

Class 12: Transcriptomics and the Analysis of RNA-Seq data

Angela Liu

Bioconductor and DESeq2 Setup

Here we will download Bioconductor and DESeq2.

```
#install.packages("BiocManager")
#BiocManager::install()

# For this class, you'll also need DESeq2:
#BiocManager::install("DESeq2")

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, rowCummmins, rowCumprods,
rowCumsums, rowDiffss, rowIQRDiffss, rowIQRs, rowLogSumExps,
rowMadDiffss, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
rowSdDiffss, rowSds, rowSums2, rowTabulates, rowVarDiffss, 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
```

Section 2: Import countData and colData

To start, let's read in the csv files we downloaded with the counts and metadata.

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

head(counts)
```

	SRR1039508	SRR1039509	SRR1039512	SRR1039513	SRR1039516
ENSG000000000003	723	486	904	445	1170
ENSG000000000005	0	0	0	0	0
ENSG000000000419	467	523	616	371	582
ENSG000000000457	347	258	364	237	318
ENSG000000000460	96	81	73	66	118
ENSG000000000938	0	0	1	0	2
	SRR1039517	SRR1039520	SRR1039521		
ENSG000000000003	1097	806	604		
ENSG000000000005	0	0	0		
ENSG000000000419	781	417	509		
ENSG000000000457	447	330	324		
ENSG000000000460	94	102	74		
ENSG000000000938	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?

There are 38694 genes in the dataset.

```
nrow(counts)
```

```
[1] 38694
```

```
ncol(counts)
```

```
[1] 8
```

and the metadata aka “colData”

```
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.

We can use the `all()` function to check that all its inputs are true.

```
metadata$id
```

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

```
colnames(counts)
```

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

```
# see if all the items in the vecotrs are the same
all(metadata$id == colnames(counts))
```

```
[1] TRUE
```

```
if(!all(c(T,T,T,F))){
  cat("Wow something is wrong")
}
```

```
Wow something is wrong
```

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

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

```
[1] 4
```

We have four controlled cell lines.

Analysis by hand

```
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	ENSG000000000419	ENSG000000000457	ENSG000000000460
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?

A better approach to make it more robust is to use the format shown below so that it adjusts automatically to any changes in the columns.

Let's extract our counts for control samples as I want to compare this to the counts for treated (i.e. with drug) samples.

```
metadata$dex == "control"
```

```
[1] TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE
```

Find the id for the controlled samples:

```
control inds <- metadata$dex == "control"
# access the IDs for samples that are controlled
control.ids <- metadata$id[control inds]
# view just the columns of the controlled samples
```

```

control.counts <- counts[ , control.ids]
head(control.counts)

```

	SRR1039508	SRR1039512	SRR1039516	SRR1039520
ENSG000000000003	723	904	1170	806
ENSG000000000005	0	0	0	0
ENSG000000000419	467	616	582	417
ENSG000000000457	347	364	318	330
ENSG000000000460	96	73	118	102
ENSG000000000938	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.

```

#apply(control.counts, 1, mean)
control.mean <- rowMeans(control.counts)
head(control.mean)

```

ENSG000000000003	ENSG000000000005	ENSG000000000419	ENSG000000000457	ENSG000000000460
900.75	0.00	520.50	339.75	97.25
ENSG000000000938				
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)

Now to separate our the treated samples and find the treated mean.

```

treated inds <- metadata$dex == "treated"
# access the IDs for samples that are treated
treated.ids <- metadata$id[treated inds]
# view just the columns of the controlled samples
treated.counts <- counts[ , treated.ids]
head(treated.counts)

```

	SRR1039509	SRR1039513	SRR1039517	SRR1039521
ENSG000000000003	486	445	1097	604
ENSG000000000005	0	0	0	0
ENSG000000000419	523	371	781	509
ENSG000000000457	258	237	447	324
ENSG000000000460	81	66	94	74
ENSG000000000938	0	0	0	0

```
treated.mean <- rowMeans(treated.counts)
head(treated.mean)
```

```
ENSG000000000003 ENSG000000000005 ENSG000000000419 ENSG000000000457 ENSG000000000460
       658.00          0.00        546.00        316.50        78.75
ENSG000000000938
       0.00
```

To help us stay organized, let's make a new data.frame to store these results together.

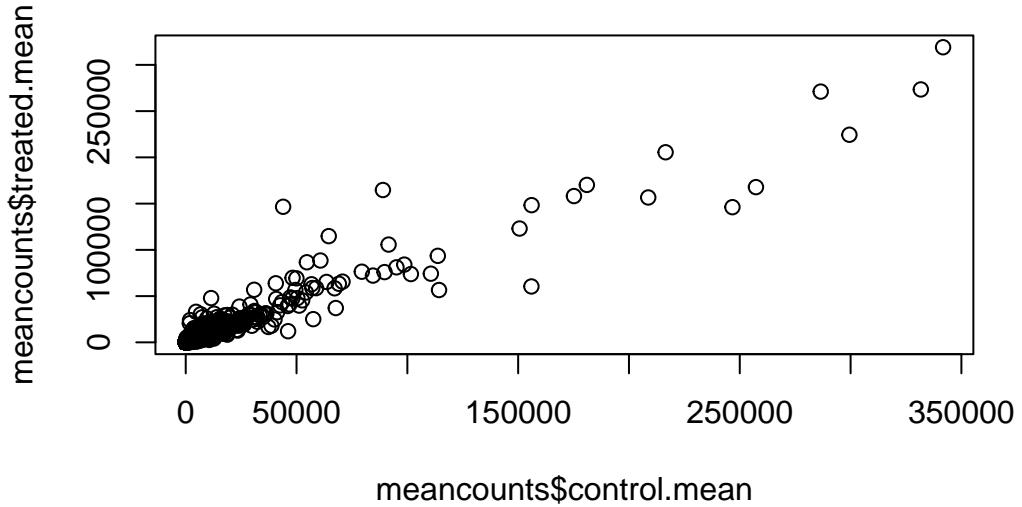
```
meancounts <- data.frame(control.mean, treated.mean)
head(meancounts)
```

	control.mean	treated.mean
ENSG000000000003	900.75	658.00
ENSG000000000005	0.00	0.00
ENSG000000000419	520.50	546.00
ENSG000000000457	339.75	316.50
ENSG000000000460	97.25	78.75
ENSG000000000938	0.75	0.00

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

And make a plot to see how we are doing

```
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()`

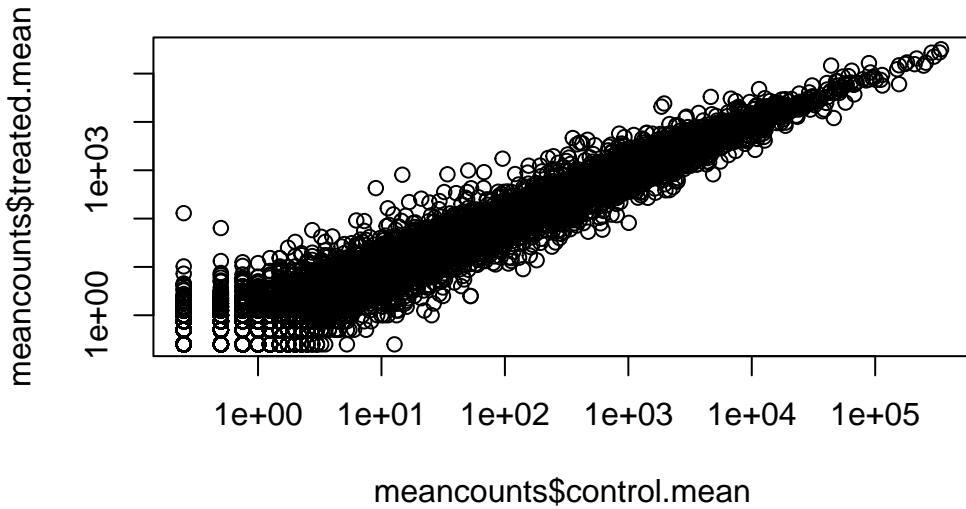
Q6. Try plotting both axes on a log scale. What is the argument to `plot()` that allows you to do this?

Let's transform this plot into an x and y scale on the log.

```
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



If drug has no effect, the treated and control points will lie on the diagonal line. The points above the line show that there's more expression in the treated than control while points below the line show more expression in the control. This is an instance of fold change.

Log2 transformations are most useful and convenient to understand.

```
# log2 values of zero mean no change
log2(20/20)
```

[1] 0

Doubling:

```
# one times fold change means it doubled
log2(40/20)
```

[1] 1

Halved:

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

```
[1] -1
```

```
log2(80/20)
```

```
[1] 2
```

0 means no change, negative means reduced expression, positive means increased expression.

Add a “log2 fold-change” to plot:

```
# add a new column log2fc with fold change values  
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
ENSG000000000419	520.50	546.00	0.06900279
ENSG000000000457	339.75	316.50	-0.10226805
ENSG000000000460	97.25	78.75	-0.30441833
ENSG000000000938	0.75	0.00	-Inf

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

```
# set the meancounts of zero to true, then set them to false  
to.keep <- rowSums(meancounts[,1:2] == 0) == 0  
mycounts <- meancounts[to.keep, ]  
head(mycounts)
```

	control.mean	treated.mean	log2fc
ENSG000000000003	900.75	658.00	-0.45303916
ENSG000000000419	520.50	546.00	0.06900279
ENSG000000000457	339.75	316.50	-0.10226805
ENSG000000000460	97.25	78.75	-0.30441833
ENSG000000000971	5219.00	6687.50	0.35769358
ENSG00000001036	2327.00	1785.75	-0.38194109

```

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
ENSG000000000003	900.75	658.00	-0.45303916
ENSG000000000419	520.50	546.00	0.06900279
ENSG000000000457	339.75	316.50	-0.10226805
ENSG000000000460	97.25	78.75	-0.30441833
ENSG000000000971	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 arr.ind = TRUE causes the which() function to give back the locations of the TRUE values. With the locations/positions, the zero counted samples are ignored. The unique() function makes sure that each row is counted only once.

```

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(mycounts$log2fc > 2)
```

```
[1] 250
```

There are 250 upregulated genes greater than a 2 fc level.

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(mycounts$log2fc < -2)
```

```
[1] 367
```

There are 367 downregulated genes.

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

No, it does not show if the results are statistically significant since all we did was take the mean. There are no statistical aspects to these results.

DESeq2 analysis

```
#1 message: false
library(DESeq2)
```

Like most bioconductor packages DESeq wants its input and output in a very specific format. It has its own object type.

```
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
```

```
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

The main DESeq function is called DESeq. It is extracting the columns, means, calculating fold change and p values.

```
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>
ENSG000000000003	747.1942	-0.3507030	0.168246	-2.084470	0.0371175
ENSG000000000005	0.0000	NA	NA	NA	NA
ENSG000000000419	520.1342	0.2061078	0.101059	2.039475	0.0414026
ENSG000000000457	322.6648	0.0245269	0.145145	0.168982	0.8658106
ENSG000000000460	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

```

```
head(rownames(res))
```

```
[1] "ENSG00000000003" "ENSG00000000005" "ENSG00000000419" "ENSG00000000457"
[5] "ENSG00000000460" "ENSG00000000938"
```

Padj = P value adjusted, very important.

```
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

```

```

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

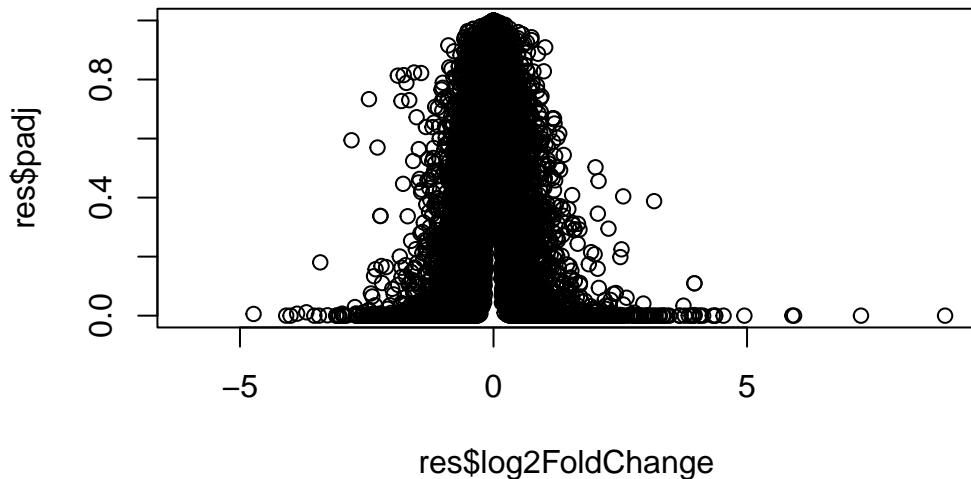
out of 25258 with nonzero total read count
adjusted p-value < 0.05
LFC > 0 (up)      : 1236, 4.9%
LFC < 0 (down)     : 933, 3.7%
outliers [1]        : 142, 0.56%
low counts [2]       : 9033, 36%
(mean count < 6)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results

```

Data Visualization – Volcano Plots

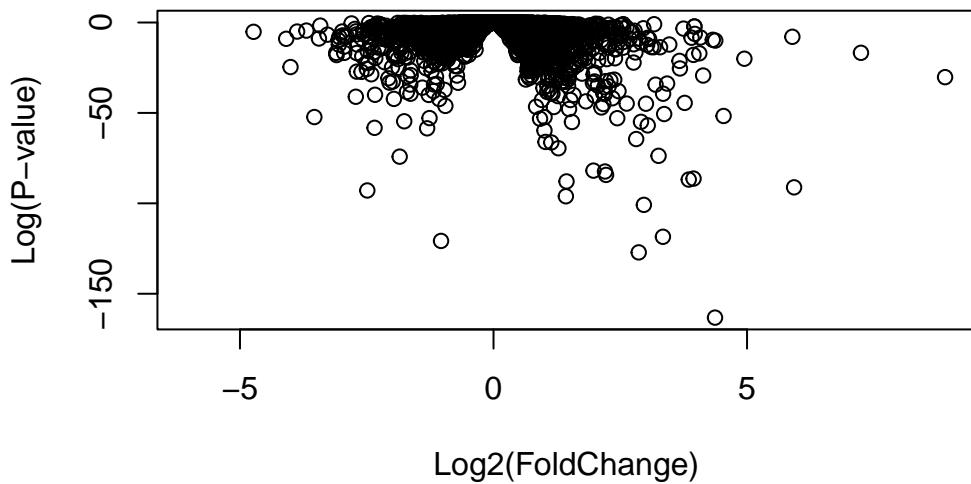
A major summary figure of this type of analysis is called a volcano plot - the idea here is to keep our inner biologist and statistician happy with the plot.

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



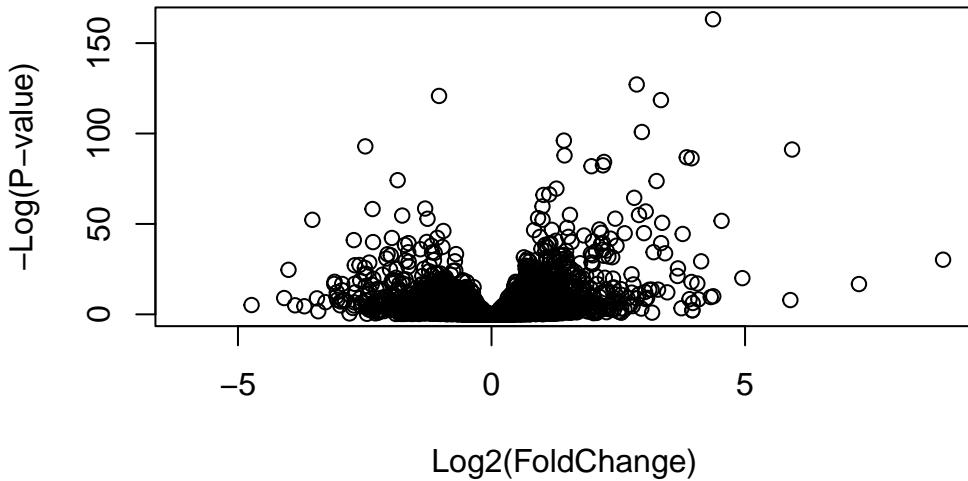
Care about the points close to the zero. We can improve this plot by taking the log of that p-value axis.

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



This plot is oriented the wrong way. We want to flip the y-axis.

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



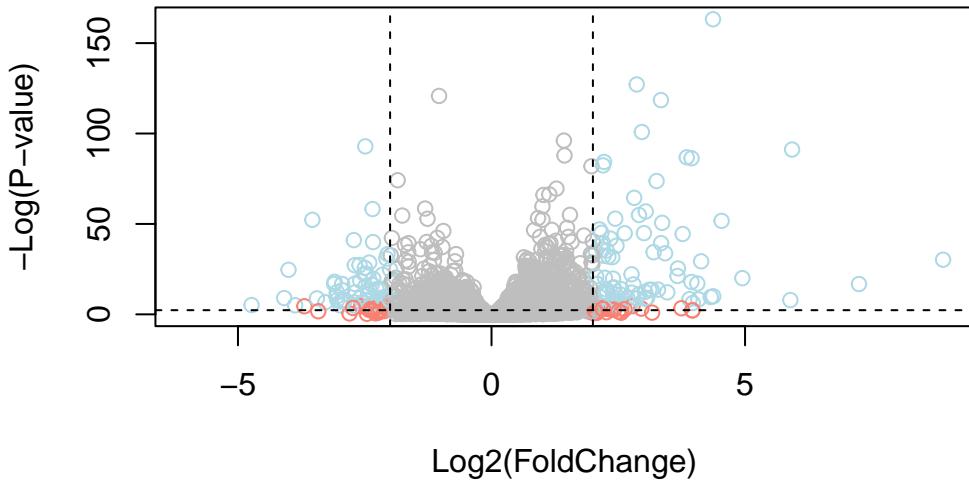
Move in negative x direction see down regulated, move in positive x direction see unregulated.
Moving up in y-axis show more statistically significant value – the points exploding away are our focus points.

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

inds <- (res$padj < 0.01) & (abs(res$log2FoldChange) > 2 )
mycols[ inds ] <- "lightblue"

# 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="black", lty=2)
abline(h=-log(0.1), col="black", lty=2)
```



Gene annotation

Let's open Bioconductor's annotation packages so that we can map between different ID schemes. We will load the `AnnotationDbi` package and annotation data package for humans `org.Hs.eg.db`.

```
# BiocManager::install("AnnotationDbi")
# BiocManager::install("org.Hs.eg.db")
library("AnnotationDbi")
```

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

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

```
select
```

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

The “org” is the organisms annotation package, “Hs” is for homo sapiens, “eg” is for entrez gene IDs, and “db” is database package.

Look at types of IDs I can translate from the `org.Hs.eg.db` package with the `columns()` function.

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

[1] "ACNUM"      "ALIAS"       "ENSEMBL"      "ENSEMLPROT"   "ENSEMLTRANS"
[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"

res$symbol <- mapIds(x = org.Hs.eg.db,
                      column = "SYMBOL", # the new format we want to add
                      keys = rownames(res), # our genenames
                      keytype = "ENSEMBL") # format of current genenames

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

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

```
res$entrez <- mapIds(x = org.Hs.eg.db,
                      column = "ENTREZID", # the new format we want to add
                      keys = rownames(res), # our genenames
                      keytype = "ENSEMBL") # format of current genenames

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

res$genenames <- mapIds(x = org.Hs.eg.db,
                        column = "GENENAME", # the new format we want to add
                        keys = rownames(res), # our genenames
                        keytype = "ENSEMBL") # format of current genenames

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

```

res$uniprot <- mapIds(x = org.Hs.eg.db,
                      column = "UNIPROT", # the new format we want to add
                      keys = rownames(res), # our genenames
                      keytype = "ENSEMBL") # format of current genenames

'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>
ENSG000000000003 747.194195 -0.3507030  0.168246 -2.084470 0.0371175
ENSG000000000005  0.000000      NA        NA        NA        NA
ENSG000000000419 520.134160  0.2061078  0.101059  2.039475 0.0414026
ENSG000000000457 322.664844  0.0245269  0.145145  0.168982 0.8658106
ENSG000000000460 87.682625  -0.1471420  0.257007 -0.572521 0.5669691
ENSG000000000938 0.319167  -1.7322890  3.493601 -0.495846 0.6200029
  padj      symbol      entrez      genenames
  <numeric> <character> <character> <character>
ENSG000000000003 0.163035    TSPAN6      7105      tetraspanin 6
ENSG000000000005   NA        TNMD       64102      tenomodulin
ENSG000000000419 0.176032    DPM1       8813      dolichyl-phosphate m..
ENSG000000000457 0.961694    SCYL3      57147      SCY1 like pseudokina..
ENSG000000000460 0.815849    C1orf112    55732      chromosome 1 open re..
ENSG000000000938   NA        FGR       2268      FGR proto-oncogene, ..

  uniprot
  <character>
ENSG000000000003 AOA024RCI0
ENSG000000000005 Q9H2S6
ENSG000000000419 O60762
ENSG000000000457 Q8IZE3
ENSG000000000460 AOA024R922
ENSG000000000938 P09769

```

Pathway Analysis

We'll finish this lab w a quick pathway analysis. Here, we can use the **GAGE package** (Generally Applicable Gene set Enrichment) to do **KEGG pathway enrichment analysis**.

```
library(pathview)
```

```
#####
# Pathview is an open source software package distributed under GNU General
# Public License version 3 (GPLv3). Details of GPLv3 is available at
# http://www.gnu.org/licenses/gpl-3.0.html. Particullary, users are required to
# formally cite the original Pathview paper (not just mention it) in publications
# or products. For details, do citation("pathview") within R.
```

The pathview downloads and uses KEGG data. Non-academic uses may require a KEGG license agreement (details at <http://www.kegg.jp/kegg/legal.html>).

```
library(gage)
```

```
library(gageData)
```

```
data(kegg.sets.hs)
```

```
# Examine the first 2 pathways in this kegg set for humans
head(kegg.sets.hs, 2)
```

```
$`hsa00232 Caffeine metabolism`  
[1] "10"    "1544"  "1548"  "1549"  "1553"  "7498"  "9"
```

```
$`hsa00983 Drug metabolism - other enzymes`  
[1] "10"     "1066"   "10720"  "10941"  "151531" "1548"   "1549"   "1551"  
[9] "1553"   "1576"   "1577"   "1806"   "1807"   "1890"   "221223" "2990"  
[17] "3251"   "3614"   "3615"   "3704"   "51733"  "54490"  "54575"  "54576"  
[25] "54577"  "54578"  "54579"  "54600"  "54657"  "54658"  "54659"  "54963"  
[33] "574537" "64816"  "7083"   "7084"   "7172"   "7363"   "7364"   "7365"  
[41] "7366"   "7367"   "7371"   "7372"   "7378"   "7498"   "79799" "83549"  
[49] "8824"   "8833"   "9"      "978"
```

The main `gage()` function needs a named vector of fold changes, where the names of the values are the Entrez gene IDs.

Note that we used the `mapIDs()` function above to obtain Entrez gene IDs (stored in `res$entrez`) and we have the fold change results from DESeq2 analysis (stored in `res$log2FoldChange`).

```
c(barry=4, angela = 3, becky = 2)
```

```
barry angela becky
4      3      2
```

```
foldchanges <- res$log2FoldChange
names(foldchanges) <- res$entrez

head(foldchanges)
```

```
7105      64102      8813      57147      55732      2268
-0.35070302      NA  0.20610777  0.02452695 -0.14714205 -1.73228897
```

Now let's run the `gage` pathway analysis

```
# do an overlap analysis
keggres = gage(foldchanges, gsets=kegg.sets.hs)
```

Now let's look at the object returned from `gage()`, ie our results here:

```
attributes(keggres)
```

```
$names
[1] "greater" "less"    "stats"
```

“greater” are upregulated pathways, “less” are downregulated.

```
# Look at first three down(less) pathways
head(keggres$less, 3)
```

```
          p.geomean stat.mean      p.val
hsa05332 Graft-versus-host disease 0.0004250461 -3.473346 0.0004250461
hsa04940 Type I diabetes mellitus  0.0017820293 -3.002352 0.0017820293
```

hsa05310 Asthma	0.0020045888	-3.009050	0.0020045888
	q.val	set.size	exp1
hsa05332 Graft-versus-host disease	0.09053483	40	0.0004250461
hsa04940 Type I diabetes mellitus	0.14232581	42	0.0017820293
hsa05310 Asthma	0.14232581	29	0.0020045888

Let's pull up the highlighted pathways and show our differentially expressed genes on the pathway. I will use the "hsa" KEGG id to get the pathway from KEGG and my `foldchange` vector to show my genes.

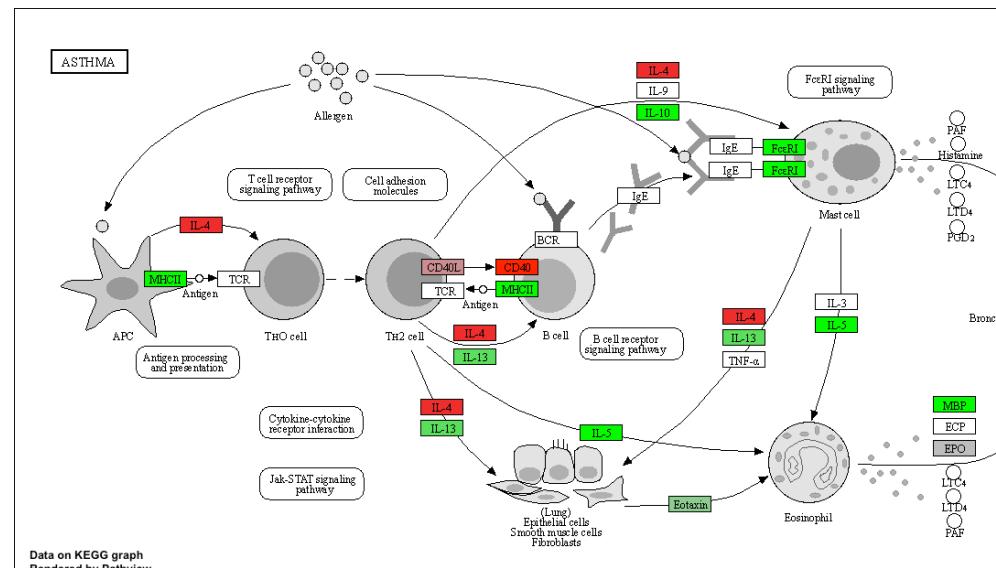
```
pathview(gene.data=foldchanges, pathway.id="hsa05310")
```

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

Info: Working in directory /Users/angelaliu/Documents/bimm143/class12

Info: Writing image file hsa05310.pathview.png

This downloads the pathway figure data from KEGG and adds our results.



Put the image into the document.