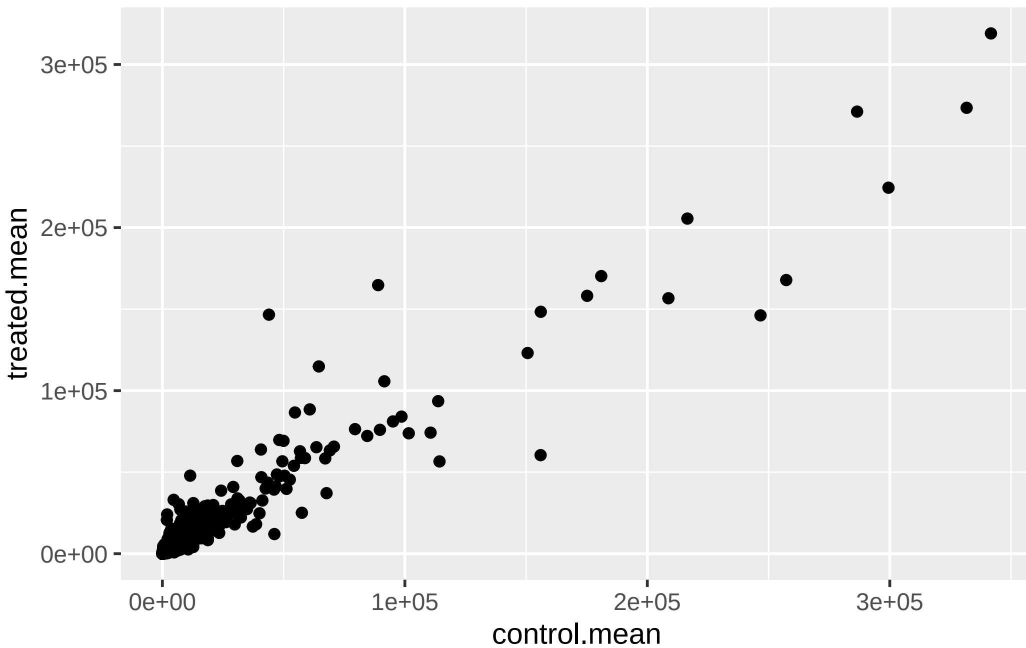
```
plot(meancounts$control.mean, meancounts$treated.mean)
```



> 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?

`geom_point()` is the function used

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

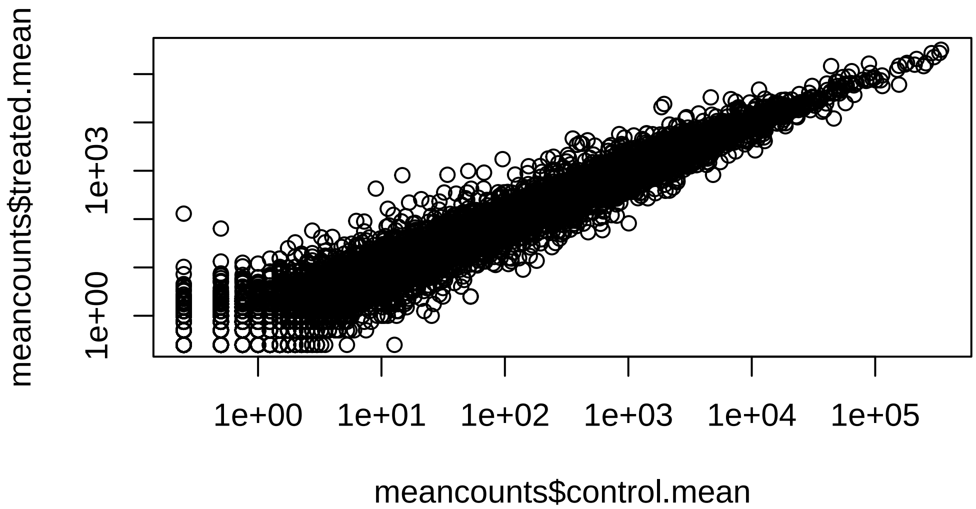


Lets put this on a log scale >Q6. Try plotting both axes on a log scale. What is the argument to plot() that allows you to do this?

```
plot(meancounts$control.mean, meancounts$treated.mean, log="xy")
```

```
Warning in xy.coords(x, y, xlabel, ylabel, log): 15032 x values <= 0 omitted
from logarithmic plot
```

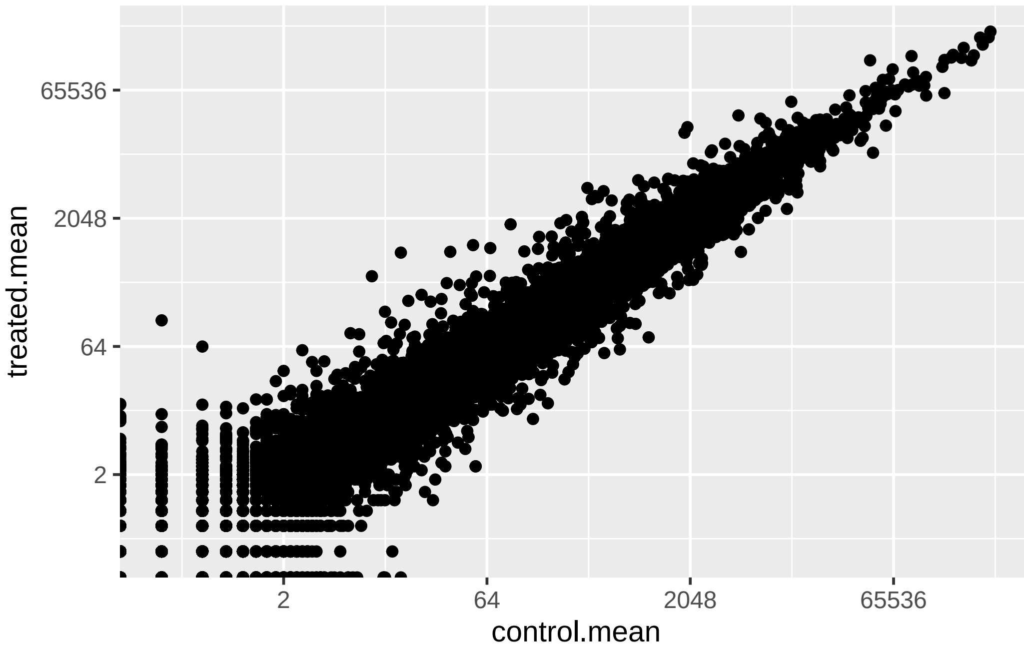
```
Warning in xy.coords(x, y, xlabel, ylabel, log): 15281 y values <= 0 omitted
from logarithmic plot
```




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

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

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



The most useful and most straightforward to understand is log2 transformation

Let's 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
```

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

> 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?

```r
#alternative way
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
```

`arr.ind()` returns the positions of the parts of mean counts from columns 1 to 2 that are equal to 0. We only take the first column and call them unique on the first column in order to ensure that the rows are not removed twice if there are zeros for both values in the samples.

Using this method:

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

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

```
                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
```

```r
nrow(my.counts)
```

```
[1] 21817
```

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

How many genes are upregulated at the log2fc level of +2

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

```
[1] 314
```

There are 314 upregulated genes

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

How many are suppresed/downregulated?

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

```
[1] 485
```

485 are downregulated

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

We didn't do any statistical tests, we don't know if the changes are consistent or significant.

DESeq2 analysis

```
library(DESeq2)
```

Like most bioconductor packages, DESeq2 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
```

The main DESeq function is called DESeq

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

```
estimating size factors

estimating dispersions

gene-wise dispersion estimates

mean-dispersion relationship

final dispersion estimates

fitting model and testing
```

```r
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
```

```r
summary(res)
```

```
out of 25258 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 1563, 6.2%
LFC < 0 (down)     : 1188, 4.7%
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
```

The results function contains a number of arguments to customize the results table. By default the argument alpha is set to 0.1. If the adjusted p value cutoff will be a value other than 0.1, alpha should be set to that value:

```r
res05 <- results(dds, alpha=0.05)
summary(res05)
```

```
out of 25258 with nonzero total read count
adjusted p-value < 0.05
LFC > 0 (up)       : 1236, 4.9%
LFC < 0 (down)     : 933, 3.7%
outliers [1]       : 142, 0.56%
low counts [2]     : 9033, 36%
(mean count < 6)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

Adding Annotation Data

Load packages

```r
library("AnnotationDbi")
```

```
Attaching package: 'AnnotationDbi'
```

```
The following object is masked from 'package:dplyr':

    select
```

```r
library("org.Hs.eg.db")
```

All the key types we can use to label

```r
columns(org.Hs.eg.db)
```

```
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
 [6] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
[11] "GENETYPE"     "GO"           "GOALL"        "IPI"          "MAP"
[16] "OMIM"         "ONTOLOGY"     "ONTOLOGYALL"  "PATH"         "PFAM"
[21] "PMID"         "PROSITE"      "REFSEQ"       "SYMBOL"       "UCSCKG"
[26] "UNIPROT"
```

We can use the mapIds() function to add individual columns to our results table. We provide the row names of our results table as a key, and specify that keytype=ENSEMBL. The column argument tells the mapIds() function which information we want, and the multiVals argument tells the function what to do if there are multiple possible values for a single input value. Here we ask to just give us back the first one that occurs in the database.

```r
res$symbol <- mapIds(org.Hs.eg.db,
                     keys=row.names(res), # Our genenames
                     keytype="ENSEMBL",        # The format of our genenames
                     column="SYMBOL",          # The new format we want to add
                     multiVals="first")
```

```
'select()' returned 1:many mapping between keys and columns
```

```r
head(res)
```

```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 7 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      symbol
                <numeric> <character>
ENSG00000000003  0.163035      TSPAN6
ENSG00000000005        NA        TNMD
ENSG00000000419  0.176032        DPM1
ENSG00000000457  0.961694       SCYL3
ENSG00000000460  0.815849    C1orf112
ENSG00000000938        NA         FGR
```

> Q11. Run the mapIds() function two more times to add the Entrez ID and UniProt accession and GENENAME as new columns called res$entrez$, resuniprot and res$genename.

```r
res$entrez <- mapIds(org.Hs.eg.db,
                     keys=row.names(res),
                     column="ENTREZID",
                     keytype="ENSEMBL",
                     multiVals="first")
```

```
'select()' returned 1:many mapping between keys and columns
```

```r
res$uniprot <- mapIds(org.Hs.eg.db,
                      keys=row.names(res),
                      column="UNIPROT",
                      keytype="ENSEMBL",
                      multiVals="first")
```

```
'select()' returned 1:many mapping between keys and columns
```

```r
res$genename <- mapIds(org.Hs.eg.db,
                       keys=row.names(res),
                       column="GENENAME",
```

```
                     keytype="ENSEMBL",
                     multiVals="first")
```

```
'select()' returned 1:many mapping between keys and columns
```

```
head(res)
```

```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 10 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      symbol      entrez     uniprot
                <numeric> <character> <character> <character>
ENSG00000000003  0.163035      TSPAN6        7105  A0A024RCI0
ENSG00000000005        NA        TNMD       64102      Q9H2S6
ENSG00000000419  0.176032        DPM1        8813      O60762
ENSG00000000457  0.961694       SCYL3       57147      Q8IZE3
ENSG00000000460  0.815849    C1orf112       55732  A0A024R922
ENSG00000000938        NA         FGR        2268      P09769
                              genename
                           <character>
ENSG00000000003          tetraspanin 6
ENSG00000000005            tenomodulin
ENSG00000000419 dolichyl-phosphate m..
ENSG00000000457 SCY1 like pseudokina..
ENSG00000000460 chromosome 1 open re..
ENSG00000000938 FGR proto-oncogene, ..
```

Arrange and order by adjusted p-value

```
ord <- order( res$padj )
#View(res[ord,])
head(res[ord,])
```

```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 10 columns
                 baseMean log2FoldChange     lfcSE      stat      pvalue
                <numeric>      <numeric> <numeric> <numeric>   <numeric>
ENSG00000152583   954.771        4.36836 0.2371268   18.4220 8.74490e-76
ENSG00000179094   743.253        2.86389 0.1755693   16.3120 8.10784e-60
ENSG00000116584  2277.913       -1.03470 0.0650984  -15.8944 6.92855e-57
ENSG00000189221  2383.754        3.34154 0.2124058   15.7319 9.14433e-56
ENSG00000120129  3440.704        2.96521 0.2036951   14.5571 5.26424e-48
ENSG00000148175 13493.920        1.42717 0.1003890   14.2164 7.25128e-46
                       padj      symbol      entrez     uniprot
                  <numeric> <character> <character> <character>
ENSG00000152583 1.32441e-71     SPARCL1        8404  A0A024RDE1
ENSG00000179094 6.13966e-56        PER1        5187      O15534
ENSG00000116584 3.49776e-53     ARHGEF2        9181      Q92974
ENSG00000189221 3.46227e-52        MAOA        4128      P21397
ENSG00000120129 1.59454e-44       DUSP1        1843      B4DU40
ENSG00000148175 1.83034e-42        STOM        2040      F8VSL7
                              genename
                           <character>
ENSG00000152583           SPARC like 1
ENSG00000179094 period circadian reg..
ENSG00000116584 Rho/Rac guanine nucl..
ENSG00000189221    monoamine oxidase A
ENSG00000120129 dual specificity pho..
ENSG00000148175               stomatin
```

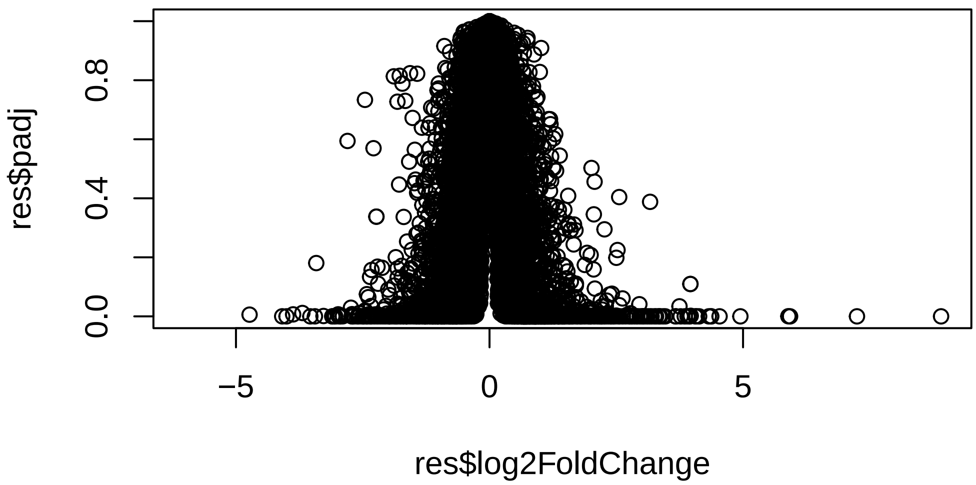
Write the ordered significant results

```
write.csv(res[ord,], "deseq_results.csv")
```

Volcano Plots

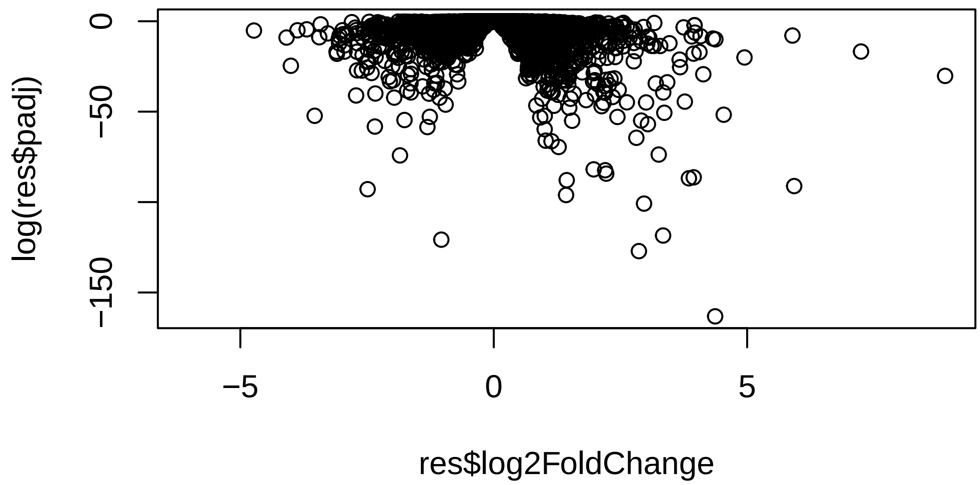
A major summary figure of this type of analysis is called a volcano plot, the idea is to keep our inner biologist and inner stats person happy

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



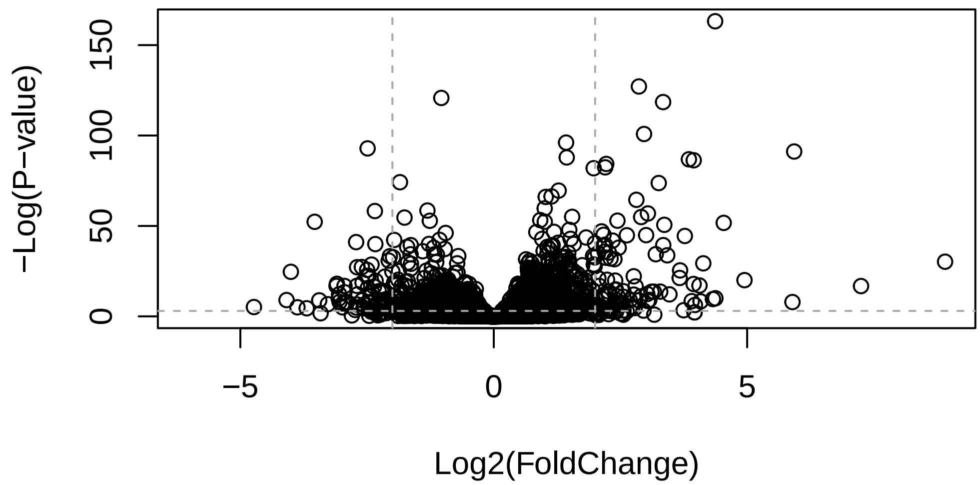
Improve this plot by taking the log of the p-value axis

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



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

```
plot(res$log2FoldChange, -log(res$padj), xlab="Log2(FoldChange)", ylab="-Log(P-value)")
# Add some cut-off lines
abline(v=c(-2,2), col="darkgray", lty=2)
abline(h=-log(0.05), col="darkgray", lty=2)
```



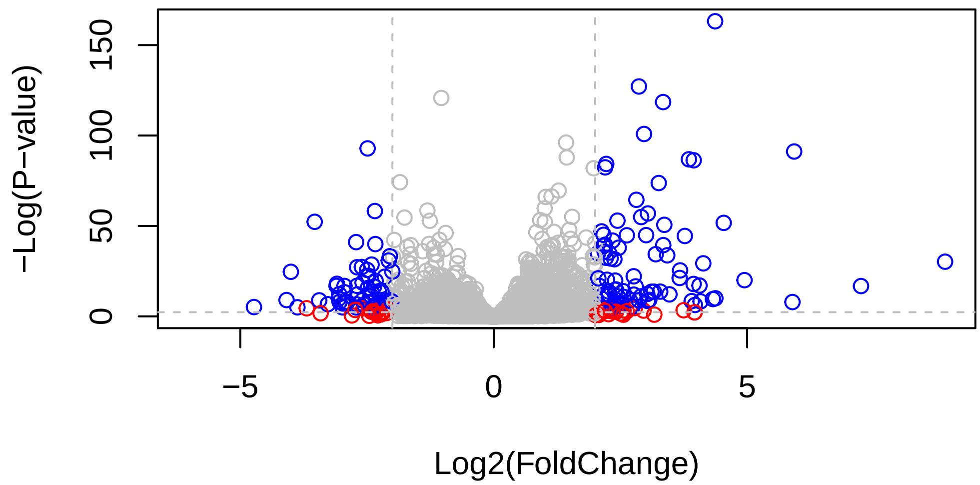
Colors

```
# 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)
```


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