Class13

Georgina Canto-Encalada

Import countData and colData

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).

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

Have we peal at what they look like:

```
head(counts)
```

```
                SRR1039508 SRR1039509 SRR1039512 SRR1039513 SRR1039516
ENSG00000000003        723        486        904        445       1170
ENSG00000000005          0          0          0          0          0
ENSG00000000419        467        523        616        371        582
ENSG00000000457        347        258        364        237        318
ENSG00000000460         96         81         73         66        118
ENSG00000000938          0          0          1          0          2
                SRR1039517 SRR1039520 SRR1039521
ENSG00000000003       1097        806        604
ENSG00000000005          0          0          0
ENSG00000000419        781        417        509
ENSG00000000457        447        330        324
ENSG00000000460         94        102         74
ENSG00000000938          0          0          0
```

```
head(metadata)
```

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

Q1. How many genes are in this dataset?

ANS: 38694

```
nrow(counts)
```

```
[1] 38694
```

Q. How many total samples?

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

```
control treated
      4       4
```

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

ANS: 4

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

```
[1] 4
```

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

Let's make sure our `metadata` matches the `counts`

```
if(all(metadata$id == colnames(counts) ))
  print("They match")
```

```
[1] "They match"
```

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)

We have a plan - first extract out the control samples (i.e. columns) - next calculate the row wise means (i.e. mean counts for each gene)

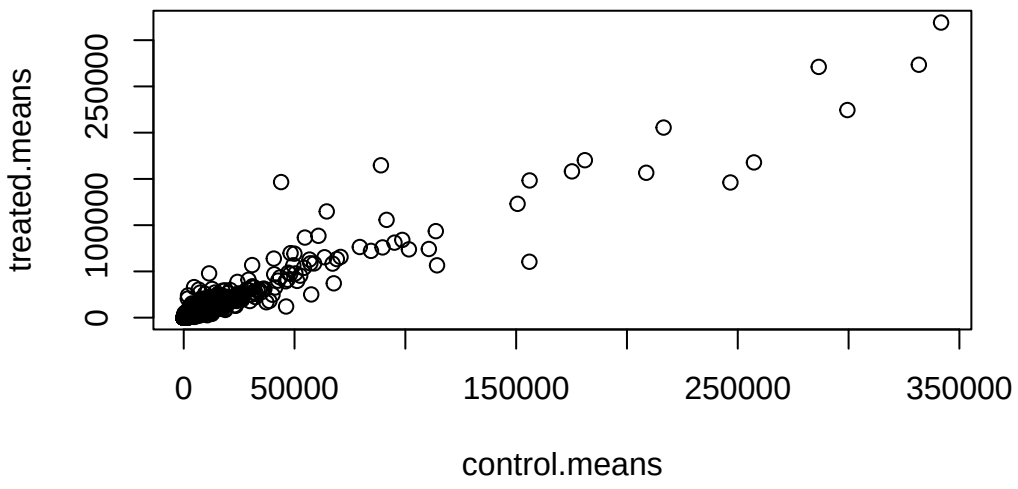
```
# First where are the "control" samples
control.counts <- counts[,metadata$dex=="control"]
control.means<-apply(control.counts,1,mean)

# Then where are the "treated" samples
treated.counts <- counts[,metadata$dex=="treated"]
treated.means<-apply(treated.counts,1,mean)
```

Store these together for ease of book-keeping

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

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.

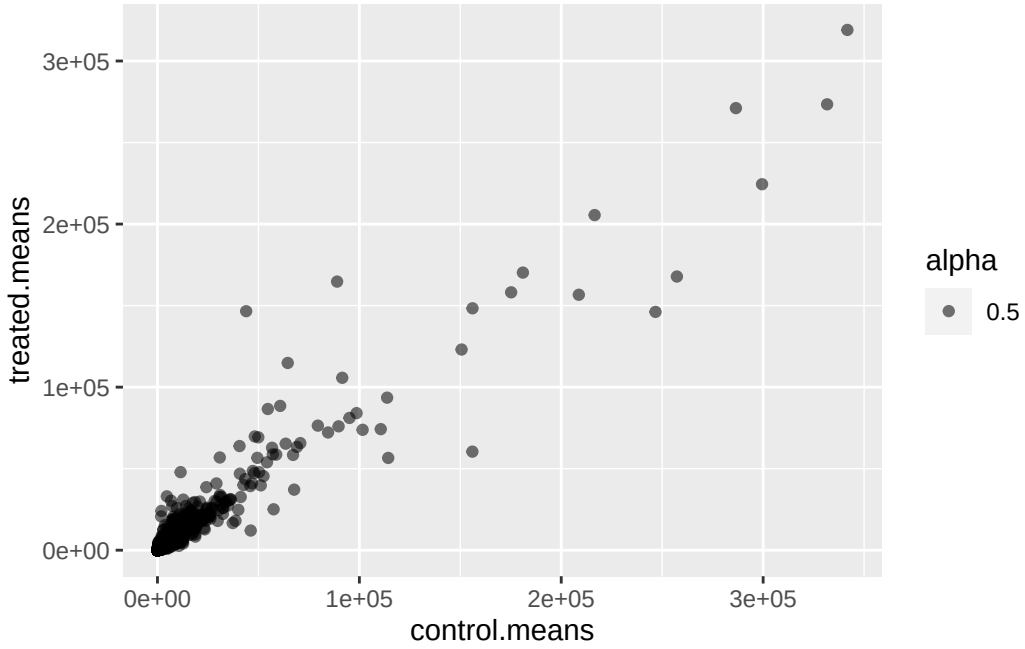
```
plot(meancounts)
```



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?

ANS: geom_point()

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




```
head(meancounts)
```

```
                control.means treated.means
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
```

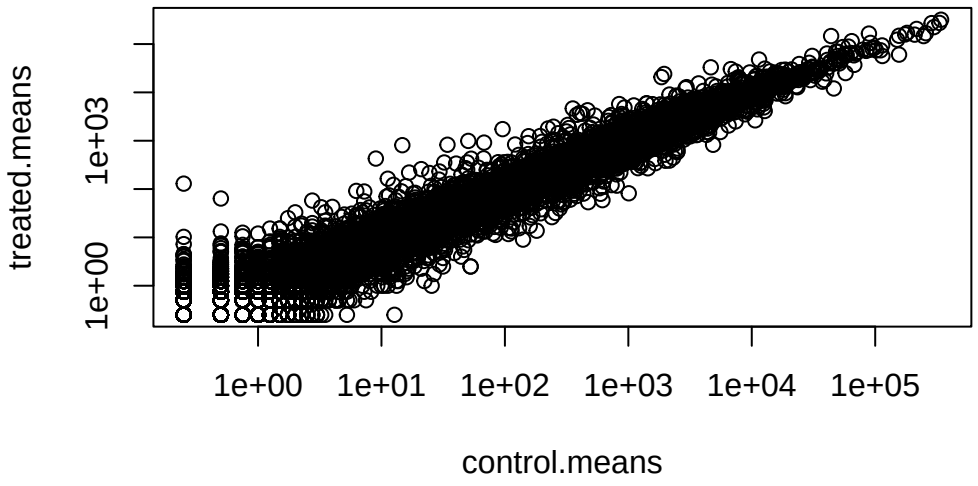
Q6. Try plotting both axes on a log scale. What is the argument to plot() that allows you to do this?

We have such skewed data over a wide range and we ultimately care about orders of magnitude change anyway so lets log transform

```
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 look at differences here via a little fraction. Divide one condition by another. E.g. Treated/control

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

```
[1] 0
```

```
log2(40/20)
```

```
[1] 1
```

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

```
[1] -1
```

Log2 fold change for our treated/control

```
meancounts$log2fc<-log2(meancounts$treated.mean/meancounts$control.mean)
```

There are some weird results in here because of our zero count genes. We can't say anything about these genes anyway so it is common practice to remove them/filter them aout

How can we identify zero count genes in our `meancounts`

- First approach

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

to.rm <- unique(zero.vals[,1])
mycounts <- meancounts[-to.rm,]
head(mycounts)
```

```
                control.means treated.means      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?

ANS: Instead of returning a single vector of indices, it returns a matrix of indices where each row corresponds to the indices of a TRUE element, with the first column representing row indices and the second column representing column indices. The purpose of taking the first column of this output (zero.vals[,1]) is to identify all rows in meancounts that contain zeroes in either of the first two columns. Calling unique() on this vector of row indices is necessary because there might be multiple zeroes in different columns of the same row. Without calling unique(), you could end up with duplicate row indices in your vector

- Second approach

```
zero.sums <- rowSums(meancounts[,1:2]==0)
to.rm.ind <- zero.sums >0
mycounts <- meancounts[!to.rm.ind,]
```

A common threshold for calling something "differentially expressed" is a log2 fold-change value of +2 or -2

> Q9. Can you determine how many up regulated genes we have at the greater than 2 fc level?

ANS: 250

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

```
[1] 250
```

> Q8 Can you determine how many down regulated genes we have at the greater than 2 fc level?

ANS: 367

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

```
[1] 367
```

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

ANS: I do not since we are not considering the p-value

DESeq2 Analysis

Let's do this the right way with DESeq2

```
#/ message: false
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, 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 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 object is masked from 'package:grDevices':

    windows

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
```

DESeq2 wants our data in a very particular object called a deseq object and we can set this up with functions from within the DESeq2 package

```
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 analysis function is called `DESeq()` and we can now pass it our setup input object with all the data we need

```
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 function `results()`

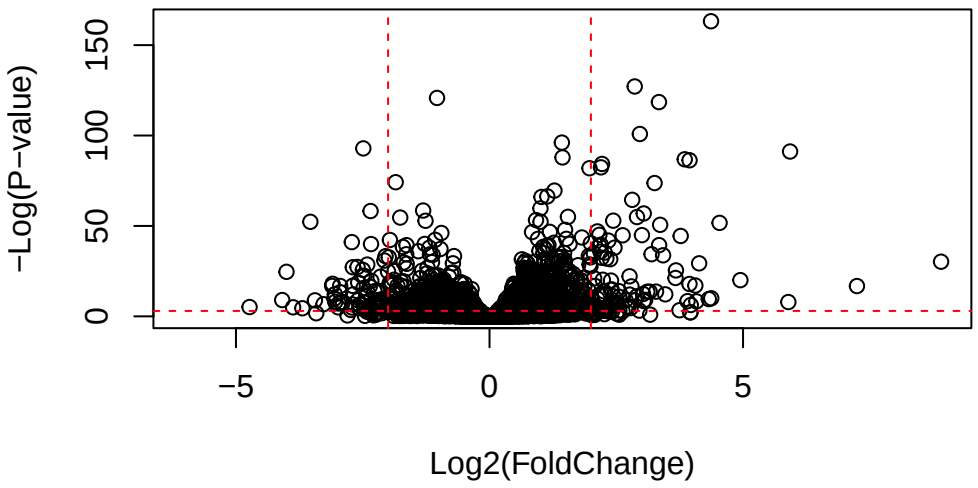
```
res <- results(dds)
head(res)
```

```
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
```

Volcano plot

A common visualization for this type of data is called a volcano plot. It is plot of log2 fold change against p-value

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



Save our results thus far

```
write.csv(res,file='myresults.csv')
```

Adding annotation data

Our result table so far only contains the Ensembl gene IDs. However, we want to be able to make sense of these genes. Minimally

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

```
res$symbol <- mapIds(org.Hs.eg.db,
                     keys=row.names(res), # Our genenames
                     keytype="ENSEMBL",        # The format of our genenames
                     column="ENTREZID",          # The new format we want to add
                     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 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        7105
ENSG00000000005        NA       64102
ENSG00000000419  0.176032        8813
ENSG00000000457  0.961694       57147
ENSG00000000460  0.815849       55732
ENSG00000000938        NA        2268
```

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

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

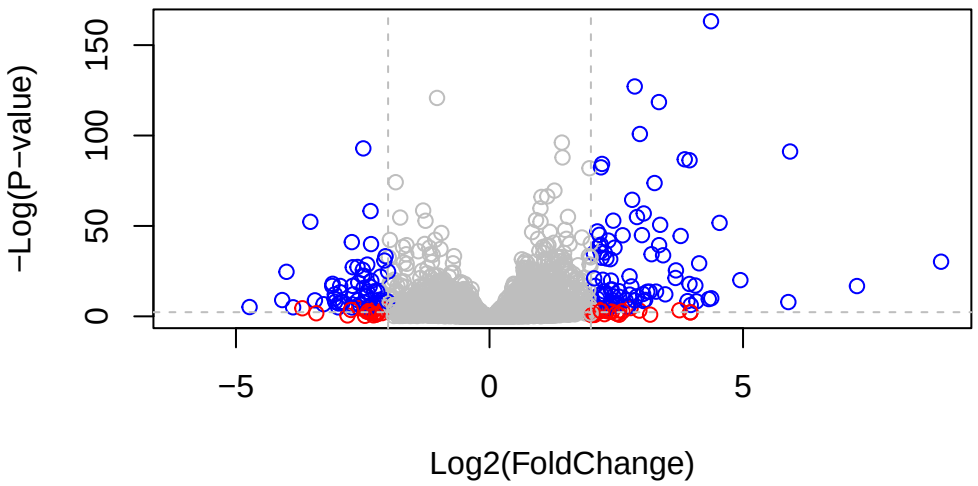
```
log2 fold change (MLE): dex treated vs control
Wald test p-value: dex treated vs control
DataFrame with 6 rows and 7 columns
                 baseMean log2FoldChange     lfcSE      stat      pvalue
                <numeric>      <numeric> <numeric> <numeric>   <numeric>
ENSG00000152583   954.771        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        8404
ENSG00000179094 6.13966e-56        5187
ENSG00000116584 3.49776e-53        9181
ENSG00000189221 3.46227e-52        4128
ENSG00000120129 1.59454e-44        1843
ENSG00000148175 1.83034e-42        2040
```

Finally, let's write out the ordered significant results with annotations.

```
write.csv(res[ord,], "deseq_results.csv")
```

Volcano plots

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

inds <- (res$padj < 0.01) & (abs(res$log2FoldChange) > 2 )
mycols[ inds ] <- "blue"

# Volcano plot with custom colors
plot( res$log2FoldChange,  -log(res$padj),
 col=mycols, ylab="-Log(P-value)", xlab="Log2(FoldChange)" )

# Cut-off lines
abline(v=c(-2,2), col="gray", lty=2)
abline(h=-log(0.1), col="gray", lty=2)
```


−Log(P−value)
0
50
100
150
−5
0
5
Log2(FoldChange)