Class 13

Job Rocha PID: 59023124

Transcriptomics and the analysis of RNA-Seq data

Background.

The data for this hands-on session comes from a published RNA-seq experiment where airway smooth muscle cells were treated with dexamethasone, a synthetic glucocorticoid steroid with anti-inflammatory effects (Himes et al. 2014).

Glucocorticoids are used, for example, by people with asthma to reduce inflammation of the airways. The anti-inflammatory effects on airway smooth muscle (ASM) cells has been known for some time but the underlying molecular mechanisms are unclear.

Himes et al. used RNA-seq to profile gene expression changes in four different ASM cell lines treated with dexamethasone glucocorticoid. They found a number of differentially expressed genes comparing dexamethasone-treated to control cells, but focus much of the discussion on a gene called CRISPLD2. This gene encodes a secreted protein known to be involved in lung development, and SNPs in this gene in previous GWAS studies are associated with inhaled corticosteroid resistance and bronchodilator response in asthma patients. They confirmed the upregulated CRISPLD2 mRNA expression with qPCR and increased protein expression using Western blotting.

In the experiment, four primary human ASM cell lines were treated with 1 micromolar dexamethasone for 18 hours. For each of the four cell lines, we have a treated and an untreated sample. They did their analysis using Tophat and Cufflinks similar to our last day's hands-on session. For a more detailed description of their analysis see the PubMed entry 24926665 and for raw data see the GEO entry GSE52778.

In this session we will read and explore the gene expression data from this experiment using base R functions and then perform a detailed analysis with the DESeq2 package from Bioconductor.

1. Bioconductor and DESeq2 setup.

As we already noted back in Class 6 Bioconductor is a large repository and resource for R packages that focus on analysis of high-throughput genomic data.

Bioconductor packages are installed differently than "regular" R packages from CRAN. To install the core Bioconductor packages, copy and paste the following two lines of code into your R console one at a time.

N.B. Remember not to put these install commands in your Rmd report as it will at best re-install the packages every time you kint your report, which is not what you want, and at worst cause a confusing knit error.

```
#install.packages("BiocManager")

# For this class, you'll also need DESeq2:
#BiocManager::install("DESeq2")
```

The entire install process can take some time as there are many packages with dependencies on other packages. For some important notes on the install process please see our Bioconductor setup notes. Your install process may produce some notes or other output. Generally, as long as you don't get an error message, you're good to move on. If you do see error messages then again please see our Bioconductor setup notes for debugging steps.

Finally, check that you have installed everything correctly by closing and reopening RStudio and entering the following two commands at the console window:

```
library(BiocManager)
library(DESeq2)
```

If you get a message that says something like: Error in library(DESeq2) : there is no package called 'DESeq2', then the required packages did not install correctly. Please see our Bioconductor setup notes and let us know so we can debug this together.

Side-note: Aligning reads to a reference genome.

The computational analysis of an RNA-seq experiment begins from the FASTQ files that contain the nucleotide sequence of each read and a quality score at each position. These reads must first be aligned to a reference genome or transcriptome. The output of this alignment step is commonly stored in a file format called SAM/BAM. This is the workflow we followed last day.

Once the reads have been aligned, there are a number of tools that can be used to count the number of reads/fragments that can be assigned to genomic features for each sample. These

often take as input SAM/BAM alignment files and a file specifying the genomic features, e.g. a GFF3 or GTF file specifying the gene models as obtained from ENSEMBLE or UCSC.

In the workflow we'll use here, the abundance of each transcript was quantified using kallisto (software, paper) and transcript-level abundance estimates were then summarized to the gene level to produce length-scaled counts using the R package txImport (software, paper), suitable for using in count-based analysis tools like DESeq. This is the starting point - a "count matrix", where each cell indicates the number of reads mapping to a particular gene (in rows) for each sample (in columns). This is where we left off last day when analyzing our 1000 genome data.

Note: This is one of several well-established workflows for data pre-processing. The goal here is to provide a reference point to acquire fundamental skills with DESeq2 that will be applicable to other bioinformatics tools and workflows. In this regard, the following resources summarize a number of best practices for RNA-seq data analysis and pre-processing.

Conesa, A. et al. "A survey of best practices for RNA-seq data analysis." Genome Biology 17:13 (2016). Soneson, C., Love, M. I. & Robinson, M. D. "Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences." F1000Res. 4:1521 (2016). Griffith, Malachi, et al. "Informatics for RNA sequencing: a web resource for analysis on the cloud." PLoS Comput Biol 11.8: e1004393 (2015).

DESeq2 Required Inputs.

As input, the DESeq2 package expects (1) a data.frame of count data (as obtained from RNA-seq or another high-throughput sequencing experiment) and (2) a second data.frame with information about the samples - often called sample metadata (or colData in DESeq2-speak because it supplies metadata/information about the columns of the countData matrix).

The "count matrix" (called the countData in DESeq2-speak) the value in the i-th row and the j-th column of the data.frame tells us how many reads can be assigned to gene i in sample j. Analogously, for other types of assays, the rows of this matrix might correspond e.g. to binding regions (with ChIP-Seq) or peptide sequences (with quantitative mass spectrometry).

For the sample metadata (i.e. colData in DESeq2-speak) samples are in rows and metadata about those samples are in columns. Notice that the first column of colData must match the column names of countData (except the first, which is the gene ID column).

Note from the DESeq2 vignette: The values in the input contData object should be counts of sequencing reads/fragments. This is important for DESeq2's statistical model to hold, as only counts allow assessing the measurement precision correctly. It is important to never provide counts that were pre-normalized for sequencing depth/library size, as the statistical model is most powerful when applied to un-normalized counts, and is designed to account for library size differences internally.

2. Import countData and colData.

First, create a new RStudio project (File > New Project > New Directory > New Project) and download the input airway_scaledcounts.csv and airway_metadata.csv into your project directory.

Begin a new R script and use the read.csv() function to read these count data and metadata files.

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

Now, take a look at each.

```
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
```

You can also use the View() function to view the entire object. Notice something here. The sample IDs in the metadata sheet (SRR1039508, SRR1039509, etc.) exactly match the column names of the countdata, except for the first column, which contains the Ensembl gene ID. This is important, and we'll get more strict about it later on.

Q1. How many genes are in this dataset? *38,694 genes.*

```
nrow(counts)
```

```
[1] 38694
```

Q2. How many 'control' cell lines do we have? *4 control cell lines.*

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

```
[1] 4
```

3. Toy differential gene expression

Lets perform some exploratory differential gene expression analysis. Note: this analysis is for demonstration only. NEVER do differential expression analysis this way!

Look at the metadata object again to see which samples are control and which are drug treated. You can also see this in the metadata printed table below:

Note that the control samples are SRR1039508, SRR1039512, SRR1039516, and SRR1039520. This bit of code will first find the sample id for those labeled control. Then calculate the mean counts per gene across these samples:

I want to compare the control to the treated columns. To do this I will:

- Step 1. Identify and extract the "control" columns.
- Step 2. Calcultate the mean value per gene for all these "control columns and save as `control.mean`
- Step 3: Do the same for treated.
- Step 4. Compare the `control.mean` and `treated.mean` values.

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

```
head(control.mean)
```

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

Side-note: An alternative way to do this same thing using the dplyr package from the tidyverse is shown below. Which do you prefer and why?

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

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

Q3. How would you make the above code in either approach more robust? *Using romMeans instead of rowSums and converting it into a function.*

```
calculate.mean <- function(counts, medatadata, group){
  subgroup <- metadata %>% filter(dex==group)
  subgroup.counts <- counts %>% select(subgroup$id)
  return (rowMeans(subgroup.counts))
}
```

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.mean <- calculate.mean(counts, metadata, "treated")
head(treated.mean)
```

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

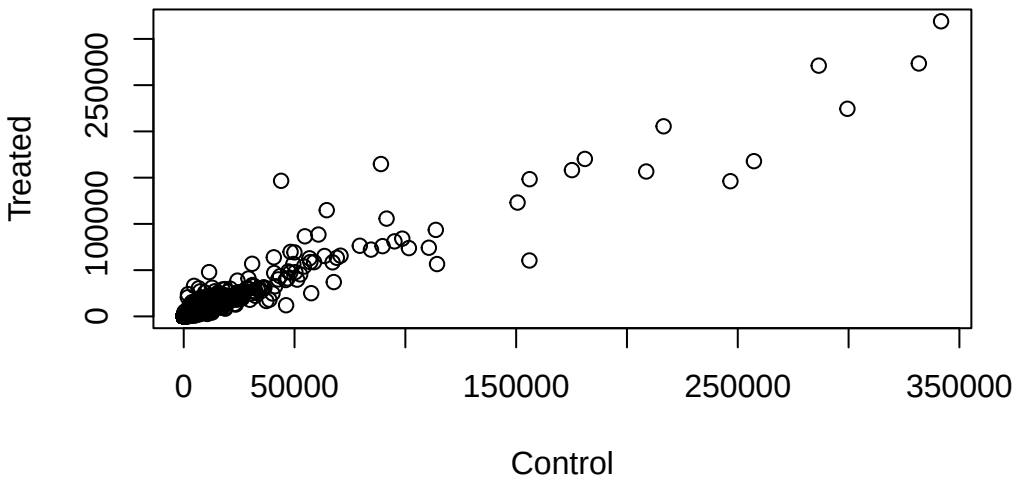
We will combine our meancount data for bookkeeping purposes.

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

Directly comparing the raw counts is going to be problematic if we just happened to sequence one group at a higher depth than another. Later on we'll do this analysis properly, normalizing by sequencing depth per sample using a better approach. But for now, colSums() the data to show the sum of the mean counts across all genes for each group. Your answer should look like this:

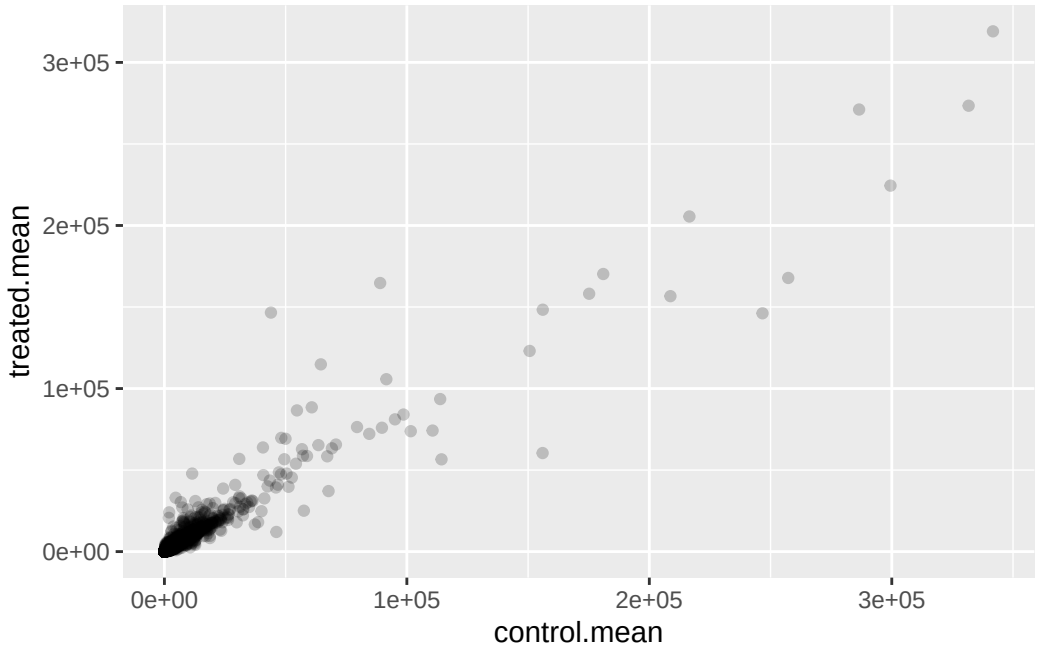
Q5 (a). Create a scatter plot showing the mean of the treated samples against the mean of the control samples. Your plot should look something like the following.

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



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*

```
library(ggplot2)
ggplot(meancounts, aes(x=control.mean, y=treated.mean)) +
  geom_point(alpha=0.2)
```



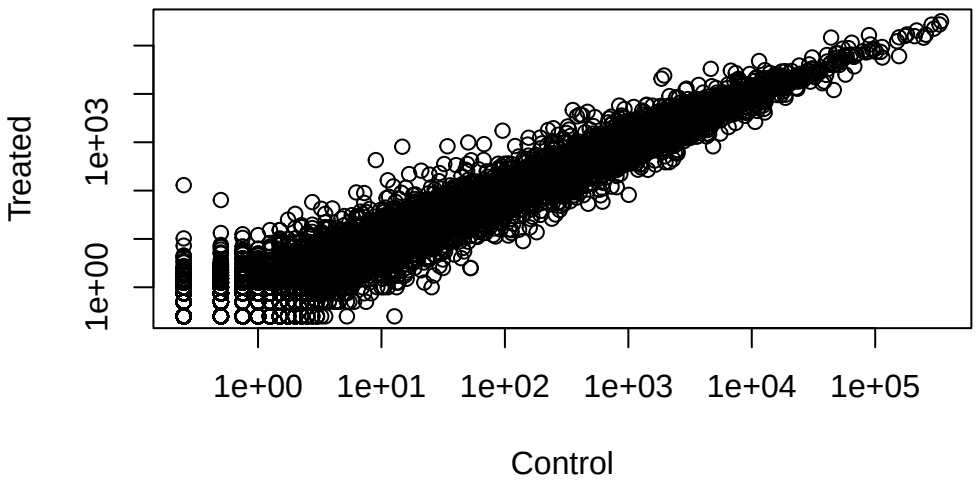
Wait a sec. There are 60,000-some rows in this data, but I'm only seeing a few dozen dots at most outside of the big clump around the origin

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

```
plot(meancounts, xlab="Control", ylab="Treated", 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
```



Logs are super useful when we have such skewed data they are also handy when we are most interested in

```
#Treated / control
log2(10/10)
```

```
[1] 0
```

```
log2(20/10)
```

```
[1] 1
```

```
log2(10/20)
```

```
[1] -1
```

If you are using ggplot have a look at the function scale_x_continuous(trans="log2") and of course do the same for the y axis.

We can find candidate differentially expressed genes by looking for genes with a large change between control and dex-treated samples. We usually look at the log2 of the fold change, because this has better mathematical properties.

Here we calculate log2foldchange, add it to our meancounts data.frame and inspect the results either with the head() or the View() function for example.

Add log2(Fold-change) values to aur wee results table.

```
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.

The NaN is returned when you divide by zero and try to take the log. The -Inf is returned when you try to take the log of zero. It turns out that there are a lot of genes with zero expression. Let's filter our data to remove these genes. Again inspect your result (and the intermediate steps) to see if things make sense to you

I need to exclude any genes with zero counts as we can't say anithing aboth them anyway form this experiment and it causes math pain.

```
# What values in the first two colus are zero
to.rm.inds <- rowSums(meancounts[,1:2]==0) > 0
mycounts <- meancounts[!to.rm.inds,]
mycounts$log2fc <- log2(mycounts[,"treated.mean"]/mycounts[,"control.mean"])
nrow(mycounts)
```

```
[1] 21817
```

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? It is used to return a logical vector that includes labels.

Q8. How many genes are "up regulated" i.e. have a log2(fold-change) grater than +2? ***258***

```
sum(mycounts$log2fc > 2)
```

```
[1] 250
```

Q9. How many genes are "up regulated" i.e. have a log2(fold-change) less than -2? ***367***

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

```
[1] 367
```

Q10. Do you trust these results? Why or why not? ***Not really, I dont have any statistical metrics to tell how confident are the results.***

5. Setting up for DESeq

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

Bioconductor packages usually have great documentation in the form of vignettes. For a great example, take a look at the DESeq2 vignette for analyzing count data. This 40+ page manual is packed full of examples on using DESeq2, importing data, fitting models, creating visualizations, references, etc.

Just like R packages from CRAN, you only need to install Bioconductor packages once (instructions here), then load them every time you start a new R session.

```
library(DESeq2)
citation("DESeq2")
```

```
To cite package 'DESeq2' in publications use:

  Love, M.I., Huber, W., Anders, S. Moderated estimation of fold change
  and dispersion for RNA-seq data with DESeq2 Genome Biology 15(12):550
  (2014)

A BibTeX entry for LaTeX users is

  @Article{,
    title = {Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2
    author = {Michael I. Love and Wolfgang Huber and Simon Anders},
    year = {2014},
    journal = {Genome Biology},
    doi = {10.1186/s13059-014-0550-8},
    volume = {15},
    issue = {12},
    pages = {550},
  }
```

Take a second and read through all the stuff that flies by the screen when you load the DESeq2 package. When you first installed DESeq2 it may have taken a while, because DESeq2 depends on a number of other R packages (S4Vectors, BiocGenerics, parallel, IRanges, etc.) Each of these, in turn, may depend on other packages. These are all loaded into your working environment when you load DESeq2. Also notice the lines that start with The following objects are masked from 'package:….

Importing data

Bioconductor software packages often define and use custom class objects for storing data. This helps to ensure that all the needed data for analysis (and the results) are available. DESeq works on a particular type of object called a DESeqDataSet. The DESeqDataSet is a single object that contains input values, intermediate calculations like how things are normalized, and all results of a differential expression analysis.

You can construct a DESeqDataSet from (1) a count matrix, (2) a metadata file, and (3) a formula indicating the design of the experiment.

We have talked about (1) and (2) previously. The third needed item that has to be specified at the beginning of the analysis is a design formula. This tells DESeq2 which columns in the sample information table (colData) specify the experimental design (i.e. which groups the samples belong to) and how these factors should be used in the analysis. Essentially, this formula expresses how the counts for each gene depend on the variables in colData.

Take a look at metadata again. The thing we're interested in is the dex column, which tells us which samples are treated with dexamethasone versus which samples are untreated controls. We'll specify the design with a tilde, like this: design=~dex. (The tilde is the shifted key to the left of the number 1 key on my keyboard. It looks like a little squiggly line).

We will use the DESeqDataSetFromMatrix() function to build the required DESeqDataSet object and call it dds, short for our DESeqDataSet. If you get a warning about "some variables in design formula are characters, converting to factors" don't worry about it. Take a look at the dds object once you create 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
```

```
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
```

To run DESeq analysis we call the main function form the package called `DESeq(dds)`

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

```
estimating size factors
```

```
estimating dispersions
```

```
gene-wise dispersion estimates
```

```
mean-dispersion relationship
```

```
final dispersion estimates
```

```
fitting model and testing
```

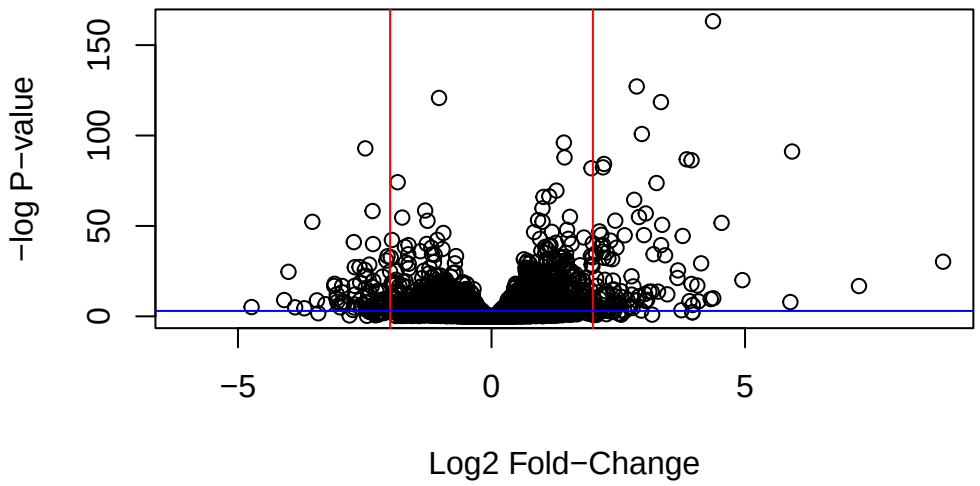
To get the results out of this `dds` object we can use the DESeq `results()` function.

```
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
```

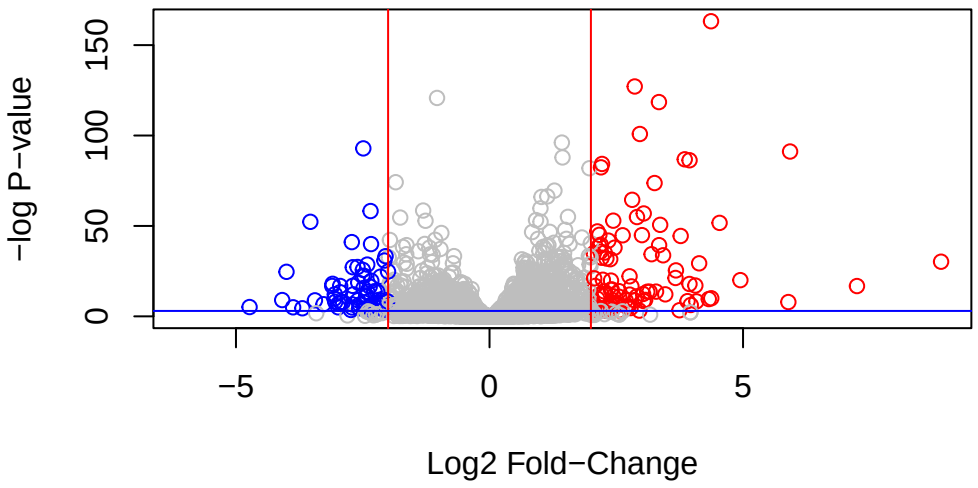
A common summary visualization is called a Volcano plot.

```
plot(res$log2FoldChange, -log(res$padj),
     xlab = "Log2 Fold-Change",
     ylab="-log P-value")
abline(v=c(-2, 2), col="red")
abline(h=-log(0.05), col="blue")
```




```
mycols <- rep("gray", nrow(res))
mycols[res$log2FoldChange > 2] <- "red"
mycols[res$log2FoldChange < -2] <- "blue"
mycols[res$padj > 0.05] <- "grey"
```

```
plot(res$log2FoldChange, -log(res$padj),
     xlab = "Log2 Fold-Change",
     ylab="-log P-value",
     col=mycols)
abline(v=c(-2, 2), col="red")
abline(h=-log(0.05), col="blue")
```



Save our results to date

```
write.csv(res, "DEG_results.csv")
```

Adding annotation data

We need to translate or "map" our ensemble IDs into more undestandable gene names and the identifiers that other useful databases use.

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

library(AnnotationDbi)
```

```
Warning: package 'AnnotationDbi' was built under R version 4.3.2
```

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

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

    select
```

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

```
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
```

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$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
```

```
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
```

```
  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
```

```
  # Run in your R console (i.e. not your Rmarkdown doc!)
  #BiocManager::install( c("pathview", "gage", "gageData") )
```

```
  library(pathview)
  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"
```

```
  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
```

```
# Get the results
keggres = gage(foldchanges, gsets=kegg.sets.hs)
```

```
attributes(keggres)
```

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

```
# Look at the first three down (less) pathways
head(keggres$less, 3)
```

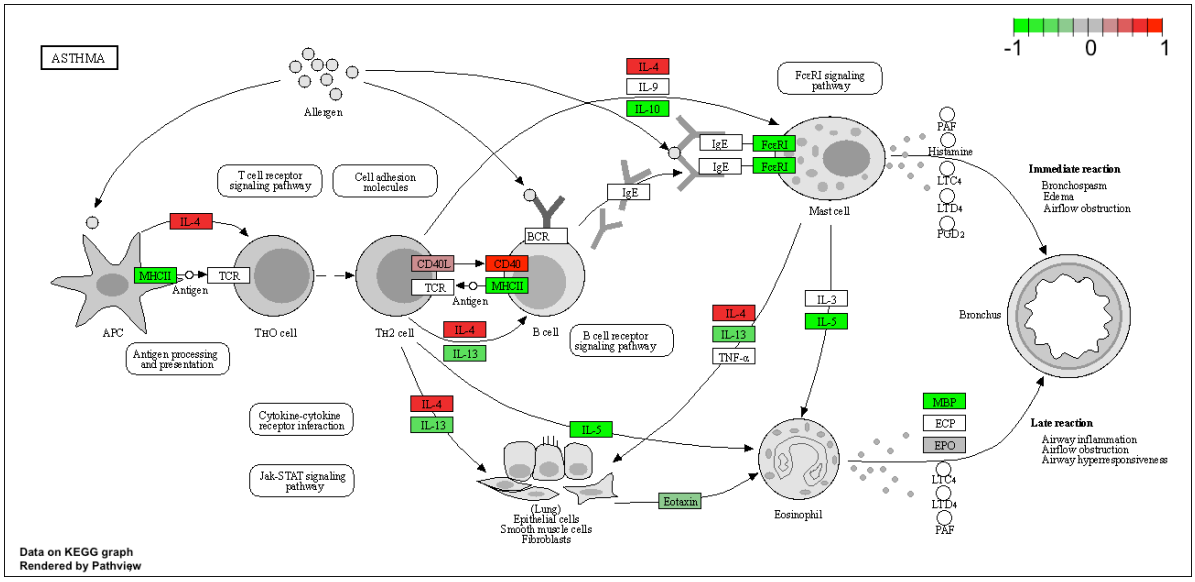
```
                                     p.geomean stat.mean        p.val
hsa05332 Graft-versus-host disease 0.0004250461 -3.473346 0.0004250461
hsa04940 Type I diabetes mellitus  0.0017820293 -3.002352 0.0017820293
hsa05310 Asthma                    0.0020045888 -3.009050 0.0020045888
                                        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
```

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

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

```
Info: Working in directory /Users/jrocha/Documents/PhD/Classes/Fall 2023/BGGN213/Class 12
```

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


ASTHMA
-1
0
1
Allegen
IL-4
IL-9
IL-10
FcεRI signaling pathway
IgE
FcεRI
Mast cell
PAF
Histamine
LTC4
LTD4
PGD2
Immediate reaction
Bronchospasm
Edema
Airflow obstruction
T cell receptor signaling pathway
Cell adhesion molecules
BCR
MHCII
TCR
Antigen
APC
TH0 cell
TH2 cell
B cell
CD40L
CD40
IL-4
IL-13
B cell receptor signaling pathway
Antigen processing and presentation
IL-3
IL-5
IL-13
TNF-α
Bronchus
Cytokine-cytokine receptor interaction
Jak-STAT signaling pathway
(Lung)
Epithelial cells
Smooth muscle cells
Fibroblasts
Eotaxin
Eosinophil
MBP
ECP
EPO
Late reaction
Airway inflamnation
Airflow obstruction
Airway hyperresponsiveness
Data on KEGG graph
Rendered by Pathview