Class.11

Mirte Ciz Marieke Kuijpers

23/02/2022

Introduction

Today we will run differential expression analysis of some publshed data from Himes et. al.

First we need to do some set up, namely, we must load the required packages. Note that loading a package often causes several messages to be printed, while these are important, we do not necessarily want these messages in the report. Thus I have added `message = FALSE` to the r chunk below to prevent printing of these warning messages.

```
# Load packages
library("BiocManager")
library("DESeq2")
library("ggplot2")
```

The input data must also be loaded and assigned to r objects.

```
# Assign data to r objects
counts <- read.csv("airway_scaledcounts.csv", row.names=1)
metadata <-  read.csv("airway_metadata.csv")

# Inspect both
str(counts)
```

```
## 'data.frame':    38694 obs. of  8 variables:
##  $ SRR1039508: num  723 0 467 347 96 ...
##  $ SRR1039509: num  486 0 523 258 81 ...
##  $ SRR1039512: num  904 0 616 364 73 1 6000 2640 692 531 ...
##  $ SRR1039513: num  445 0 371 237 66 ...
##  $ SRR1039516: num  1170 0 582 318 118 ...
##  $ SRR1039517: num  1097 0 781 447 94 ...
##  $ SRR1039520: num  806 0 417 330 102 ...
##  $ SRR1039521: num  604 0 509 324 74 ...
```

```
head(counts)
```

```
##                 SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516
## ENSG00000000003        723        486        904        445       1170
## ENSG00000000005          0          0          0          0          0
## ENSG00000000419        467        523        616        371        582
## ENSG00000000457        347        258        364        237        318
## ENSG00000000460         96         81         73         66        118
```

```
## ENSG00000000938          0          0          1          0          2
##                  SRR1039517 SRR1039520 SRR1039521
## ENSG00000000003       1097        806        604
## ENSG00000000005          0          0          0
## ENSG00000000419        781        417        509
## ENSG00000000457        447        330        324
## ENSG00000000460         94        102         74
## ENSG00000000938          0          0          0

str(metadata)

## 'data.frame':    8 obs. of  4 variables:
##  $ id      : chr  "SRR1039508" "SRR1039509" "SRR1039512" "SRR1039513" ...
##  $ dex     : chr  "control" "treated" "control" "treated" ...
##  $ celltype: chr  "N61311" "N61311" "N052611" "N052611" ...
##  $ geo_id  : chr  "GSM1275862" "GSM1275863" "GSM1275866" "GSM1275867" ...

head(metadata)

##           id     dex celltype     geo_id
## 1 SRR1039508 control   N61311 GSM1275862
## 2 SRR1039509 treated   N61311 GSM1275863
## 3 SRR1039512 control  N052611 GSM1275866
## 4 SRR1039513 treated  N052611 GSM1275867
## 5 SRR1039516 control  N080611 GSM1275870
## 6 SRR1039517 treated  N080611 GSM1275871
```

The counts data contains quantitative measurements of expression for 38694 genes for 8 conditions/observations. The metadata gives detail on these 8 conditions/observations.

Question 1

The number of genes, as stated above, is 38694.

Question 2

The number of control experiments is 4 (obtained using sum(metadata$dex == “control”) as embedded code in the text). As there are 8 conditions this means we have 4 treated (with drug) and 4 control (without drug) experiments/conditions.

Invesitigating drug effect

We can start to investigate whether the drug has any meaningful effect by doing exploratory differential expression analysis. First check that the metadata and the counts have the same conditions

```
# Do the conditions match
if(all(colnames(counts) == metadata$id)){
  print("The conditions in the two objects are in matching order and analysis
can begin.")
```

```
}else{
  print("The conditions in the two objects do not match or are not in
matching order, please do not continue to analysis until all conditions are
in both objects and are in matching order.")
}
```

```
## [1] "The conditions in the two objects are in matching order and analysis
can begin."
```

```
   # all() checks if ALL inputs are true
```

To simplify the analysis, we can first average control conditions and separately average treatment conditions. To do this first find all control conditions.

```
# Find IDs for control conditions and use them to pull control columns in
counts
control.counts <- counts[ ,metadata$dex == "control"]

# Alternatively, this can be done using multiple steps
control.inds <- which(metadata$dex == "control") # gives the index/positions
of the control conditions
control.counts <- counts[, control.inds] # uses the index vector to get the
control conditions in counts

# Check the data
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
```

The control conditions are: `metadata[metadata$dex == "control", "id"]`.

We should do the same for the treated samples.

```
# Find IDs for treated conditions and use them to pull treated columns in
counts
counts.treated <- counts[ ,metadata$dex == "treated"]

# Check the data
head(counts.treated)
```

```
##                 SRR1039509 SRR1039513 SRR1039517 SRR1039521
## ENSG00000000003        486        445       1097        604
## ENSG00000000005          0          0          0          0
## ENSG00000000419        523        371        781        509
## ENSG00000000457        258        237        447        324
```

```
## ENSG00000000460          81          66          94          74
## ENSG00000000938           0           0           0           0
```

Find the mean count value for each row (i.e. gene). We could use `apply()` with `mean()` or `rowMeans()`.

```
# Get average counts for both subsets
cc.avg <- rowMeans(control.counts)
ct.avg <- rowMeans(counts.treated)
```

Q3 and 4

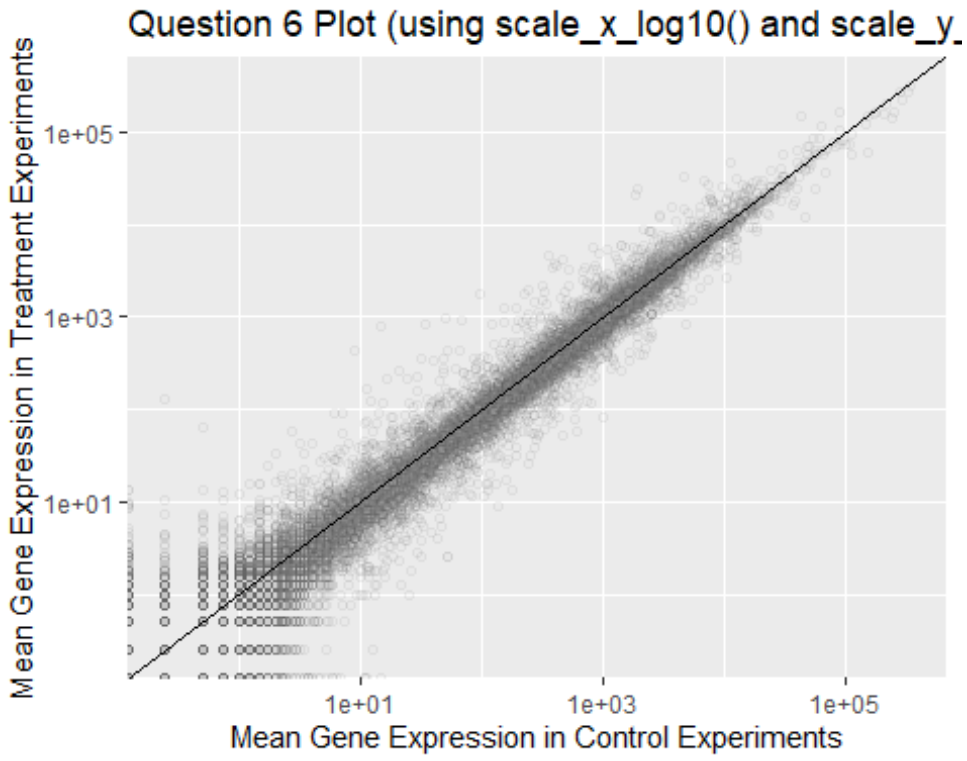
Note that in the tutorial/handout, the following code is used: `library(dplyr) control <- metadata %>% filter(dex=="control") control.counts <- counts %>% select(control$id) control.mean <- rowSums(control.counts)/4 head(control.mean)` Question 3 asks how this could be improved, one thing that could be done is the 4 could be changed to `length(control.counts)` so that if the number of controls changes then the division will not be incorrect. Alternatively, one could just use `rowMeans()` which would be more efficient then the combination of `rowSums()/length()`

Q5

These average values can then be plotted. Anything off a straight 1:1 line suggests differential expression.

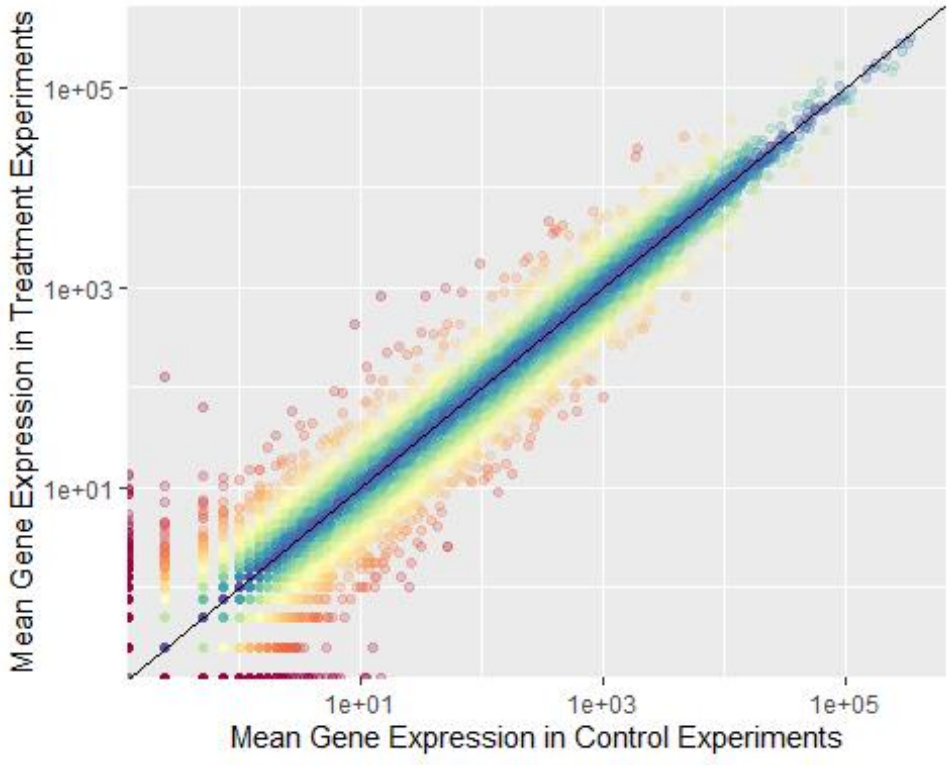
```
# Make a single object input for ggplot.
avgs <- as.data.frame(cbind(cc.avg, ct.avg))

# Make plot for Q5
ggplot(avgs, aes(x = cc.avg, y = ct.avg)) +
  geom_point(pch = 21, alpha = 0.5, fill = "grey") +
  geom_abline(slope = 1) +
  labs(title = "Question 5 Plot (using geom_point)", x = "Mean Gene
Expression in Control Experiments", y = "Mean Gene Expression in Treatment
Experiments")
```


```
# Make plot with log scales Q6
ggplot(avgs, aes(x = cc.avg, y = ct.avg)) +
  geom_point(pch = 21, alpha = 0.05, fill = "grey") +
  geom_abline(slope = 1) +
  scale_x_log10() +
  scale_y_log10() +
  labs(title = "Question 6 Plot (using scale_x_log10() and scale_y_log10)", x
= "Mean Gene Expression in Control Experiments", y = "Mean Gene Expression in
Treatment Experiments")

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

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



It would be nice to colour by distance from the line.

```
# Make a single object input for ggplot.
avgs <- as.data.frame(cbind(cc.avg, ct.avg, abs(as.numeric((cc.avg -
ct.avg)/(cc.avg + ct.avg)))))
colnames(avgs) <- c("Control", "Treatment", "Difference")

# Make a good colour scale (code from stackoverflow)
library("RColorBrewer")
myPalette <- colorRampPalette(rev(brewer.pal(11, "Spectral")))
sc <- scale_colour_gradientn(colours = myPalette(100), limits=c(0, 1))

# Make plot
ggplot(avgs, aes(x = Control, y = Treatment)) +
  geom_point(aes(colour = Difference), alpha = 0.25, show.legend = FALSE) +
  geom_abline(slope = 1) +
  scale_x_log10() +
  scale_y_log10() +
  sc +
  labs(x = "Mean Gene Expression in Control Experiments", y = "Mean Gene
Expression in Treatment Experiments")

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

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



Note while the above graph gives a nice visual effect, the colour scale is not meaningful (does not indicate significance) and is simply for reference.

A log 2 transformation can often be easier to interpret.

```
# A log 2 fold change of 1 is a doubling
log2(40/20)
```

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

```
# Obtain log 2 changes of the input data
log2fc <- log2(ct.avg/cc.avg)
```

It is worth making a dataframe at this point to store all important data together.

```
# Keep data organised by putting it together
meancounts <- data.frame(cc.avg, ct.avg, log2fc)
head(meancounts)
```

```
##                 cc.avg ct.avg      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
```

This also shows that we have some problems, there are some values that are `NaN` (Not a Number) and `-Inf` (negative Infinity). To avoid this we can remove all rows with a zero in them.

```
# Find positions of zeros
ind <- unique(which(meancounts[, 1:2] == 0, arr.ind = TRUE)[, "row"]) #adding the [, "row"] gives just the row and no the column of where the 0 is (as any row with a 0 will be removed, regardless of the column where the 0 is found); unique ensures that row numbers are not repeated if both columns have a 0 in them

# Remove rows which have a zero
meancounts.nz <- meancounts[-ind,]

# Check successful
head(meancounts.nz)
```

```
##                   cc.avg  ct.avg      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
```

Question 7

An explanation of how `ind` was created is founded - `which()` will find all instances in columns 1 and 2 (thanks to [,1:2]) which are 0 (thanks to == 0), however it will return these as if the two columns where one single vector, with the 1st value in the 2nd column equal to 1 + length of first column etc etc. - The `arr.ind` argument, when true, causes it to instead return both the row and column value of each TRUE. - The [, "row"] tells r we only require the column of row numbers `which(, arr.ind = TRUE)` returns because it does not matter which column the zero is present in, if a zero is present in either column the whole row must be removed. - The `unique()` function ensures that if there are some rows where both columns are zero, and thus the row number is returned twice, it is only shown once in the ind vector.

Analysis of the refined dataset.

It is worth doing some analysis of this refined dataset.

```
# How many genes are left?
gl <- nrow(meancounts.nz)
print(paste("Of", nrow(meancounts), "original genes present in the dataset,", nrow(meancounts.nz), "are left after removing all rows with a zero. This is a difference of", nrow(meancounts) - nrow(meancounts.nz), "genes."))
```

```
## [1] "Of 38694 original genes present in the dataset, 21817 are left after removing all rows with a zero. This is a difference of 16877 genes."
```

```
# How many have a log 2 fold change (log2fc) > 2?
g2 <- sum(meancounts.nz[,3] > 2)
```

There are 21817 genes left after removing all rows with a zero. There are 250 genes among these which have a log 2 fold change greater than 2. This is 1.1458954% of the genes with non-zero expression in both conditions.

However, these differences may have no significance, we need to consider the variation within the control group before comparing their average to that of the treatment group. We are going to calculate p values and use multiple testing correction given we are analyzing data for so many genes.

Additionally, we may be missing important changes by removing entries where only one column has 0, to include these would require using pseudocounts which we will not consider today.

Questions 8, 9 and 10

```
up.ind <- meancounts.nz$log2fc > 2
down.ind <- meancounts.nz$log2fc < (-2)
```

Q8 & 9: There are 250 genes which have a greater than log2 fold upregulation in gene expression, and 367 are downregulated with a magnitude greater than a log2 fold change.

Q10: Whether these figures can be trusted has already been partially discussed above. Without knowledge of variation within the control it is hard to determine whether variation from the mean of the control by the mean of the treatment is significant or simply normal. Clearly we require statistical testing to generate results we can interpret with confidence.

DESeq2

As explained at the end of the previous section, we need to include statistics, we can do so with DESeq2. This package was already loaded in the set-up at the beginning of this document. Thus we can move straight to data conversion into a format DESeq2 can read.

```
# Convert data into a format the DESeq2 package can use
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

# Check this object
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
```

We can then run `DESeq` to obtain results including statistics.

```
# Obtain results
dds.r <- DESeq(dds)

## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

res <- results(dds.r)

# Inspect results
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
```

A volcano plot will be a nice summary figure of these results, plotting p value (or more specifically an adjusted version of it) and log2 fold change.

```
# ggplot

# Make a good colour scale (code inspired by stackoverflow)
myPalette <- colorRampPalette(rev(brewer.pal(11, "Spectral")))
sc2 <- scale_colour_gradientn(colours = myPalette(100), limits=c(0, 175))

# plot
ggplot(as.data.frame(res), aes(x = log2FoldChange, y = -log(padj))) +
  geom_point(alpha = 0.75, aes(col = -log(padj))) +
  labs(y = "-log of Multiple Testing Adjusted p Value", x = "Log2 Fold Change
in Gene Expression") +
  sc2 # The colour scheme initiated earlier

## Warning: Removed 23549 rows containing missing values (geom_point).
```

The colour scale above is not as meaningful, rather it would be better to colour with thresholds in mind. Specifically: 0.05 for p value -2 and +2 for log2 fold change

```
# plot with a meaningful colour scheme

#first add another column for one threshold
res.c <- cbind(as.data.frame(res), (res$padj < 0.05))

#add another column for other threshold
res.c2 <- cbind(as.data.frame(res.c), (abs(res$log2FoldChange) > 2))

#fix names
cn <- colnames(as.data.frame(res))
colnames(res.c2) <- c(cn, "t.padj", "t.log2fc")

ggplot(res.c2, aes(x = log2FoldChange, y = -log(padj))) +
  geom_point(alpha = 0.75, aes(col = t.padj)) +
  labs(y = "-log of Multiple Testing Adjusted p Value", x = "Log2 Fold Change
in Gene Expression")

## Warning: Removed 23549 rows containing missing values (geom_point).
```


```
ggplot(res.c2, aes(x = log2FoldChange, y = -log(padj))) +
  geom_point(alpha = 0.75, aes(col = t.log2fc)) +
  labs(y = "-log of Multiple Testing Adjusted p Value", x = "Log2 Fold Change
in Gene Expression")

## Warning: Removed 23549 rows containing missing values (geom_point).
```


```
# To get one plot that has both requires one more column
res.c3 <- cbind(res.c2, (res.c2$t.padj + res.c2$t.log2fc) == 2)
colnames(res.c3) <- c(colnames(res.c2), "col.both")

ggplot(res.c3, aes(x = log2FoldChange, y = -log(padj))) +
  geom_point(alpha = 0.75, aes(col = col.both)) +
  labs(y = "-log of Multiple Testing Adjusted p Value", x = "Log2 Fold Change
in Gene Expression", col = "Passes both thresholds?") +
  geom_vline(xintercept=c(-2, 2), col="red") +
  geom_hline(yintercept=-log10(0.05), col="red")

## Warning: Removed 23549 rows containing missing values (geom_point).
```

The code to make this last plot was a little excessive, but only because I worked slowly through how I created the plot, to make sure that I plotted correctly. A more experienced user could have used far less code to make this plot.

Question 11 onwards are after this point, which was where the class ended on Wednesday.

Continue on from this point on Friday.

Adding annotation data

To help interpret our results we need to understand what the differentially expressed genes are. A first step here is to get the gene names in a format we can understand (i.e. gene SYMBOLs).

For this the following packages needed to be installed as follows: - BiocManager::install("AnnotationDbi") - BiocManager::install("org.Hs.eg.db")

Then we can load the packages (this must be done every time and thus we can include it in a code chunk, unlike the package installation).

```
# Load annotation package
library("AnnotationDbi")

# Load package containing all annotations for the human genome
library("org.Hs.eg.db")

##

# Inspect the org.Hs.eg.db package
columns(org.Hs.eg.db)

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

The columns in `org.Hs.eg.db` represent the possible databases we can take gene names/identifiers from.

We can now proceed with annotation using the `mapIDs()` function from the `AnnotationDbi` package to map gene identifiers of a more human-friendly (readable) format to each row (using the row names which are themselves gene identifiers). We can then add these to the results object we have already made using `DESeq2`.

```
# mapID() function
res$symbol <- mapIds(org.Hs.eg.db,
                     keys=row.names(res), # Our gene identifiers
                     keytype="ENSEMBL",        # The format of the gene
identifiers from keys
                     column="SYMBOL",          # The new format we want to
add
                     multiVals="first")     # There can be multiple
transcripts per gene, we only want the first (the highest frequency version),
note this one to many mapping

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

# Check this was successful
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
```

Question 11

We can also use the same function to add further columns with the identifiers other databases give our gene identifiers (currently the row names in the `res` object).

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

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

# Add uniprot gene identifiers
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

# Add gene names
res$gene.name <- mapIds(org.Hs.eg.db,
                     keys=row.names(res),
                     keytype="ENSEMBL",
                     column="GENENAME",
                     multiVals="first")

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

# Check this was successful
head(res)
```

```
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 6 rows and 10 columns
##                   baseMean log2FoldChange     lfcSE      stat    pvalue
##                  <numeric>      <numeric> <numeric> <numeric> <numeric>
## ENSG00000000003 747.194195     -0.3507030  0.168246 -2.084470 0.0371175
## ENSG00000000005   0.000000             NA        NA        NA        NA
## ENSG00000000419 520.134160      0.2061078  0.101059  2.039475 0.0414026
## ENSG00000000457 322.664844      0.0245269  0.145145  0.168982 0.8658106
## ENSG00000000460  87.682625     -0.1471420  0.257007 -0.572521 0.5669691
## ENSG00000000938   0.319167     -1.7322890  3.493601 -0.495846 0.6200029
##                      padj      symbol      entrez     uniprot
##                 <numeric> <character> <character> <character>
## ENSG00000000003  0.163035      TSPAN6        7105  A0A024RCI0
## ENSG00000000005        NA        TNMD       64102      Q9H2S6
## ENSG00000000419  0.176032        DPM1        8813      O60762
## ENSG00000000457  0.961694       SCYL3       57147      Q8IZE3
## ENSG00000000460  0.815849    C1orf112       55732  A0A024R922
## ENSG00000000938        NA         FGR        2268      P09769
##                           gene.name
##                         <character>
## ENSG00000000003          tetraspanin 6
## ENSG00000000005            tenomodulin
## ENSG00000000419 dolichyl-phosphate m..
## ENSG00000000457 SCY1 like pseudokina..
## ENSG00000000460 chromosome 1 open re..
## ENSG00000000938 FGR proto-oncogene, ..
```

The results can also be ordered for better viewing.

```
# Order the results by p value and place the index of these orders in a
vector
ord <- order( res$padj )

# View the results ordered by the index vector of ordered p values
head(res[ord,])
```

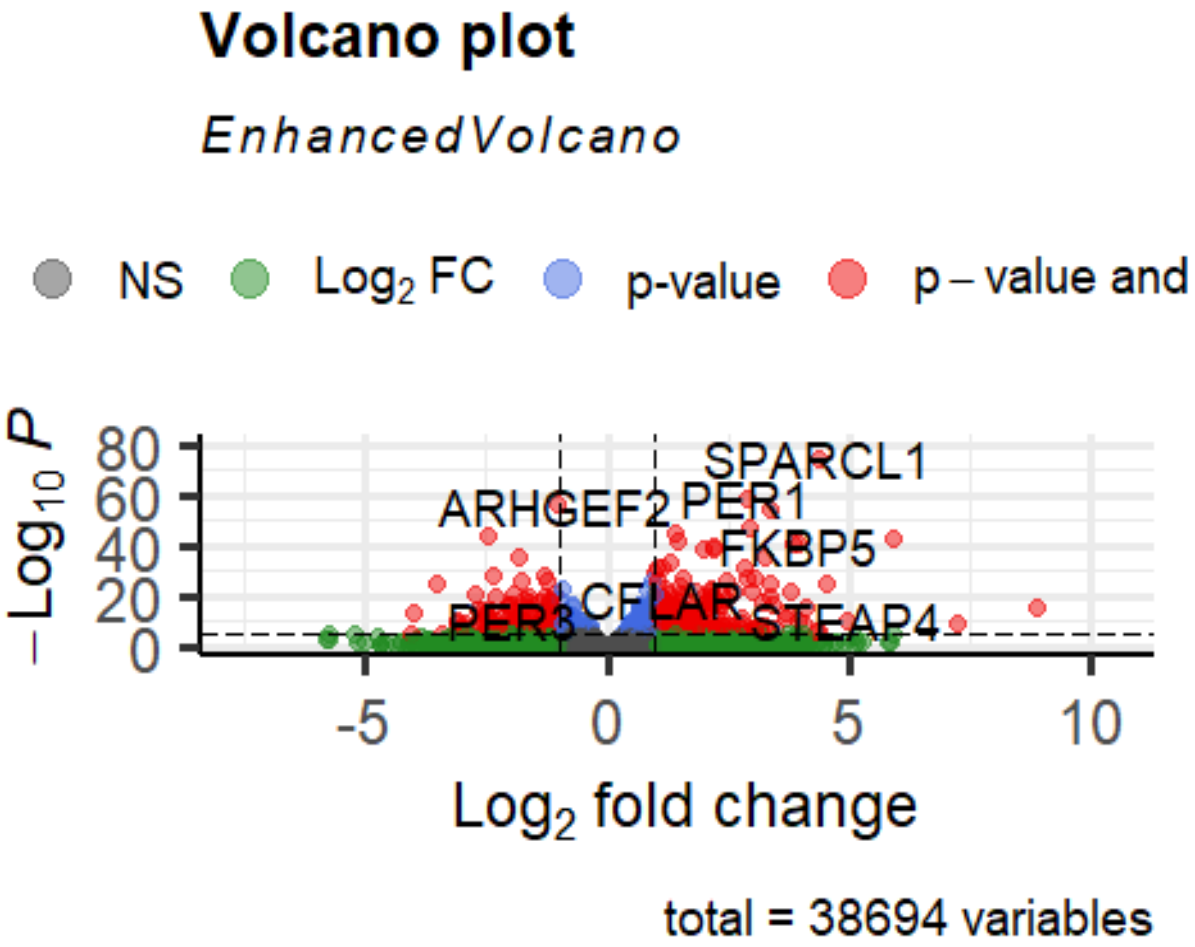
```
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 6 rows and 10 columns
##                   baseMean log2FoldChange     lfcSE      stat      pvalue
##                  <numeric>      <numeric> <numeric> <numeric>   <numeric>
## ENSG00000152583   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
##                                gene.name
##                               <character>
## ENSG00000152583           SPARC like 1
## ENSG00000179094 period circadian reg..
## ENSG00000116584 Rho/Rac guanine nucl..
## ENSG00000189221    monoamine oxidase A
## ENSG00000120129 dual specificity pho..
## ENSG00000148175               stomatin

# Save these results for later
write.csv(res[ord,], "deseq_results.csv")
```

Improved Volcano Plot

We can also now make an improved volcano plot using the gene names we have added to the results object `res`.

```
# Load package for better volcano plots (make sure this package has already
been installed)
library("EnhancedVolcano")

## 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

# Plot with this package
x <- as.data.frame(res)

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



Pathway analysis

We can now do pathway analysis, GO is one of the most commonly used first for this. KEGG is good for metabolic pathways. First we need to load some packages for data analysis (make sure these are already installed before loading).

```
# Load packages
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)

# Load data
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"
```

Gage requires a named vector of fold changes, and requires names to be of the Entrez gene IDs type.

```
# Make a fold change vector
foldchanges <- res$log2FoldChange

# Set names of each value to the entrez gene ID
names(foldchanges) <- res$entrez

# Check this has worked
head(foldchanges)

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

Now that the data is in an appropriate format we can run the analysis.

```
# Run KEGG analysis
keggres <- gage(foldchanges, gsets=kegg.sets.hs, same.dir=TRUE)

# Check results
attributes(keggres)

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

str(keggres)

## List of 3
##  $ greater: num [1:229, 1:6] 0.00331 0.01232 0.01711 0.02524 0.04343 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:229] "hsa00500 Starch and sucrose metabolism" "hsa00330
Arginine and proline metabolism" "hsa04910 Insulin signaling pathway"
"hsa04510 Focal adhesion" ...
##   .. ..$ : chr [1:6] "p.geomean" "stat.mean" "p.val" "q.val" ...
##  $ less   : num [1:229, 1:6] 0.000425 0.001782 0.002005 0.006043 0.007368
...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:229] "hsa05332 Graft-versus-host disease" "hsa04940 Type
I diabetes mellitus" "hsa05310 Asthma" "hsa04672 Intestinal immune network
for IgA production" ...
##   .. ..$ : chr [1:6] "p.geomean" "stat.mean" "p.val" "q.val" ...
##  $ stats  : num [1:229, 1:2] 2.77 2.28 2.13 1.96 1.73 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:229] "hsa00500 Starch and sucrose metabolism" "hsa00330
Arginine and proline metabolism" "hsa04910 Insulin signaling pathway"
"hsa04510 Focal adhesion" ...
##   .. ..$ : chr [1:2] "stat.mean" "exp1"
```

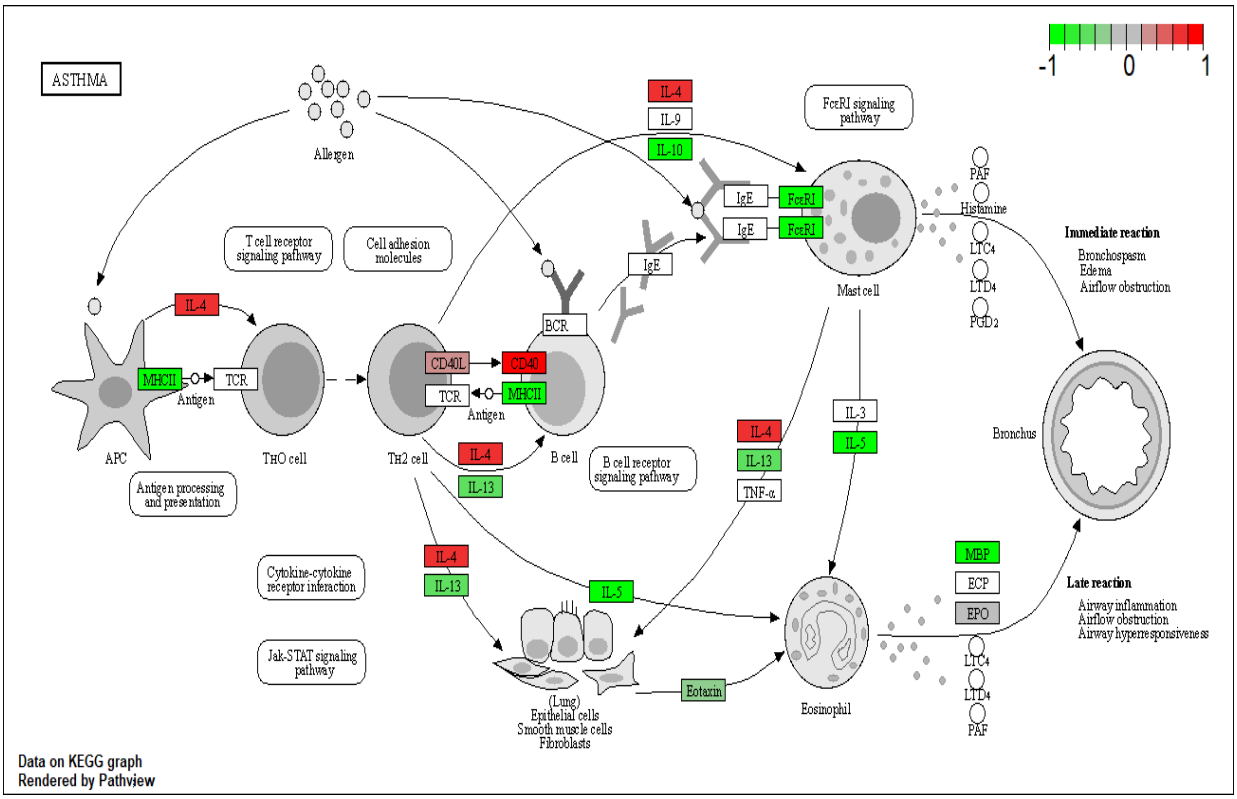
We can also look within the these attributes, for example, we can look at the pathways which have a lot of down-regulated and up-regulated genes.

```
# 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

# Look at the first three up (greater) pathways
head(keggres$greater, 3)
```

```
##                                             p.geomean stat.mean       p.val
## hsa00500 Starch and sucrose metabolism   0.003306262  2.772644 0.003306262
## hsa00330 Arginine and proline metabolism 0.012317455  2.280002 0.012317455
## hsa04910 Insulin signaling pathway       0.017110962  2.129511 0.017110962
##                                                q.val set.size        exp1
## hsa00500 Starch and sucrose metabolism   0.7042337       52 0.003306262
## hsa00330 Arginine and proline metabolism 0.7774866       54 0.012317455
## hsa04910 Insulin signaling pathway       0.7774866      138 0.017110962
```

The data from this analysis can be plotted using the `pathview()` function from the pathview package. To simplify the initial test of this package, I will initially view a single pathway, using `pathway.id` and choisng the asthma pathway which showed up in the top three down-regulated pathways above.

```
# Diagram of asthma pathway up and down regulation of genes
pathview(gene.data=foldchanges, pathway.id="hsa05310")

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

## Info: Working in directory C:/Users/mirte/Documents/UCSD/Foundations of
Bioinformatics/Week.08/Class.11

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

# 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/mirte/Documents/UCSD/Foundations of
Bioinformatics/Week.08/Class.11

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



The image created by the code above

Question 12

The same logic can be used for other pathways in our dataset. For example, we can look at the top two down-regulated pathways.

```
# Find the top 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
```

```
# Diagram of these pathways
pathview(gene.data=foldchanges, pathway.id="hsa05332")
```

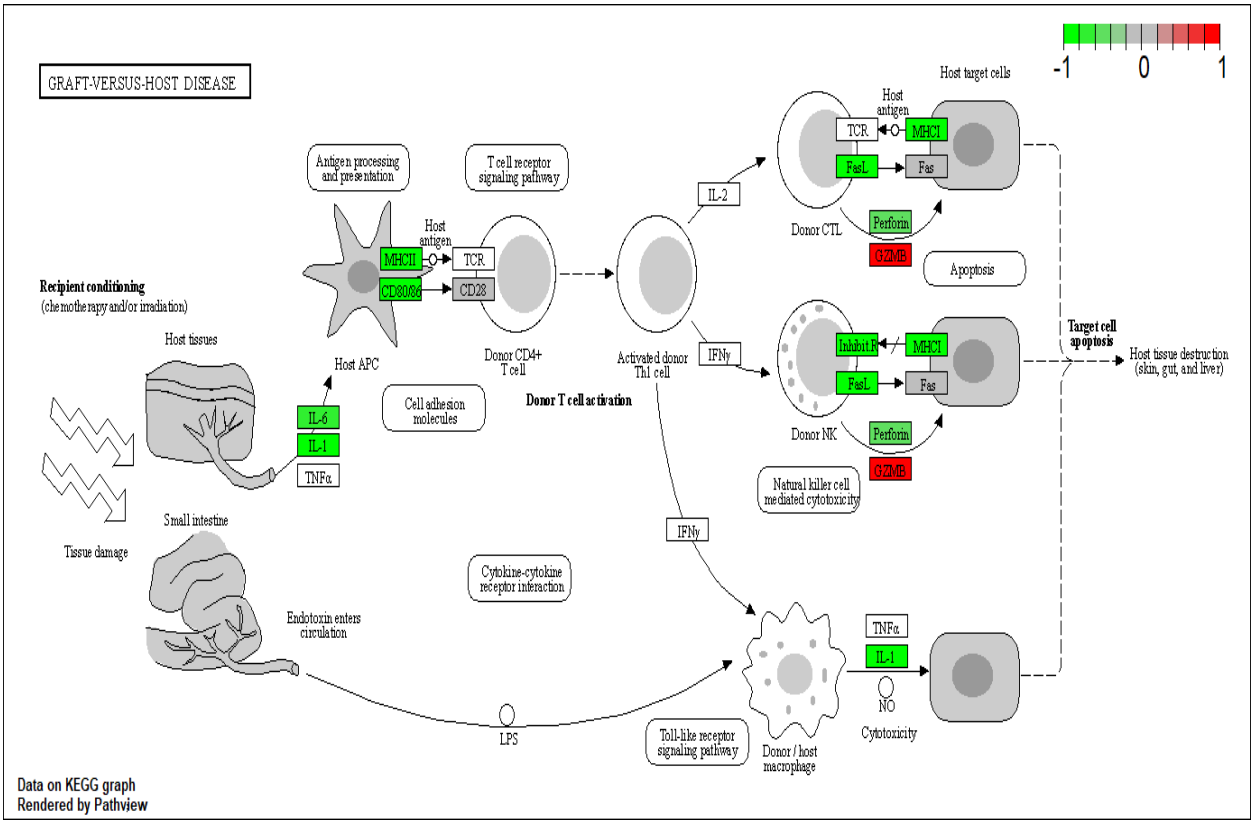
```
## 'select()' returned 1:1 mapping between keys and columns
```

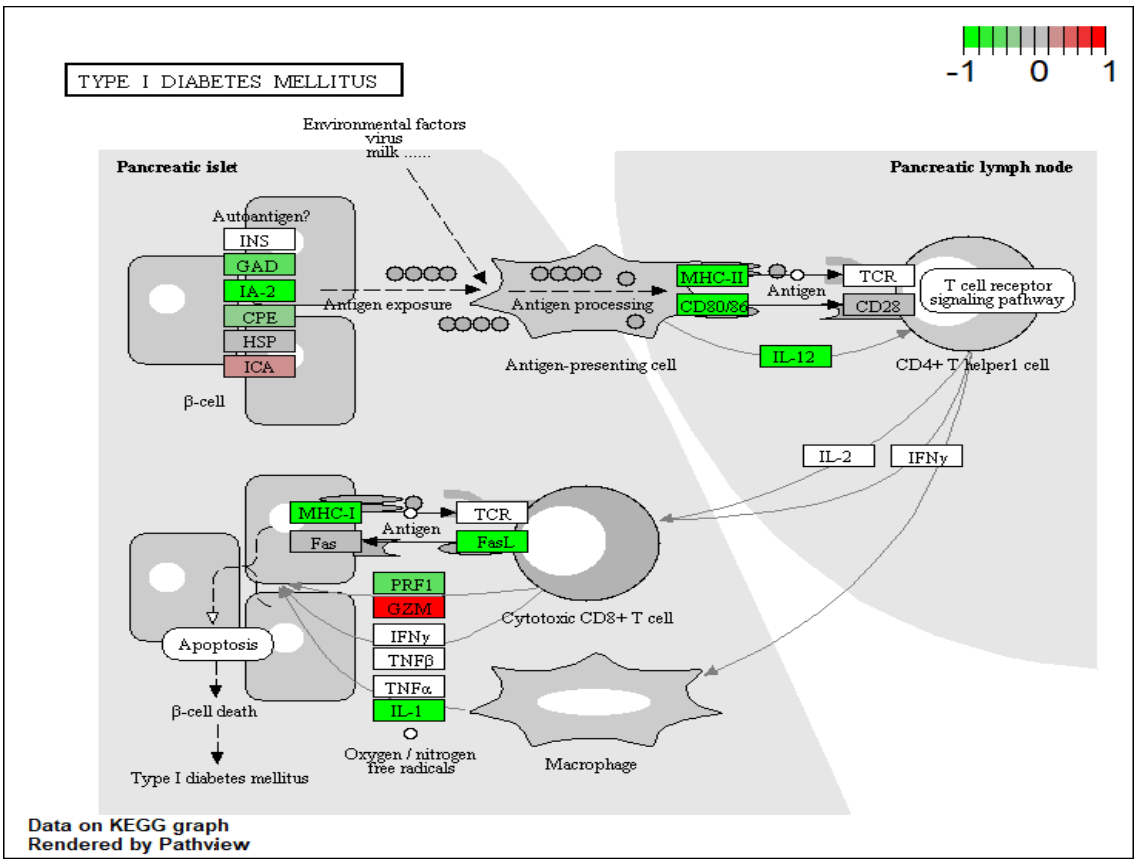
```
## Info: Working in directory C:/Users/mirte/Documents/UCSD/Foundations of
Bioinformatics/Week.08/Class.11
```

```
## 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 C:/Users/mirte/Documents/UCSD/Foundations of
Bioinformatics/Week.08/Class.11

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





Optional section on plotting genes of interest

First we can find a gene of interest within our results, for example, we could focus on CRISPLD2.

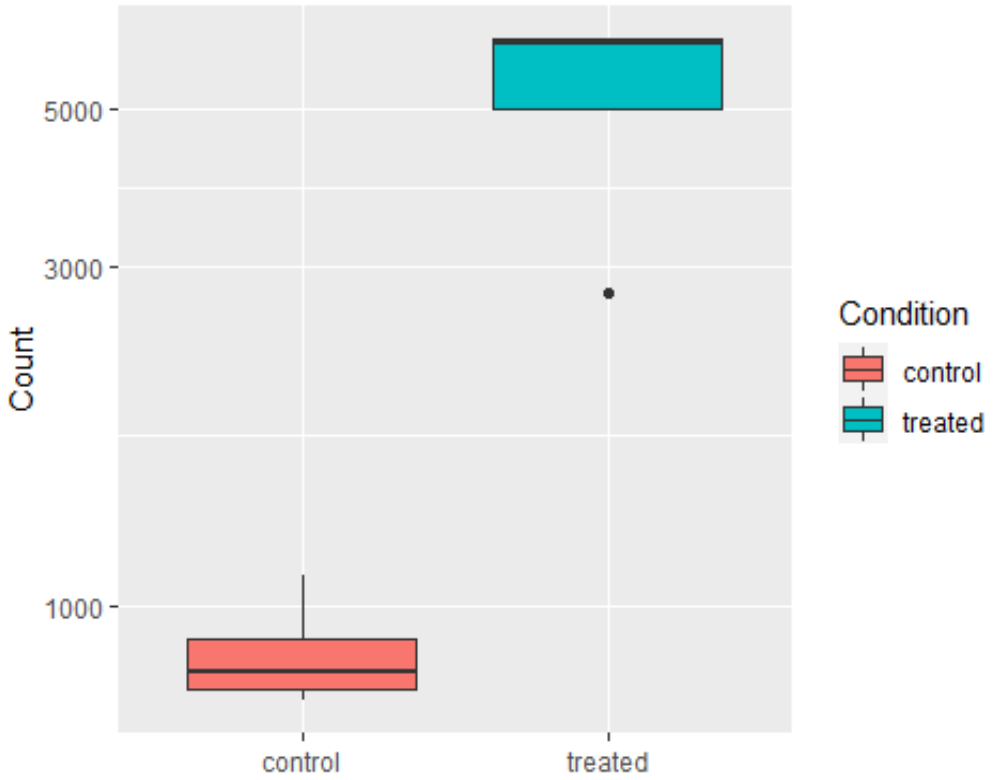
```
# Find index of CRISPLD2
i <- grep("CRISPLD2", res$symbol)

# Call this index
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
##                            gene.name
##                          <character>
## ENSG00000103196 cysteine rich secret..
```

```
# Find the rowname for CRISPLD2
rownames(res[i,])

## [1] "ENSG00000103196"

# Return the data
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
```

This data can then be plotted.

```
# Plot using ggplot
ggplot(d, aes(dex, count, fill=dex)) +
  geom_boxplot() +
  scale_y_log10() +
  labs(main = "CRISPLD2", fill = "Condition", x = "", y = "Count")
```



Session wrap up

Include session information at the end of the document for repeatability and transparency.

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 22000)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252
## [2] LC_CTYPE=English_United Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] gageData_2.32.0             gage_2.44.0
##  [3] pathview_1.34.0             EnhancedVolcano_1.12.0
##  [5] ggrepel_0.9.1               org.Hs.eg.db_3.14.0
##  [7] AnnotationDbi_1.56.2        RColorBrewer_1.1-2
##  [9] ggplot2_3.3.5               DESeq2_1.34.0
## [11] SummarizedExperiment_1.24.0 Biobase_2.54.0
## [13] MatrixGenerics_1.6.0        matrixStats_0.61.0
## [15] GenomicRanges_1.46.1        GenomeInfoDb_1.30.1
## [17] IRanges_2.28.0              S4Vectors_0.32.3
## [19] BiocGenerics_0.40.0         BiocManager_1.30.16
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-7           bit64_4.0.5            ash_1.0-15
##  [4] httr_1.4.2             Rgraphviz_2.38.0       tools_4.1.2
##  [7] utf8_1.2.2             R6_2.5.1               vipor_0.4.5
## [10] KernSmooth_2.23-20     DBI_1.1.2              colorspace_2.0-2
## [13] withr_2.4.3            ggrastr_1.0.1          tidyselect_1.1.2
## [16] ggalt_0.4.0            bit_4.0.4              compiler_4.1.2
## [19] extrafontdb_1.0        graph_1.72.0           cli_3.2.0
## [22] DelayedArray_0.20.0    labeling_0.4.2         KEGGgraph_1.54.0
## [25] scales_1.1.1           proj4_1.0-11           genefilter_1.76.0
## [28] stringr_1.4.0          digest_0.6.29          rmarkdown_2.11
## [31] XVector_0.34.0         pkgconfig_2.0.3        htmltools_0.5.2
## [34] extrafont_0.17         fastmap_1.1.0          highr_0.9
## [37] maps_3.4.0             rlang_1.0.1            rstudioapi_0.13
## [40] RSQLite_2.2.10         generics_0.1.2         farver_2.1.0
```

```
## [43] BiocParallel_1.28.3    dplyr_1.0.8            RCurl_1.98-1.6
## [46] magrittr_2.0.2         GO.db_3.14.0           GenomeInfoDbData_1.2.7
## [49] Matrix_1.3-4           ggbeeswarm_0.6.0       Rcpp_1.0.8
## [52] munsell_0.5.0          fansi_1.0.2            lifecycle_1.0.1
## [55] stringi_1.7.6          yaml_2.2.2             MASS_7.3-54
## [58] zlibbioc_1.40.0        grid_4.1.2             blob_1.2.2
## [61] parallel_4.1.2         crayon_1.5.0           lattice_0.20-45
## [64] Biostrings_2.62.0      splines_4.1.2          annotate_1.72.0
## [67] KEGGREST_1.34.0        locfit_1.5-9.4         knitr_1.37
## [70] pillar_1.7.0           geneplotter_1.72.0     XML_3.99-0.8
## [73] glue_1.6.1             evaluate_0.15          png_0.1-7
## [76] vctrs_0.3.8            Rttf2pt1_1.3.10        gtable_0.3.0
## [79] purrr_0.3.4            cachem_1.0.6           xfun_0.29
## [82] xtable_1.8-4           survival_3.2-13        tibble_3.1.6
## [85] beeswarm_0.4.0         memoise_2.0.1          ellipsis_0.3.2
```