

Class 13

Karleen Guerrero A16791042

This week we are looking at differential expression analysis.

import/read the data from Himes et al.

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

Let's have a peak at this data

```
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

Sanity check on correspondence of counts and metadata

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

```
[1] TRUE
```

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

```
[1] FALSE
```

Q1. How many genes are in this dataset?

There are 38694 `nrow(counts)` genes

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

```
n.control <- sum(metadata$dex == 'control')
```

There are '4 n.control' control cell lines in this data set.

Extract and summarize the control samples

To find out where the control samples are we need the metadata

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

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

```
ENSG000000000003 ENSG000000000005 ENSG000000000419 ENSG000000000457 ENSG000000000460
          900.75           0.00           520.50           339.75           97.25
ENSG0000000000938
          0.75
```

Extract and summarize the treated (i.e. drug) samples

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

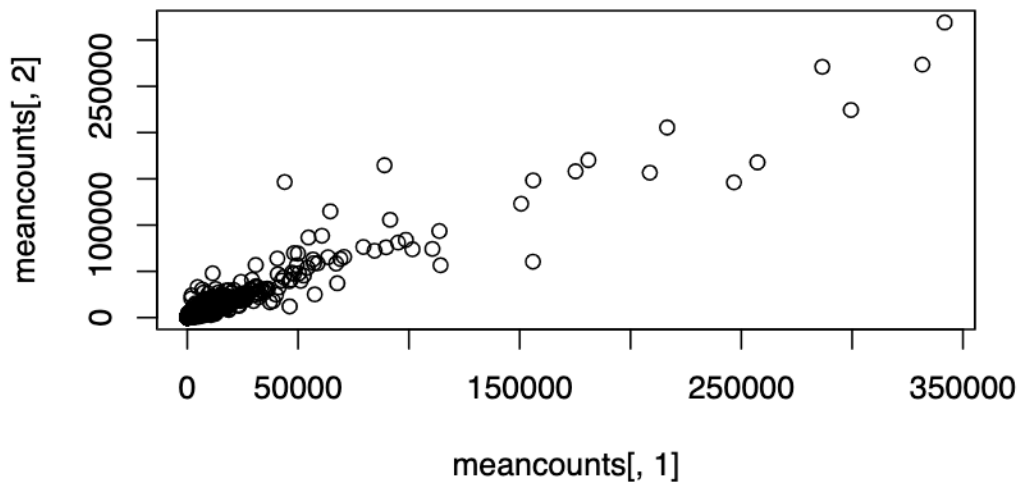
Store these results together in a new data frame called 'meancounts'

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

Let's make a plot to explore the results a little..

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.

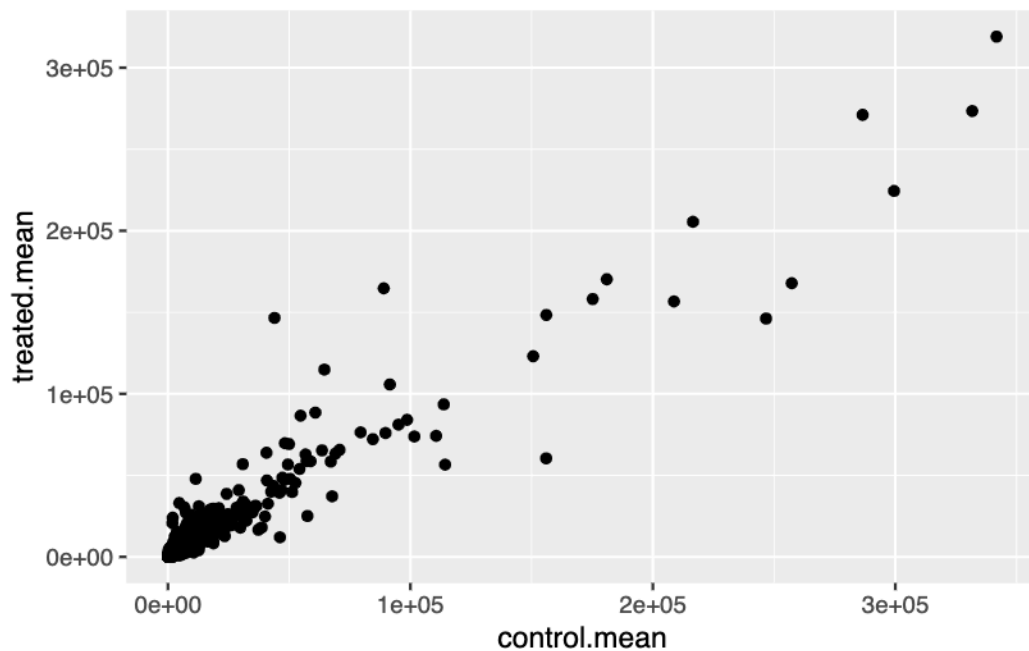
```
plot(meancounts[,1], meancounts[,2])
```



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()
```



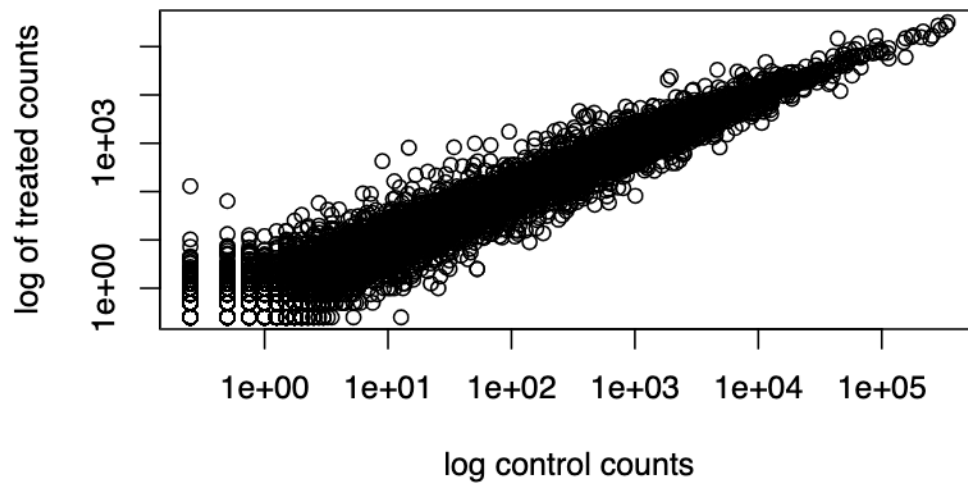
We will make a log-log plot to draw out this skewed data and see what is going on.

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

```
plot(meancounts[,1], meancounts[,2], log="xy",
     xlab="log control counts",
     ylab="log of treated counts")
```

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 often log2 transformations when dealing with this sort of data.

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

```
[1] 0
```

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

```
[1] 1
```

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

```
[1] -1
```

```
log2(80/20)
```

```
[1] 2
```

This log2 transformation has this nice property where if there is no change the log2 value will be zero and if it double the log2 value will be 1 and if halved it will be -1.

So lets add a log2 fold change column to our results so far

```
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
ENSG0000000000419	520.50	546.00	0.06900279
ENSG0000000000457	339.75	316.50	-0.10226805
ENSG0000000000460	97.25	78.75	-0.30441833
ENSG0000000000938	0.75	0.00	-Inf

We need to get rid of zero count genes that we can not say anything about

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 will return 'which()' back to the row and column. The 'unique()' function is needed to notice the zero entries and not repeat a row.

```
zero.values <- which( meancounts[,1:2]==0, arr.ind=TRUE )
to.rm <- unique(zero.values[,1])
mycounts <- meancounts[-to.rm,]
```

```
head(mycounts)
```

	control.mean	treated.mean	log2fc
ENSG000000000003	900.75	658.00	-0.45303916
ENSG0000000000419	520.50	546.00	0.06900279
ENSG0000000000457	339.75	316.50	-0.10226805
ENSG0000000000460	97.25	78.75	-0.30441833
ENSG0000000000971	5219.00	6687.50	0.35769358
ENSG000000001036	2327.00	1785.75	-0.38194109

How many genes are remaining?

```
nrow(mycounts)
```

```
[1] 21817
```

Use fold change to see up and down regulated genes

A common threshold used for calling something differentially expressed is a $\log_2(\text{FoldChange})$ of greater than 2 or less than -2. Let's filter the dataset both ways to see how many genes are up or down-regulated.

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

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

```
[1] 250
```

and down-regulated

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(mycounts$log2fc < -2)
```

```
[1] 367
```

Q10. Do we trust these results?

Well not fully because we don't yet know if these changes are significant...

DESeq2 analysis

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.

```
# load up DESeq2
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, saveRDS, setdiff,
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

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

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

estimating size factors

estimating dispersions

gene-wise dispersion estimates

mean-dispersion relationship

final dispersion estimates

fitting model and testing

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

log2 fold change (MLE): dex treated vs control

Wald test p-value: dex treated vs control

DataFrame with 38694 rows and 6 columns

	baseMean	log2FoldChange	lfcSE	stat	pvalue
	<numeric>	<numeric>	<numeric>	<numeric>	<numeric>
ENSG000000000003	747.1942	-0.3507030	0.168246	-2.084470	0.0371175
ENSG000000000005	0.0000	NA	NA	NA	NA
ENSG000000000419	520.1342	0.2061078	0.101059	2.039475	0.0414026
ENSG000000000457	322.6648	0.0245269	0.145145	0.168982	0.8658106
ENSG000000000460	87.6826	-0.1471420	0.257007	-0.572521	0.5669691
...
ENSG00000283115	0.000000	NA	NA	NA	NA
ENSG00000283116	0.000000	NA	NA	NA	NA
ENSG00000283119	0.000000	NA	NA	NA	NA
ENSG00000283120	0.974916	-0.668258	1.69456	-0.394354	0.693319
ENSG00000283123	0.000000	NA	NA	NA	NA
	padj				
	<numeric>				
ENSG000000000003	0.163035				
ENSG000000000005	NA				
ENSG000000000419	0.176032				
ENSG000000000457	0.961694				
ENSG000000000460	0.815849				
...	...				
ENSG00000283115	NA				
ENSG00000283116	NA				
ENSG00000283119	NA				
ENSG00000283120	NA				
ENSG00000283123	NA				

We can get some basic summary tallies using the 'summary()' function

```
summary(res, alpha=0.05)
```

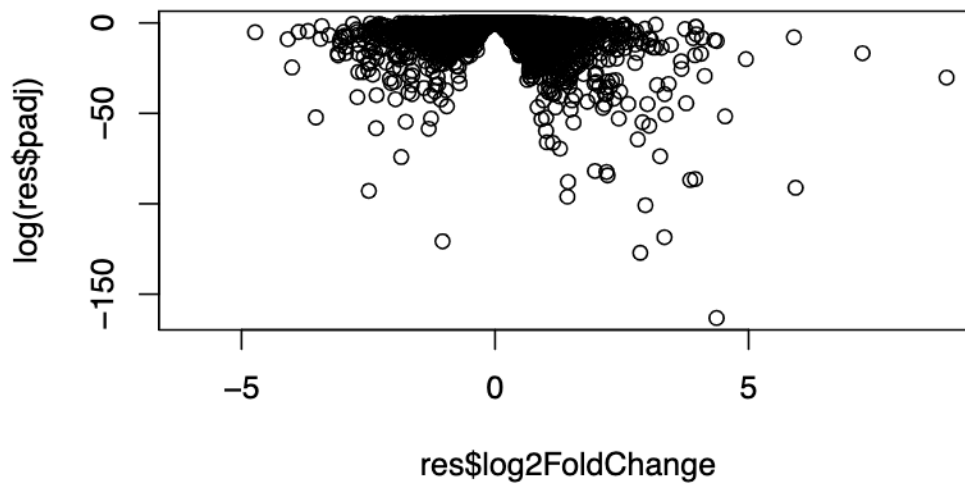
```
out of 25258 with nonzero total read count
adjusted p-value < 0.05
LFC > 0 (up)      : 1242, 4.9%
LFC < 0 (down)    : 939, 3.7%
outliers [1]      : 142, 0.56%
low counts [2]    : 9971, 39%
(mean count < 10)
```

```
[1] see 'cooksCutoff' argument of ?results  
[2] see 'independentFiltering' argument of ?results
```

Volcano plot

Make a summary plot of our results.

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



```
log(0.1)
```

```
[1] -2.302585
```

```
log(0.05)
```

```
[1] -2.995732
```

Finish for today by saving our results

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

DAY2

```
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
ENSG0000000000419	520.134160	0.2061078	0.101059	2.039475	0.0414026
ENSG0000000000457	322.664844	0.0245269	0.145145	0.168982	0.8658106
ENSG0000000000460	87.682625	-0.1471420	0.257007	-0.572521	0.5669691
ENSG0000000000938	0.319167	-1.7322890	3.493601	-0.495846	0.6200029
	padj				
	<numeric>				
ENSG000000000003	0.163035				
ENSG000000000005	NA				
ENSG0000000000419	0.176032				
ENSG0000000000457	0.961694				
ENSG0000000000460	0.815849				
ENSG0000000000938	NA				

I need to translate our gene identifiers “ENSG00000...” into gene names that the rest of the world can understand.

To this “annotation” I will use the “AnnotationDbi” package. I can install this with ‘BiocManager::install()’

```
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"				

I will use the 'mapIds()' function to "map" identifiers to those from different databases. I will go between "ENSEMBL" and "SYMBOL" (and then after "GENENAME").

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

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

```
#head(res)
```

Add "GENENAME"

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

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

And add ENTREZID

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

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

Save our annotated results object.

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

##Pathway Analysis

Now that we have our results with added annotation we can do some pathway mapping

Let's use the **gage** package to look for KEGG pathways in our results (genes of interest). I will use the **pathview** package to draw little pathway figures.

```
library(pathview)
```

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

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

```
#####
```

```
library(gage)
```

```
library(gageData)
```

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

```
# Examine the first 3 pathways
```

```
# head(kegg.sets.hs, 3)
```

What **gage** wants as input is not my big table/data.frame of results. It just want a “vector of importance”. For RNAseq data like we have this is our Log2FC values...

```
oldchanges = res$log2FoldChange  
names(oldchanges) = res$entrez  
head(oldchanges)
```

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

Now lets run the gage pathway analysis...

```
#get the results
keggres= gage(oldchanges, gsets=kegg.sets.hs)
```

What is in this 'keggres' object?

```
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

Let's use the pathview package to look at one of these highlighted KEGG pathways with our genes highlighted. "hsa05310 Asthma"

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

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

Info: Working in directory /Users/gabygonzalez/Downloads/Class 13 Dseq

Info: Writing image file hsa05310.pathview.png

