Class 13, RNAseq analytics

James Woolley A16440072

Today we're looking at RNAseq data about asthma response to Dexamethasone, a common antiinflamatory drug used as treatment.

##Importing Data First things first, we need to read the data.

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

Q1 How many genes are in this dataset?

First, let's take a look at the data so we can see what function to use to get the gene number.

```
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
```

We can see that each row corresponds to a different gene, so we can use the `nrow` function to call the number of rows.

```
nrow(counts)
```

```
[1] 38694
```

This reveals that there are 38,694 genes in this dataset.

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

We can take a look at the metadata to see which are marked with 'control'.

```
table(metadata$dex)
```

```
control treated
      4       4
```

Tabling this data reveals that there are 4 controls.

##Toy differential gene expression Let's do a quick demonstration of what gene expression analysis looks like. (you shouldn't normally do gene expression analysis like this) We want to compare the control means to the treated means, so we can run code for both treated means and control means. Basically, we're finding which columns in `counts` correspond to control samples, then we cancluate the mean value of the genes in each column, finally we save that data as a variable called `control.mean`

```
control <- metadata[metadata[,"dex"]=="control",] #we want to look at the dex column becau
control.counts <- counts[ ,control$id] #this will give data for ONLY the control experimen
control.mean <- rowSums( control.counts )/4 #some math to calculate the mean
head(control.mean) #taking a look at the means we calculated.
```

```
ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
         900.75             0.00           520.50           339.75            97.25
ENSG00000000938
           0.75
```

```
library(dplyr) #you could do the same thing with tidyverse dplyr.
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

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

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

Q3. How would you make the above code in either approach more robust? Is there a function that could help here?

Adding more data would result in a miscalculated mean because we're directly calculating mean by dividing by 4. We could instead use the `rowMeans` function to calculate everything.

```
#rowMeans(control.counts) #notice how it gives the same values as `control.mean`
#commented out because it otherwise prints 20 pages of data.
```

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)

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

Now that we have everything calculated we can table both sets of data into one data frame

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

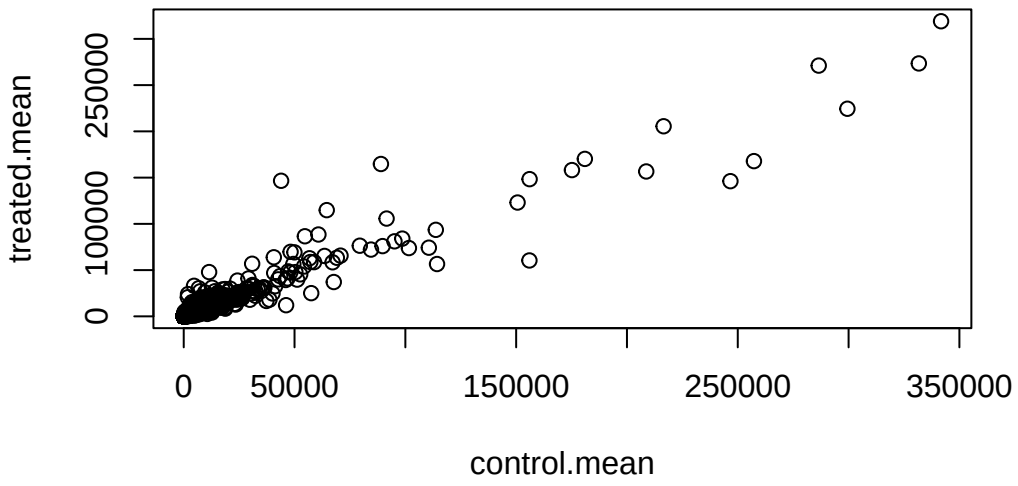
```
                control.mean treated.mean
ENSG00000000003       900.75       658.00
ENSG00000000005         0.00         0.00
ENSG00000000419       520.50       546.00
ENSG00000000457       339.75       316.50
ENSG00000000460        97.25        78.75
ENSG00000000938         0.75         0.00
```

```
colSums(meancounts)
```

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

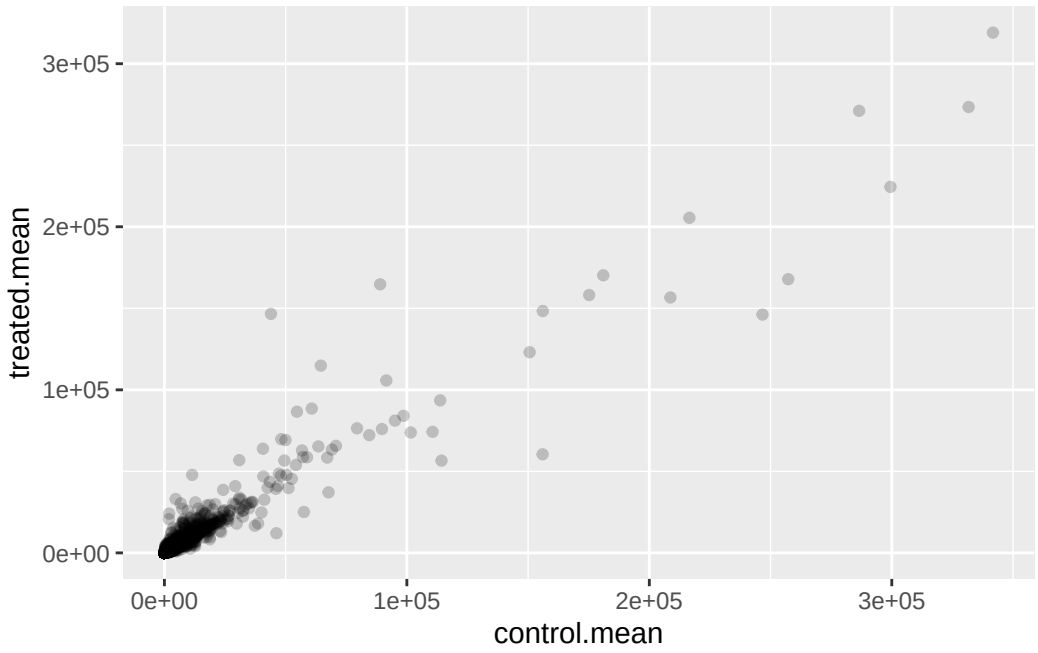
Q5 (a). Create a scatter plot showing the mean of the treated samples against the mean of the control samples.

```
plot(meancounts)
```



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(control.mean, treated.mean) +
geom_point(alpha=0.2)
```



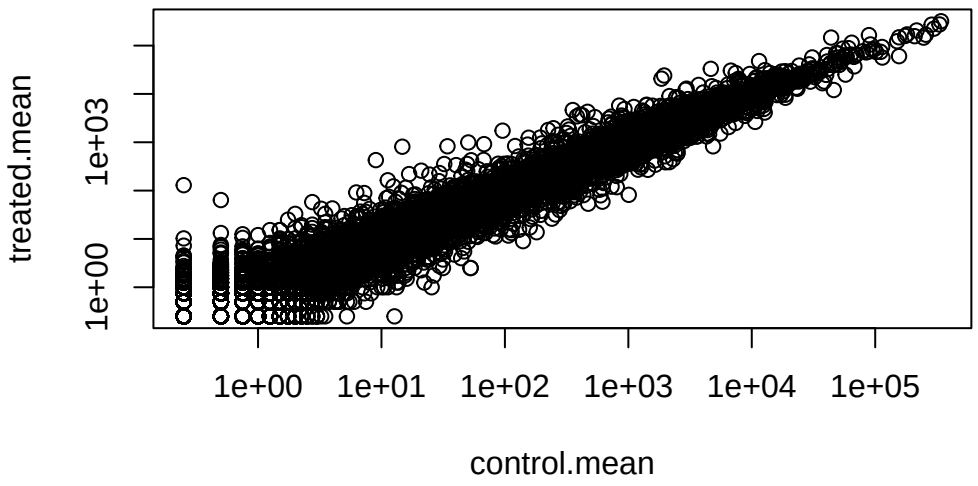
We can see that these graphs are incredibly unhelpful, because all the data is squeezed into the bottom left. We need to perform a log transformation to spread out the data some more.

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

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

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

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



We can use the `log=""` argument to make both axes on a log scale. Generally, we prefer to use `log2` to do log scales because it makes the interpretation of the data much easier. So let's add a log2 fold-change column 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
```

This is good because it provides an easy to see picture about the effects of the treatment, but we also get some werid data like NaN and -Inf. We can fix this by:

```
zero.vals <- which(meancounts[,1:2]==0, arr.ind=TRUE) #this is checking if there are 0 col
```

```
to.rm <- unique(zero.vals[,1]) #this defines all the zeros into their own group
mycounts <- meancounts[-to.rm,] #this kills all the zeroes
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) #many less rows now
```

```
[1] 21817
```

The same can be accomplished easier through:

```
to.rm.inds <- rowSums(meancounts[,1:2] == 0)>0
mycounts1 <- meancounts[!to.rm.inds,]
```

```
head(mycounts1)
```

```
                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
```

```
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
```

Notice how `mycounts` and `mycounts1` are the same.

> 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 makes `which()` return row and column indicies where the value is `TRUE`, which shows which zeroes we need to remove. The `unique` function makes sure that each count is unique so that rows don't get counted multiple times.

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

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

```
sum(up.ind)
```

```
[1] 250
```

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

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

```
[1] 367
```

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

We can't trust these numbers because we haven't established a standard for what changes are statistically significant.

##Setting up for DESeq We can use the DESeq package to do the analysis with proper statistical considerations in mind.

```
library(DESeq2)
```

```
Loading required package: S4Vectors
```

```
Loading required package: stats4

Loading required package: BiocGenerics


Attaching package: 'BiocGenerics'

The following objects are masked from 'package:dplyr':

    combine, intersect, setdiff, union

The following objects are masked from 'package:stats':

    IQR, mad, sd, var, xtabs

The following objects are masked from 'package:base':

    anyDuplicated, aperm, 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:dplyr':

    first, rename

The following object is masked from 'package:utils':

    findMatches

The following objects are masked from 'package:base':

    expand.grid, I, unname

Loading required package: IRanges
```

```
Attaching package: 'IRanges'

The following objects are masked from 'package:dplyr':

    collapse, desc, slice

Loading required package: GenomicRanges

Loading required package: GenomeInfoDb

Loading required package: SummarizedExperiment

Warning: package 'SummarizedExperiment' was built under R version 4.3.2

Loading required package: MatrixGenerics

Loading required package: matrixStats


Attaching package: 'matrixStats'

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

    count


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,#setting up the input
                              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
```

Now we can run DESeq analysis.

```
dds <- DESeq(dds) #running analysis
```

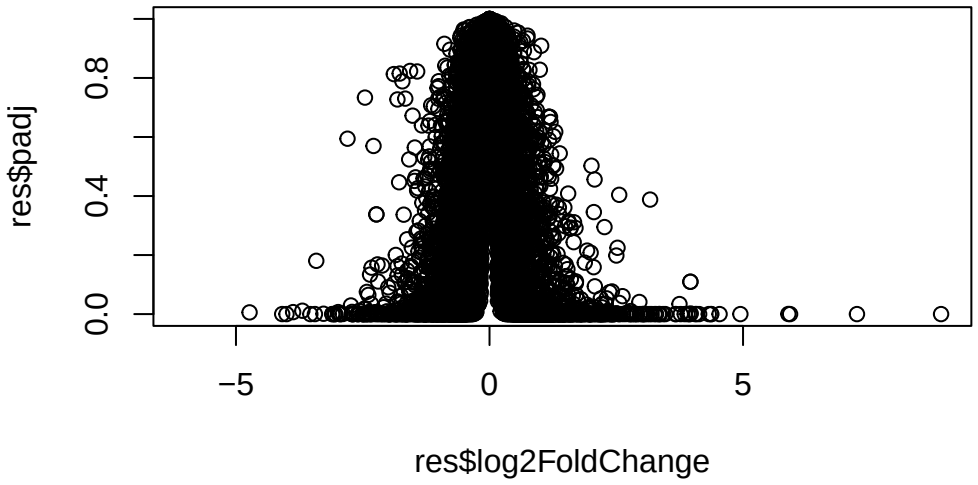
```
estimating size factors

estimating dispersions

gene-wise dispersion estimates

mean-dispersion relationship

final dispersion estimates

fitting model and testing
```

```
res <- results(dds) #viewing results
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
```

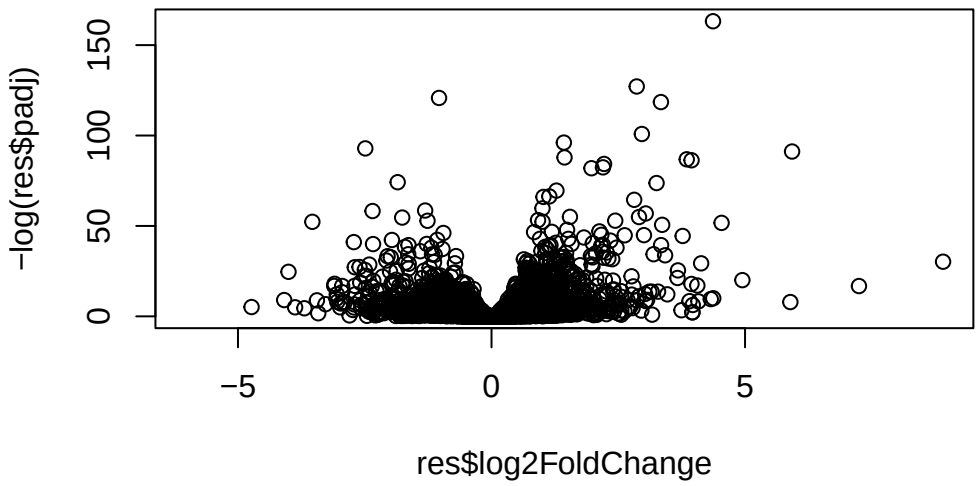
##Principal Component Analysis We can make a volcano plot, which is helpful because it shows both p-values and log2(foldchanges).

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



This is a pretty useless plot, so let's change some things to make it more intelligible. We don't really care about the genes with really high p-values

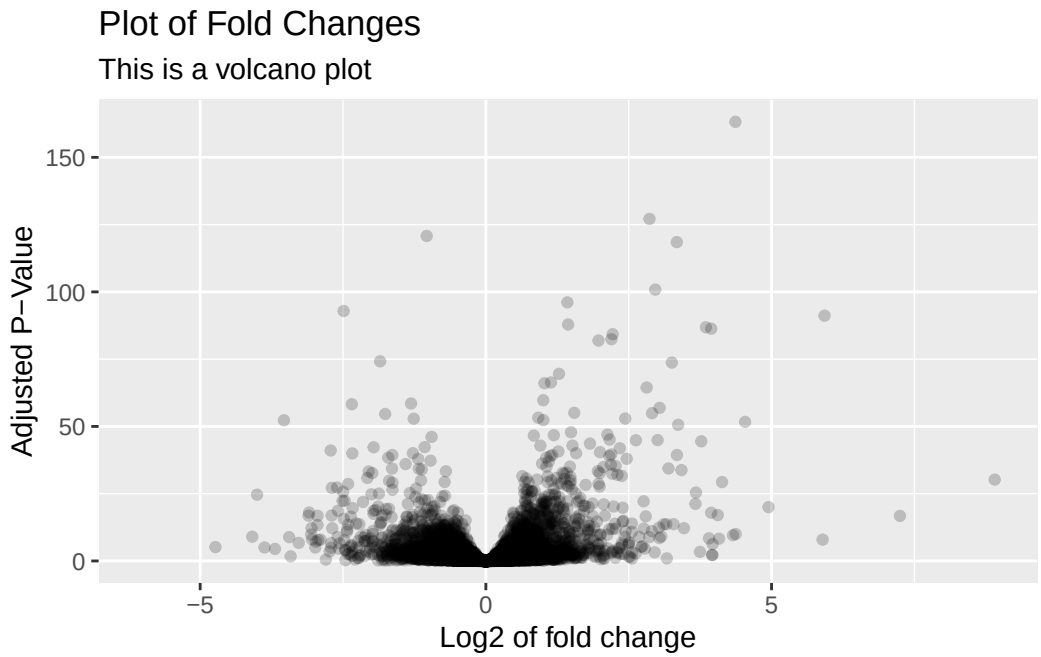
```
plot(res$log2FoldChange, -log(res$padj))
```



Let's make a nicer looking plot.

```
x <- as.data.frame(res)
library(ggplot2)
   ggplot(x) +
     aes(x$log2FoldChange, -log(x$padj)) +
     geom_point(alpha=0.2) +
     labs(title="Plot of Fold Changes",
          x="Log2 of fold change",
          y="Adjusted P-Value",
          subtitle="This is a volcano plot",)
```

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



Save our results to date

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

```
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
```

Adding Annotation Data

The remaining content in this lab report was finished during Class 14 16/11/23 We want to add annotation data because the information in the dataset only includes geneIDs, which isn't very helpful for interpretation by humans. We want to link them to actual names.

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

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

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

    select
```

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

```
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're going to use the `mapid` function to map the ensembl gene IDs we have to the actual names of the genes.

```
head(row.names(res))
```

```
[1] "ENSG00000000003" "ENSG00000000005" "ENSG00000000419" "ENSG00000000457"
[5] "ENSG00000000460" "ENSG00000000938"
```

These aren't very helpful. We want to translate them to symbols.

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

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

That should've added a column of the symbols.

```
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       FIRRM
ENSG00000000938        NA         FGR
```

And we can see that it has been added after the padj row! We can also add the gene names and others to make it even better.

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

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

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

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

```
head(res)
```

```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 9 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
```

##Pathway Analysis

We will use the **gage** package and the **pathview** package to do pathway analysis and find generation.

```
library(pathview)
library(gage)
library(gageData)

data(kegg.sets.hs)
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"
```

To use `gage()` we need our genes in ENTREZID format with a measure of their importance in a vector, such as fold-change.

```
foldchanges <- res$log2FoldChange
head(foldchanges)
```

```
[1] -0.35070302          NA  0.20610777  0.02452695 -0.14714205 -1.73228897
```

We can add ENTERZIDS as `names()` to the `foldchanges` vector.

```
names(foldchanges) <- res$entrez
head(foldchanges)
```

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

Now that they're properly labelled, we can run `gage()` to examine the enrichment.

```
keggres = gage(foldchanges, gsets=kegg.sets.hs)
```

Now let's visualise the results

```
attributes(keggres)
```

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

```
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
```

We want to take a look at Dex's most common use, which is with asthma.

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

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

```
Info: Working in directory /Users/jameswoolley/Library/Mobile Documents/com~apple~CloudDocs/U
```

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

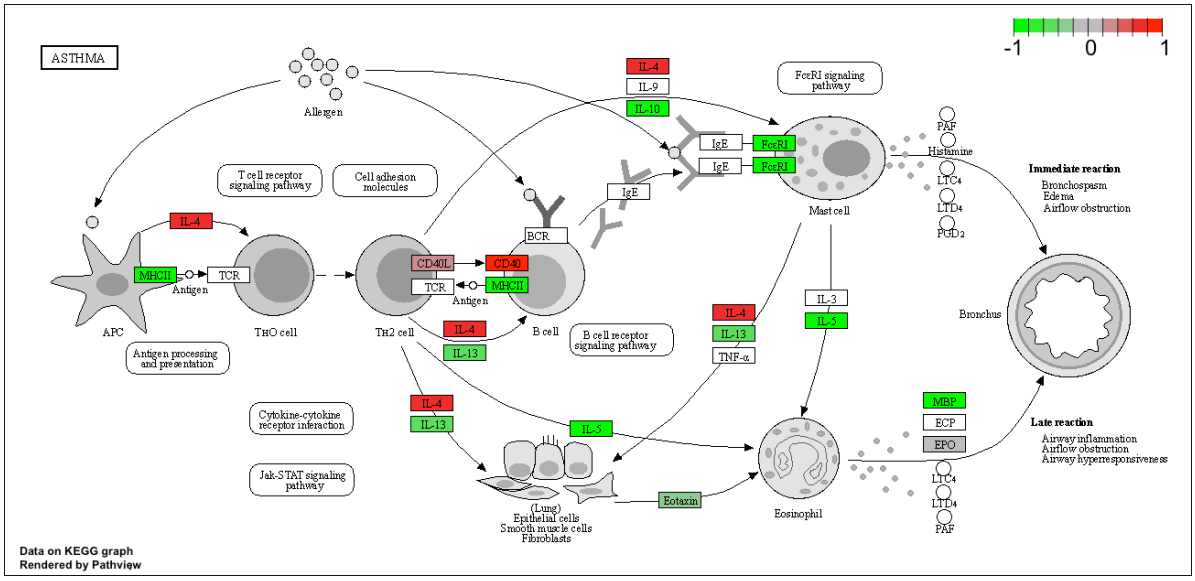


Figure 1: Gene Pathway