

# class13

Mason Lew (PID :A17533139)

2025-02-24

## Table of contents

3. Import countData and colData . . . . .	4
4. Toy differential gene expression . . . . .	5
5. Setting up for DESeq . . . . .	10
6. Principal Component Analysis (PCA) . . . . .	11
7. DESeq analysis . . . . .	12
8. Adding annotation data . . . . .	14
9. Data Visualization . . . . .	15
10. Pathway analysis . . . . .	18

### ##1. Background

Glucocorticoids are used, for example, by people with asthma to reduce inflammation of the airways. The anti-inflammatory effects on airway smooth muscle (ASM) cells has been known for some time but the underlying molecular mechanisms are unclear

Himes et al. used RNA-seq to profile gene expression changes in four different ASM cell lines treated with dexamethasone glucocorticoid.

They found a number of differentially expressed genes but focus much of the discussion on a gene called CRISPLD2.

This gene encodes a secreted protein known to be involved in lung development, and SNPs in this gene in previous GWAS studies are associated with inhaled corticosteroid resistance and bronchodilator response in asthma patients.

### ##2. Bioconductor setup

```
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, saveRDS, setdiff,
table, tapply, union, unique, unsplit, which.max, which.min
```

```
Attaching package: 'S4Vectors'
```

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

```
findMatches
```

```
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, colCummmaxs, 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, rowCummmaxs, 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
```

### 3. Import countData and colData

Loading our data:

```
metaFile <- "/Users/Mason/Desktop/BIMM143/class13/airway_metadata (1).csv"  
countFile <- "/Users/Mason/Desktop/BIMM143/class13/airway_scaledcounts (1).csv"  
  
# Import metadata and take a peak  
metadata = read.csv(metaFile)  
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
```

```
# Import countdata  
  
counts = read.csv(countFile, row.names=1)  
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		

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. Toy differential gene expression

```
control <- metadata[metadata[, "dex"] == "control",]
control.counts <- counts[, control$id]
control.mean <- rowSums(control.counts) / 4
head(control.mean)
```

ENSG000000000003	ENSG000000000005	ENSG000000000419	ENSG000000000457	ENSG000000000460
900.75	0.00	520.50	339.75	97.25
ENSG000000000938				
0.75				

Q3. How would you make the above code in either approach more robust? Is there a function that could help here?

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

ENSG00000000003	ENSG00000000005	ENSG000000000419	ENSG000000000457	ENSG000000000460
658.00	0.00	546.00	316.50	78.75
ENSG000000000938				
0.00				

We will combine our meancount data for bookkeeping purposes.

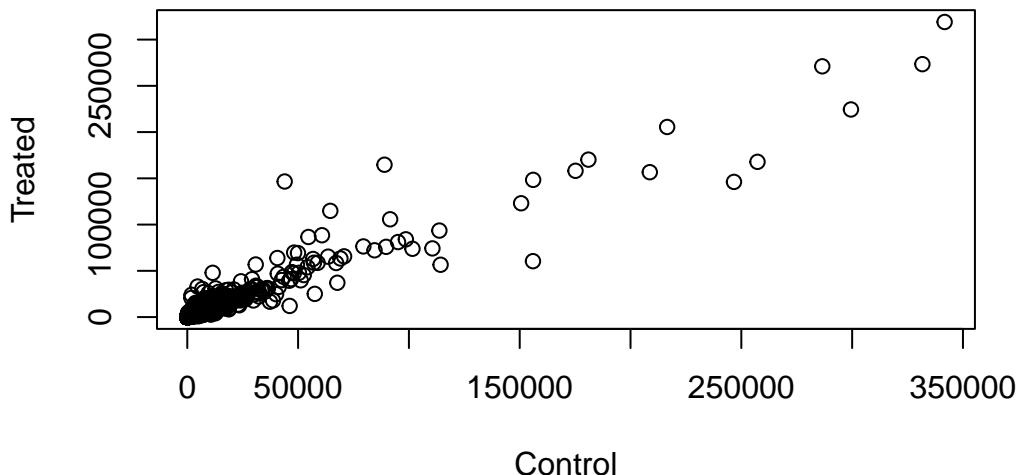
```
meancounts <- data.frame(control.mean, treated.mean)

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. Your plot should look something like the following.

```
plot(meancounts[, 1], meancounts[, 2], xlab="Control", ylab="Treated")
```

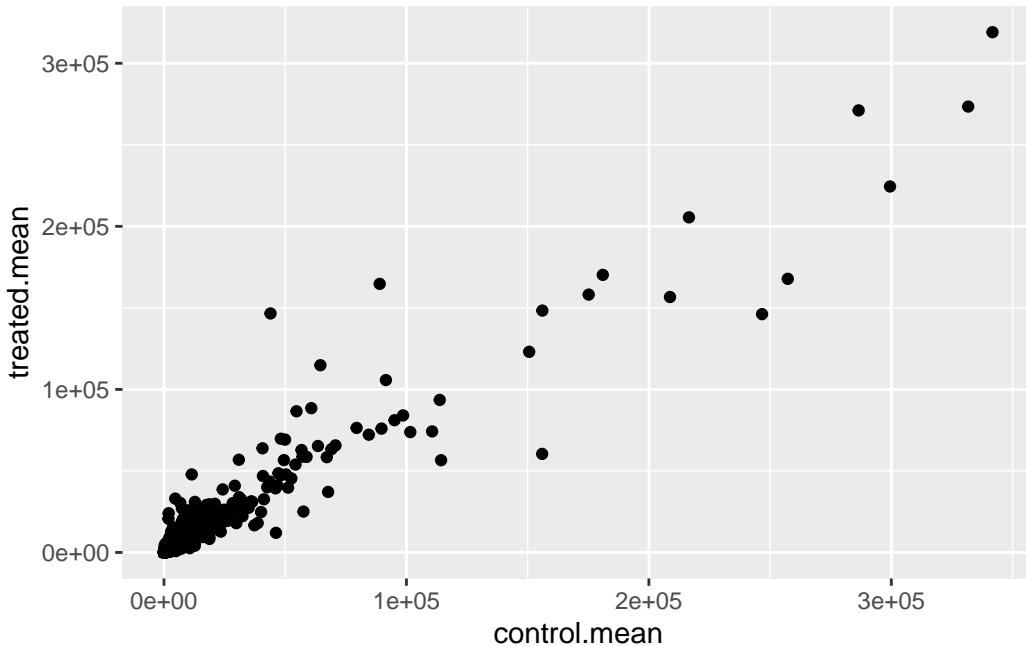


Q5 (b). You could also use the ggplot2 package to make this figure producing the plot below. What geom\_?() function would you use for this plot?

```
geom_point()
```

```
library(ggplot2)

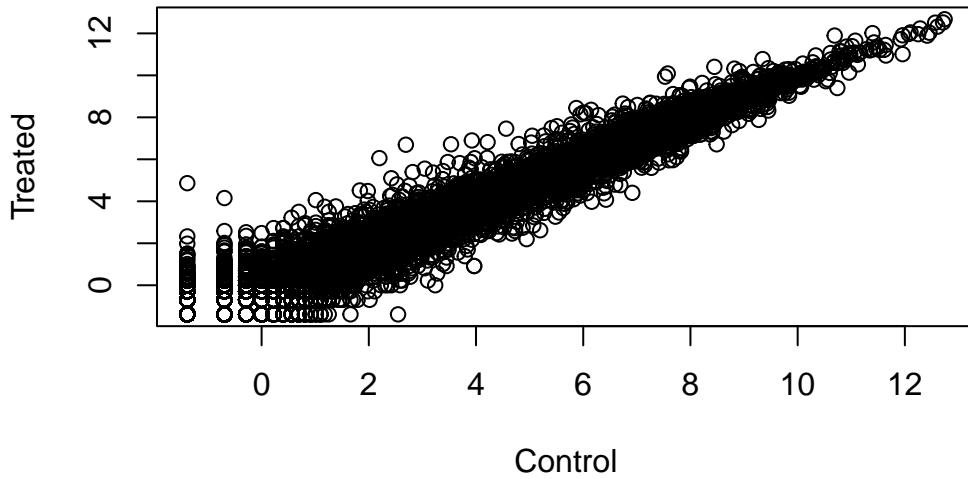
ggplot(meancounts) +
  aes(x = control.mean, y = treated.mean) +
  geom_point()
```



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

```
log()
```

```
plot(log(meancounts[,1]),log(meancounts[,2]), xlab="Control", ylab="Treated")
```



Here we calculate `log2foldchange`, add it to our `meancounts` data.frame and inspect the results either with the `head()` or the `View()` function for example.

```
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

The NaN is returned when you divide by zero and try to take the log. The -Inf is returned when you try to take the log of zero. It turns out that there are a lot of genes with zero expression. Let's filter our data to remove these genes. Again inspect your result (and the intermediate steps) to see if things make sense to you

```
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?

arr.ind causes the which() function to return both row and column positions where there are true values.

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

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?

No, have not done any sort of statistical analysis of the data to determine p-values

## 5. Setting up for DESeq

```
library(DESeq2)
```

```
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

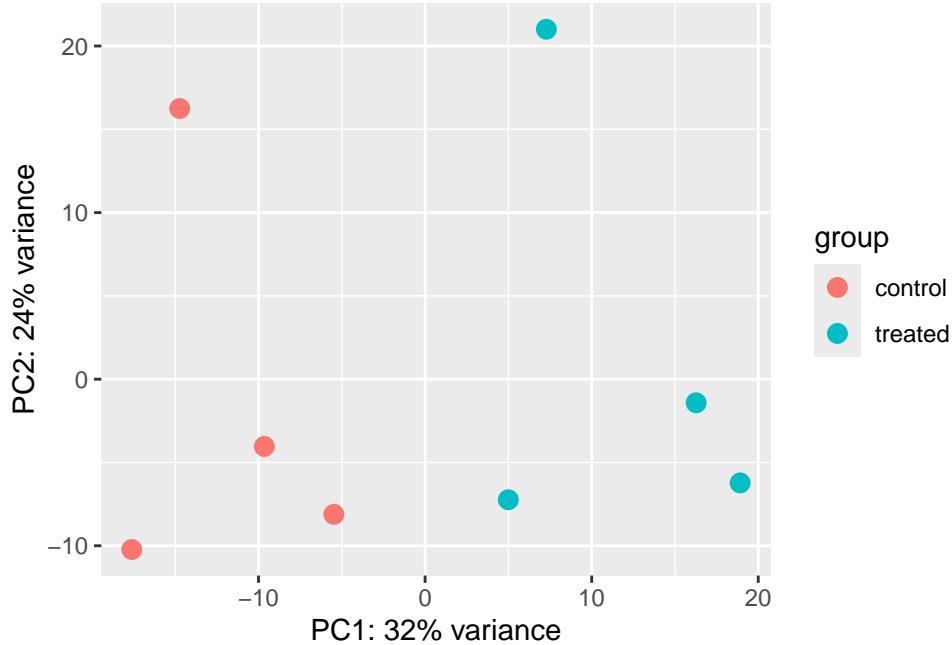
```
dds
```

```
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
```

## 6. Principal Component Analysis (PCA)

```
vsd <- vst(dds, blind = FALSE)
plotPCA(vsd, intgroup = c("dex"))
```

using ntop=500 top features by variance



```
pcaData <- plotPCA(vsd, intgroup=c("dex"), returnData=TRUE)
```

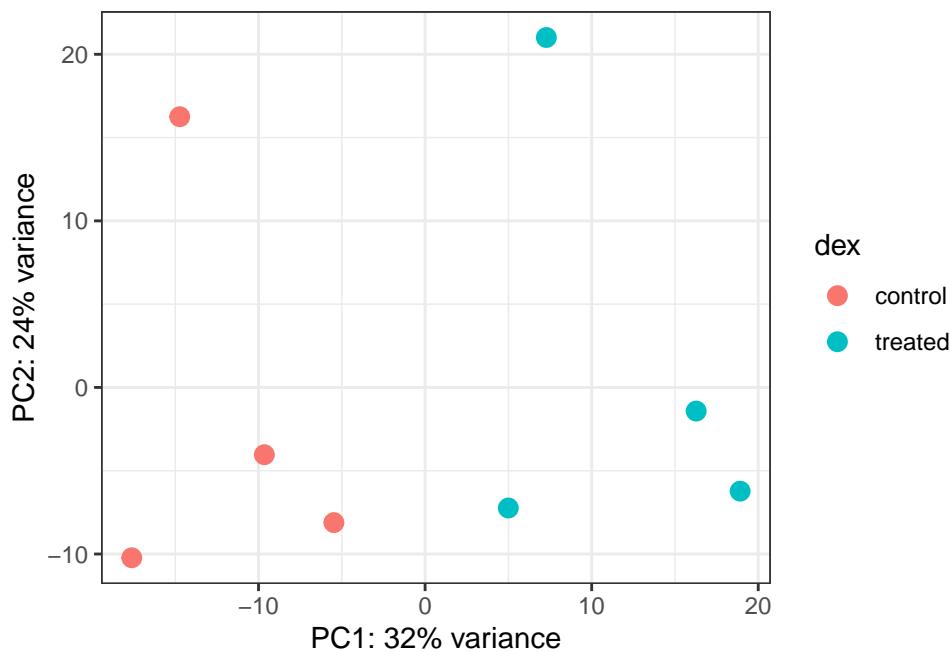
using ntop=500 top features by variance

```
head(pcaData)
```

	PC1	PC2	group	dex	name
SRR1039508	-17.607922	-10.225252	control	control	SRR1039508
SRR1039509	4.996738	-7.238117	treated	treated	SRR1039509
SRR1039512	-5.474456	-8.113993	control	control	SRR1039512
SRR1039513	18.912974	-6.226041	treated	treated	SRR1039513
SRR1039516	-14.729173	16.252000	control	control	SRR1039516
SRR1039517	7.279863	21.008034	treated	treated	SRR1039517

```
# Calculate percent variance per PC for the plot axis labels
percentVar <- round(100 * attr(pcaData, "percentVar"))
```

```
ggplot(pcaData) +
  aes(x = PC1, y = PC2, color = dex) +
  geom_point(size = 3) +
  xlab(paste0("PC1: ", percentVar[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar[2], "% variance")) +
  coord_fixed() +
  theme_bw()
```



## 7. DESeq analysis

```
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>
ENSG000000000003 0.163035
ENSG000000000005     NA
ENSG000000000419 0.176032
ENSG000000000457 0.961694
ENSG000000000460 0.815849
...
...
ENSG00000283115     NA
ENSG00000283116     NA
ENSG00000283119     NA
ENSG00000283120     NA
ENSG00000283123     NA

```

```
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

```

## 8. Adding annotation data

We will use some bioconductor packages to “map” the ENSEMBLE ids to more useful gene SYMBOL names/ids

we can install these packages with: `BiocManager::install("AnnotationDbi")`

```

library(AnnotationDbi)
library(org.Hs.eg.db)

```

What database identifiers can I translate between here:

```
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"

```

We can now use the `mapIDs()` function to translate/map between these different identifier formats

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`.

Let's add SYMBOL, GENENAME and ENTREZID

```
res$symbol <- mapIds(org.Hs.eg.db,
                      keys = rownames(res),
                      keytype = "ENSEMBL",
                      column = "SYMBOL")  
  
'select()' returned 1:many mapping between keys and columns
```

```
res$genename <- mapIds(org.Hs.eg.db,
                        keys = rownames(res),
                        keytype = "ENSEMBL",
                        column = "GENENAME")
```

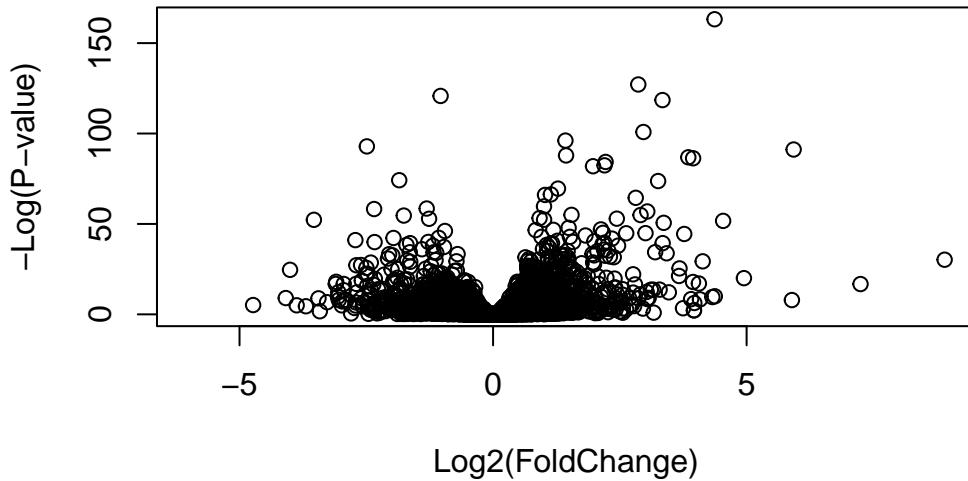
```
'select()' returned 1:many mapping between keys and columns
```

```
res$entrez <- mapIds(org.Hs.eg.db,
                      keys = rownames(res),
                      keytype = "ENSEMBL",
                      column = "ENTREZID")
```

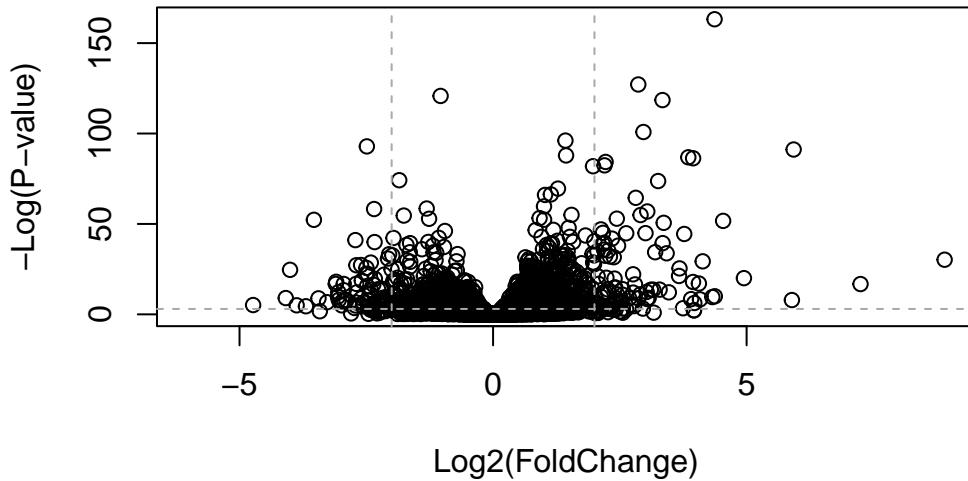
```
'select()' returned 1:many mapping between keys and columns
```

## 9. Data Visualization

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



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



```

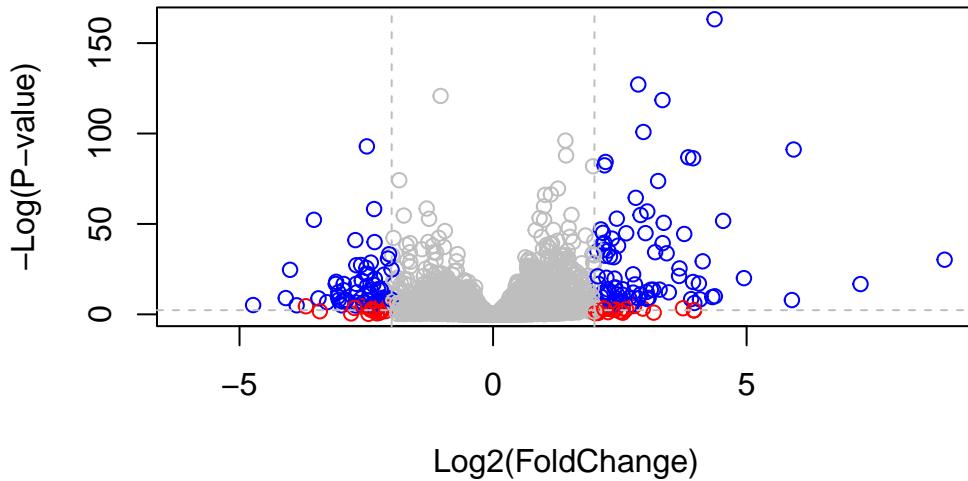
# Setup our custom point color vector
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2 ] <- "red"

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)

```



## 10. Pathway analysis

Now I know the gene names and their IDs in different databases I want to know what type of biology they are involved in...

This is the job of “pathway analysis” (aka “gene set enrichment”)

there are tones of different BioCONductor packages for pathway analysis here. We use just one of them called **gage** and **pathview**. I will install these packages with `BiocManager::install(c("gage", "pathview", "gageData"))`

```
library(gage)
```

```
library(gageData)
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>).

```
#####
#####
```

Load up our Kegg gentrics

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

We will use these KEGG genesets (aka pathways) and our `res` results to see what overlaps To do this we will use the `gage()` function.

For inputs `gage()` wants just a vector of importance - in our case FoldChange values

```
foldchanges <- res$log2FoldChange
```

Vectos in R cna have “names” that are useful for bookkepping so we know what a given balue coressponds to.

Lets put names on our `foldchanges` vecotr - here we will use `res$entrez`

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

Now we can run “pathway analysis”

```

keggres = gage(foldchanges, gsets=kegg.sets.hs)

head(keggres$less)

          p.geomean stat.mean
hsa05332 Graft-versus-host disease      0.0004250461 -3.473346
hsa04940 Type I diabetes mellitus       0.0017820293 -3.002352
hsa05310 Asthma                         0.0020045888 -3.009050
hsa04672 Intestinal immune network for IgA production 0.0060434515 -2.560547
hsa05330 Allograft rejection            0.0073678825 -2.501419
hsa04340 Hedgehog signaling pathway      0.0133239547 -2.248547
                                         p.val      q.val
hsa05332 Graft-versus-host disease      0.0004250461 0.09053483
hsa04940 Type I diabetes mellitus       0.0017820293 0.14232581
hsa05310 Asthma                         0.0020045888 0.14232581
hsa04672 Intestinal immune network for IgA production 0.0060434515 0.31387180
hsa05330 Allograft rejection            0.0073678825 0.31387180
hsa04340 Hedgehog signaling pathway      0.0133239547 0.47300039
                                         set.size      exp1
hsa05332 Graft-versus-host disease      40 0.0004250461
hsa04940 Type I diabetes mellitus       42 0.0017820293
hsa05310 Asthma                         29 0.0020045888
hsa04672 Intestinal immune network for IgA production 47 0.0060434515
hsa05330 Allograft rejection            36 0.0073678825
hsa04340 Hedgehog signaling pathway      56 0.0133239547

```

We can get a pathway image file with our geneset highlighted via the `pathview()` function.

```

pathview(foldchanges, pathway.id = "hsa05310")

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

Info: Working in directory C:/Users/Mason/Desktop/BIMM143/class13

Info: Writing image file hsa05310.pathview.png

```

Insert this figure in my report

