

# Class 15 Transcriptomics and the analysis of RNA-Seq data

Anita Wang (PID: A15567878)

11/16/2021

#Background Our 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).

## 2. Import countData and colData

Read the countData and colData

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

Let's have a look at these

```
head(counts)
```

```
##          SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516
## ENSG00000000003      723        486        904        445       1170
## ENSG00000000005       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
## ENSG00000000003     1097        806        604
## ENSG00000000005       0         0         0
## ENSG000000000419     781        417        509
## ENSG000000000457     447        330        324
## ENSG000000000460      94        102        74
## ENSG000000000938      0         0         0
```

```
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
```

Q1. How many genes are in this dataset?

```
nrow(counts)
```

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

- 38694 genes

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

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

```
## [1] 4
```

- There are 4 control cell lines

#3. 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!

Let’s develop a bit of code that will first find the sample id for those labeled control and then calculate the mean counts per gene across these samples:

First I need to extract all the “control” columns. Then I will take the row wise mean to get the average count values for all genes in these four experiments.

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

```
control mean <- rowMeans(control counts)
```

Q3. How would you make the above code in either approach more robust?

- You should use rowMeans() instead of rowSums() + dividing by 4 because what if the dataset were updated? If the dataset were updated to have more than 4 control groups, the written code using rowSums() would prove not robust and would not work.

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 do the same for the drug treated experiments (i.e. columns)

```

treated inds <- metadata$dex == "treated"
treated counts <- counts[, treated inds]
head(treated counts)

##          SRR1039509 SRR1039513 SRR1039517 SRR1039521
## ENSG000000000003     486       445      1097       604
## ENSG000000000005      0         0         0         0
## ENSG00000000419      523       371      781       509
## ENSG00000000457      258       237      447       324
## ENSG00000000460      81        66       94        74
## ENSG00000000938      0         0         0         0

```

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

We will combine our meancount data for bookkeeping purposes.

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

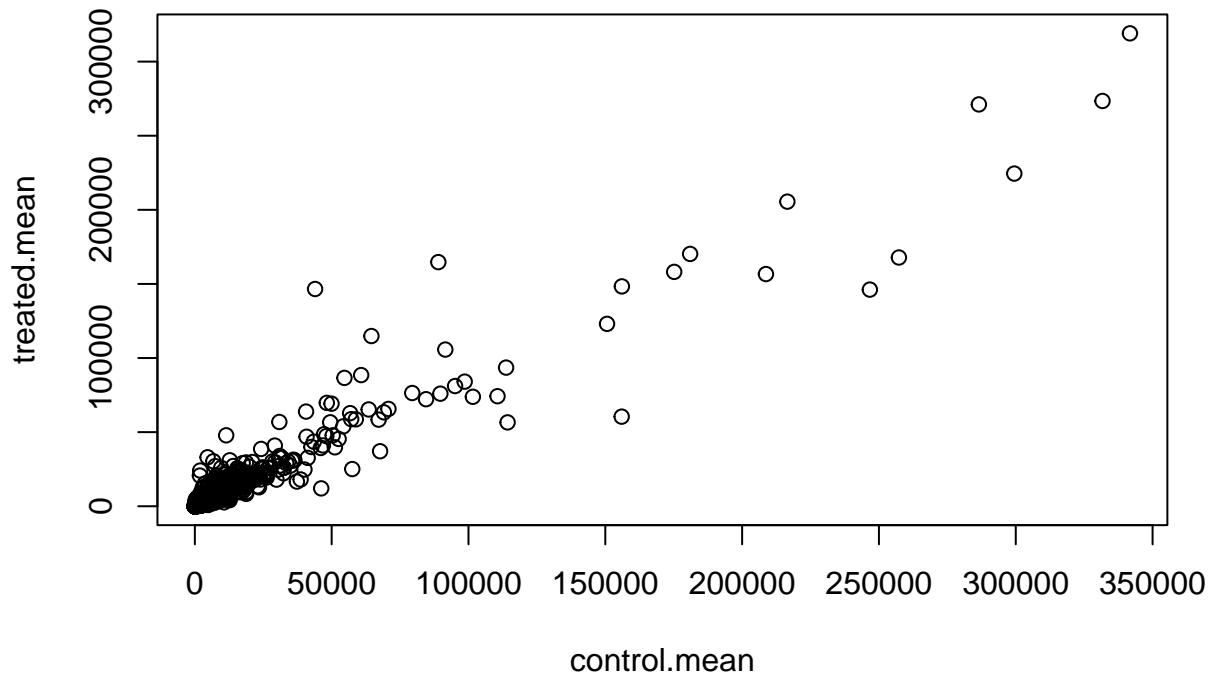
Now show the sum of the mean counts across all genes for each group:

```
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)
```



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?

- You would use `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.

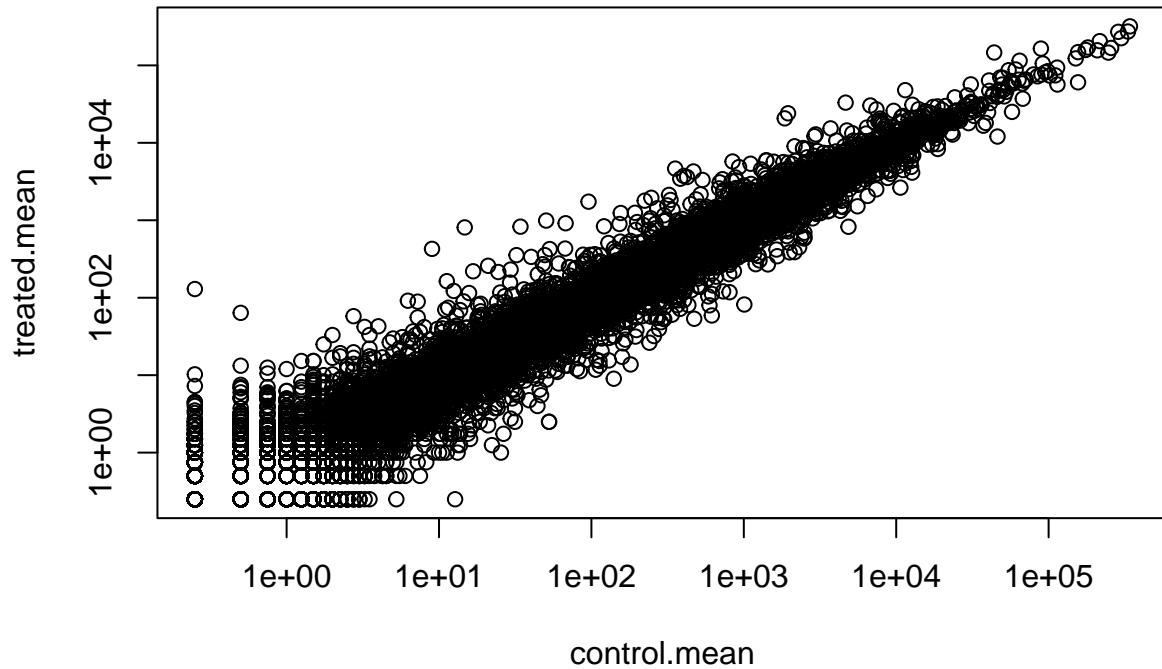
This plot indicates that we need to transform our data!

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

```
plot(meancounts, 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
```



We often use `log2` in this field because it has nice math properties that make interpretation easier.

Ex: if  $\log_2(\text{somenum})=0$ , this means that theres no change. Thus deviations from 0 indicate change (either positive or negative)

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

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

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

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

```
log2(40/10)
```

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

```
log2(5/10)
```

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

We see 0 values for no change and + values for increases and - values for decreases. This nice property leads us to work with **log2(fold-change)** all the time in the genomics and proteomics field.

Let's add the `log2(fold-change)` values to our `meancounts` dataframe.

We can find candidate differentially expressed genes by looking for genes with a large change between control and dex-treated samples. We usually look at the  $\log_2$  of the fold change, because this has better mathematical properties.

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
```

There are a couple of “weird” results. Namely, the `NaN` (“not a number”) and `-Inf` (negative infinity) results.

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

Let's just **exclude** these weird values for genes (i.e. rows) that we can't say anything about since we have no data for them:

```
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
## ENSG000000001036    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` argument, when set to `TRUE`, returns array indices (positions) of where `x` is in the array (in the table). The `arr.ind=TRUE` argument gets the columns and rows where the `TRUE` values are (i.e. the zero counts in our case)
- We then take the first column of the output and call the `unique()` function because we would like to ensure that we aren't double-counting any row

A common threshold used for calling something differentially expressed is a  $\log_2(\text{FoldChange})$  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 up-regulated genes at the greater than 2fc 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(down.ind)
```

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

There are 367 down-regulated genes at the greater than 2 fc level

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

- No. We're missing statistical analyses to help us ensure that the fold changes are significant. Even large fold-changes are not necessarily statistically significant. Thus, these current results may be misleading and inaccurate.
- We need to utilize the DESeq2 package in order to properly perform analysis.

#### #4. DESeq2 analysis – ADDING THE STATISTICS

Let's do this the right way. DESeq2 is an R package specifically for analyzing count-based NGS data like RNA-seq.

```
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, 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, colAnyNs, colAvgsPerRowSet, colCollapse,
##   colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##   colDiffss, colIQRDiffss, colIQRs, colLogSumExps, colMadDiffss,
##   colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##   colProds, colQuantiles, colRanges, colRanks, colSdDiffss, colSds,
##   colSums2, colTabulates, colVarDiffss, colVars, colWeightedMads,
##   colWeightedMeans, colWeightedMedians, colWeightedSds,
##   colWeightedVars, rowAlls, rowAnyNAs, rowAnyNs, rowAvgsPerColSet,
##   rowCollapse, rowCounts, rowCummaxs, rowCummins, 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

citation("DESeq2")

##
##      Love, M.I., Huber, W., Anders, S. Moderated estimation of fold change
##      and dispersion for RNA-seq data with DESeq2 Genome Biology 15(12):550
##      (2014)
##
## A BibTeX entry for LaTeX users is
##
## @Article{,
##   title = {Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2},
##   author = {Michael I. Love and Wolfgang Huber and Simon Anders},
##   year = {2014},
##   journal = {Genome Biology},
##   doi = {10.1186/s13059-014-0550-8},
##   volume = {15},
##   issue = {12},
##   pages = {550},
## }

```

### Importing data

DESeq works on a particular type of object called a DESeqDataSet.

The DESeqDataSet is a single object that contains input values, intermediate calculations like how things are normalized, and all results of a differential expression analysis.

You can construct a DESeqDataSet from (1) a count matrix, (2) a metadata file, and (3) a formula indicating the design of the experiment.

(3): tells DESeq2 which columns in the sample information table (colData) specify the experimental design (i.e. which groups the samples belong to) and how these factors should be used in the analysis. Essentially, this formula expresses how the counts for each gene depend on the variables in colData.

Take a look at metadata again. The thing we're interested in is the dex column, which tells us which samples are treated with dexamethasone versus which samples are untreated controls. We'll specify the design with a tilde, like this: design=~dex.

Use the `DESeqDataSetFromMatrix()` function to build the required `DESeqDataSet` object and call it `dds`, short for our `DESeqDataSet`:

```
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
```

### *DESeq analysis*

Let's run the `DESeq` analysis pipeline on the dataset, and reassign the resulting object back to the same variable. Note that before we start, `dds` is a bare-bones `DESeqDataSet`. The `DESeq()` function takes a `DESeqDataSet` and returns a `DESeqDataSet`, but with additional information filled in (including the differential expression results we are after). Notice how if we try to access these results before running the analysis, nothing exists.

Now we can run `DESeq` analysis:

```
dds <- DESeq(dds)

## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

### *Getting results*

Since we've got a fairly simple design (single factor, two groups, treated versus control), we can get results out of the object simply by calling the `results()` function on the `DESeqDataSet` that has been run through the pipeline.

```

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>
## 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
##           <numeric>
## ENSG000000000003 0.163035
## ENSG000000000005  NA
## ENSG000000000419 0.176032
## ENSG000000000457 0.961694
## ENSG000000000460 0.815849
## ENSG000000000938  NA

```

Convert the res object to a data.frame with the as.data.frame() function and then pass it to View() to bring it up in a data viewer:

```

dat.fr.res <- as.data.frame(res)
View(dat.fr.res)

```

Now summarize some basic tallies using the summary function:

```
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:

```

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
```

## #5. Adding annotation data for our genes

Our result table so far only contains the Ensembl gene IDs. However, alternative gene names and extra annotation are usually required for informative interpretation of our results. In this section we will add this necessary annotation data to our results.

For this, we need two bioconductor packages:

```
#First, install packages
#BiocManager::install("AnnotationDbi")
#BiocManager::install("org.Hs.eg.db")

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

```
##
```

Let's then get a list of all available key types that we can use to map between:

```
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 then 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.

```
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
```

```

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, res\$uniprot and res\$genename.

```

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

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

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

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

res$genename <- mapIds(org.Hs.eg.db,
                      keys=row.names(res),
                      keytype="ENSEMBL",
                      column="GENENAME",
                      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>
## 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      uniprot
##           <numeric> <character> <character> <character>
## ENSG000000000003 0.163035      TSPAN6      7105  AOA024RC10
## ENSG000000000005          NA      TNMD      64102  Q9H2S6
## ENSG000000000419 0.176032      DPM1      8813  060762
## ENSG000000000457 0.961694      SCYL3      57147  Q8IZE3
## ENSG000000000460 0.815849      C1orf112  55732  AOA024R922
## ENSG000000000938          NA      FGR      2268  P09769
##           genename
##           <character>
## ENSG000000000003      tetraspanin 6
## ENSG000000000005      tenomodulin
## ENSG000000000419      dolichyl-phosphate m..
## ENSG000000000457      SCY1 like pseudokina..
## ENSG000000000460      chromosome 1 open re..
## ENSG000000000938      FGR proto-oncogene, ..
```

You can arrange and view the results by the 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  AOA024RDE1
## ENSG00000179094 6.13966e-56      PER1        5187  015534
## 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

```

Finally, let's write out the ordered significant results with annotations:

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

#Save our results

Write out whole results dataset (including genes that don't change significantly)

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

Setting a cutoff for alpha:

Focus in on those genes with a small p-value (i.e. show a significant change):

```
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

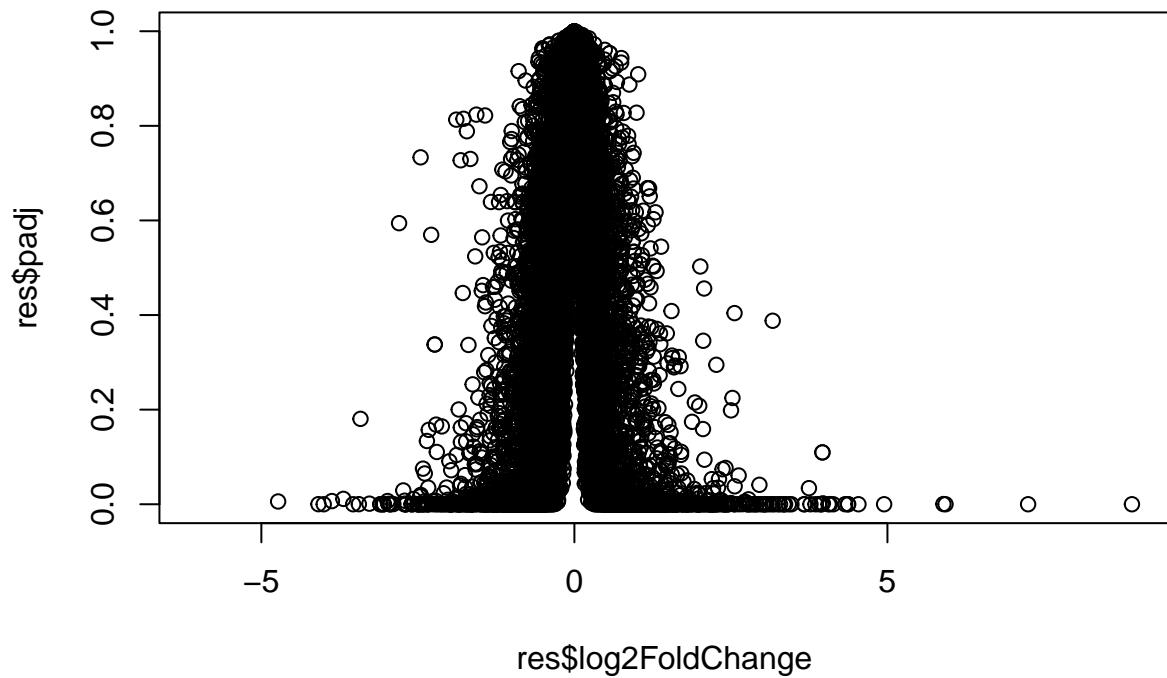
```

#6. Data Visualization

*Volcano Plots*

Let's make a commonly produced visualization from this data, namely a so-called Volcano plot. These summary figures are frequently used to highlight the proportion of genes that are both significantly regulated and display a high fold change.

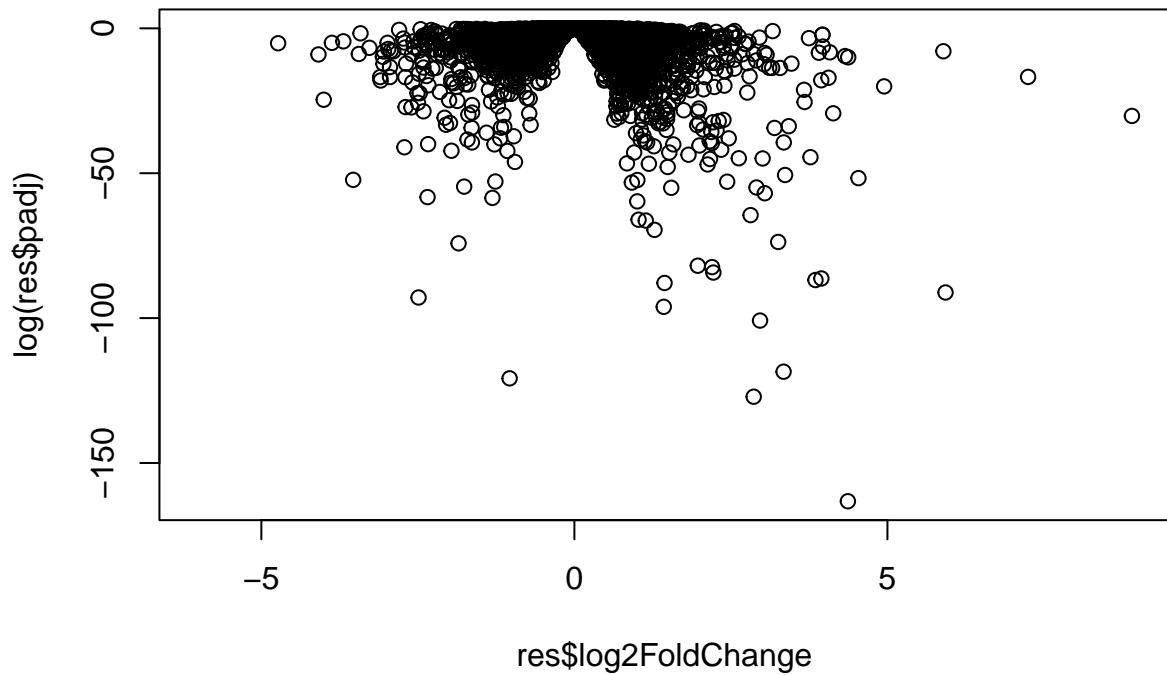
```
plot(res$log2FoldChange, res$padj)
```



This plot isn't too useful as all the small p-values are hidden at the bottom of the plot and we can't really see them. Taking the log will help.

Lets improve it:

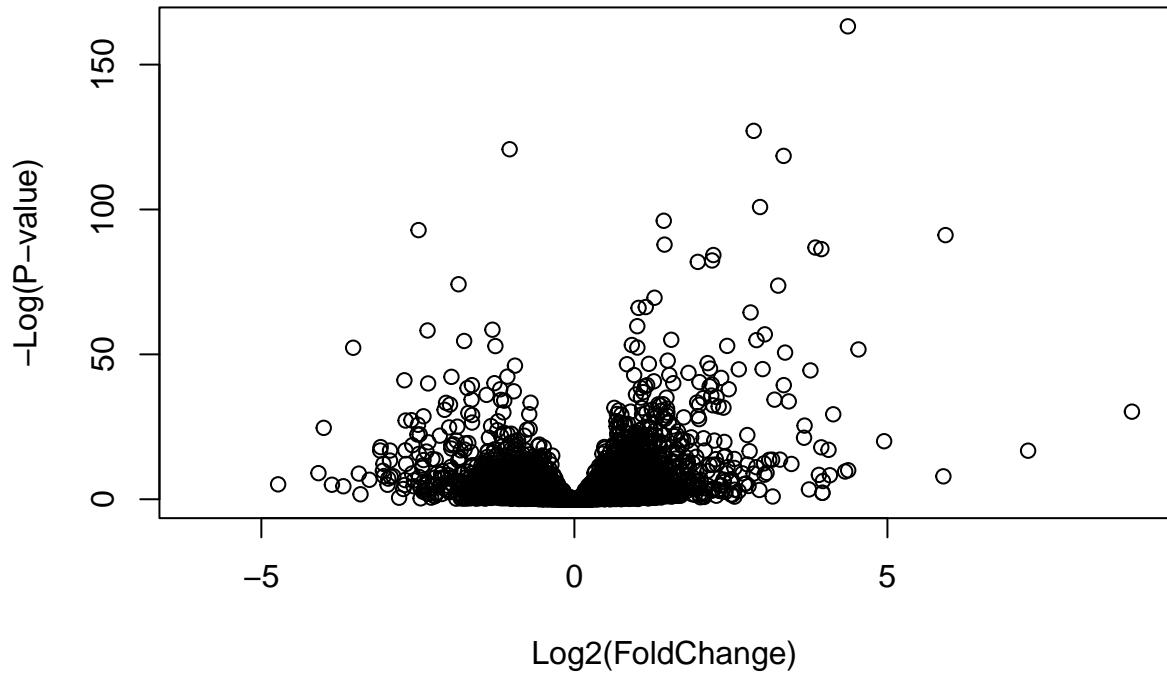
```
plot( res$log2FoldChange, log(res$padj))
```



And some more:

We can flip this p-value axis by adding a minus sign:

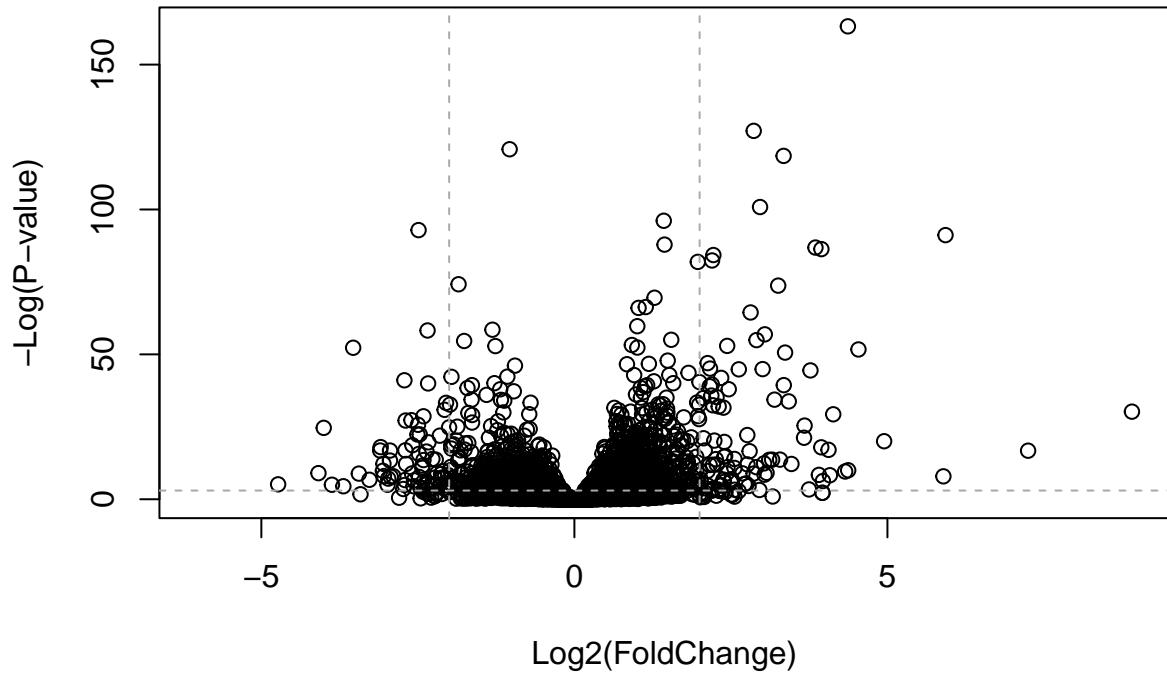
```
plot( res$log2FoldChange, -log(res$padj),
      xlab="Log2(FoldChange)",
      ylab="-Log(P-value)")
```



Now lets make this plot more useful (add some guidelines (with the abline() function) and color (with a custom color vector) highlighting genes that have  $\text{padj} < 0.05$  and the absolute  $\text{log2FoldChange} > 2$ ):

```
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)
```



Finally let's add some color to this plot to draw attention to the genes (i.e. points) we care about - that is those with large fold-change and low p-values (i.e. high  $-\log(p\text{-value})$ )

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

```

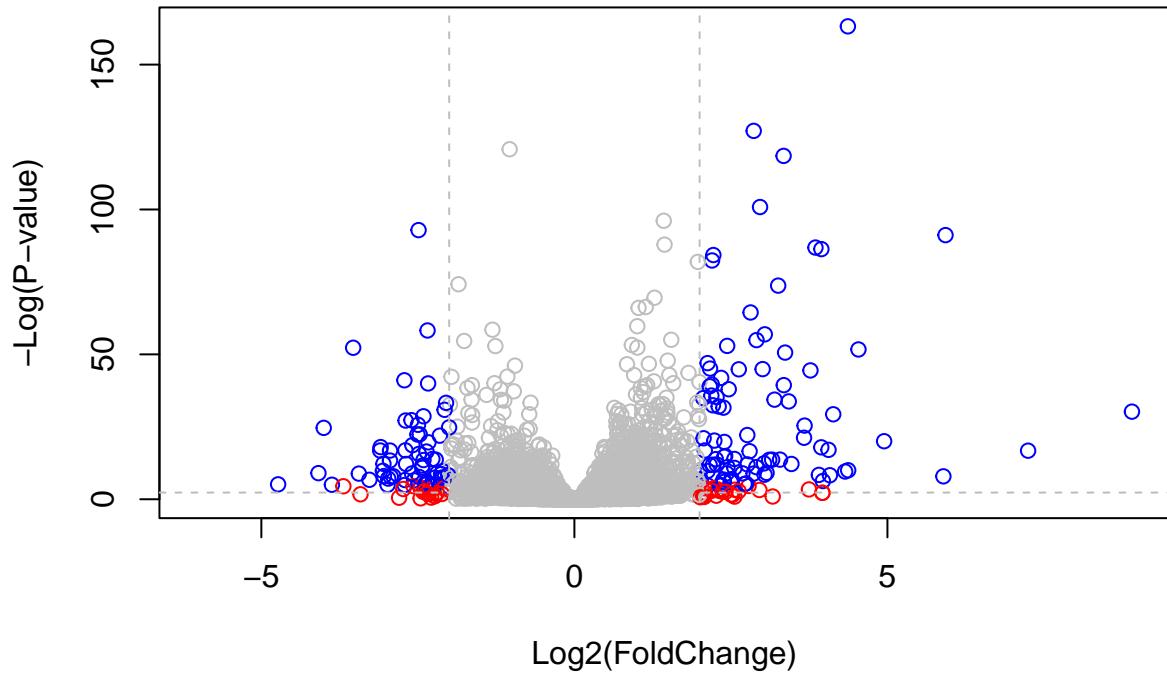
# Setup our custom point color vector
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2 ] <- "red"

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)

```



NEXT CLASS 11/18: For even more customization you might find the EnhancedVolcano bioconductor package useful (Note. It uses ggplot under the hood):

First we will add the more understandable gene symbol names to our full results object `res` as we will use this to label the most interesting genes in our final plot.

11/18: We've added gene labels. Now let's make another volcano plot with some gene labels

For this we can use the *EnhancedVolcano* package:

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

#^ installed but we haven't added gene symbols yet so we'll get back to this next class

library(EnhancedVolcano)

## Loading required package: ggplot2

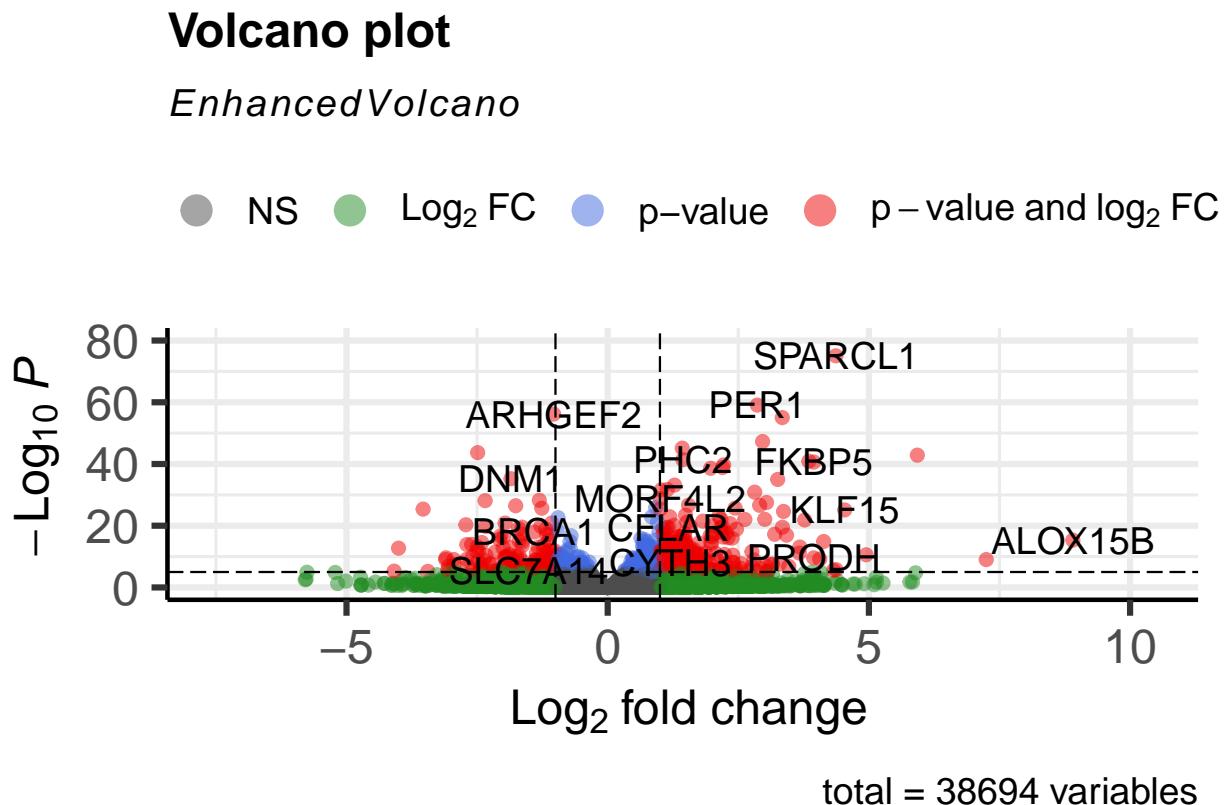
## Loading required package: ggrepel

## Registered S3 methods overwritten by 'ggalt':
##   method           from
##   grid.draw.absoluteGrob  ggplot2
##   grobHeight.absoluteGrob ggplot2
##   grobWidth.absoluteGrob ggplot2
##   grobX.absoluteGrob    ggplot2
##   grobY.absoluteGrob    ggplot2
```

Now plot:

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

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



## #7. Pathway Analysis

Now we will find out how to derive biological (and hopefully) mechanistic insight from the subset of our most interesting genes highlighted in these types of plots:

Pathway analysis (also known as gene set analysis or over-representation analysis), aims to reduce the complexity of interpreting gene lists via mapping the listed genes to known (i.e. annotated) biological pathways, processes and functions.

### Pathway analysis with R and Bioconductor

In this analysis, we check for coordinated differential expression over gene sets from KEGG pathways instead of changes of individual genes. The assumption here is that consistent perturbations over a given pathway (gene set) may suggest mechanistic changes.

Once we have a list of enriched pathways from gage we will use the pathview package to draw pathway diagrams, coloring the molecules in the pathway by their degree of up/down-regulation.

First do a one time install of these required bioconductor packages:

```

# Run in your R console (i.e. not your Rmarkdown doc!)
#BiocManager::install( c("pathview", "gage", "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).
## #####
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 requires a named vector of fold changes, where the names of the values are the Entrez gene IDs

```

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.:

```
# Get the results
keggres = gage(foldchanges, gsets=kegg.sets.hs)
```

Now lets look at the object returned from gage():

attributes(keggres)

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

This separates out results by “greater” and “less” i.e. those that are up-regulated and those that are down-regulated.

Lets look at the first few down (less) pathway results:

```
# Look at the 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
```

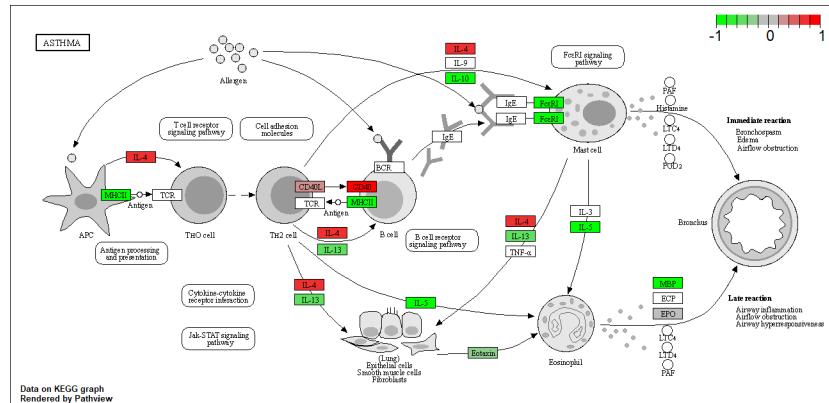
Now, let's try out the `pathview()` function from the `pathview` package to make a pathway plot with our RNA-Seq expression results shown in color:

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

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

## Info: Working in directory C:/Users/anita/Documents/Bimm143/Week4/bimm143/class15

## Info: Writing image file hsa05310.pathview.png



You can play with the other input arguments to `pathview()` to change the display in various ways including generating a PDF graph. For example:

```
# A different PDF based output of the same data
pathview(gene.data=foldchanges, pathway.id="hsa05310", kegg.native=FALSE)
```

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

```
## Info: Working in directory C:/Users/anita/Documents/Bimm143/Week4/bimm143/class15
```

```
## Info: Writing image file hsa05310.pathview.pdf
```

Q12. Can you do the same procedure as above to plot the pathview figures for the top 2 down-regulated pathways?

```
# Look at the first two down-regulated pathways
head(keggres$less, 2)
```

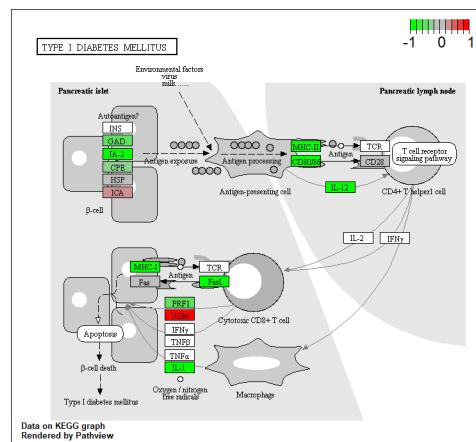
```
##                                     p.geomean stat.mean      p.val
## hsa05332 Graft-versus-host disease 0.0004250461 -3.473346 0.0004250461
## hsa04940 Type I diabetes mellitus 0.0017820293 -3.002352 0.0017820293
##                                     q.val set.size      exp1
## hsa05332 Graft-versus-host disease 0.09053483      40 0.0004250461
## hsa04940 Type I diabetes mellitus 0.14232581      42 0.0017820293
```

```
pathview(gene.data=foldchanges, pathway.id="hsa04940")
```

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

```
## Info: Working in directory C:/Users/anita/Documents/Bimm143/Week4/bimm143/class15
```

```
## Info: Writing image file hsa04940.pathview.png
```



```
pathview(gene.data=foldchanges, pathway.id="hsa05332")
```

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

```
## Info: Working in directory C:/Users/anita/Documents/Bimm143/Week4/bimm143/class15
```

## Info: Writing image file hsa05332.pathview.png

