class15.Rmd

Anu Chaparala

11/16/2021

##Background Our data for today came from Himes et. al. RNASeq analysis of the drug dexamethasone, a synthetic glucocorticoid steroid with anti-inflammatory effects (Himes et al. 2014).

Import and Read countData and colData

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

Let's look at our data!

```
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
```

```
head(metadata)
```

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

Q1. How many genes are in this dataset?

38694

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

4

```
dim(counts)
```

```
## [1] 38694     8
```

```
dim(metadata)
```

```
## [1] 8 4
```

```
#more robust method
sum(metadata$dex == "control")
```

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

First I need to extract all the "control" columns. Then I will take the rowwise mean to get the average count calues for all genes in these four experiements.

```
control.inds <- metadata$dex == "control"
control.counts1 <- counts[ , control.inds]
head(control.counts1)
```

```
##                 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
```

Store control means.

```
control.mean1 <- rowMeans(control.counts1)
```

Now let's do the same thing for the treated samples.

```
treated.inds <- metadata$dex == "treated"
treated.counts1 <- counts[ , treated.inds]
head(treated.counts1)
```

```
##                 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
```

Store treated means.

```
treated.mean1 <- rowMeans(treated.counts1)
```

Put the two mean data sets into the same dataframe for further analysis/dipslay.

```
meancounts1 <- data.frame(control.mean1, treated.mean1)
colSums(meancounts1)
```

```
## control.mean1 treated.mean1
##      23005324      22196524
```

```
#correct!
```

This bit of code will first find the sample id for those labeled control. Then, it will calculate the mean counts per gene across these samples. (This is code based on the Handout, in order to answer the followng questions. The above chunks are what was done in lab, and I kept both for personal understanding. Thank you.)

```
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 using the dyplr package from tidyverse:

```
#library(dplyr)
#control <- metadata %>% filter(dex=="control")
#control.counts <- counts %>% select(control$id)
#control.mean <- rowSums(control.counts)/4
#head(control.mean)
```

Q3. How would you make the above code in either approach more robust? (ie what would happen if we have more than 4 rows?)

```
#I can sub the hard coded denominator of "4" for nrow(control), so that this code chunk will now be abl

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

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

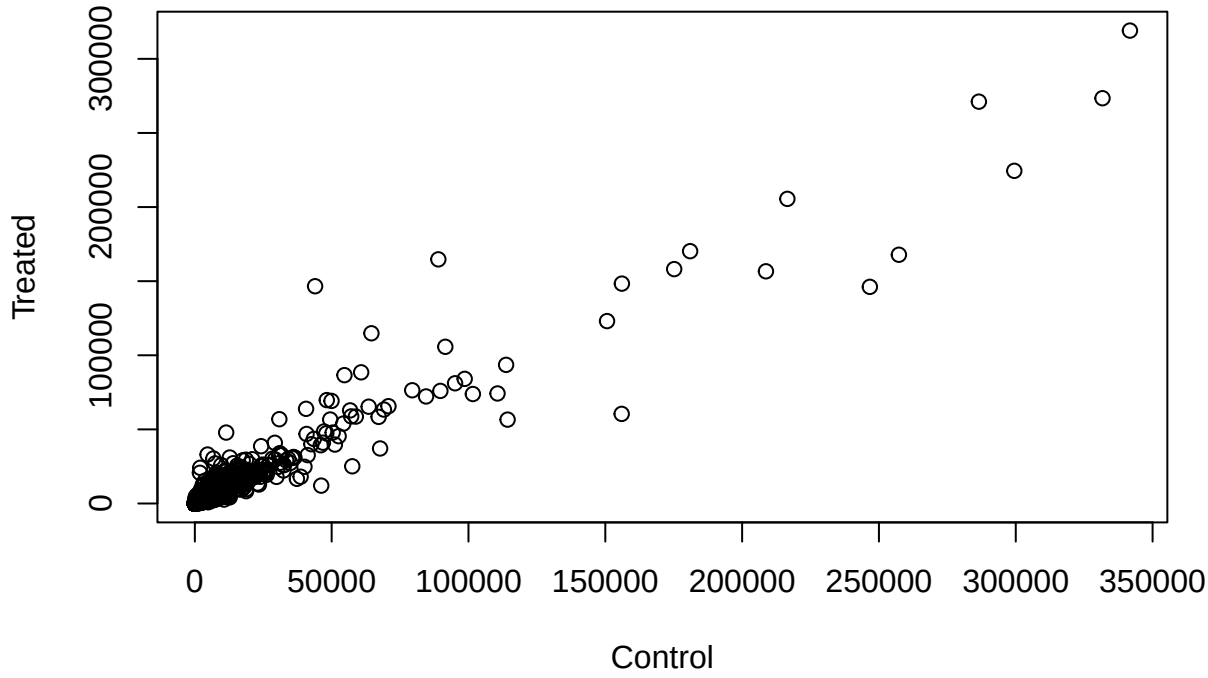
Combine the meancount data for possible use later on.

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

```
## control.mean treated.mean
##     23005324     22196524
```

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

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




```
#points along the diagonal would suggest that the drug compared to control had very little difference i
```

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

geom_point()

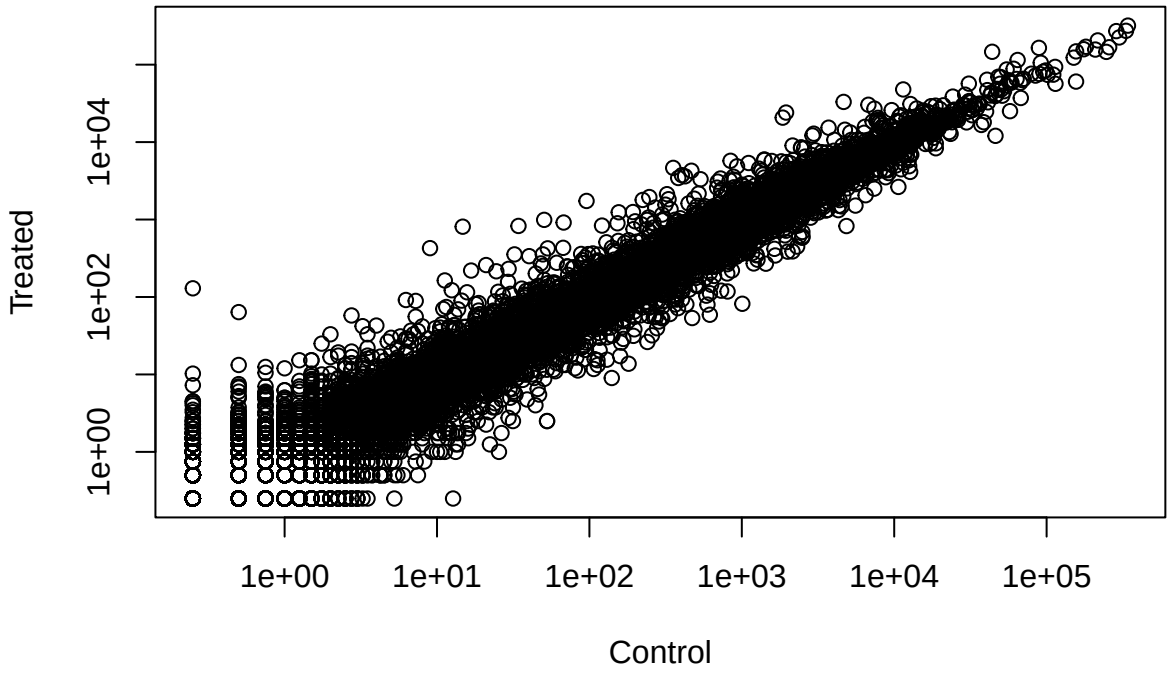
Q6. Try plotting both axes on a log scale. What is the argument to plot() that allows you to do this?

log = "xy"

```
plot(meancounts, log="xy", xlab="Control", ylab="Treated")
```

```
## 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 thsi field because it has nice math properties that make interpretation easier. For log2, we see 0 values for no change, + calues for increases, and - values for decreases. This nivce property lead us to work with **log2(fold-change)** all the time in the genomics and proteomics field.

Let's add the **log2(fold-change)** to the 'meancounts' data frame.

```
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
```

There are a couple of "weird" results. Namely, the NaN ("not a number") and -Inf (negative infinity) results. Let's get rid of these zero count results. I can use the **which()** function with the 'arr.ind=TRUE' argument to get the columns and rows where the TRUE values are. (ie the zero counts in our case).

```
zero.vals <- which(meancounts[,1:2]==0, arr.ind=TRUE)

to.rm <- unique(zero.vals[,"row"])
#sort(to.rm)

#Now remove these from our meancounts dataframe.
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
```

```
nrow(mycounts)
```

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

```
#there are now 21817 genes left after we omitted the zero count results.
```

> 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 in the which() function, is a logical that when TRUE will lead the which() function to return the row and column (position) of the data point for which it is TRUE. For our purposes, this return/true value will tell us when the genes/rows and samples/columns have zero counts, so that we can ignore the zero counts in any sample/column and just use the genes/row answers. Calling the unique() function will make sure that we do not count any row twice if it has zero entries in both samples.

Now, 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)

table(up.ind)
```

```
## up.ind
## FALSE  TRUE
## 21567   250
```

```
table(down.ind)
```

```
## down.ind
## FALSE  TRUE
## 21450   367
```

> Q8. Using the up.ind vector above can you determine how many up regulated genes we have at the greater than 2 fc level?

250 upregulated genes are greater than 2 fc level.

> Q9. Using the down.ind vector above can you determine how many down regulated genes we have at the greater than 2 fc level?

367 downregulated genes are greater than 2 fc level.

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

Not quite yet, since all we have done so far is configured the data's fold changes; however, we have yet to understand if these fold changes are statistically significant differences between the experimental and control groups.

##DESeq2 Analysis

Call the previously installed DESeq2 BiocManager package to our Studio.

```
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

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: SummarizedExperiment

## Loading required package: MatrixGenerics

## Loading required package: matrixStats

##
## Attaching package: 'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

##
## Attaching package: 'Biobase'
```

```
## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians

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

```
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 Here, we will run the DESeq pipeline on the dds object, and reassign the whole thing back to dds, which will now be a DESeqDataSet populated with all those values. The **DESeq()** function calls a number of other functions within the package to essentially run the entire pipeline (normalizing by library size by estimating the "size factors," estimating dispersion for the negative binomial model, and fitting models and getting statistics for each gene for the design specified when you imported the data).

```
dds <- DESeq(dds)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

Now, we can simply get the results using the **results()** function. Note, our experiment for today is relatively simple: 2 gropus, single factored, treated vs. untreated.

```
res <- as.data.frame(results(dds))
head(res)
```

```
##                    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
```

Summarize some basic tallies using the summary function.

```
summary(res)
```

```
##     baseMean        log2FoldChange       lfcSE            stat
##  Min.   :     0.0   Min.   :-6.030   Min.   :0.057   Min.   :-15.894
##  1st Qu.:     0.0   1st Qu.:-0.425   1st Qu.:0.174   1st Qu.: -0.643
##  Median :     1.1   Median :-0.009   Median :0.445   Median : -0.027
##  Mean   :   570.2   Mean   :-0.011   Mean   :1.136   Mean   :  0.045
##  3rd Qu.:   201.8   3rd Qu.: 0.306   3rd Qu.:1.848   3rd Qu.:  0.593
##  Max.   :329280.4   Max.   : 8.906   Max.   :3.534   Max.   : 18.422
##                     NA's   :13436    NA's   :13436   NA's   :13436
##      pvalue           padj
##  Min.   :0.000   Min.   :0.000
##  1st Qu.:0.168   1st Qu.:0.203
##  Median :0.533   Median :0.606
##  Mean   :0.495   Mean   :0.539
##  3rd Qu.:0.800   3rd Qu.:0.866
##  Max.   :1.000   Max.   :1.000
##  NA's   :13578   NA's   :23549
```

By default the argument alpha is set to 0.1. If the adjusted p value cutoff will be a value other than 0.1, we can set the alpha value. Let's create a variable that elminates all data with p-value over threshold 0.05.

```
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
```

##Adding annotation data

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.

For this we need 2 Bioconductor packages: AnnotationDbi and org.Hs.eg.db

```
library("AnnotationDbi")
```

```
## Warning: package 'AnnotationDbi' was built under R version 4.1.2
```

```
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] "GENETYPE"     "GO"           "GOALL"        "IPI"          "MAP"
## [16] "OMIM"         "ONTOLOGY"     "ONTOLOGYALL"  "PATH"         "PFAM"
## [21] "PMID"         "PROSITE"      "REFSEQ"       "SYMBOL"       "UCSCKG"
## [26] "UNIPROT"
```

We will use the 'mapIDs' function to translate between identifiers from different databases.

> 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$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
```

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

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

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

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

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

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

```
head(res)
```

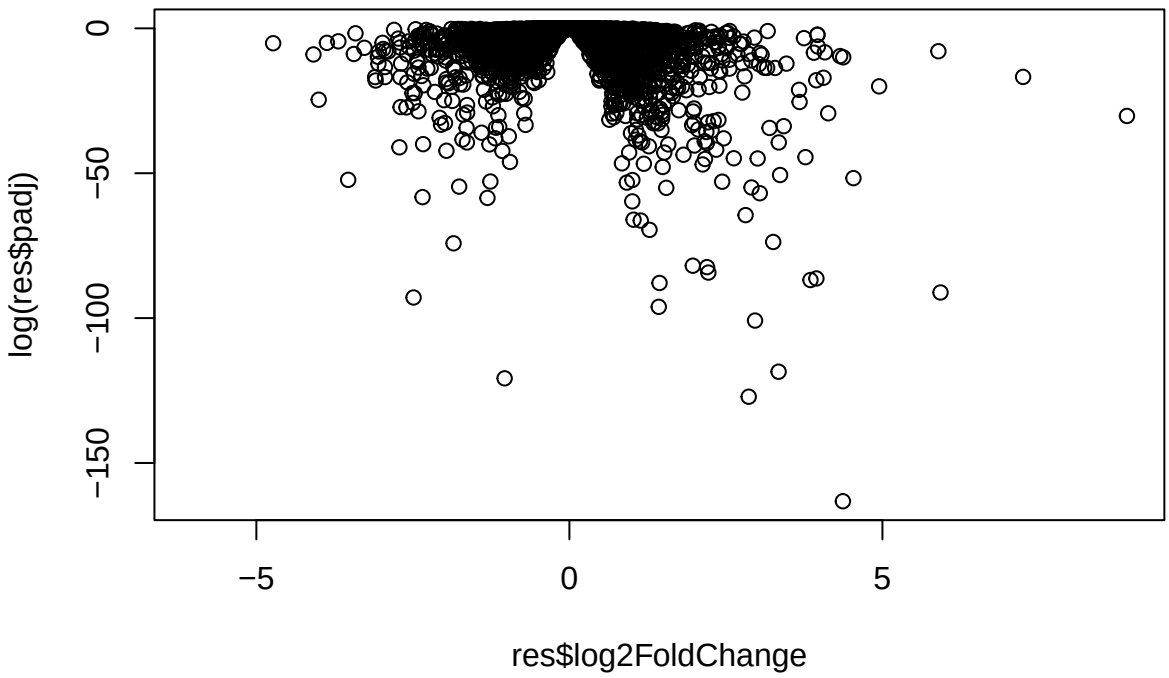
```
##                    baseMean log2FoldChange     lfcSE       stat     pvalue
## ENSG00000000003 747.1941954    -0.35070302 0.1682457 -2.0844697 0.03711747
## ENSG00000000005   0.0000000             NA        NA         NA         NA
## ENSG00000000419 520.1341601     0.20610777 0.1010592  2.0394752 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   symbol entrez    uniprot
## ENSG00000000003 0.1630348   TSPAN6   7105 A0A024RCI0
## ENSG00000000005        NA     TNMD  64102     Q9H2S6
## ENSG00000000419 0.1760317     DPM1   8813     O60762
## ENSG00000000457 0.9616942    SCYL3  57147     Q8IZE3
## ENSG00000000460 0.8158486 C1orf112  55732 A0A024R922
## ENSG00000000938        NA      FGR   2268     P09769
##                                                                genename
## ENSG00000000003                                           tetraspanin 6
## ENSG00000000005                                             tenomodulin
## ENSG00000000419 dolichyl-phosphate mannosyltransferase subunit 1, catalytic
## ENSG00000000457                                  SCY1 like pseudokinase 3
## ENSG00000000460                       chromosome 1 open reading frame 112
## ENSG00000000938           FGR proto-oncogene, Src family tyrosine kinase
```

##Data Visualization

Let's make a commmonly 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.

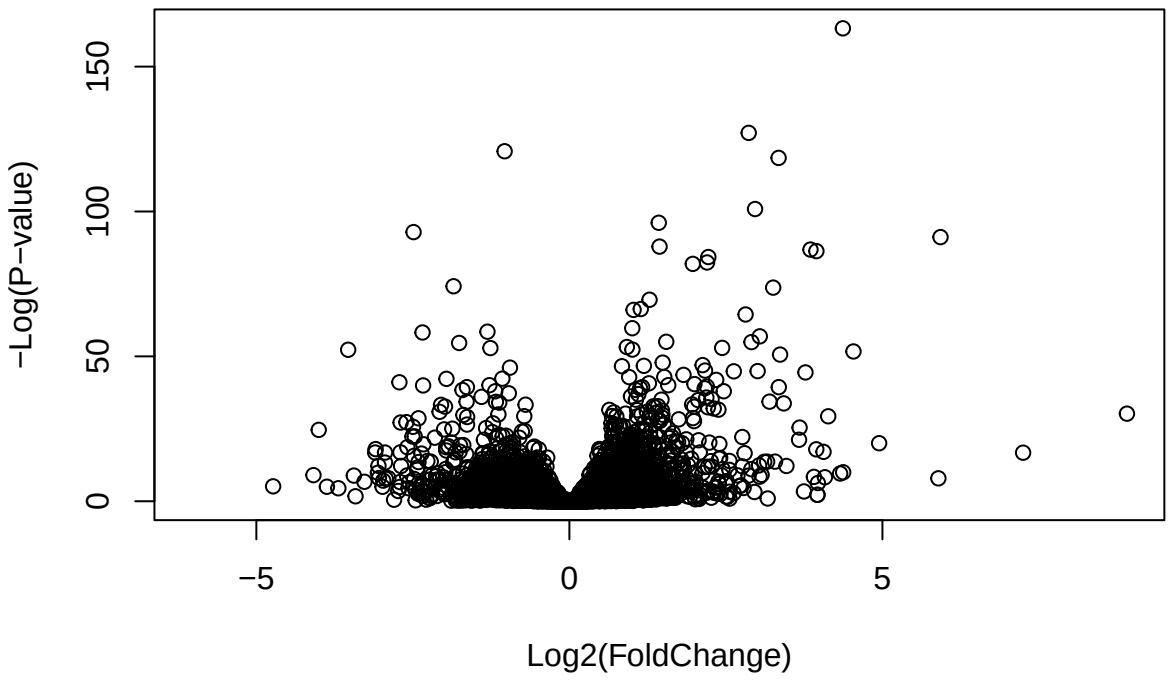
Typically these plots shows the log fold change on the X-axis, and the $-\log10$ of the p-value on the Y-axis (the more significant the p-value, the larger the $-\log10$ of that value will be). A very dull (i.e. non colored and labeled) version can be created with a quick call to plot() like so:

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



This is not a useful plot becuase all the small p-values are hidden at the bottom of the plot and we can't really see them. -Log will help.

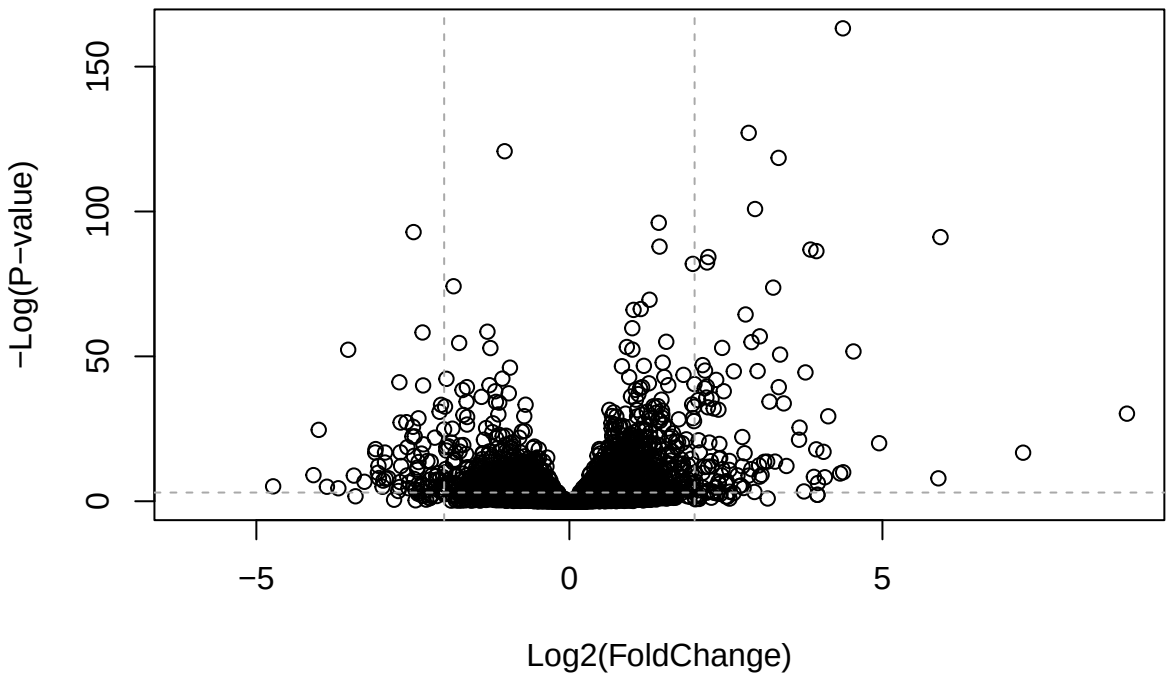
```
plot( res$log2FoldChange,  -log(res$padj),
      xlab="Log2(FoldChange)",
      ylab="-Log(P-value)")
```



To make this more useful we can add some guidelines (with the abline() function) and color (with a custom color vector) highlighting genes that have padj<0.05 and the absolute 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)
```



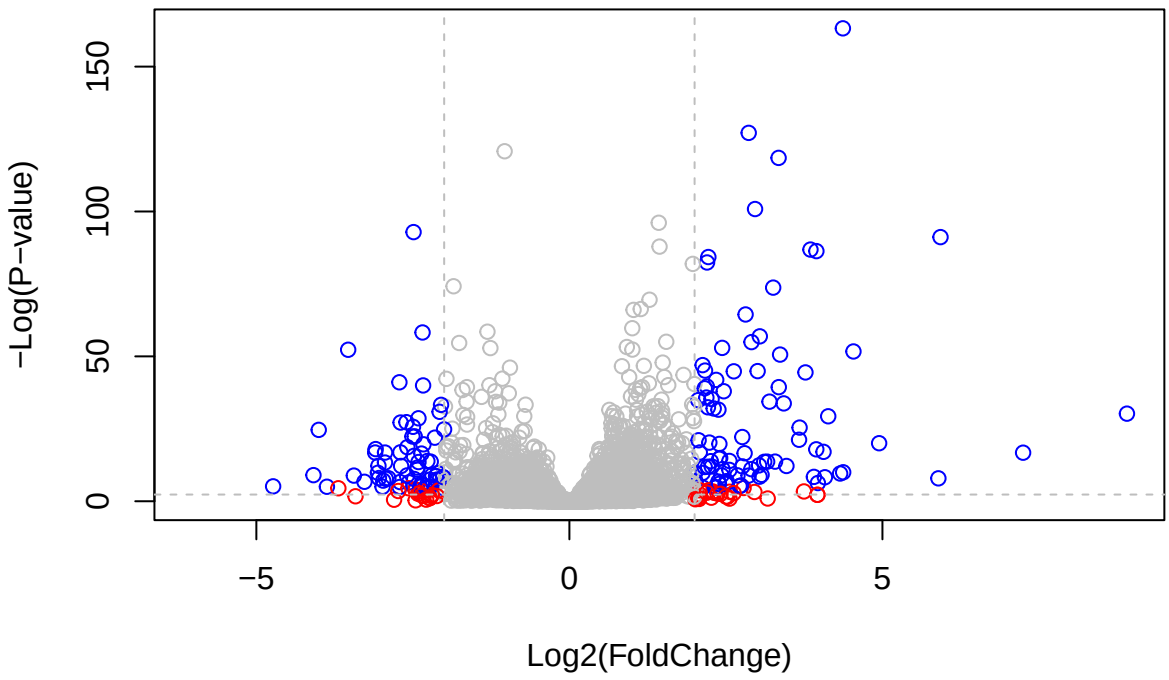
Finally, let's add some color to the points by setting up a custom color vector indicating transcripts with large fold change and significant differences between conditions (ie. the genes/points we care about - that is those with large fold-change and low p-values (ie high -log(p-values))).

```
# Setup my 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)
```



##Save Our Results

Write out whole data set, including genes that did not change significantly.

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

Let's make an Enhanced Volcano Plot! (adding gene labels and more differentiating color)

```
#Load EnhancedVolcano package from BiocManager
library(EnhancedVolcano)
```

```
## Loading required package: ggplot2
```

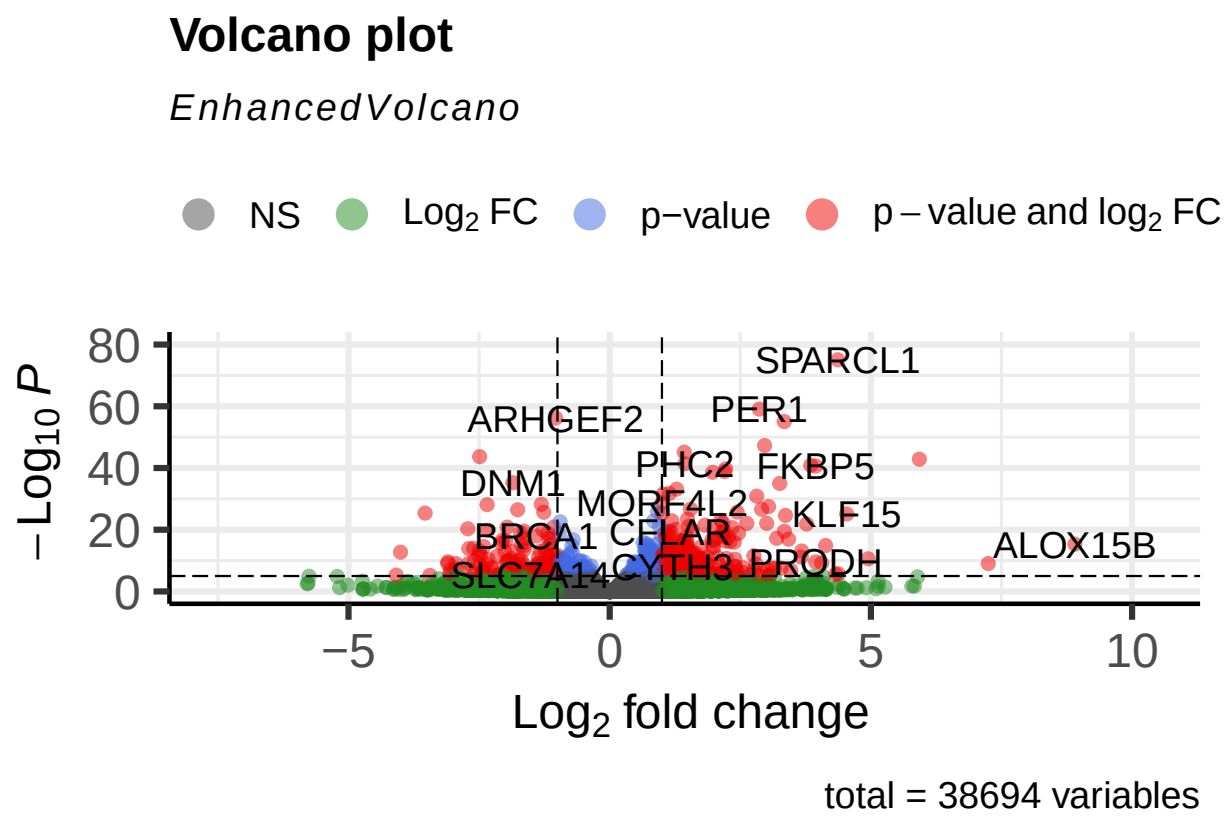
```
## Loading required package: ggrepel
```

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

Make Plot

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

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



##Pathway Analysis/Gene Set Annotation

```
#load the pakages we need from GAGE
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 the KEGG data-sets I need
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"
```

```
#This is the data storage from when we used the mapIDs() function above to obtain Entrez gene IDs (stor

#res$entrez
foldchanges = res$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. The main **gage()** function requires a named vector of fold changes, where the names of the values are the Entrez gene IDs.

```
# Get our results
keggres = gage(foldchanges, gsets=kegg.sets.hs, same.dir=TRUE)
```

```
#Now, let's look at the object returned by the gage() function
#This separates out results by "greater" and "less" i.e. those that are up regulated and those that are
attributes(keggres)
```

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

```
#Let's 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. First, we will try mannually entering the pathway.id (the third one from above as "hsa05310 Asthma").

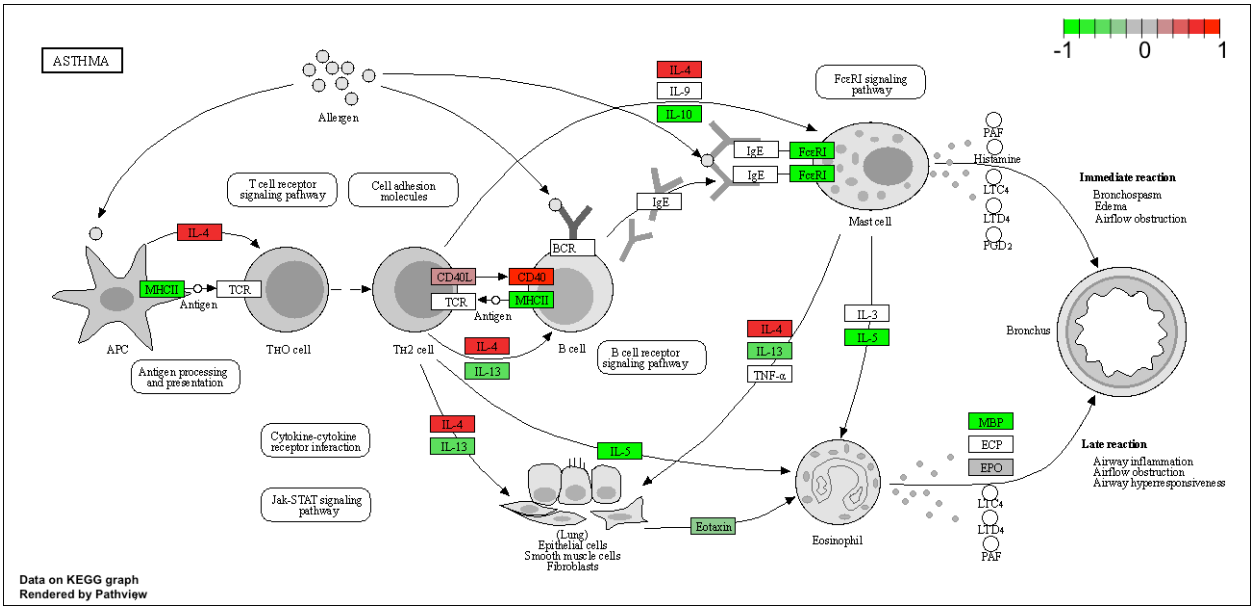
```
pathview(gene.data=foldchanges, pathway.id="hsa05310")
```

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

```
## Info: Working in directory /Users/Anu/Desktop/BIMM 143A/Bimm143_github/class 15
```

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

Insert pathway generated.



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

```
#top down-regulated pathway
pathview(gene.data=foldchanges, pathway.id="hsa05332")
```

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

```
## Info: Working in directory /Users/Anu/Desktop/BIMM 143A/Bimm143_github/class 15
```

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

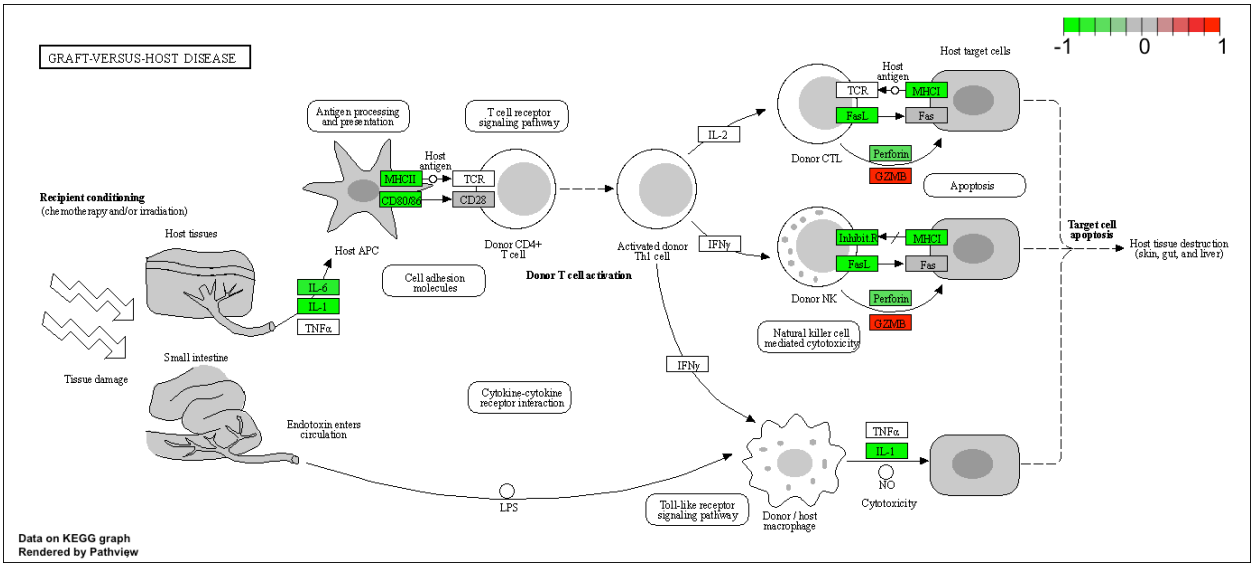
```
#second highest down-regulated pathway
pathview(gene.data=foldchanges, pathway.id="hsa04940")
```

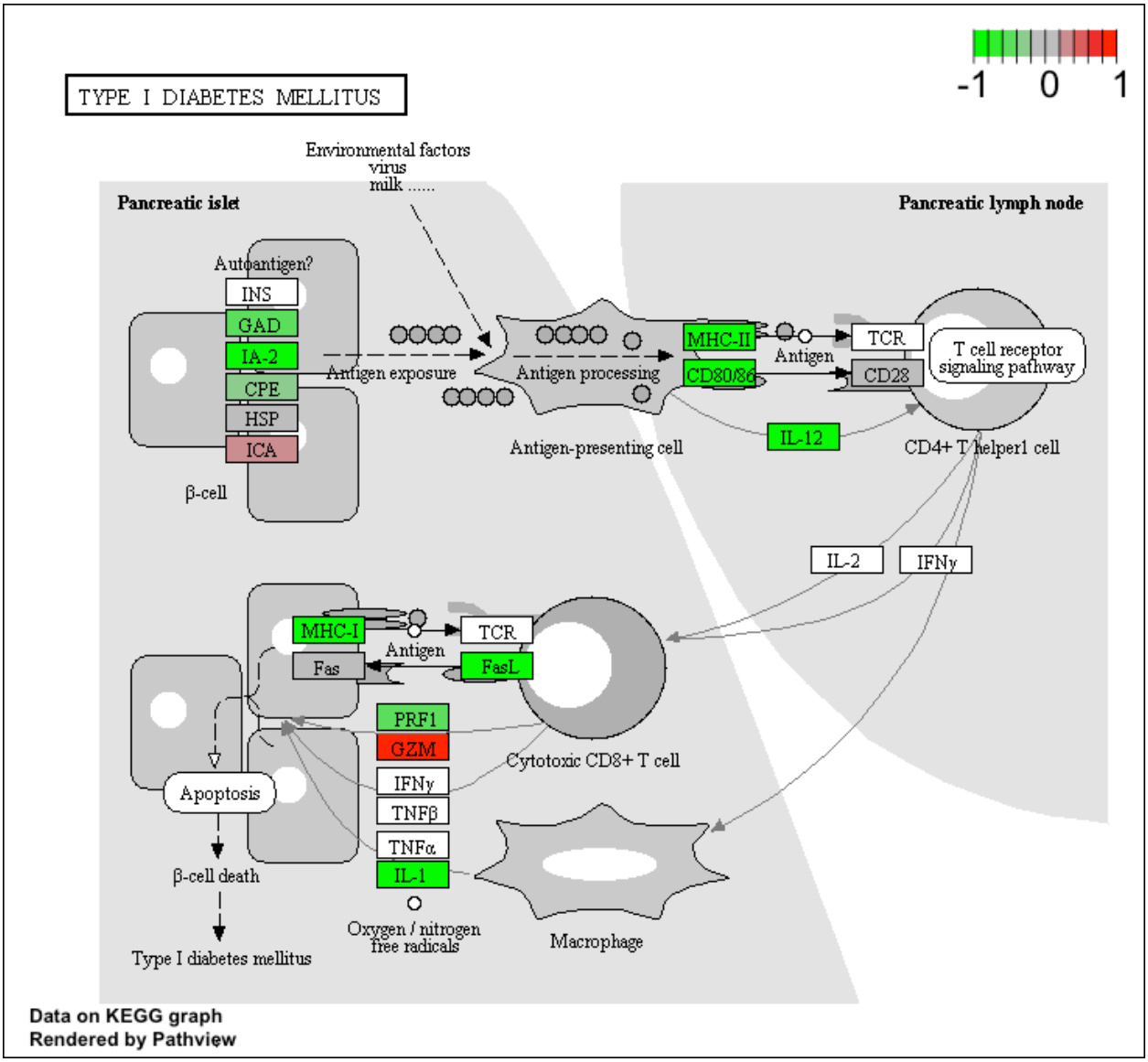
```
## 'select()' returned 1:1 mapping between keys and columns

## Info: Working in directory /Users/Anu/Desktop/BIMM 143A/Bimm143_github/class 15

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

Insert pathways generated.




-1 0 1
TYPE I DIABETES MELLITUS
Environmental factors
virus
milk .....
Pancreatic islet
Pancreatic lymph node
Autoantigen?
INS
GAD
IA-2
CPE
HSP
ICA
β-cell
Antigen exposure
Antigen processing
MHC-II
CD80/86
Antigen
TCR
CD28
T cell receptor signaling pathway
IL-12
Antigen-presenting cell
CD4+ T helper1 cell
IL-2
IFNγ
MHC-I
Fas
Antigen
TCR
FasL
PRF1
GZM
IFNγ
TNFβ
TNFα
IL-1
Cytotoxic CD8+ T cell
Apoptosis
β-cell death
Type I diabetes mellitus
Oxygen / nitrogen
free radicals
Macrophage
Data on KEGG graph
Rendered by Pathview