

Class13: RNASeq analysis with DESeq2

Sawyer (PID: A16335277)

In today's class we will explore and analyse data 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).

Data Import

We have two input files, so-called "count data" and "col data".

```
# Complete the missing code
counts <- read.csv("airway_scaledcounts.csv", row.names=1)
metadata <- read.csv("airway_metadata.csv")
```

Take a look at head of each

```
head(counts)
```

	SRR1039508	SRR1039509	SRR1039512	SRR1039513	SRR1039516
ENSG000000000003	723	486	904	445	1170
ENSG000000000005	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		
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
```

Data Explore

Q1. How many genes are in this dataset

```
nrow(counts)
```

```
[1] 38694
```

There are 38694 genes.

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

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

```
[1] 4
```

We have 4 control cell lines.

Toy differential gene expression

Time to do some analysis.

We have 4 control and 4 treated samples/experiments/columns

Make sure the metadata id column matches the columns in our count data.

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

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

To check that all elements of a vector are TRUE we can use the `all()` function.

```
all(c(T,T,F))  
  
[1] FALSE  
  
all(colnames(counts) == metadata$id)  
  
[1] TRUE
```

To start I will calculate the `control.mean` and `treated.mean` values and compare them.

- Identify and extract the `control` only columns
- Determine te mean value for each gene (i.e., row)
- Do the same for `treated`.

Lab worksheet approach:

```
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  
ENSG00000000938  
0.75
```

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

The 4 is specific to the number of columns we are averaging over so it may not be robust to other datasets. The below approach using `apply` is much more helpful and generalizable.

```
control inds <- metadata$id[metadata$dex == 'control']  
control.counts <- counts[, control.inds]  
control.mean <- apply(control.counts, 1, mean)
```

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`)

Now do the same for the treated samples to get `treated.mean` ...

```
treated inds <- metadata$id[metadata$dex == 'treated']
treated counts <- counts[,treated inds]
treated mean <- apply(treated counts, 1, mean)
```

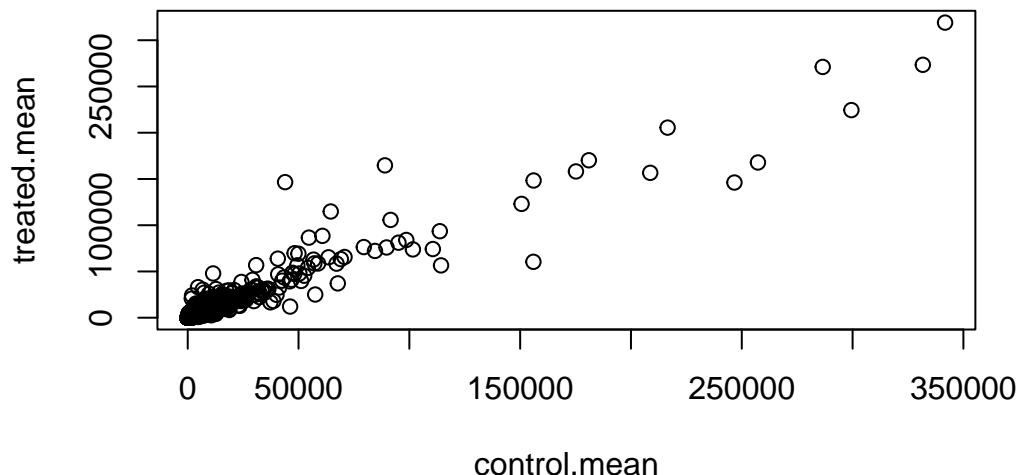
Let's store these together for ease of book-keeping

```
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.

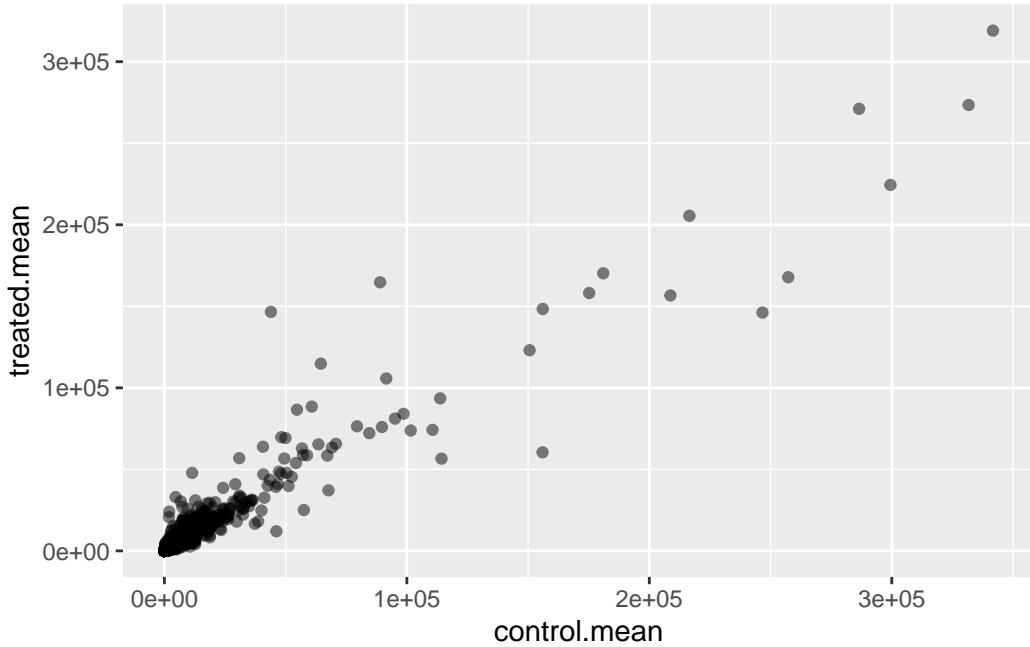
Have a quick view of this data:

```
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(x=control.mean, y=treated.mean) + geom_point(alpha=0.5)
```



Use `geom_point()`

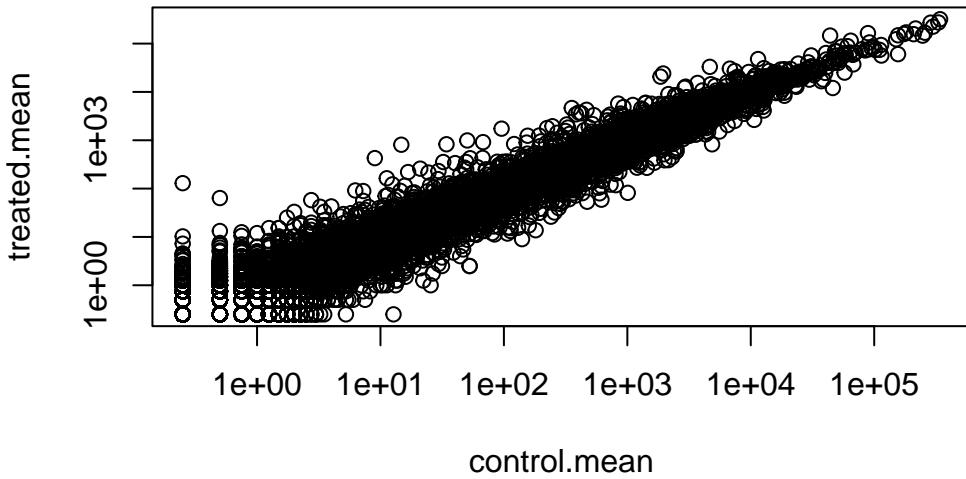
Q6. Try plotting both axes on a log scale. What is the argument to `plot()` that allows you to do this?

This data is screaming at us to log transform as it is so heavily skewed and over such a wide range.

```
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



I want to compare the treated and the control values here and we will use fold change in log₂ units to do this. $\log_2(\text{Treated}/\text{Control})$

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

No difference

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

```
[1] 0
```

```
log2(40/10)
```

```
[1] 2
```

A common rule of thumb cut-off for calling a gene “differentially expressed” is a log₂ fold-change value of either $> +2$ or < -2 for “up regulated” and “down regulated” respectively.

```
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

We first need to remove zero count genes - as we can't say anything about these genes anyway and their division of log values are messing things up (divide by zero) or the -infinity log problem.

```
sum(meancounts$log2fc > +2, na.rm=T)

[1] 1846

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
ENSG000000000003	900.75	658.00	-0.45303916
ENSG000000000419	520.50	546.00	0.06900279
ENSG000000000457	339.75	316.50	-0.10226805
ENSG000000000460	97.25	78.75	-0.30441833
ENSG000000000971	5219.00	6687.50	0.35769358
ENSG000000001036	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?

The arr.ind ensures the function returns array indices when the input argument, x, is an array.

The first column of the output corresponds to row indices with zero values. We want all distinct indices to be removed from meancounts, hence the unique function.

In class, we used a different approach.

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

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(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 cannot entirely trust these results because we have not proven statistical significance using statistical tests.

DESeq analysis

Let's do this properly with the help of the DESeq2 package

```
library(DESeq2)
```

We have to use a specific data object for working with DESeq.

```
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
```

Run our main analysis with the `DESeq()` function

```

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 our `dds` object we can use the `DESeq` function called `results()`:

```

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>
ENSG00000000003   747.1942    -0.3507030  0.168246 -2.084470  0.0371175
ENSG00000000005    0.0000       NA        NA        NA        NA
ENSG00000000419   520.1342    0.2061078  0.101059  2.039475  0.0414026
ENSG00000000457   322.6648    0.0245269  0.145145  0.168982  0.8658106
ENSG00000000460   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>
ENSG00000000003   0.163035
ENSG00000000005       NA

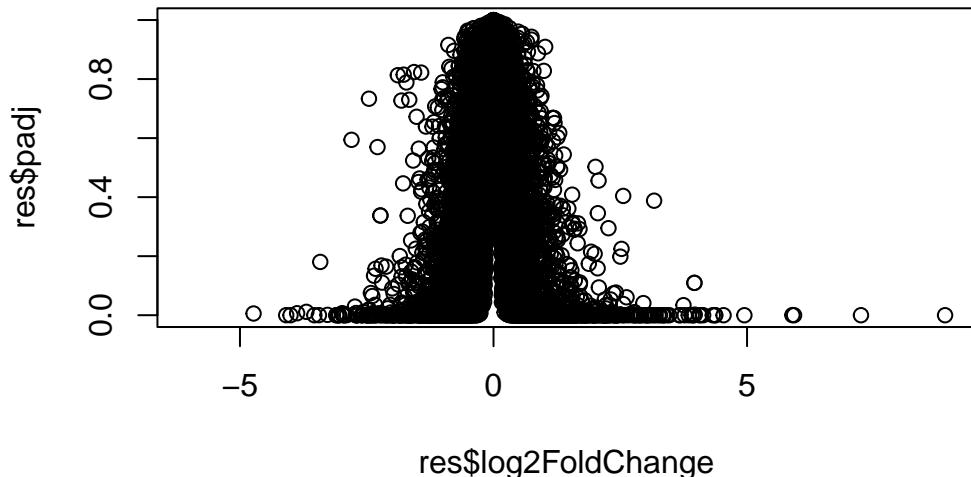
```

```
ENSG00000000419  0.176032
ENSG00000000457  0.961694
ENSG00000000460  0.815849
...
ENSG00000283115  NA
ENSG00000283116  NA
ENSG00000283119  NA
ENSG00000283120  NA
ENSG00000283123  NA
```

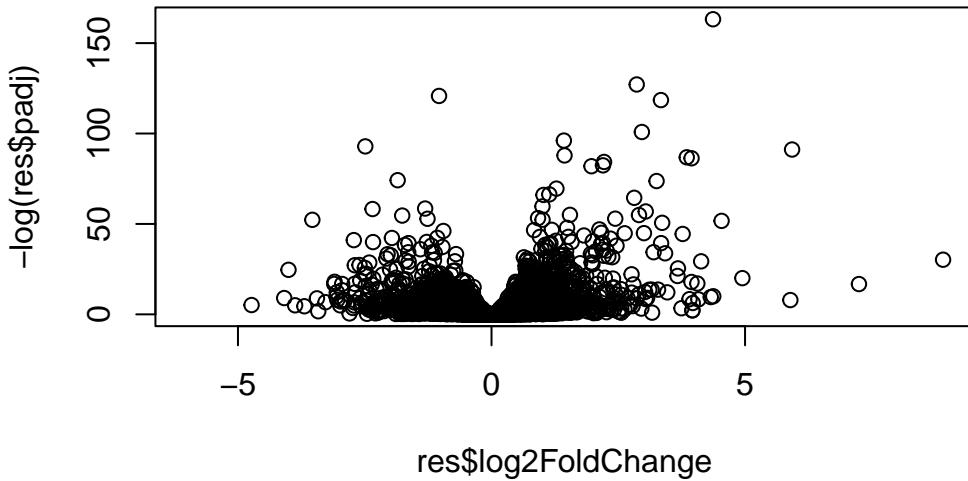
Volcano Plot

A very common and useful summary results figure from this type of analysis is called a volcano plot - a plot of log2FC vs P-value. We use the `padj` the adjusted P-value for multiple testing.

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



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

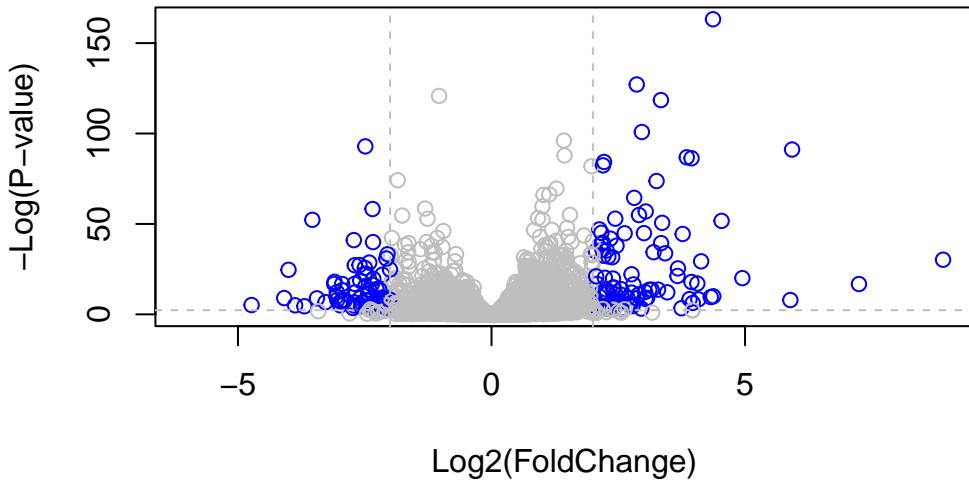


Add some color and nice labels for this plot (as done in class).

```
# Setup our custom point color vector
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2 ] <- "blue"
mycols[ res$padj > 0.05] <- "gray"

# 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)
```



Add annotation data

We will use one of Bioconductor's main annotation packages to help with mapping between various ID schemes. Here we

```
# head(res)

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

library("AnnotationDbi")
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"

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

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

res$entrez <- 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

And gene name.

res$genename <- mapIds(org.Hs.eg.db,
                      keys=row.names(res), # Our genenames
                      keytype="ENSEMBL",      # The format of our genenames
                      column="GENENAME",       # 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 9 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	symbol	entrez		genename	
	<numeric>	<character>	<character>		<character>	
ENSG000000000003	0.163035	TSPAN6	7105		tetraspanin 6	
ENSG000000000005	NA	TNMD	64102		tenomodulin	
ENSG000000000419	0.176032	DPM1	8813	dolichyl-phosphate m..		
ENSG000000000457	0.961694	SCYL3	57147	SCY1 like pseudokina..		
ENSG000000000460	0.815849	FIRRM	55732	FIGNL1 interacting r..		
ENSG000000000938	NA	FGR	2268	FGR proto-oncogene, ..		

Pathway analysis

Now that I have added the necessary annotation data I can talk to different databases that use these IDs.

We will use the `gage` package to do geneset analysis (a.k.a. pathway analysis, geneset enrichment, overlap analysis)

```
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)
```

We will use KEGG first ()

```
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"
```

The main `gage()` function requires a named vector of fold changes, where the names of the values are the Entrez gene IDs.

```
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

Run the analysis

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

Let's look at what is in our results here

```
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

I can now use the returned pathway IDs from KEGG as input to the `pathview` package to make pathway figures with our DEGs highlighted.

```
# A different PDF based output of the same data
pathview(gene.data=foldchanges, pathway.id="hsa05310")
```

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

Info: Working in directory C:/Users/soysa/Documents/BIMM 143/class13

Info: Writing image file hsa05310.pathview.png

