

# class15\_transcriptomics

Jocelyn Olvera

11/17/2021

```
library(BiocManager)
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

```

## 2. Import countData and colData

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

### 1. Count data

2: colDAta the metadata that tells us about the disgn of the experiment.

```

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

```

Getting the first 6 rows of the data counts.

```

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
## ENSG000000000003    1097    806    604
## ENSG000000000005      0      0      0
## ENSG000000000419    781     417    509
## ENSG000000000457    447     330    324
## ENSG000000000460     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?

There are 38694 genes.

There are 38694 row/genes in this dataset.

```
nrow(counts)
```

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

Another way to get the number of genes.

```
dim(counts)
```

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

```
colnames(counts)
```

```
## [1] "SRR1039508" "SRR1039509" "SRR1039512" "SRR1039513" "SRR1039516"
## [6] "SRR1039517" "SRR1039520" "SRR1039521"
```

## Q2. How many ‘control’ cell lines do we have?

4 control cell lines.

Side-note:

lets check the correspondence of the metadata and count data setup.

```
metadata$id
```

two different methods

```
## [1] "SRR1039508" "SRR1039509" "SRR1039512" "SRR1039513" "SRR1039516"
## [6] "SRR1039517" "SRR1039520" "SRR1039521"
```

Use '=='to return a vector as “FALSE” or “TRUE” if columns and rows are the same.

```
all(metadata$id == colnames(counts))
```

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

Function that allows you to check if any of the columns/rows do not match.

```
all( c(T, T, T))
```

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

### 3. Toy differential gene expression

```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:Biobase':
##
##     combine

## The following object is masked from 'package:matrixStats':
##
##     count

## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union

## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect

## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union

## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union

## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

## CONTROL

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

```

## ENSG00000000003 ENSG00000000005 ENSG000000000419 ENSG000000000457 ENSG000000000460
##          900.75           0.00        520.50       339.75        97.25
## ENSG000000000938
##          0.75

```

**Q3.** How would you make the above code in either approach more robust?

```

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

## ENSG00000000003 ENSG00000000005 ENSG000000000419 ENSG000000000457 ENSG000000000460
##          900.75           0.00        520.50       339.75        97.25
## ENSG000000000938
##          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**

```

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

## ENSG00000000003 ENSG00000000005 ENSG000000000419 ENSG000000000457 ENSG000000000460
##          658.00           0.00        546.00       316.50        78.75
## ENSG000000000938
##          0.00

```

**Compare Control to treated.**

First we need to access all the control columns in our counts data.

```

control inds <- metadata$dex == "control"
control.id <- metadata[control.inds,]$id
control.id

## [1] "SRR1039508" "SRR1039512" "SRR1039516" "SRR1039520"

```

Use these ids to access just the control columns for our ‘counts’ data.

```

head(counts[, control.id])

##          SRR1039508 SRR1039512 SRR1039516 SRR1039520
## ENSG00000000003      723      904     1170      806
## ENSG00000000005       0         0         0         0
## ENSG000000000419     467      616      582      417
## ENSG000000000457     347      364      318      330
## ENSG000000000460      96       73      118      102
## ENSG000000000938      0         1         2         0

```

Taking the mean of the controls using ‘rowMeans’

```
control.mean <- rowMeans(control.counts)
head(control.mean)

## ENSG00000000003 ENSG00000000005 ENSG000000000419 ENSG000000000457 ENSG000000000460
##          900.75           0.00        520.50       339.75        97.25
## ENSG000000000938
##          0.75
```

Now do the same with the ‘treated’ group data.

```
treated.id <- metadata[metadata$dex == "treated",]$id
treated.id

## [1] "SRR1039509" "SRR1039513" "SRR1039517" "SRR1039521"
head(counts[,treated.id])

##          SRR1039509 SRR1039513 SRR1039517 SRR1039521
## ENSG00000000003      486       445     1097      604
## ENSG00000000005        0         0         0         0
## ENSG000000000419      523       371      781      509
## ENSG000000000457      258       237      447      324
## ENSG000000000460       81        66       94       74
## ENSG000000000938       0         0         0         0

treated.mean <- rowMeans(treated.counts)
head(treated.mean)

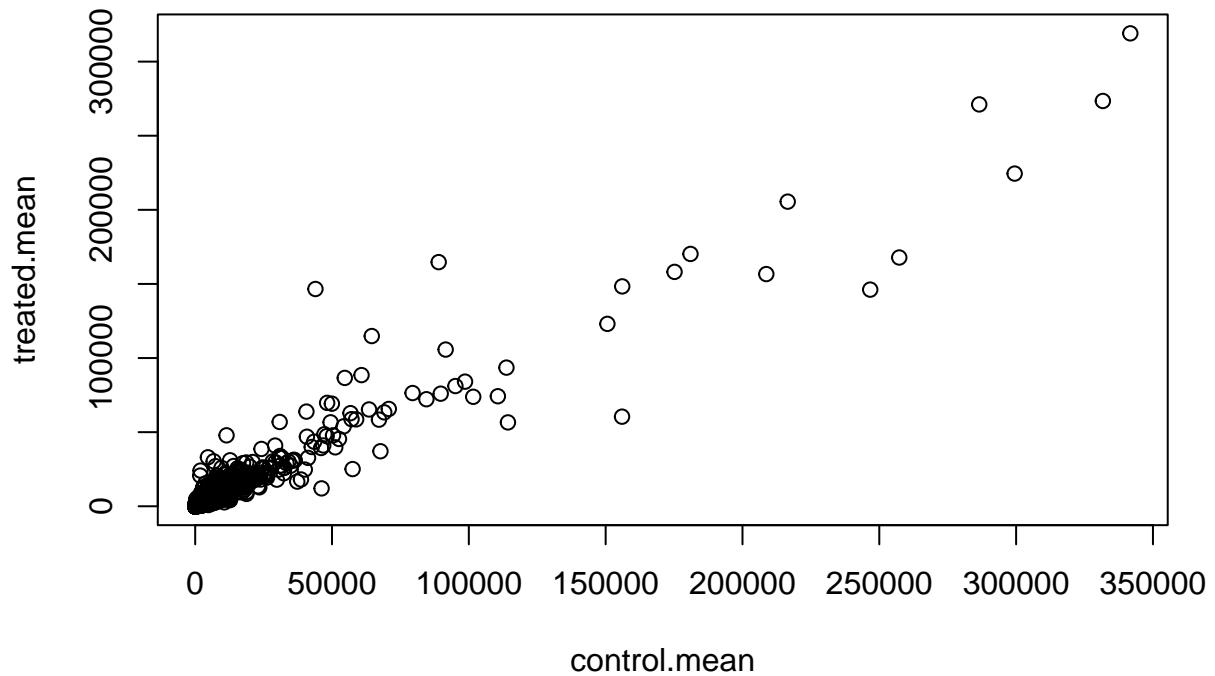
## ENSG00000000003 ENSG00000000005 ENSG000000000419 ENSG000000000457 ENSG000000000460
##          658.00           0.00        546.00       316.50        78.75
## ENSG000000000938
##          0.00
```

We will combine our meancount data for bookeeping purposes.

```
meancounts <- data.frame(control.mean, treated.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  
Easy plotting of control mean counts against treated means.

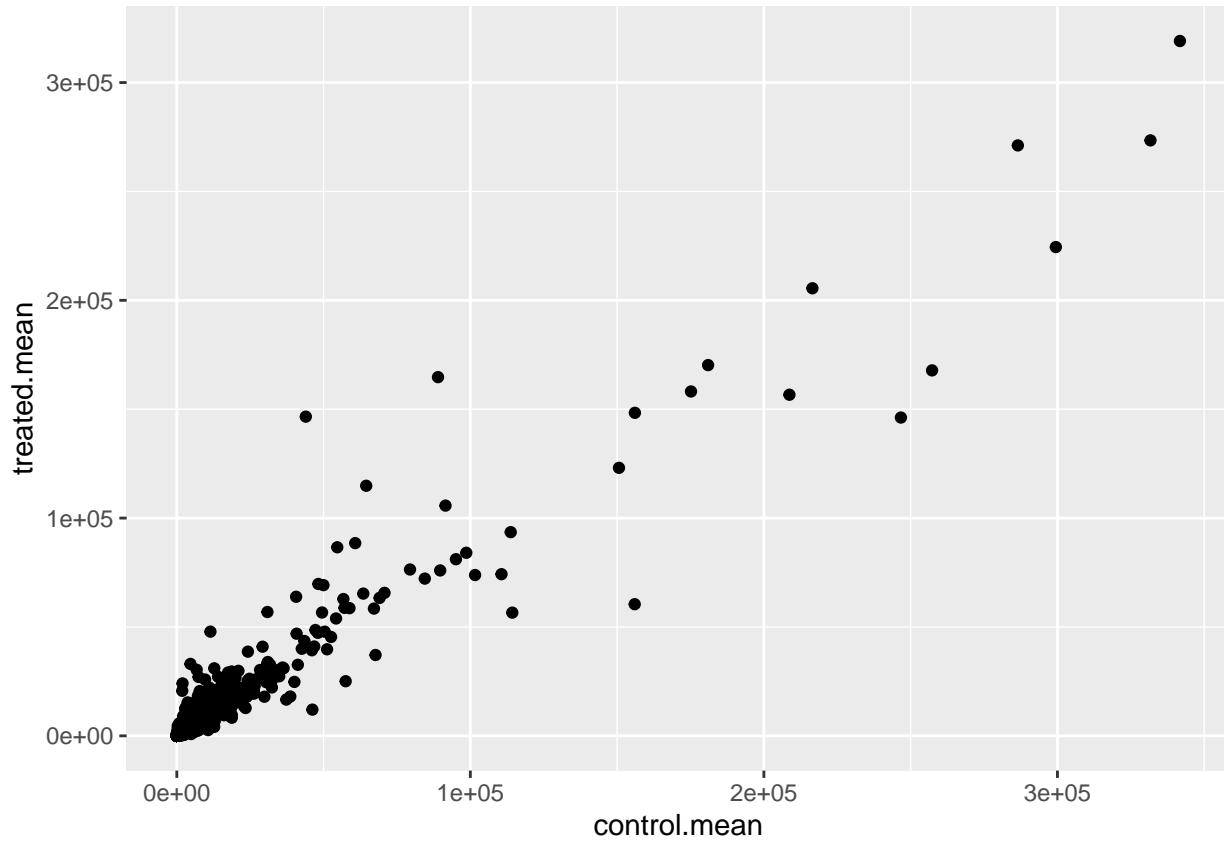
```
plot(meancounts)
```



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?

```
geom_point()
```

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



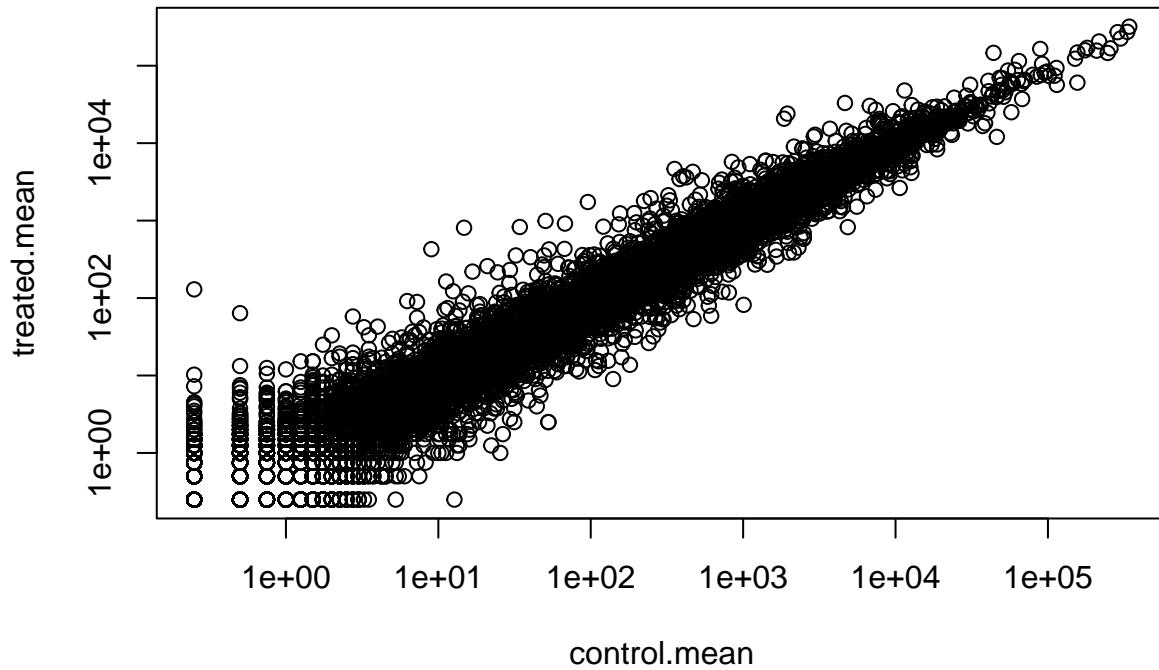
**Q6.** Try plotting both axes on a log scale. What is the argument to `plot()` that allows you to do this?

This would benefit from a log transform! Lets plot on a log scale.

```
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
```



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.

Warm-up

```
log2(40/20)
## [1] 1
```

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

```
meancounts$log2fc <- log2(meancounts[, "treated.mean"]/meancounts[, "control.mean"])
head(meancounts)

##           control.mean treated.mean      log2fc
## ENSG000000000003     900.75    658.00 -0.45303916
## ENSG000000000005      0.00     0.00       NaN
## ENSG000000000419    520.50    546.00  0.06900279
## ENSG000000000457    339.75    316.50 -0.10226805
## ENSG000000000460     97.25     78.75 -0.30441833
## ENSG000000000938      0.75     0.00      -Inf
```

Verifies where we have NaN values in our table.

```
head(meancounts[, 1:2]==0)

##           control.mean treated.mean
## ENSG000000000003      FALSE      FALSE
## ENSG000000000005      TRUE       TRUE
## ENSG000000000419      FALSE      FALSE
## ENSG000000000457      FALSE      FALSE
```

```
## ENSG00000000460      FALSE      FALSE
## ENSG00000000938      FALSE      TRUE
```

The `which()` function tells us the indices of TRUE entries in a logical vector.

```
which(c(T, T, F, T))
```

```
## [1] 1 2 4
```

However, it is not that useful in default mode on our type of multi column input.

```
which.mean <- which(meancounts[,1:2]==0)
head(which.mean)
```

```
## [1] 2 65 70 73 121 123
```

**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?

Instead, we can argue with `which()` function.

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

```
##           row col
## ENSG00000000005   2   1
## ENSG00000004848  65   1
## ENSG00000004948  70   1
## ENSG00000005001  73   1
## ENSG00000006059 121   1
## ENSG00000006071 123   1
```

Let's filter our data to remove NaN genes.

```
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
```

Breaking it down. Here we only care about the rows (if there is a zero in my column, I will exclude this row)

```
to.rm <- unique(sort(inds[, "row"]))
head(to.rm)
```

```
## [1] 2 6 65 70 73 81
```

## Double checking

```
# head(meancounts[to.rm,])  
head(meancounts[-to.rm,]) # to view only the data that do not have zeros.
```

```
## control.mean treated.mean log2fc  
## ENSG000000000003 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
```

## How many genes are remaining?

We now have 21817 genes remaining.

```
nrow(mycounts)
```

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

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

Which genes are upregulated/downregulated?

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

```
## [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?

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

```
## [1] 367
```

Which genes are upregulated? Round up.

```
round((sum(mycounts$log2fc > +2)/nrow(mycounts))*100,2)
```

```
## [1] 1.15
```

Which genes are downregulated? Round up.

```
round((sum(mycounts$log2fc < +2)/nrow(mycounts))*100,2)
```

```
## [1] 98.56
```

## 4. DESeq2 Analysis

First need to setup the DESeq input object.

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

## 
##   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},
## }
```

(1)a count matrix, (2)a metadata file, and (3)a formula indicating the design of the experiment.

```
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
dds <- DESeq(dds)

## estimating size factors
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
```

```

## final dispersion estimates
## fitting model and testing
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>
## ENSG000000000003 747.194195    -0.3507030  0.168246 -2.084470 0.0371175
## ENSG000000000005  0.000000      NA        NA        NA        NA
## ENSG000000000419 520.134160     0.2061078  0.101059  2.039475 0.0414026
## ENSG000000000457 322.664844     0.0245269  0.145145  0.168982 0.8658106
## ENSG000000000460 87.682625     -0.1471420  0.257007 -0.572521 0.5669691
## ENSG000000000938  0.319167     -1.7322890  3.493601 -0.495846 0.6200029
##           padj
##           <numeric>
## ENSG000000000003  0.163035
## ENSG000000000005  NA
## ENSG000000000419  0.176032
## ENSG000000000457  0.961694
## ENSG000000000460  0.815849
## ENSG000000000938  NA

```

We can summarize some basic tallies using the summary function.

```

summary(res)

##
## out of 25258 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)      : 1563, 6.2%
## LFC < 0 (down)    : 1188, 4.7%
## outliers [1]       : 142, 0.56%
## low counts [2]     : 9971, 39%
## (mean count < 10)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results

```

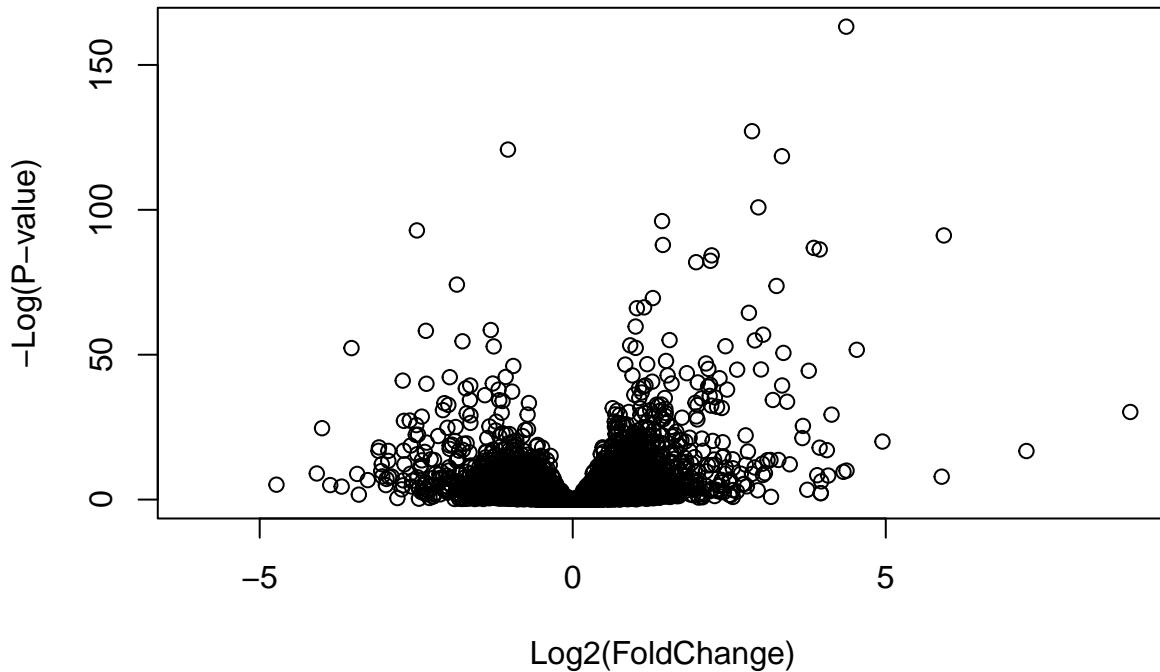
## 6. Data Visualization

Volcano plot to visualize proportion of genes. Convert to Log.

```

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

```

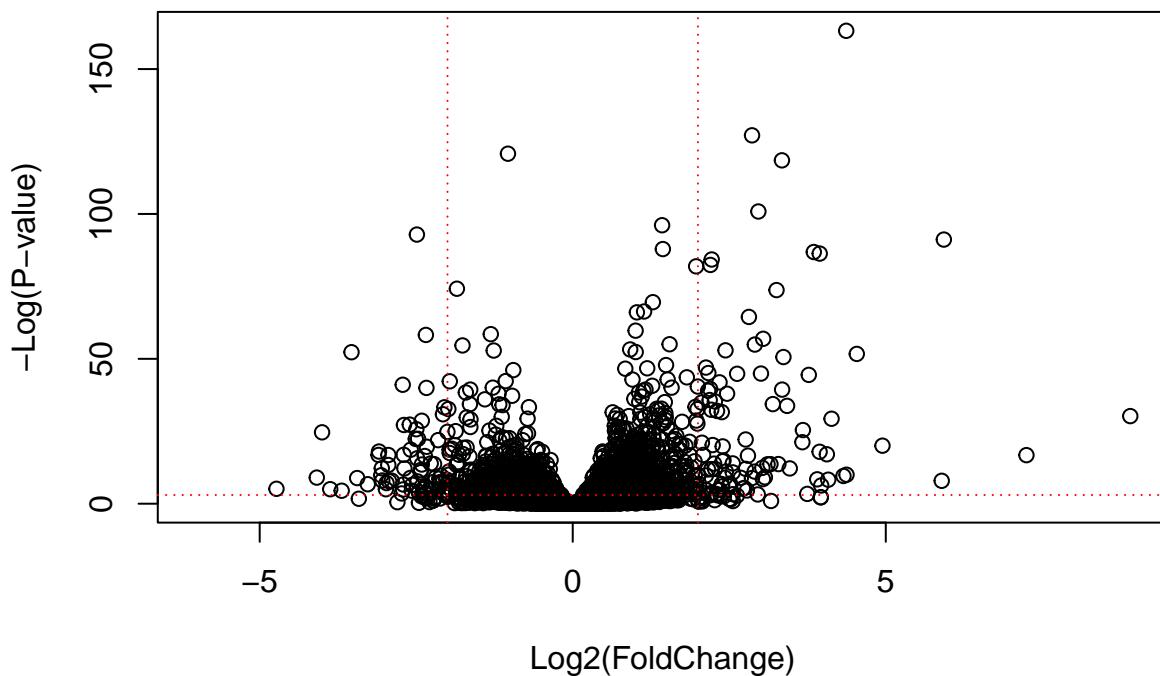


#Volcano summary figures are frequently used to highlight the proportion of genes that are both significant and differentially expressed.

### Adding cut-off vertical and horizontal lines

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

# Add some cut-off lines. We care about the points that are on either side of the lines.
abline(v=c(-2,2), col="red", lty=3)
abline(h=-log(0.05), col="red", lty=3)
```



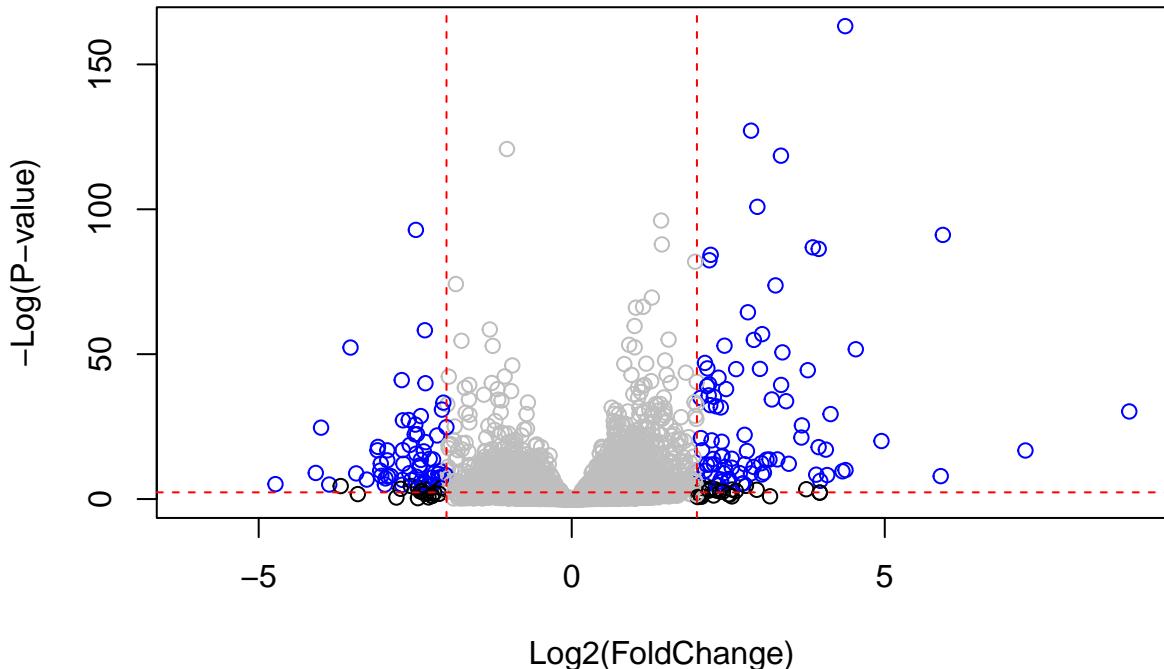
## Changing the color of our plots

```
# Setup our custom point color vector
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2 ] <- "black"

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="red", lty=2)
abline(h=-log(0.1), col="red", lty=2)
```



## 5. Adding annotations to our data.

```
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] "ACNUM"      "ALIAS"       "ENSEMBL"     "ENSEMLPROT"  "ENSEMLTRANS"
## [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 can use the `mapIds()` function to add individual columns to our results table. We provide the row names of our results table as a key, and specify that `keytype=ENSEMBL`

Here we map to “SYMBOL” the common gene name that the world understands and wants.

```
res$symbol <- mapIds(org.Hs.eg.db,
                      keys=row.names(res),           # Our gene names
                      keytype="ENSEMBL",             # The format of our gene names
                      column="SYMBOL",              # The new format we want to add
                      multiVals="first")
```

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

```
head(res$symbol)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457 ENSG00000000460
##      "TSPAN6"           "TNMD"          "DPM1"        "SCYL3"      "C1orf112"
## ENSG00000000938
##      "FGR"
```

```
head(res)
```

```
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 6 rows and 7 columns
##           baseMean log2FoldChange      lfcSE      stat      pvalue
##           <numeric>      <numeric> <numeric> <numeric> <numeric>
## ENSG00000000003 747.194195 -0.3507030  0.168246 -2.084470 0.0371175
## ENSG00000000005  0.000000      NA        NA        NA        NA
## ENSG00000000419 520.134160  0.2061078  0.101059  2.039475 0.0414026
## ENSG00000000457 322.664844  0.0245269  0.145145  0.168982 0.8658106
## ENSG00000000460 87.682625 -0.1471420  0.257007 -0.572521 0.5669691
## ENSG00000000938 0.319167 -1.7322890  3.493601 -0.495846 0.6200029
##           padj      symbol
##           <numeric> <character>
## ENSG00000000003 0.163035    TSPAN6
## ENSG00000000005  NA          TNMD
## ENSG00000000419 0.176032    DPM1
## ENSG00000000457 0.961694    SCYL3
## ENSG00000000460 0.815849    C1orf112
## ENSG00000000938  NA          FGR
```

You can arrange and view the results by the adjusted p-value

```
ord <- order( res$padj )
#View(res[ord,])
head(res[ord,])

## 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>
## ENSG00000152583    954.771      4.36836  0.2371268   18.4220 8.74490e-76
## ENSG00000179094    743.253      2.86389  0.1755693   16.3120 8.10784e-60
## ENSG00000116584   2277.913     -1.03470  0.0650984  -15.8944 6.92855e-57
## ENSG00000189221    2383.754      3.34154  0.2124058   15.7319 9.14433e-56
## ENSG00000120129    3440.704      2.96521  0.2036951   14.5571 5.26424e-48
## ENSG00000148175   13493.920      1.42717  0.1003890   14.2164 7.25128e-46
##           padj      symbol
##       <numeric> <character>
## ENSG00000152583 1.32441e-71    SPARCL1
## ENSG00000179094 6.13966e-56    PER1
## ENSG00000116584 3.49776e-53    ARHGEF2
## ENSG00000189221 3.46227e-52    MAOA
## ENSG00000120129 1.59454e-44    DUSP1
## ENSG00000148175 1.83034e-42    STOM
write.csv(res[ord,], "deseq.results.csv")
```

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.

```
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"
```

```

head(rownames(res))

## [1] "ENSG000000000003" "ENSG000000000005" "ENSG000000000419" "ENSG000000000457"
## [5] "ENSG000000000460" "ENSG000000000938"

Before we can use KEGG we need to get our gene identifiers in the correct format for KEGG, which is ENTREZ format in this case.

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$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

foldchanges <- res$log2FoldChange
head(foldchanges)

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

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)

## $names
## [1] "greater" "less"     "stats"
head(keggres$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

the pathview() function will add our genes to a KEGG pathway as colored entries:
pathview(gene.data=foldchanges, pathway.id="hsa05310")

## 'select()' returned 1:1 mapping between keys and columns
## Info: Working in directory /Users/jocelynolvera/Documents/phd_yr1_2021/bgg213/bgg213_Rstudio_git/b

```

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

