Week 9 DESeq analysis

Yash Garodia

22/02/2022

This week we are looking at differential expression analysis.

The 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).

Import/Read the data from Himes et al.

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

Lets have a wee peak at the metadata file.

```
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
```

Lets do a little sanity check on the correspondence of counts on metadata

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

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

Q1. How many genes are in this dataset?

There are 38694 genes in this dataset

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

We have 4 control cell lines in this dataset.

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
```

An alternative method is using the dplyr package:

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.0.2
```

```
##
## Attaching package: 'dplyr'
```

```
## 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?

Extract and summarize the control samples

To find out where the control samples are we need the metadata

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

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
##          900.75            0.00          520.50          339.75           97.25
## ENSG00000000938
##            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)

Lets repeat the procedure for the treated columns.

```
n.treated <- sum(metadata$dex == "treated")
```

We have 4 treated cell lines in this dataset

Extract and summarize the treated (i.e drug) samples

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

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

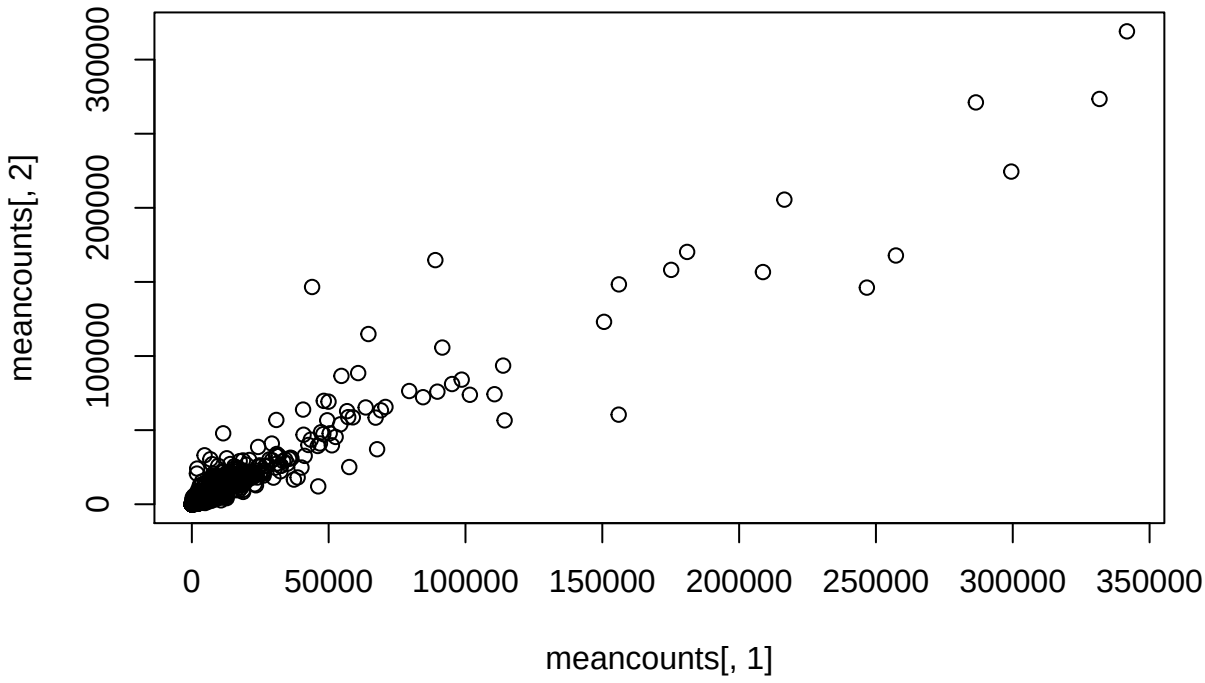
We will combine our meancount data for bookkeeping purposes. We will store these results together in a dataframe called `meancounts`

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

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

Lets make a plot to explore the results a little

```
plot(meancounts[,1], meancounts[,2])
```



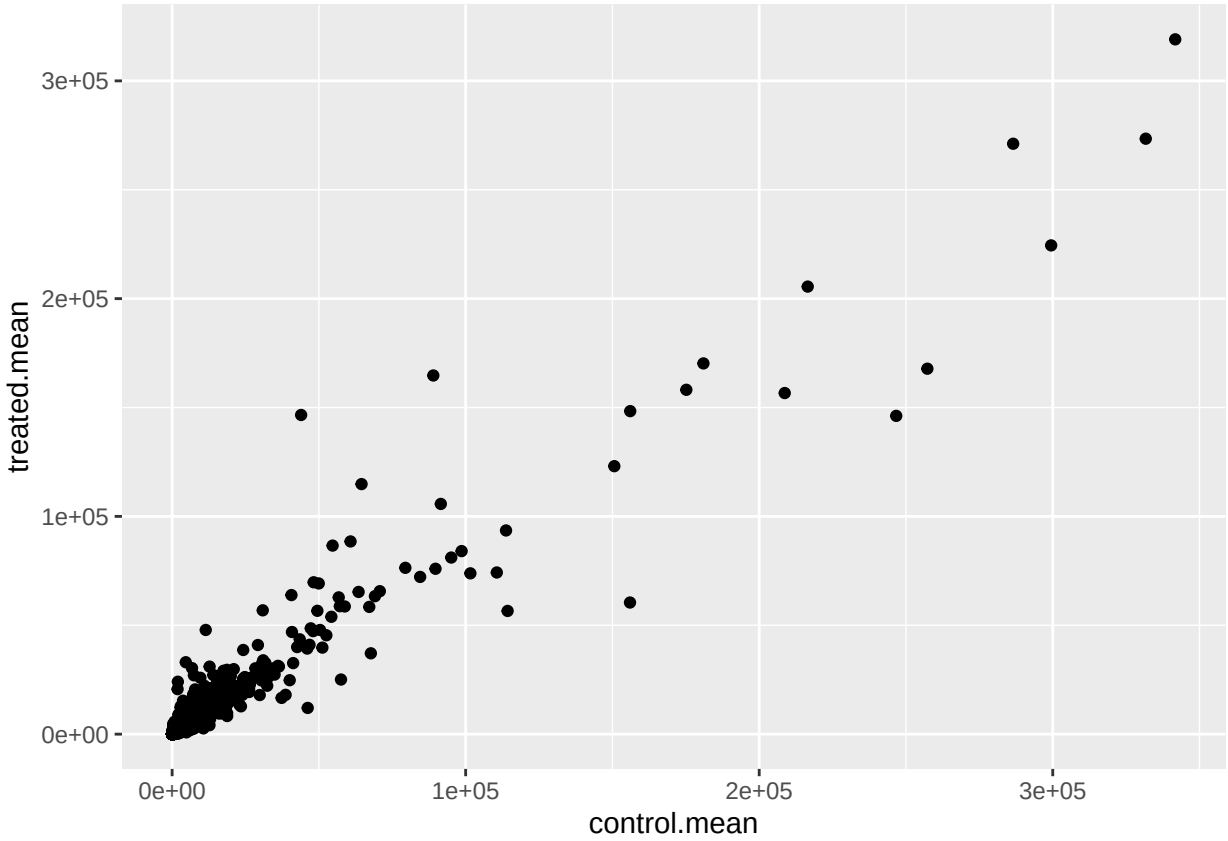
Q5 (b).You could also use the ggplot2 package to make this figure producing the plot below. What geom_?() function would you use for this plot?

We would use a geom_point() function for this plot

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

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

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

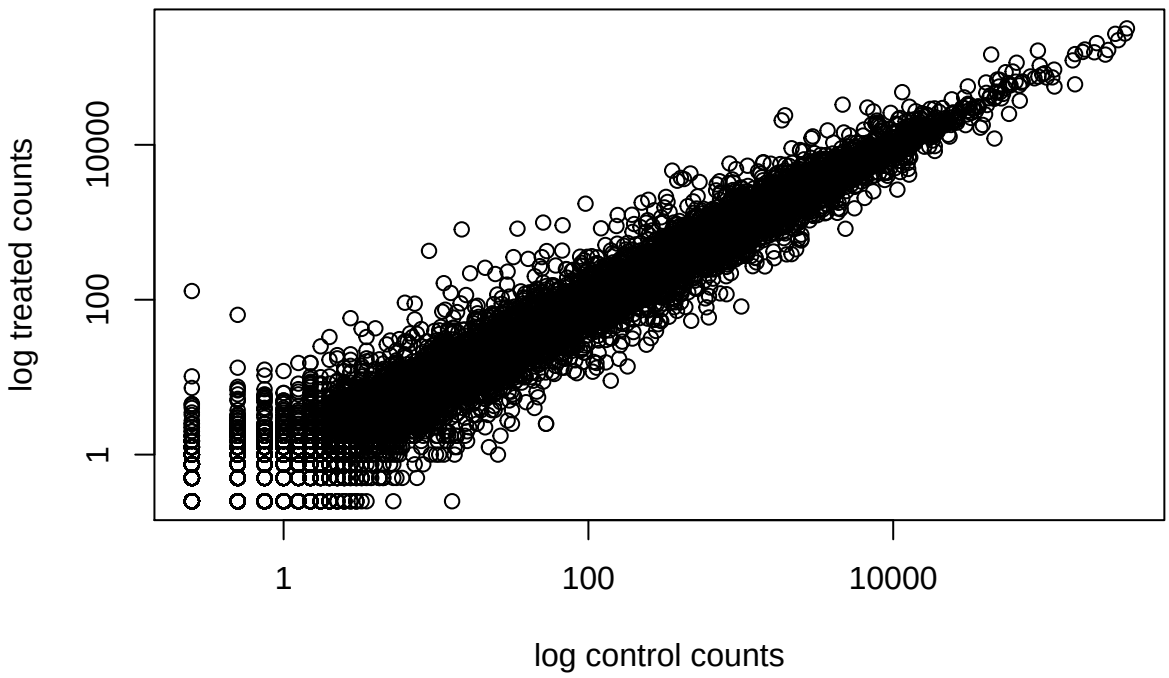
We will make a log-log plot to draw out this skewed data and see whats going on.

We can add the log = "xy" argument to plot() that will allow us to do this.

```
plot(meancounts[,1], meancounts[,2], log = "xy", xlab = "log control counts", ylab = "log treated counts
```

```
## 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
```

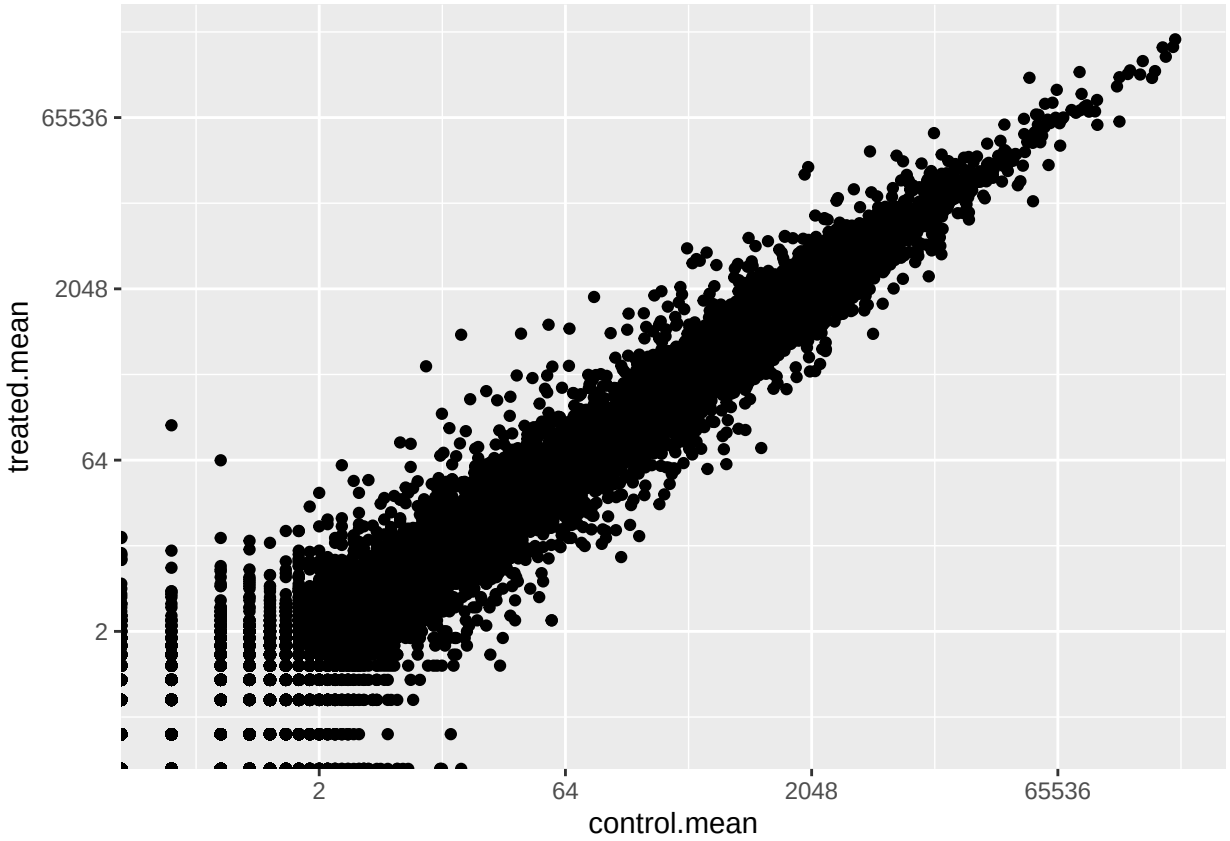


We can repeat the same for ggplot using the scale_x_continuous and scale_y_continuous function:

```
library(ggplot2)
ggplot(meancounts) + aes(control.mean, treated.mean) + geom_point() + scale_x_continuous(trans="log2") +
```

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

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



We often use log2 transformations when dealing with this sort of data.

```
# No effect of drug, so effect remains the same
log2(20/20)
```

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

```
# Suppose drug enhances things:
log2(40/20)
```

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

```
# Suppose drug inhibits things:
log2(20/40)
```

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

```
# What if the scale is different?
log2(80/20)
```

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

This log2 transformation has this nice property where if there is no change then the log2 value will be zero and if it doubles we will have a log2 value of 1 and if halved it will be -1.

So lets add log2 fold change column to our results so far.

```
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 get rid of zero count genes that we cannot say anything about

```
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
```

> 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 purpose of the arr.ind argument is to return only the location of those genes in the dataframe that have either the control.mean = 0 or treated.mean = 0. We then need to call the unique function to get rid of any duplicate genes so each gene location is only mentioned once.

How many genes are remaining?

```
nrow(mycounts)
```

```
## [1] 21817
```

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
```

There are 250 upregulated genes

```
sum(down.ind)
```

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

There are 367 downregulated genes

There are clearly more number of downregulated genes than upregulated genes

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

We cannot trust these results as we are entirely basing our results on a threshold we created. Fold change is very important, but we haven't looked at the statistic aspect and whether the fold change is significant. We haven't looked at the p-values, so we can't tell that the change is significant or not.

DESeq2 analysis

Let's do this the right way. DESeq2 is an R package specifically for analyzing count-based NGS data like RNA-seq. It is available from Bioconductor. Bioconductor is a project to provide tools for analyzing high-throughput genomic data including RNA-seq, ChIP-seq and arrays.

```
# Load up DESeq2

library(DESeq2)
```

```
## Loading required package: S4Vectors

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union

## 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, which.max, which.min

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:dplyr':
##
##     first, rename

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: SummarizedExperiment

## Warning: package 'SummarizedExperiment' was built under R version 4.0.2

## 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")'.
```

```
## Loading required package: DelayedArray

## Warning: package 'DelayedArray' was built under R version 4.0.2

## Loading required package: matrixStats

## Warning: package 'matrixStats' was built under R version 4.0.2

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians

## The following object is masked from 'package:dplyr':
##
##     count

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum
```

```
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
```

```
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>
## ENSG00000000003  747.1942     -0.3507030  0.168246 -2.084470 0.0371175
## ENSG00000000005    0.0000             NA        NA        NA        NA
## ENSG00000000419  520.1342      0.2061078  0.101059  2.039475 0.0414026
## ENSG00000000457  322.6648      0.0245269  0.145145  0.168982 0.8658106
## ENSG00000000460   87.6826     -0.1471420  0.257007 -0.572521 0.5669691
## ...                    ...            ...       ...       ...       ...
## ENSG00000283115  0.000000             NA        NA        NA        NA
## ENSG00000283116  0.000000             NA        NA        NA        NA
## ENSG00000283119  0.000000             NA        NA        NA        NA
## ENSG00000283120  0.974916      -0.668258   1.69456 -0.394354  0.693319
## ENSG00000283123  0.000000             NA        NA        NA        NA
##                       padj
##                  <numeric>
## ENSG00000000003  0.163035
## ENSG00000000005        NA
## ENSG00000000419  0.176032
## ENSG00000000457  0.961694
## ENSG00000000460  0.815849
## ...                   ...
## ENSG00000283115        NA
## ENSG00000283116        NA
## ENSG00000283119        NA
## ENSG00000283120        NA
## ENSG00000283123        NA
```

We can visualize this in a table using this function:

```
resview <- as.data.frame(res)
head(resview)
```

```
##                    baseMean log2FoldChange     lfcSE       stat     pvalue
## ENSG00000000003 747.1941954    -0.35070302 0.1682457 -2.0844697 0.03711747
## ENSG00000000005   0.0000000             NA        NA         NA         NA
## ENSG00000000419 520.1341601     0.20610777 0.1010592  2.0394752 0.04140263
## ENSG00000000457 322.6648439     0.02452695 0.1451451  0.1689823 0.86581056
## ENSG00000000460  87.6826252    -0.14714205 0.2570073 -0.5725210 0.56696907
## ENSG00000000938   0.3191666    -1.73228897 3.4936010 -0.4958463 0.62000288
##                       padj
## ENSG00000000003 0.1630348
## ENSG00000000005        NA
## ENSG00000000419 0.1760317
## ENSG00000000457 0.9616942
## ENSG00000000460 0.8158486
## ENSG00000000938        NA
```

We can get some basic summary tallies using the `summary` function.

```
summary(res, alpha = 0.05)
```

```
##
## out of 25258 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)       : 1242, 4.9%
## LFC < 0 (down)     : 939, 3.7%
## 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
```

Adding Annotation Data

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.

We will use one of Bioconductor's main annotation packages to help with mapping between various ID schemes. Here we load the AnnotationDbi package and the annotation data package for humans org.Hs.eg.db.

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

```
## Warning: package 'AnnotationDbi' was built under R version 4.0.2
```

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

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

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

```
##
```

The later of these is is the organism annotation package ("org") for Homo sapiens ("Hs"), organized as an AnnotationDbi database package ("db"), using Entrez Gene IDs ("eg") as primary key. To get a list of all available key types that we can use to map between, use the columns() function:

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

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

The main function we will use from the AnnotationDbi package is called mapIds().

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.

```
res$symbol <- mapIds(org.Hs.eg.db, keys = row.names(res), keytype = "ENSEMBL", column = "SYMBOL", multi
```

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

lets look at the results obtained:

```
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.

First, lets add the Entrez ID:

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

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

Then lets do it for Uniprot:

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

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

and finally for Genename:

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

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

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  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 regulator 1
## ENSG00000116584 Rho/Rac guanine nucleotide exchange factor 2
## ENSG00000189221                              monoamine oxidase A
## ENSG00000120129              dual specificity phosphatase 1
## ENSG00000148175                                       stomatin
```

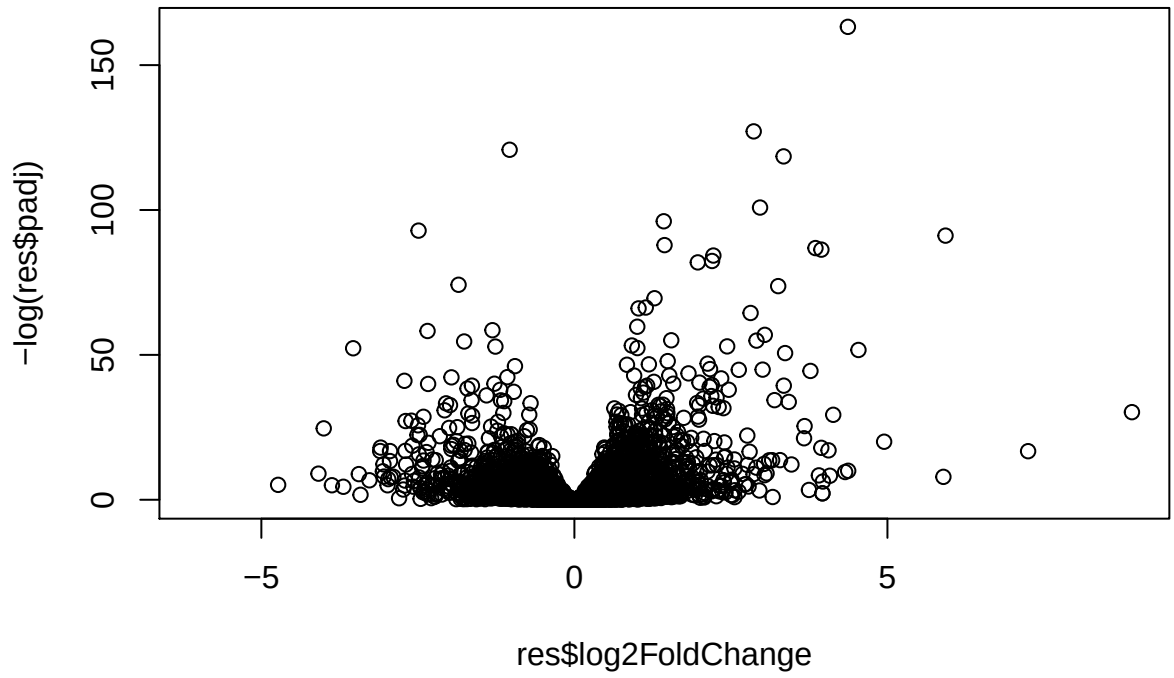
Finally, let's write out the ordered significant results with annotations. See the help for ?write.csv if you are unsure here.

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

Volcano Plot

Make a summary plot of our results:

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



The genes that change a lot have a higher level of change

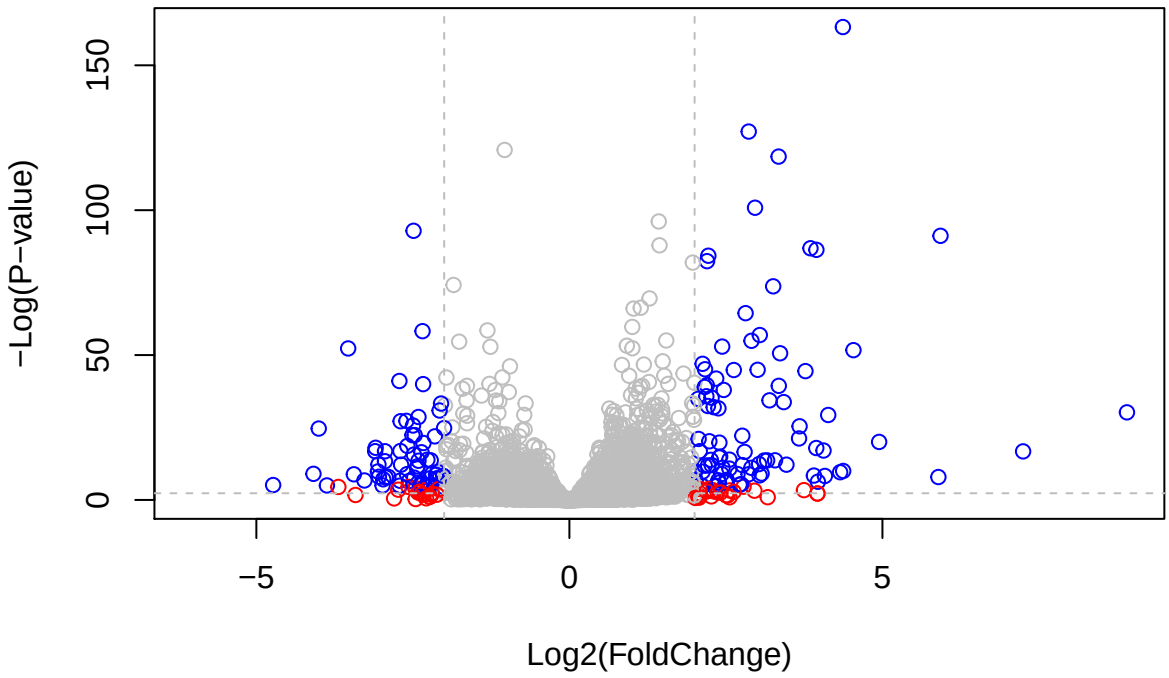
To color the points we will setup a custom color vector indicating transcripts with large fold change and significant differences between conditions:

```
# 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)
```



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

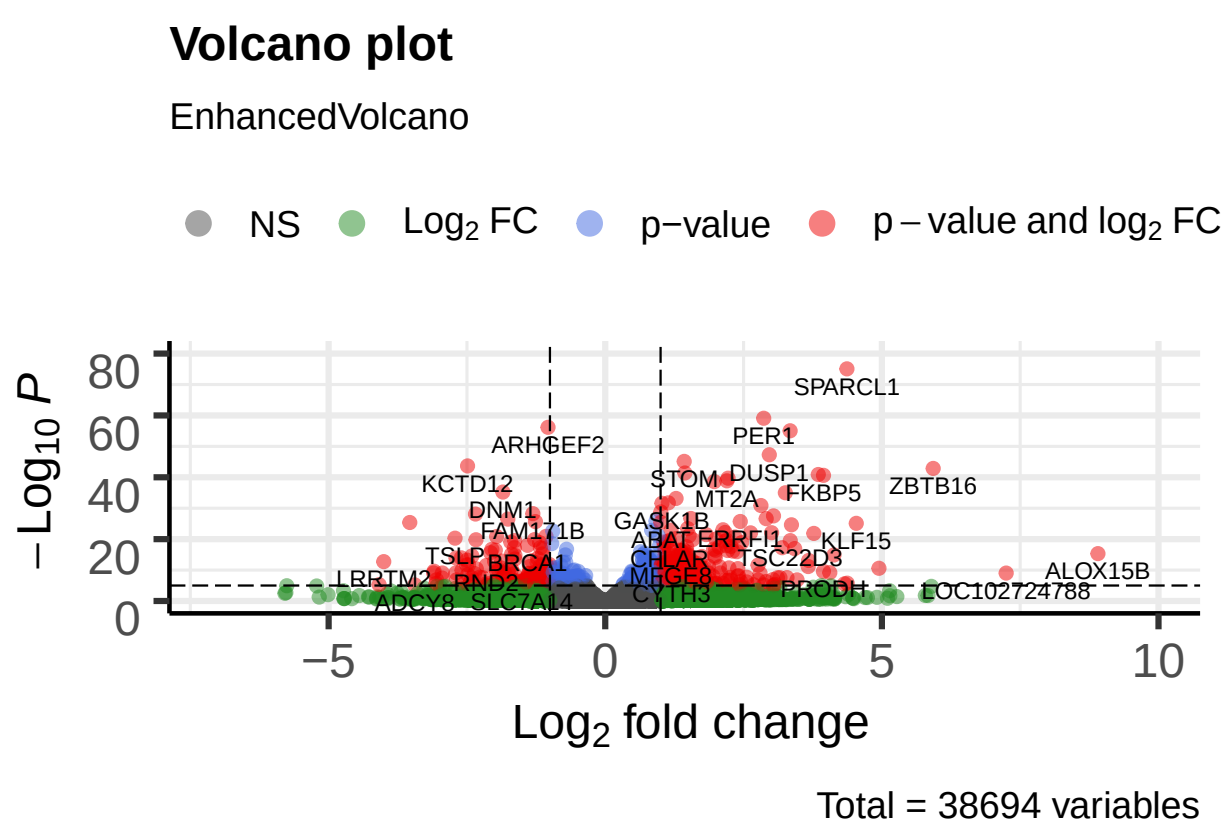
```
#BiocManager::install("EnhancedVolcano")
library(EnhancedVolcano)
```

```
## Loading required package: ggrepel
```

```
## Warning: package 'ggrepel' was built under R version 4.0.2
```

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

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



Pathway Analysis

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.

There are many freely available tools for pathway or over-representation analysis. Ath the time of writting Bioconductor alone has over 80 packages categorized under gene set enrichment and over 120 packages categorized under pathways.

Here we play with just one, the GAGE package (which stands for Generally Applicable Gene set Enrichment), to do KEGG pathway enrichment analysis on our RNA-seq based differential expression results.

```
library("pathview")
```

```
## Warning: package 'pathview' was built under R version 4.0.2
```

```
## ################################################################################
## 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")
```

```
## Warning: package 'gage' was built under R version 4.0.2
```

```
##
```

```
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.

Note that we used the mapIDs() function above to obtain Entrez gene IDs (stored in res$entrez$) and we have the fold change resul

```
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"
```

It is a list with three elements, "greater", "less" and "stats".

You can also see this in your Environmnet panel/tab window of RStudio or use the R command str(keggres).

Like any list we can use the dollar syntax to access a named element, e.g. head(keggres$greater) and head(keggres$less).

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

```
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
```

Each keggres$less and keggres greater object is data matrix with gene sets as rows sorted by p-value.

The top three Kegg pathways indicated here include Graft-versus-host disease, Type I diabetes and the Asthma pathway (with pathway ID hsa05310).

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. To begin with lets manually supply a pathway.id (namely the first part of the "hsa05310 Asthma") that we could see from the print out above.

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

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

```
## Info: Working in directory /Users/yashgarodia/Desktop/UCSD_Coursework/BIMM 143/week 09
```

```
## 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 /Users/yashgarodia/Desktop/UCSD_Coursework/BIMM 143/week 09
```

```
## 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-reguled pathways?

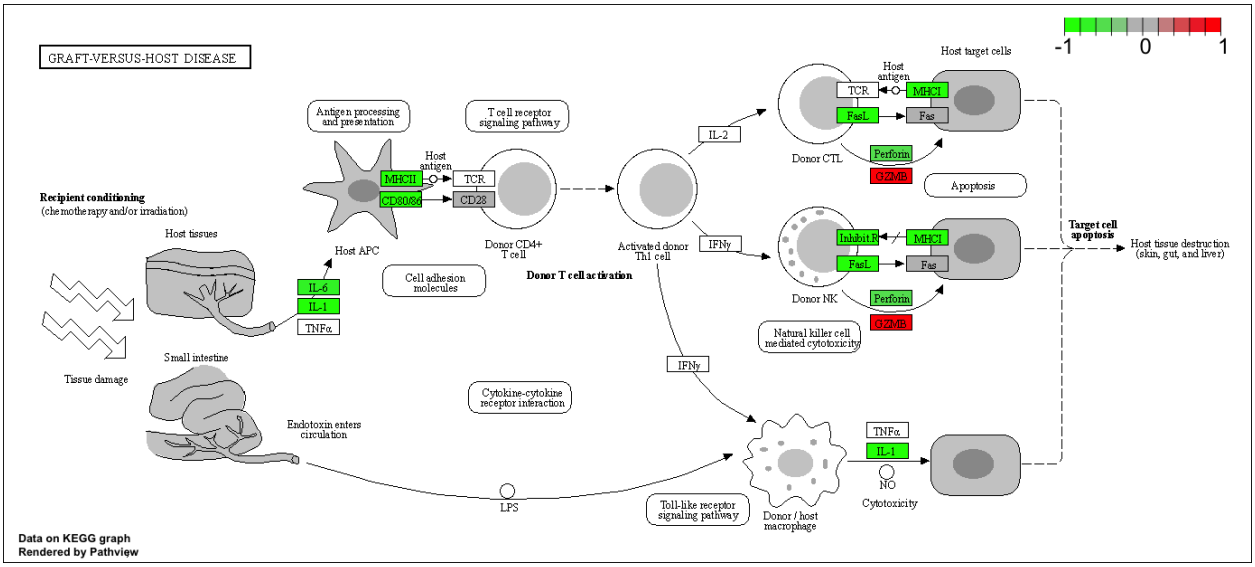
Repeating the procedure using different hsa values: in this case we have hsa05332 and hsa04940

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

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

```
## Info: Working in directory /Users/yashgarodia/Desktop/UCSD_Coursework/BIMM 143/week 09
```

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

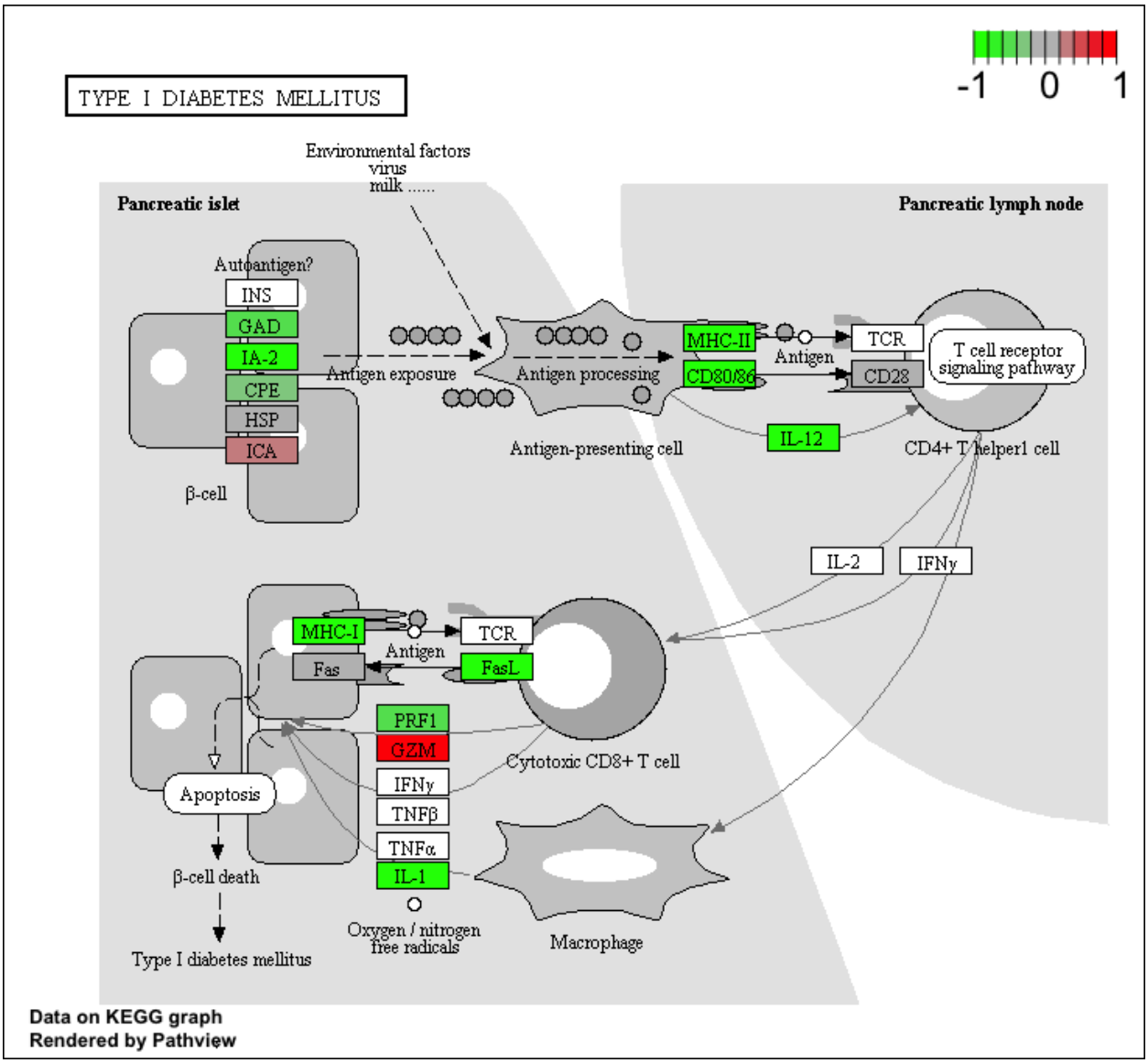



```
pathview(gene.data=foldchanges, pathway.id="hsa04940")
```

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

```
## Info: Working in directory /Users/yashgarodia/Desktop/UCSD_Coursework/BIMM 143/week 09
```

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



Optional: Plotting counts for genes of interest

DESeq2 offers a function called plotCounts() that takes a DESeqDataSet that has been run through the pipeline, the name of a gene, and the name of the variable in the colData that you're interested in, and plots those values. See the help for ?plotCounts. Let's first see what the gene ID is for the CRISPLD2 gene using:

```
i <- grep("CRISPLD2", res$symbol)
res[i,]
```

```
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 1 row and 10 columns
##                  baseMean log2FoldChange     lfcSE      stat      pvalue
##                 <numeric>      <numeric> <numeric> <numeric>   <numeric>
## ENSG00000103196   3096.16        2.62603  0.267444   9.81899 9.32747e-23
```

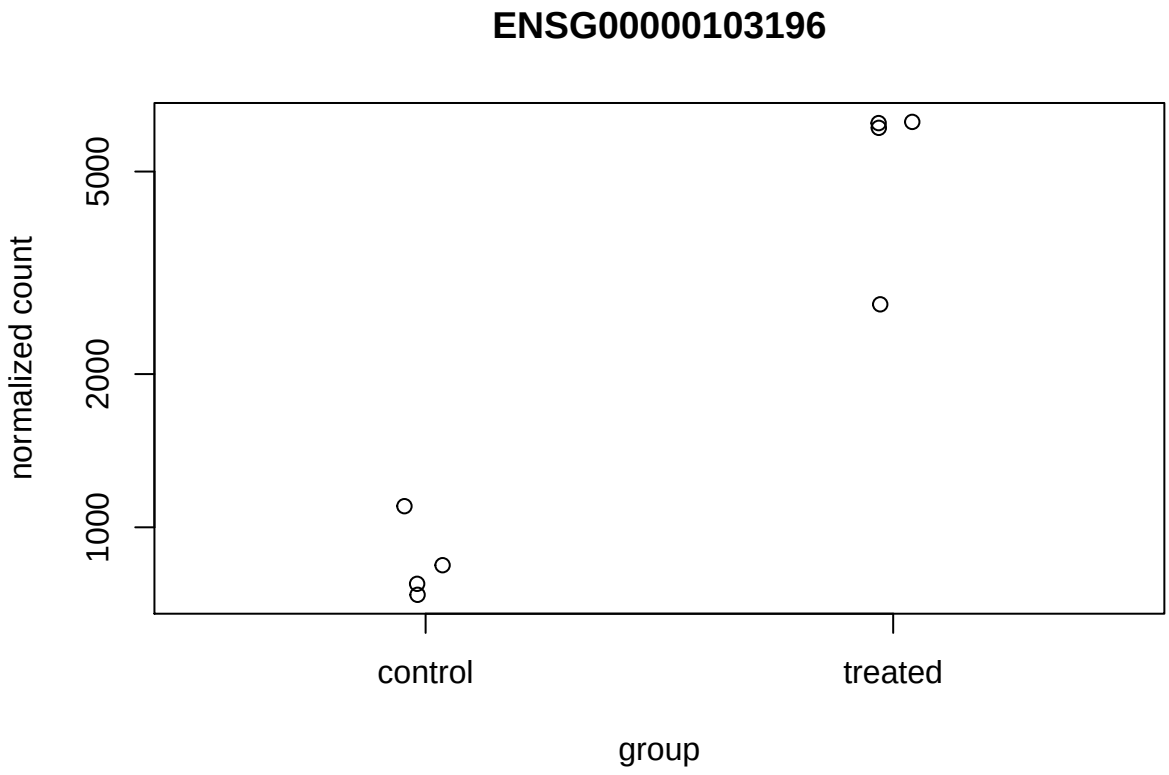
```
##                       padj      symbol      entrez     uniprot
##                  <numeric> <character> <character> <character>
## ENSG00000103196 3.36344e-20    CRISPLD2       83716  A0A140VK80
##                                                            genename
##                                                         <character>
## ENSG00000103196 cysteine rich secretory protein LCCL domain containing 2
```

```
rownames(res[i,])
```

```
## [1] "ENSG00000103196"
```

Now, with that gene ID in hand let's plot the counts, where our intgroup, or "interesting group" variable is the "dex" column.

```
plotCounts(dds, gene = "ENSG00000103196", intgroup = "dex")
```



That's just okay. Keep looking at the help for ?plotCounts. Notice that we could have actually returned the data instead of plotting. We could then pipe this to ggplot and make our own figure. Let's make a boxplot.

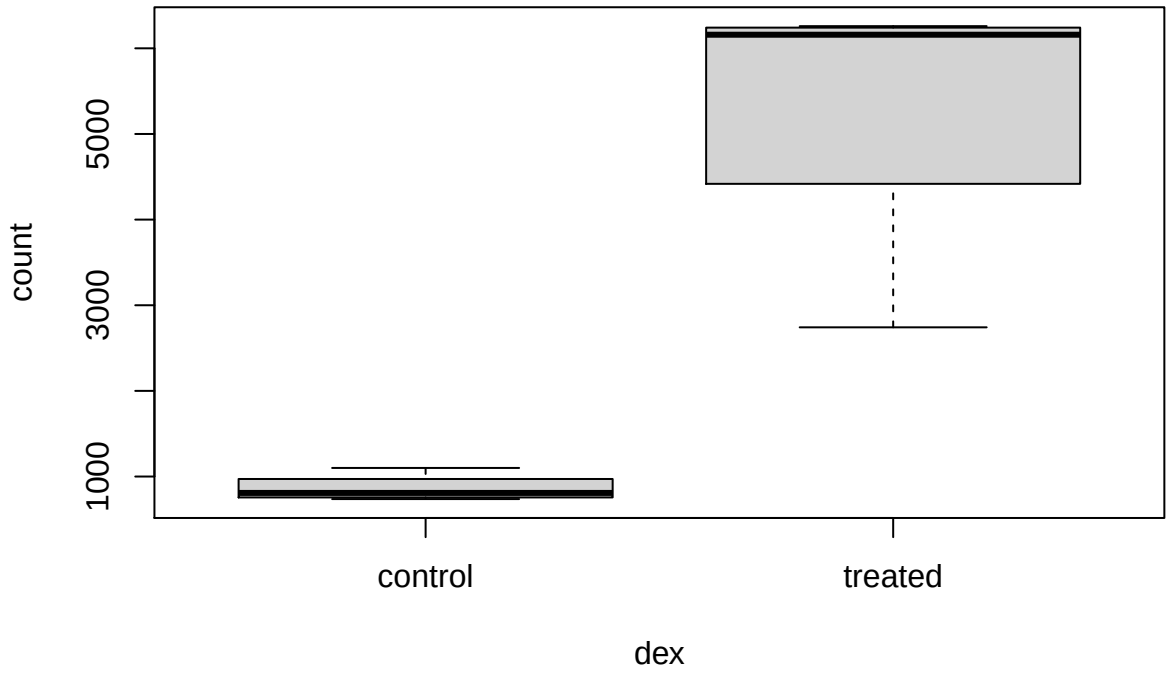
```
d <- plotCounts(dds, gene="ENSG00000103196", intgroup="dex", returnData=TRUE)
head(d)
```

```
##                count     dex
## SRR1039508  774.5002 control
```

```
## SRR1039509 6258.7915 treated
## SRR1039512 1100.2741 control
## SRR1039513 6093.0324 treated
## SRR1039516  736.9483 control
## SRR1039517 2742.1908 treated
```

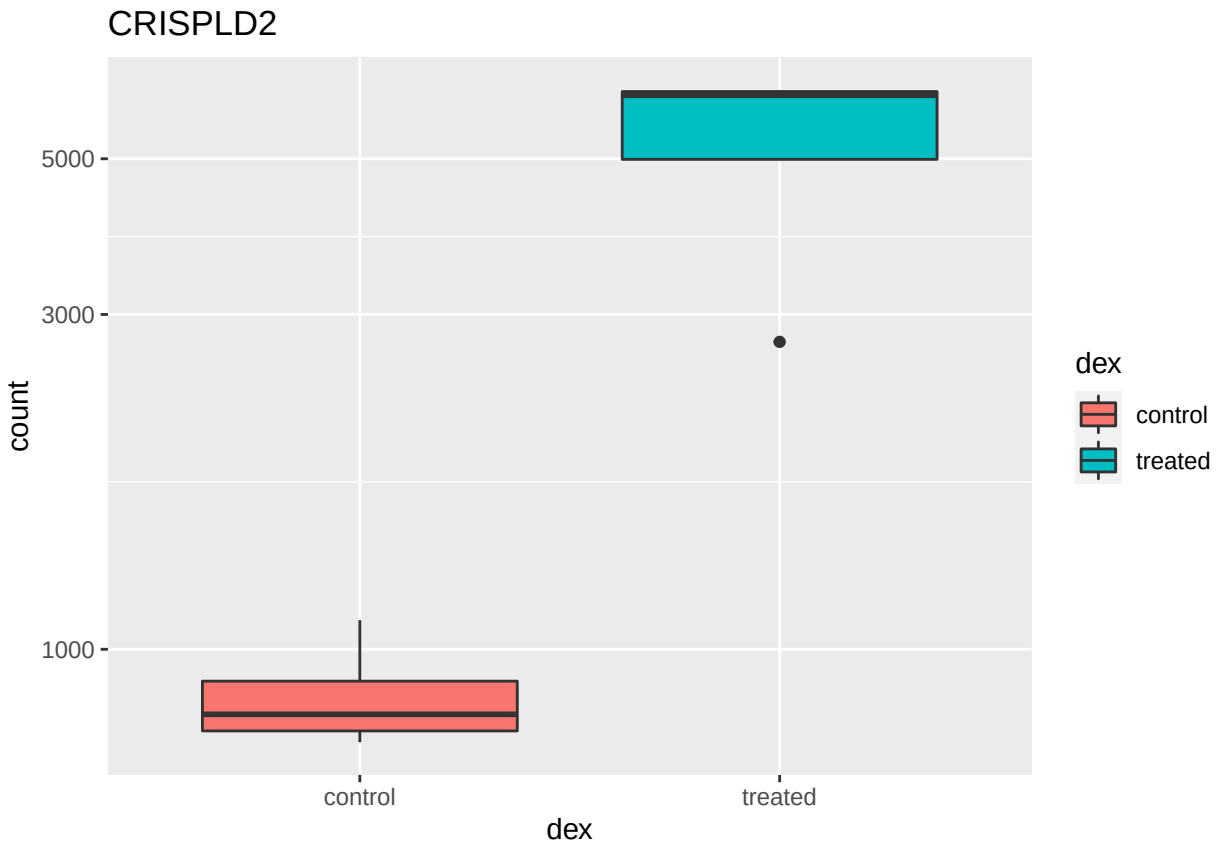
We can mow use this returned object to plot a boxplot with the base graphics function boxplot()

```
boxplot(count ~ dex , data=d)
```



As the returned object is a data.frame it is also all setup for ggplot2 based plotting. For example:

```
library(ggplot2)
ggplot(d) + aes(dex, count, fill = dex) + geom_boxplot() + scale_y_log10() + ggtitle("CRISPLD2")
```



##Session Information

The sessionInfo() prints version information about R and any attached packages. It's a good practice to always run this command at the end of your R session and record it for the sake of reproducibility in the future.

```
sessionInfo()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS  10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] gageData_2.26.0             gage_2.38.3
##  [3] pathview_1.28.1             EnhancedVolcano_1.6.0
```

```
##  [5] ggrepel_0.9.1               org.Hs.eg.db_3.11.4
##  [7] AnnotationDbi_1.50.3        DESeq2_1.28.1
##  [9] SummarizedExperiment_1.18.2 DelayedArray_0.14.1
## [11] matrixStats_0.61.0          Biobase_2.48.0
## [13] GenomicRanges_1.40.0        GenomeInfoDb_1.24.2
## [15] IRanges_2.22.2              S4Vectors_0.26.1
## [17] BiocGenerics_0.34.0         ggplot2_3.3.5
## [19] dplyr_1.0.7
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.2             bit64_4.0.5            splines_4.0.0
##  [4] assertthat_0.2.1       highr_0.9              blob_1.2.2
##  [7] GenomeInfoDbData_1.2.3 yaml_2.2.1             pillar_1.6.3
## [10] RSQLite_2.2.8          lattice_0.20-45        glue_1.6.1
## [13] digest_0.6.28          RColorBrewer_1.1-2     XVector_0.28.0
## [16] colorspace_2.0-2       htmltools_0.5.2        Matrix_1.3-4
## [19] XML_3.99-0.8           pkgconfig_2.0.3        genefilter_1.70.0
## [22] zlibbioc_1.34.0        GO.db_3.11.4           purrr_0.3.4
## [25] xtable_1.8-4           scales_1.1.1           BiocParallel_1.22.0
## [28] tibble_3.1.5           annotate_1.66.0        KEGGREST_1.28.0
## [31] generics_0.1.0         farver_2.1.0           ellipsis_0.3.2
## [34] cachem_1.0.6           withr_2.4.2            survival_3.2-13
## [37] magrittr_2.0.1         crayon_1.4.1           memoise_2.0.0
## [40] evaluate_0.14          KEGGgraph_1.48.0       fansi_0.5.0
## [43] graph_1.66.0           tools_4.0.0            lifecycle_1.0.1
## [46] stringr_1.4.0          munsell_0.5.0          locfit_1.5-9.4
## [49] Biostrings_2.56.0      compiler_4.0.0         rlang_0.4.11
## [52] grid_4.0.0             RCurl_1.98-1.5         bitops_1.0-7
## [55] labeling_0.4.2         rmarkdown_2.11         gtable_0.3.0
## [58] DBI_1.1.1              R6_2.5.1               knitr_1.36
## [61] fastmap_1.1.0          bit_4.0.4              utf8_1.2.2
## [64] Rgraphviz_2.32.0       stringi_1.7.5          Rcpp_1.0.8
## [67] png_0.1-7              vctrs_0.3.8            geneplotter_1.66.0
## [70] tidyselect_1.1.1       xfun_0.29
```

‘‘‘