lab13_report

Longmei Zhang A17012012

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

Q1. How many genes are in this dataset?

There are 38694 genes in this dataset

```
nrow(counts)
```

```
[1] 38694
```

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

We have 4 "control" cell lines

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

```
[1] 4
```

##Toy Differential expression Analysis

calculate the mean per gene count values for control samples and treated samples and compare.

1. find all control counts in csv `counts`

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

2. find the mean counts for each gene

```
control.mean <- rowSums(control.count)/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? Is there a function that could help here?

I can also use the apply function to calculate the mean

```
control.mean <- apply(control.count, 1, mean)
```

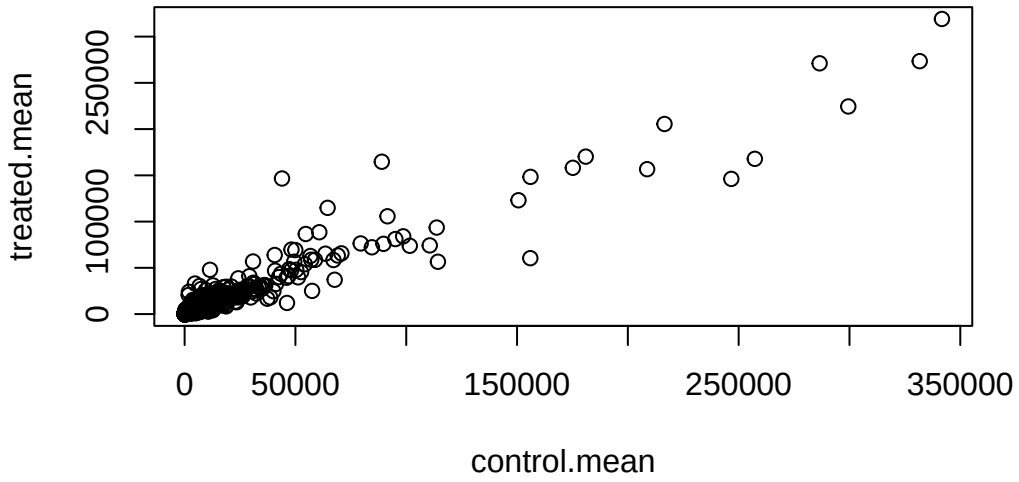
Q4. Follow the same procedure for the treated samples (i.e. calculate the mean per gene across drug treated samples and assign to a labeled vector called treated.mean)

3. do the same steps for the treated value

```
treated.inds <- metadata$dex == "treated"
treated.count <- counts[, treated.inds]
treated.mean <- apply(treated.count, 1, mean)
```

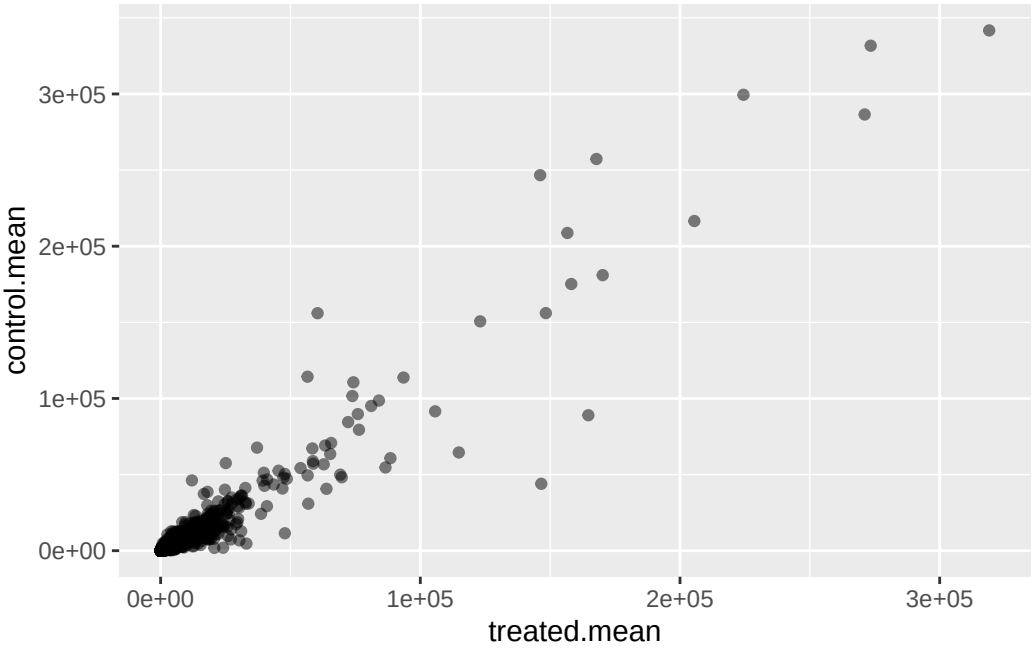
Q5 (a). Create a scatter plot showing the mean of the treated samples against the mean of the control samples. Your plot should look something like the following.

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



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

```
library(ggplot2)
ggplot(meancounts, aes(treated.mean, control.mean)) +
  geom_point(alpha = 0.5)
```

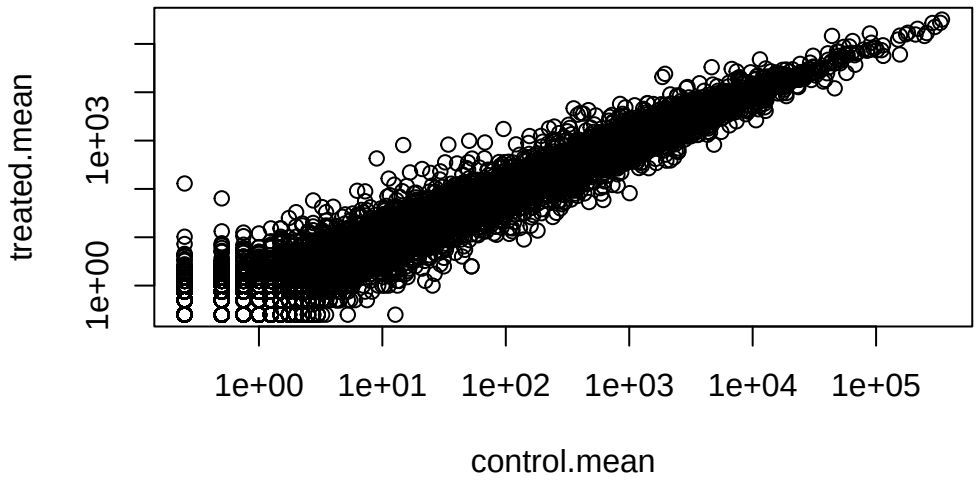


Q6. Try plotting both axes on a log scale. What is the argument to plot() that allows you to do this?

```
plot(meancounts, log = "xy")
```

```
Warning in xy.coords(x, y, xlabel, ylabel, log): 15032 x values <= 0 omitted
from logarithmic plot
```

```
Warning in xy.coords(x, y, xlabel, ylabel, log): 15281 y values <= 0 omitted
from logarithmic plot
```



most frequently use log2 transformation for this type of data. Log2 is more useful for showing the fold change (how many times the expression of one condition is higher than the other). A rule-of-tjumb is Log2 fold changes of >2 or <-2 are worth noting. Negative fc means down regulated, vice versa.

Calculate the log2(fold change) and add it to the `meancounts` dataframe.

```
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
```

```
# remove genes with no read counts
#to.rm <- rowSums(meancounts[, 1:2]==0) > 0
#mycounts <- meancounts[!to.rm, ]

#or we can use the Which() function and condition
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?

Setting arr.ind parameter to true will make the `which()` function return the positions, including the row coordinate and the column coordinate, of the values that fits the condition. In this case, the rows we want to remove are included in the vector zero.vals[,1], in which the first column stores the rows or genes that have 0 expression.

> Q how many genes do I have left after filtering out the zero reads ones

21817

```
nrow(mycounts)
```

```
[1] 21817
```

> Q8 how many genes are up-regulated by the treatment (threshold of 2 for log2fc)

1. need to extract the log2fc values
2. need to find values above threshold

250 genes are up-regulated

```
up <- mycounts$log2fc > 2
sum(up)
```

```
[1] 250
```

> Q9 how many genes are down-regulated (threshold -2 for log2fc)

367 genes are down regulated

```
down <- mycounts$log2fc < -2
sum(down)
```

```
[1] 367
```

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

We are missing statistics. Cannot determine if the difference is significant. We will do this analysis with DESeq2, which identify the differentially expressed genes with both biological data and statistics.

DESeq2

```
library(DESeq2)
```

```
Warning: package 'DESeq2' was built under R version 4.3.3
```

```
Warning: package 'GenomeInfoDb' was built under R version 4.3.3
```

```
Warning: package 'matrixStats' was built under R version 4.3.3
```

The first function that we will use will setup data in the format DESeq2 wants it.

```
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
```

The main function in the package is called DESeq, we can run it on our `dds` object

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

```
estimating size factors
```

```
estimating dispersions
```

```
gene-wise dispersion estimates

mean-dispersion relationship

final dispersion estimates

fitting model and testing
```

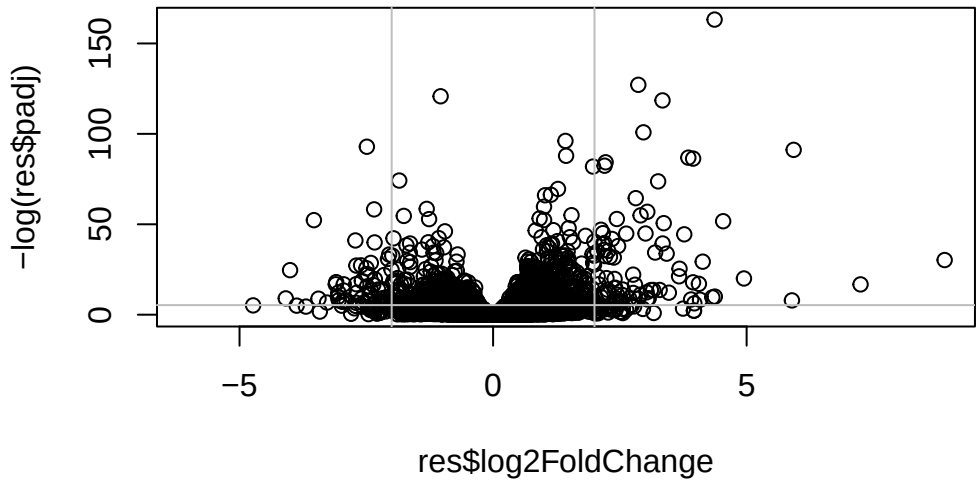
I will get the results from dds with the `results()` function

```
res <- results(dds)
head(res)
```

```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 6 columns
                  baseMean log2FoldChange     lfcSE      stat    pvalue
                 <numeric>      <numeric> <numeric> <numeric> <numeric>
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
                <numeric>
ENSG00000000003  0.163035
ENSG00000000005        NA
ENSG00000000419  0.176032
ENSG00000000457  0.961694
ENSG00000000460  0.815849
ENSG00000000938        NA
```

make a common overall results figure from this analysis. This is designed to have the biological data and statistical data. Plot fold change vs. p-value. Focus on points that change a lot and are differentially expressed. Adjustment of p-value for doing multiple tests to prevent getting significant data by change.

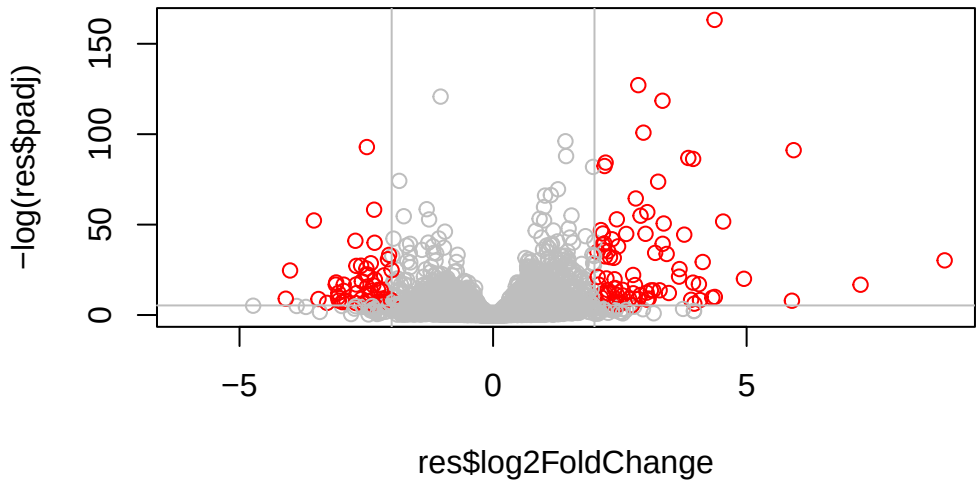
```
plot(res$log2FoldChange, -log(res$padj))
abline(v = c(-2,2), col = "grey")
abline(h = -log(0.005), col = "grey")
```



Add some color to the important plots

```
mycols <- rep("grey", nrow(res))
mycols[res$log2FoldChange > 2] <- "red"
mycols[res$log2FoldChange < -2] <- "red"
mycols[res$padj > 0.005] <- "grey"

plot(res$log2FoldChange, -log(res$padj), col = mycols)
abline(v = c(-2,2), col = "grey")
abline(h = -log(0.005), col = "grey")
```



Save my results to date out to disc

```
write.csv(res, file = "DESeq2_results.csv")
```

Annotation and gene set enrichment

translate the gene identifiers (the ensemble accession) into actual gene names. To do this "annotation" we will use the `AnnotationDbi` package. I can install it with the `Bioconductor::install()`

```
library(AnnotationDbi)
library(org.Hs.eg.db) #all the common genome databases
```

```
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"
```

will use the MapId function to map the identifiers to different databases. Go between ENSEMBL and SYMBOL

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

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

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

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

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

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

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

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

```
head(res)
```

```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 10 columns
                  baseMean log2FoldChange     lfcSE      stat    pvalue
                 <numeric>      <numeric> <numeric> <numeric> <numeric>
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               genename      entrez
                <numeric> <character>            <character> <character>
ENSG00000000003  0.163035      TSPAN6          tetraspanin 6        7105
ENSG00000000005        NA        TNMD            tenomodulin       64102
ENSG00000000419  0.176032        DPM1 dolichyl-phosphate m..        8813
ENSG00000000457  0.961694       SCYL3 SCY1 like pseudokina..       57147
ENSG00000000460  0.815849       FIRRM FIGNL1 interacting r..       55732
ENSG00000000938        NA         FGR FGR proto-oncogene, ..        2268
                    uniprot
                <character>
ENSG00000000003  A0A024RCI0
```

```
ENSG00000000005      Q9H2S6
ENSG00000000419      O60762
ENSG00000000457      Q8IZE3
ENSG00000000460  A0A024R922
ENSG00000000938      P09769
```

Save the annotated results object

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

Pathway Analysis

Now that we have our results with added annotations. We can do some pathway mapping

Let's use the **gage** package to look for KEGG pathways. Will also use **pathview** packages to visualize the pathways

```
library(pathview)
```

```
#############################################################################
Pathview is an open source software package distributed under GNU General
Public License version 3 (GPLv3). Details of GPLv3 is available at
http://www.gnu.org/licenses/gpl-3.0.html. Particullary, users are required to
formally cite the original Pathview paper (not just mention it) in publications
or products. For details, do citation("pathview") within R.

The pathview downloads and uses KEGG data. Non-academic uses may require a KEGG
license agreement (details at http://www.kegg.jp/kegg/legal.html).
#############################################################################
```

```
library(gage)
```

```
library(gageData)
```

```
data(kegg.sets.hs)

# Examine the first 2 pathways in this kegg set for humans
head(kegg.sets.hs, 2)
```

```
$`hsa00232 Caffeine metabolism`
[1] "10"   "1544" "1548" "1549" "1553" "7498" "9"

$`hsa00983 Drug metabolism - other enzymes`
 [1] "10"     "1066"   "10720"  "10941"  "151531" "1548"   "1549"   "1551"
 [9] "1553"   "1576"   "1577"   "1806"   "1807"   "1890"   "221223" "2990"
[17] "3251"   "3614"   "3615"   "3704"   "51733"  "54490"  "54575"  "54576"
[25] "54577"  "54578"  "54579"  "54600"  "54657"  "54658"  "54659"  "54963"
[33] "574537" "64816"  "7083"   "7084"   "7172"   "7363"   "7364"   "7365"
[41] "7366"   "7367"   "7371"   "7372"   "7378"   "7498"   "79799"  "83549"
[49] "8824"   "8833"   "9"      "978"
```

what **gage** wants as input is not the big table/dataframe. It just wants a one dimensional "vector of importance". For RNASeq data life we have, this is log2FC values.

```
fc <- res$log2FoldChange
names(fc) <- res$entrez
head(fc)
```

```
       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(fc, gsets=kegg.sets.hs)
```

Checking the content of this pathway analysis object

```
attributes(keggres)
```

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

```
#greater = up regulated, less = down regulated
```

```
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
```

Lets use the pathview package to look as one of these of the highlighted KEGG pathways with our genes highlighted. "hsa05310 Asthma"

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

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

```
Info: Working in directory /Users/longmeizhang/Desktop/BIMM143/lab13
```

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

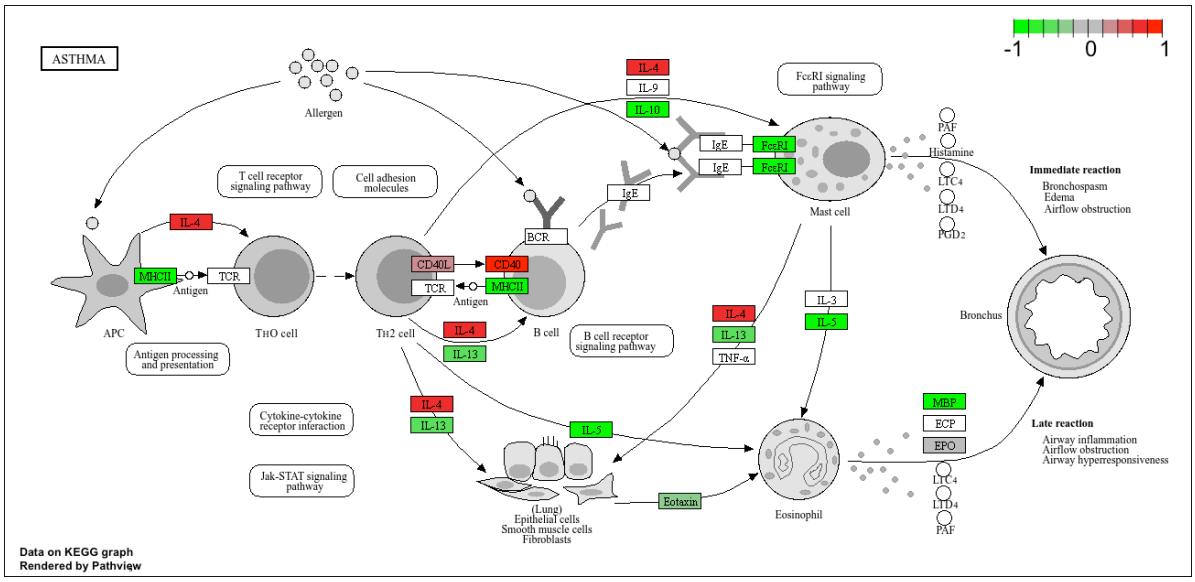


Figure 1: Asthma pathway with my DEGs

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

```
t2down = row.names(keggres$less[1:2,])
```

```
pathview(gene.data=fc, pathway.id=c("hsa05332", "hsa04940"))
```

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

```
Info: Working in directory /Users/longmeizhang/Desktop/BIMM143/lab13
```

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

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

```
Info: Working in directory /Users/longmeizhang/Desktop/BIMM143/lab13
```

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

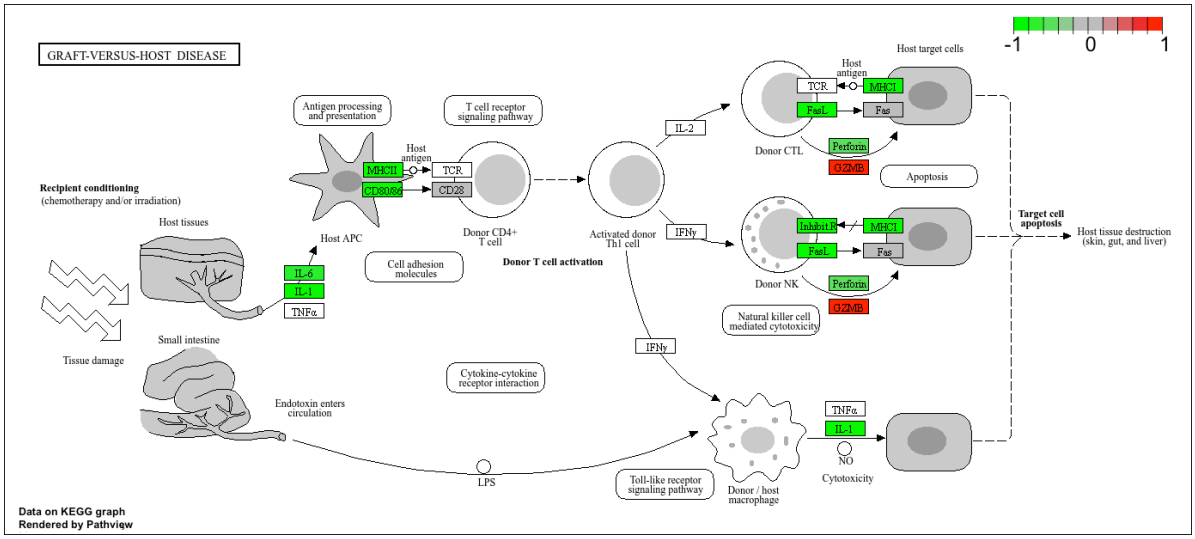


Figure 2: Graft-versus-host disease

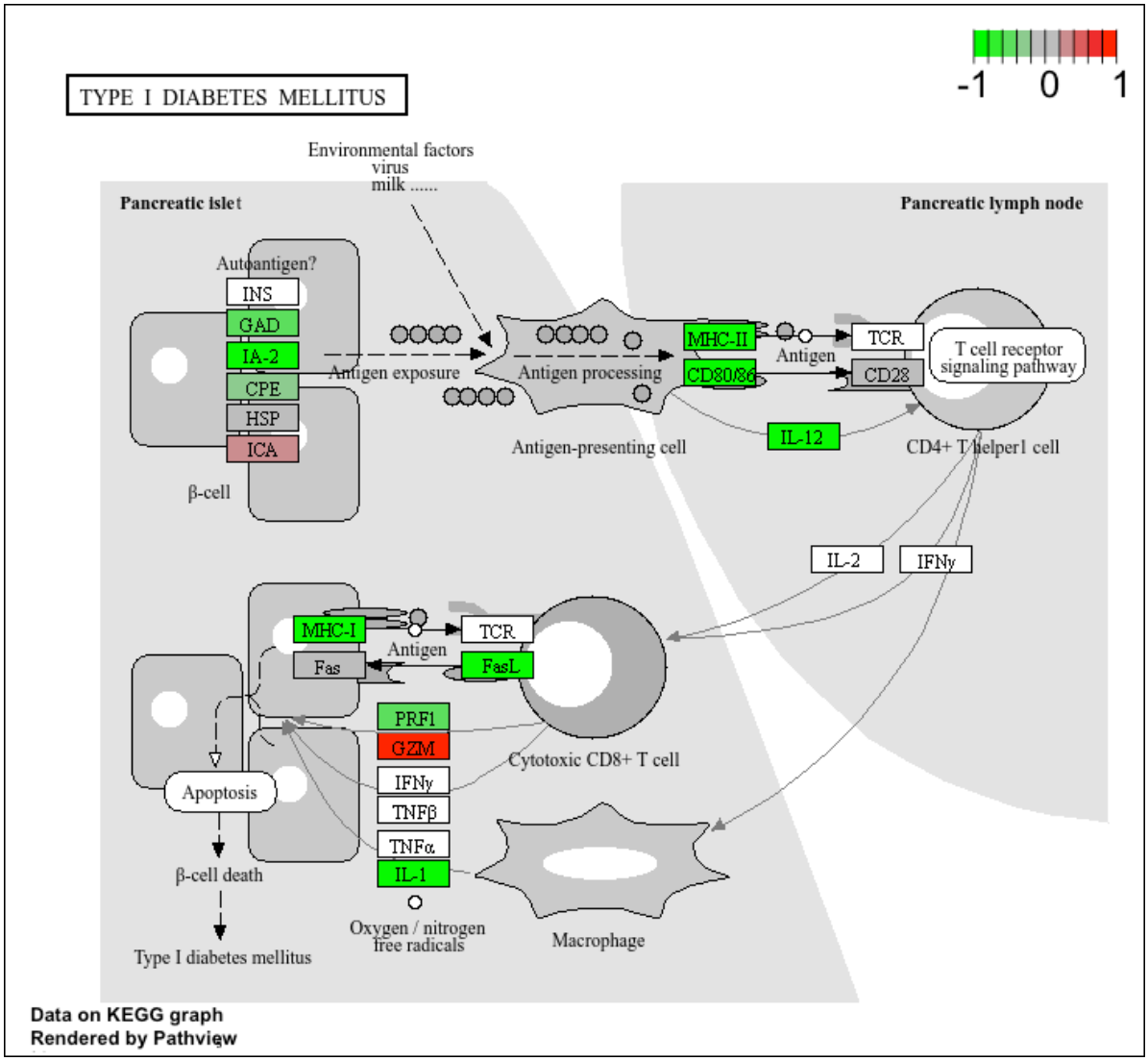


Figure 3: Graft-versus-host disease