Transcriptomics and the analysis of RNA-seq data

Joshua Cheung

02/22/2022

1. Bioconductor and DESeq2

We first install the core Bioconductor packages into the R console.

```
# We input the following commands into the console.
# install.packages("BiocManager")
# BiocManager::install()
```

Then we install the DESeq2 bioconductor package below.

```
# We input the following commands into the console.
# BiocManager::install("DESeq2")
```

Now we recall both these packages using the library function.

```
# We input the following commands into the console.
# library(BiocManager)
# library(DESeq2)
```

2. Import countData and colData

The data for this 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).

We read 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 we preview the first 6 rows of both counts and metadata.

```
# We first examine the first 6 rows of 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
## ENSG00000000003       1097        806        604
## ENSG00000000005          0          0          0
## ENSG00000000419        781        417        509
## ENSG00000000457        447        330        324
## ENSG00000000460         94        102         74
## ENSG00000000938          0          0          0
```

```
# We also examine the first 5 rows of metadata.
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
```

We check on the correspondence of counts and metadata.

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

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

So we have good correspondence of the counts and metadata.

Q1. How many genes are in this dataset?

```
nrow(counts)
```

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

We see that there are 38694 genes in this dataset.

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

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

We see that there are 4 'control' cell lines.

3. Toy differential gene expression

Note that in the metadata object, the control samples are SRR1039508, SRR1039512, SRR1039516, and SRR1039520. So we first use R to first find the sample id for those labeled control. Then calculate the mean counts per gene across these samples:

```
control <- metadata[metadata[,"dex"]=="control",]
control.counts <- counts[ ,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
```

An alternative way to do this same thing using the dplyr package from the tidyverse is shown below.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

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

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

```
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
```

We see that we get the same results using both methods.

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

For each method we can make the code more robust by using the rowMeans() instead of rowSums(). Also for each method the code divides rowsums() by 4 which is not very robust if we were to add more samples or experiments, thus we can make this more robust by finding the number of samples from the code itself rather than hard coding a single integer value.

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)

For this question we can modify the code we used earlier to perform the procedure for the treated samples as follows:

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

We will then combine the meancount data for bookkeeping purposes.

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

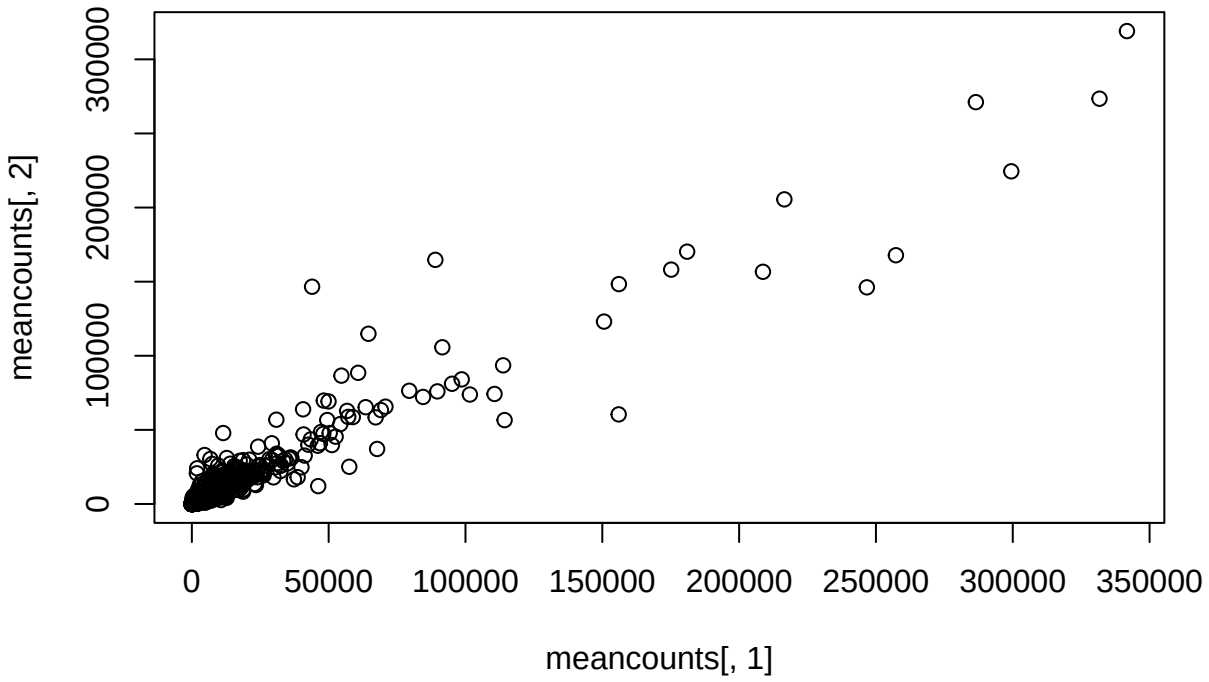
We note that directly comparing the raw counts is going to problematic if we just happened to sequence one group at a higher depth than another. Later on we'll do this analysis properly, normalizing by sequencing depth per sample using a better approach. But for now, colSums() the data to show the sum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

```
colSums(meancounts)
```

```
## control.mean treated.mean
##     23005324     22196524
```

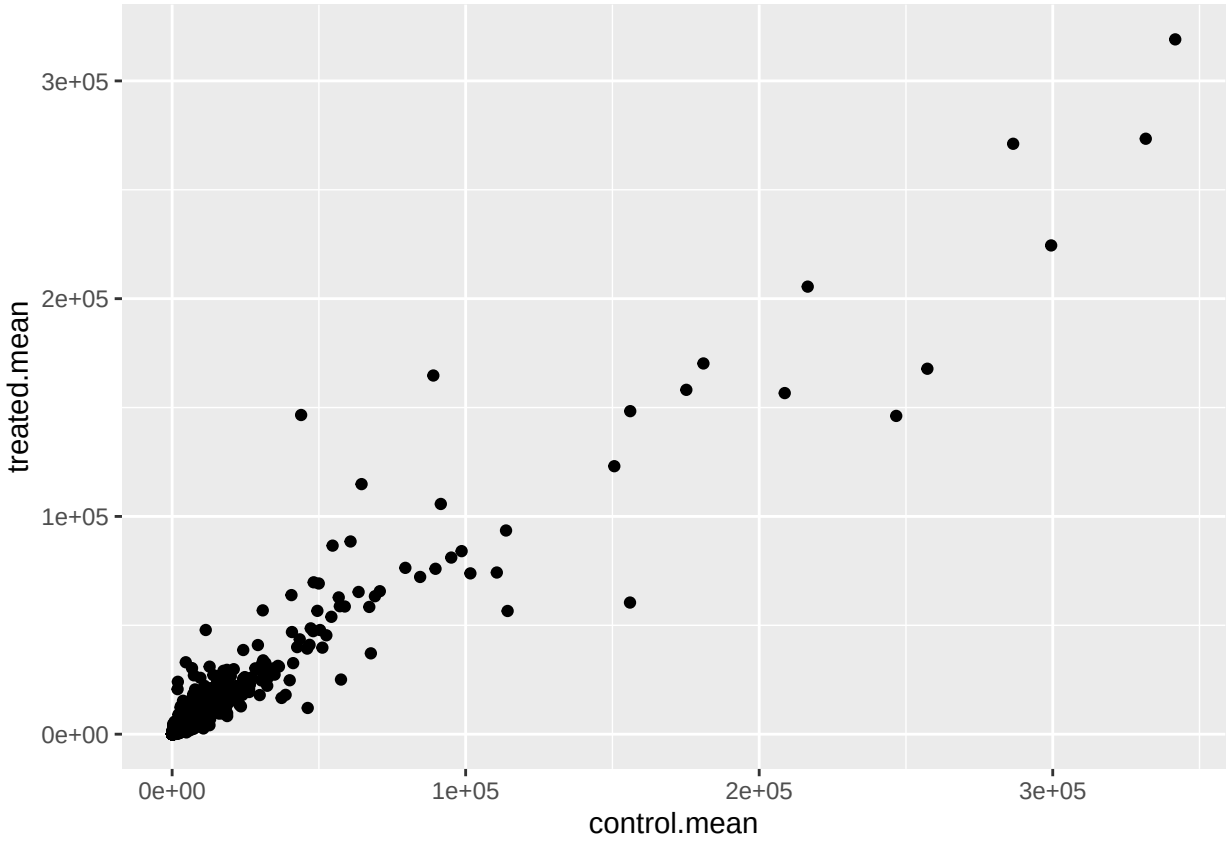
Q5 (a). Create a scatter plot showing the mean of the treated samples against the mean of the control samples.

```
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()
```



To answer question 5b, we would use the geom_point() function for this plot.

We note that there are 60,000-some rows in this data, but we are only seeing a few dozen dots at most outside of the big clump around the origin.

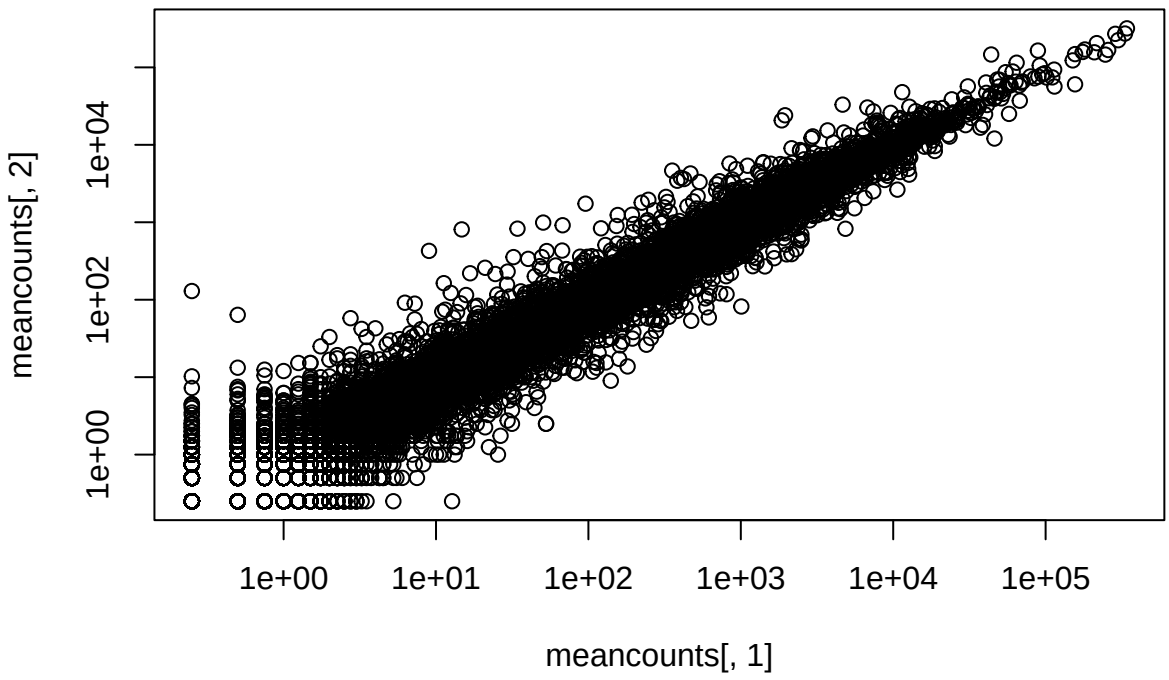
Q6. Try plotting both axes on a log scale. What is the argument to plot() that allows you to do this?

The log argument in plot allows use to plot both axes on a log scale. We change the plot using the base R function as follows.

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

```
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
```

We see that there are a couple of "odd" results. Namely, the NaN ("not a number") and -Inf (negative infinity) results.

Note the NaN is returned when we divide by zero and try to take the log. The -Inf is returned when we try to take the log of zero. since there are a lot of genes with zero expression, we filter our data to remove these genes and inspect your result to ensure things make sense.

```
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
```

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 purpose of the arr.ind argument in the which() function call above is to tell which rows (genes) and which columns (samples) have zero counts. We take the first column of it as this as we are only interested in which genes have zero values. since, we want to ignore any genes that have zero counts in any sample, we take the unique() function to ensure that we do not count any row twice, in the case that it has zero entries in both samples.

We note a common threshold used for calling something differentially expressed is a log2(FoldChange) of greater than 2 or less than -2. We filter the dataset both ways to see how many genes are up or down-regulated.

```
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
```

So we see that there are 250 up regulated gene at the greater than 2 fc level.

Q9. Using the down.ind vector above can you determine how many down regulated genes we have at the less than -2 fc level?

```
sum(down.ind)
```

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

So we see that there are 367 down regulated genes at the less than -2 fc level.

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

No we should not trust these results. So far all our analysis has been done solely on fold change. However, we have not performed any analysis in order to determine whether the fold change differences we see are statistically significant. Since fold change can be relatively large without being statistically significant the results are likely to be misleading.

4. DESeq2 analysis

We load the DESeq2 package.

```
library(DESeq2)
```

```
## Loading required package: S4Vectors

## Loading required package: stats4

## Loading required package: BiocGenerics

##
## Attaching package: 'BiocGenerics'

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

## 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:dplyr':
##
##     first, rename

## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice
```

```
## 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: 'matrixStats'

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

##
## 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
```

```
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},
##   }
```

Importing data

We will use the DESeqDataSetFromMatrix() function to build the required DESeqDataSet object and name it dds, short for our DESeqDataSet.

```
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
```

DESeq analysis

Next, we run the DESeq analysis pipeline on the dataset, and reassign the resulting object back to the same variable. Note that before we start, dds is a bare-bones DESeqDataSet. The DESeq() function takes a DESeqDataSet and returns a DESeqDataSet, but with additional information filled in (including the differential expression results we are after). Notice how if we try to access these results before running the analysis, nothing exists.

```
# The following line of code will yield will an error message.
# results(dds)
```

Now we are running the DESeq pipeline on the dds object, and reassigning the whole thing back to dds, which will now be a DESeqDataSet populated with all those values.

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

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

Getting results

Since we've got a fairly simple design (single factor, two groups, treated versus control), we can get results out of the object simply by calling the results() function on the DESeqDataSet that has been run through the pipeline.

```
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
```

We can now 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
```

The results function contains a number of arguments to customize the results table. We see that by default the argument alpha is set to 0.1. If the adjusted p value cutoff will be a value other than 0.1, alpha should be set to that value. We use alpha=0.5.

```
res05 <- results(dds, alpha=0.05)
summary(res05)
```

```
##
## out of 25258 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)       : 1236, 4.9%
## LFC < 0 (down)     : 933, 3.7%
## outliers [1]       : 142, 0.56%
## low counts [2]     : 9033, 36%
## (mean count < 6)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

5. Adding annotation data

Our result table so far only contains the Ensembl gene IDs. However, alternative gene names and extra annotation are usually required for informative interpretation of our results. In this section we will add this necessary annotation data to our results.

We will use one of Bioconductor's main annotation packages to help with mapping between various ID schemes. Here we load the AnnotationDbi package and the annotation data package for humans org.Hs.eg.db.

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

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

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

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

```
##
```

The later of these is is the organism annotation package ("org") for Homo sapiens ("Hs"), organized as an AnnotationDbi database package ("db"), using Entrez Gene IDs ("eg") as primary key. To get a list of all available key types that we can use to map between, use the columns() function:

```
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"
```

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. The column argument tells the mapIds() function which information we want, and the multiVals argument tells the function what to do if there are multiple possible values for a single input value. Here we ask to just give us back the first one that occurs in the database.

```
res$symbol <- mapIds(org.Hs.eg.db,
                     keys=row.names(res),      # These are 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
```

Now we reexamine the res object again.

```
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
```

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.

```
# For res$entrez we use the following code:
res$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
```

```
# For res$uniprot we use the following code:
res$uniprot <- mapIds(org.Hs.eg.db,
                     keys=row.names(res),
                     column="UNIPROT",
                     keytype="ENSEMBL",
                     multiVals="first")
```

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

```
# For res$genename we use the following code:
res$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
```

```
# Now we reexamine the res object.
head(res)
```

```
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 6 rows and 10 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      entrez     uniprot
##                 <numeric> <character> <character> <character>
## ENSG00000000003  0.163035      TSPAN6        7105  A0A024RCI0
## ENSG00000000005        NA        TNMD       64102      Q9H2S6
## ENSG00000000419  0.176032        DPM1        8813      O60762
## ENSG00000000457  0.961694       SCYL3       57147      Q8IZE3
## ENSG00000000460  0.815849    C1orf112       55732  A0A024R922
## ENSG00000000938        NA         FGR        2268      P09769
##                            genename
##                         <character>
## ENSG00000000003          tetraspanin 6
## ENSG00000000005            tenomodulin
## ENSG00000000419 dolichyl-phosphate m..
## ENSG00000000457 SCY1 like pseudokina..
## ENSG00000000460 chromosome 1 open re..
## ENSG00000000938 FGR proto-oncogene, ..
```

We can no arrange and view the results by the adjusted p-value as follows:

```
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 10 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      entrez     uniprot
##                 <numeric> <character> <character> <character>
## ENSG00000152583 1.32441e-71     SPARCL1        8404  A0A024RDE1
## ENSG00000179094 6.13966e-56        PER1        5187      O15534
## ENSG00000116584 3.49776e-53     ARHGEF2        9181      Q92974
```

```
## ENSG00000189221 3.46227e-52        MAOA        4128      P21397
## ENSG00000120129 1.59454e-44       DUSP1        1843      B4DU40
## ENSG00000148175 1.83034e-42        STOM        2040      F8VSL7
##                                genename
##                              <character>
## ENSG00000152583           SPARC like 1
## ENSG00000179094 period circadian reg..
## ENSG00000116584 Rho/Rac guanine nucl..
## ENSG00000189221    monoamine oxidase A
## ENSG00000120129 dual specificity pho..
## ENSG00000148175               stomatin
```

Finally, we can write out the ordered significant results with annotations as follows:

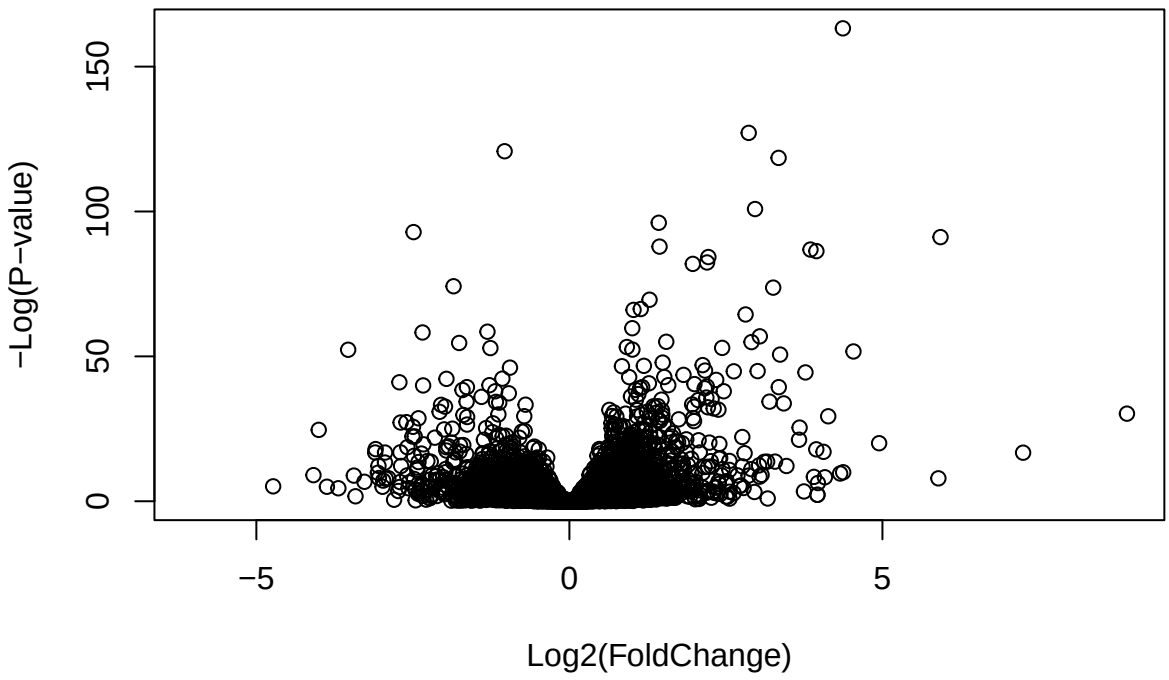
```
write.csv(res[ord,], "deseq_results.csv")
```

6. Data visualization

Volcano plots

Let's make a commonly produced visualization from this data, namely a so-called Volcano plot. These summary figures are frequently used to highlight the proportion of genes that are both significantly regulated and display a high fold change.

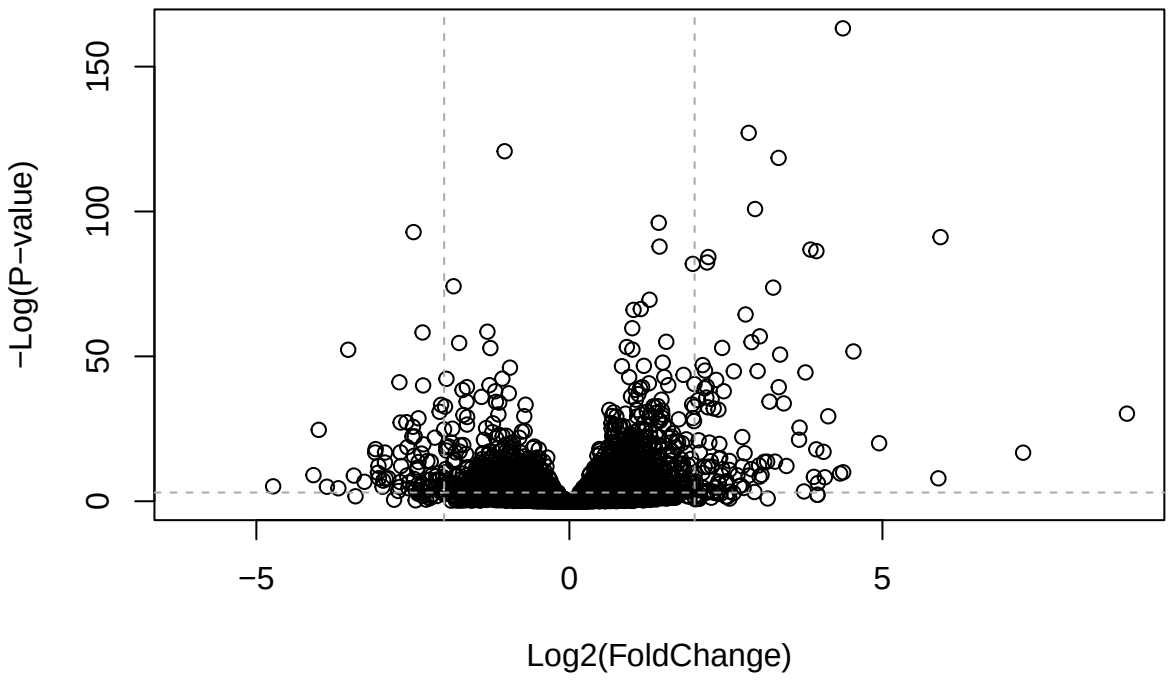
```
plot( res$log2FoldChange,  -log(res$padj),
      xlab="Log2(FoldChange)",
      ylab="-Log(P-value)")
```



To make this more useful we can add some guidelines (with the abline() function) and color (with a custom color vector) highlighting genes that have padj<0.05 and the absolute log2FoldChange>2.

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

# Now we add some cut-off lines.
abline(v=c(-2,2), col="darkgray", lty=2)
abline(h=-log(0.05), col="darkgray", lty=2)
```



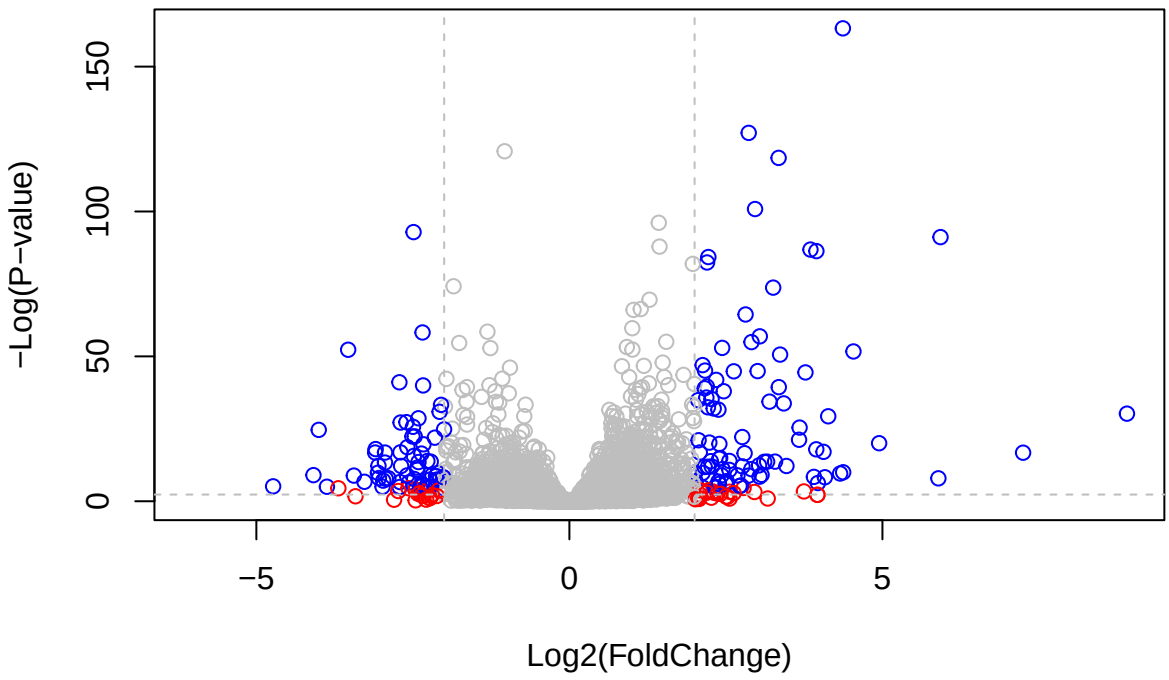
To color the points we will setup a custom color vector indicating transcripts with large fold change and significant differences between conditions as follows:

```
# We first 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"

# Our volcano plot with custom colors is as follows:
plot( res$log2FoldChange,  -log(res$padj),
 col=mycols, ylab="-Log(P-value)", xlab="Log2(FoldChange)" )

# Now we add the cut-off lines.
abline(v=c(-2,2), col="gray", lty=2)
abline(h=-log(0.1), col="gray", lty=2)
```



For even more customization we use the EnhancedVolcano bioconductor package. First we will add the more understandable gene symbol names to our full results object res as we will use this to label the most interesting genes in our final plot.

```
# After installing EnhancedVolcano in the console we call it below:
library(EnhancedVolcano)
```

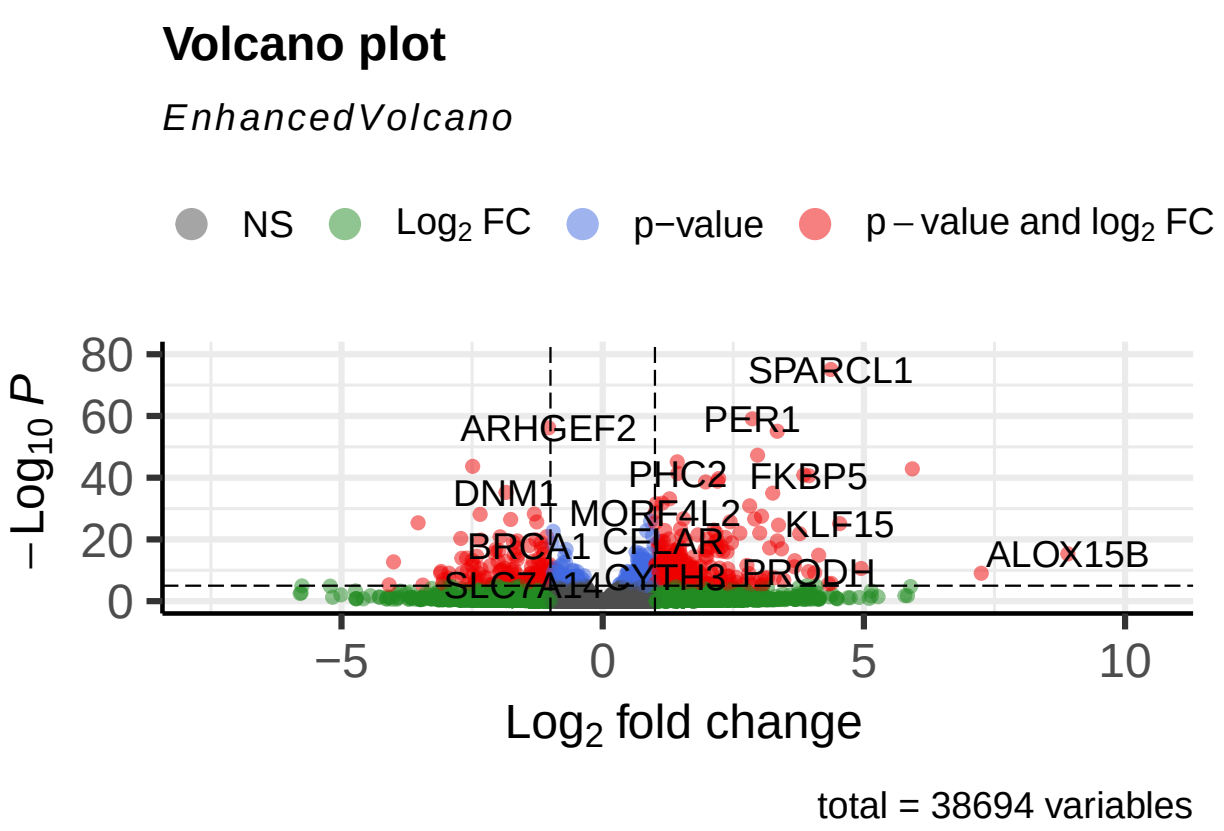
```
## Loading required package: ggrepel
```

```
## Registered S3 methods overwritten by 'ggalt':
##   method                  from
##   grid.draw.absoluteGrob  ggplot2
##   grobHeight.absoluteGrob ggplot2
##   grobWidth.absoluteGrob  ggplot2
##   grobX.absoluteGrob      ggplot2
##   grobY.absoluteGrob      ggplot2
```

Now we turn to creating our plot.

```
x <- as.data.frame(res)

EnhancedVolcano(x,
    lab = x$symbol,
    x = 'log2FoldChange',
    y = 'pvalue')
```



7. Pathway analysis

Pathway analysis (also known as gene set analysis or over-representation analysis), aims to reduce the complexity of interpreting gene lists via mapping the listed genes to known (i.e. annotated) biological pathways, processes and functions.

Pathway analysis with R and Bioconductor

We installed the necessary packages using the following code:

```
# We input the following commands into the console.
# BiocManager::install( c("pathview", "gage", "gageData") )
```

Now we can load the packages and setup the KEGG data-sets we need. The gageData package has pre-compiled databases mapping genes to KEGG pathways and GO terms for common organisms. kegg.sets.hs is a named list of 229 elements. Each element is a character vector of member gene Entrez IDs for a single KEGG pathway.

```
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)

# We 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"
```

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
```

Now we can run the gage pathway analysis.

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

Now we look at the object returned from gage().

```
attributes(keggres)
```

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

Now we look at the first few down (less) pathway results as follows:

```
# 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
```

We see that the top three Kegg pathways indicated here include Graft-versus-host disease, Type I diabetes and the Asthma pathway (with pathway ID hsa05310).

Now, we try out the pathview() function from the pathview package to make a pathway plot with our RNA-Seq expression results shown in color. We first manually supply a pathway.id (namely the first part of the "hsa05310 Asthma") that we could see from the print out above.

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

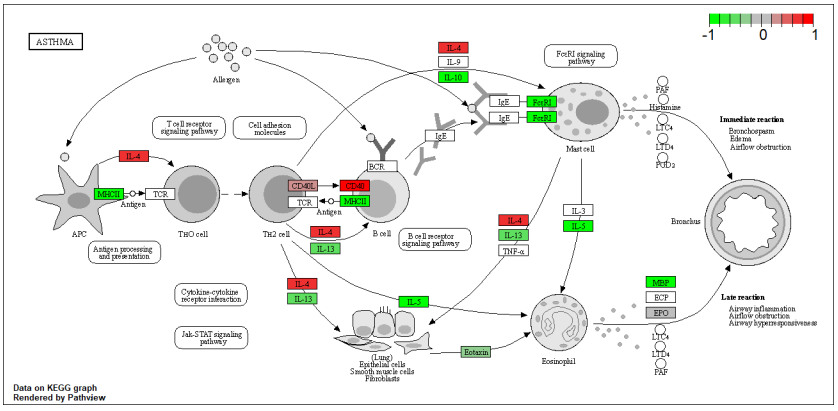
```
## Info: Downloading xml files for hsa05310, 1/1 pathways..
```

```
## Info: Downloading png files for hsa05310, 1/1 pathways..
```

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

```
## Info: Working in directory C:/Users/Caleb/Desktop/BIMM143 GIT/02-22-22_Transcriptomics_Analysis_of_RI
```

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



We can play with the other input arguments to pathview() to change the display in various ways including generating a PDF graph. For example:

```
# A different PDF based output of the same data.
# Note the PDF graph will not show up in this markdown file but, we include the
# code used to generate it.
pathview(gene.data=foldchanges, pathway.id="hsa05310", kegg.native=FALSE)
```

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

```
## Info: Working in directory C:/Users/Caleb/Desktop/BIMM143 GIT/02-22-22_Transcriptomics_Analysis_of_R
```

```
## Info: Writing image file hsa05310.pathview.pdf
```

Q12. Can you do the same procedure as above to plot the pathview figures for the top 2 down-regulated pathways?

For the first most down-regulated pathway we use the following code:

```
pathview(gene.data=foldchanges, pathway.id="hsa05332")
```

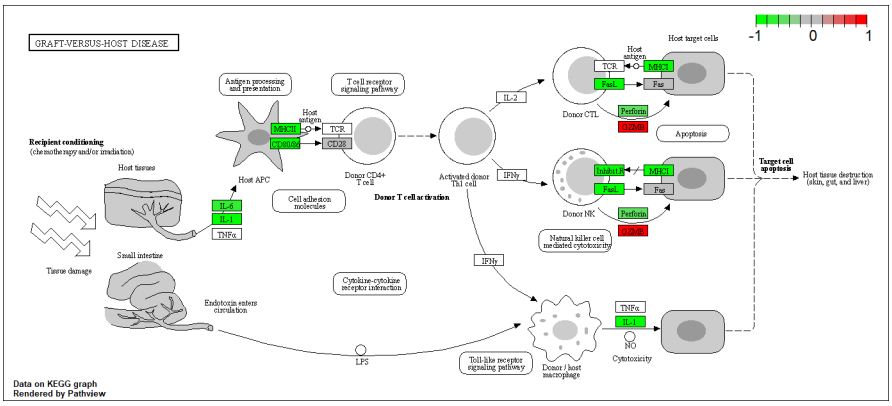
```
## Info: Downloading xml files for hsa05332, 1/1 pathways..
```

```
## Info: Downloading png files for hsa05332, 1/1 pathways..
```

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

```
## Info: Working in directory C:/Users/Caleb/Desktop/BIMM143 GIT/02-22-22_Transcriptomics_Analysis_of_R
```

```
## Info: Writing image file hsa05332.pathview.png
```



For the second most down-regulated pathway we use the following code:

```
pathview(gene.data=foldchanges, pathway.id="hsa04940")
```

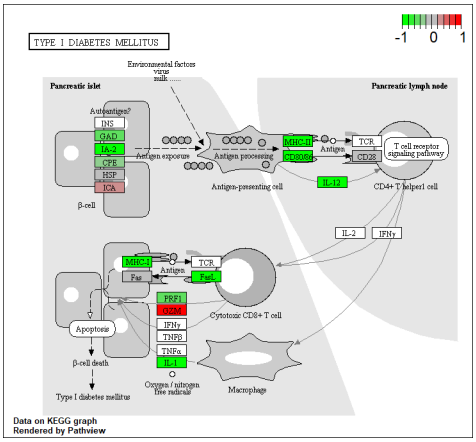
```
## Info: Downloading xml files for hsa04940, 1/1 pathways..
```

```
## Info: Downloading png files for hsa04940, 1/1 pathways..
```

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

```
## Info: Working in directory C:/Users/Caleb/Desktop/BIMM143 GIT/02-22-22_Transcriptomics_Analysis_of_Rl
```

```
## Info: Writing image file hsa04940.pathview.png
```



OPTIONAL: Plotting counts for genes of interest

We note that DESeq2 offers a function called plotCounts() that takes a DESeqDataSet that has been run through the pipeline, the name of a gene, and the name of the variable in the colData that you're interested in, and plots those values.

```
i <- grep("CRISPLD2", res$symbol)
res[i,]
```

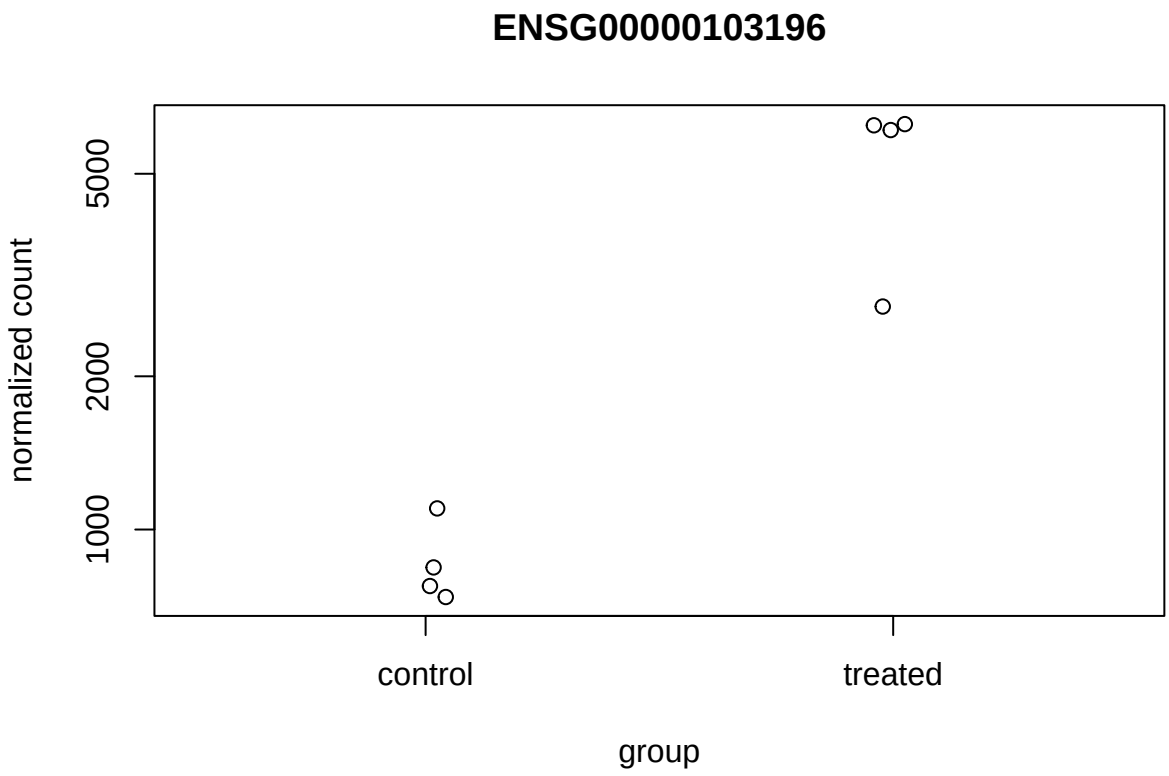
```
## log2 fold change (MLE): dex treated vs control
## Wald test p-value: dex treated vs control
## DataFrame with 1 row and 10 columns
##                   baseMean log2FoldChange     lfcSE      stat      pvalue
##                  <numeric>      <numeric> <numeric> <numeric>   <numeric>
## ENSG00000103196   3096.16        2.62603  0.267444   9.81899 9.32747e-23
##                       padj      symbol      entrez     uniprot
##                  <numeric> <character> <character> <character>
## ENSG00000103196 3.36344e-20    CRISPLD2       83716  A0A140VK80
##                             genename
##                          <character>
## ENSG00000103196 cysteine rich secret..
```

```
rownames(res[i,])
```

```
## [1] "ENSG00000103196"
```

Now, with that gene ID in hand let's plot the counts, where our intgroup, or "interesting group" variable is the "dex" column.

```
plotCounts(dds, gene="ENSG00000103196", intgroup="dex")
```



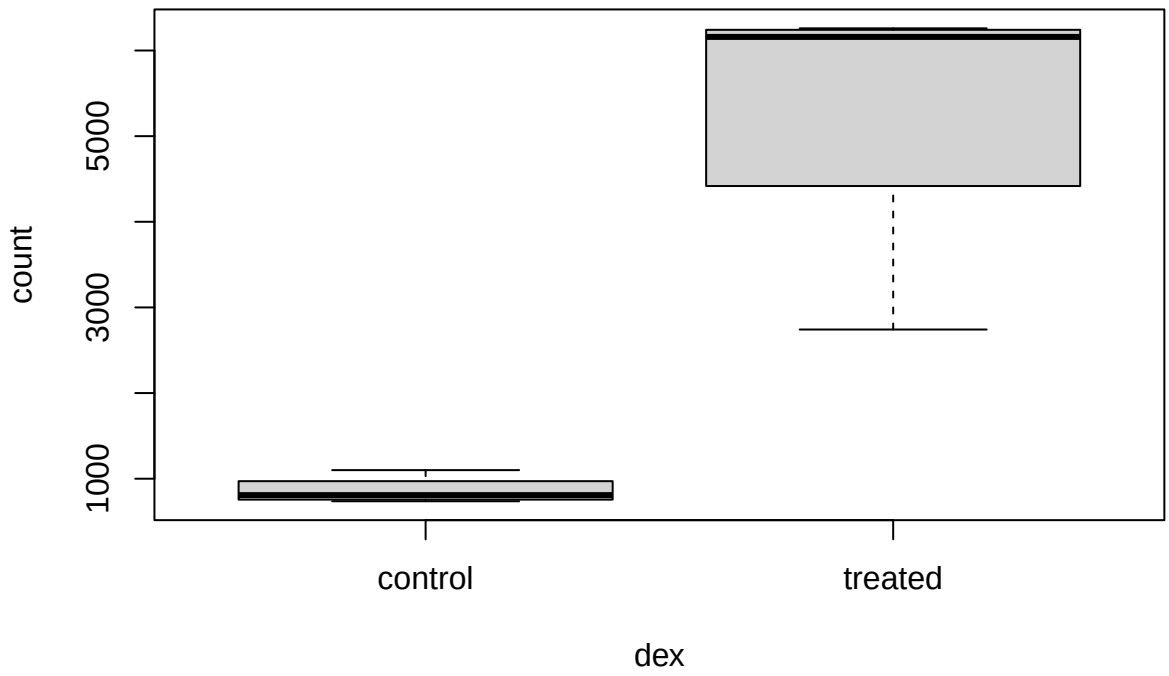
We notice that we could have actually returned the data instead of plotting. We then pipe this to ggplot and make our own figure to make a boxplot.

```
# We return the data as follows:
d <- plotCounts(dds, gene="ENSG00000103196", intgroup="dex", returnData=TRUE)
head(d)
```

```
##                count     dex
## SRR1039508  774.5002 control
## SRR1039509 6258.7915 treated
## SRR1039512 1100.2741 control
## SRR1039513 6093.0324 treated
## SRR1039516  736.9483 control
## SRR1039517 2742.1908 treated
```

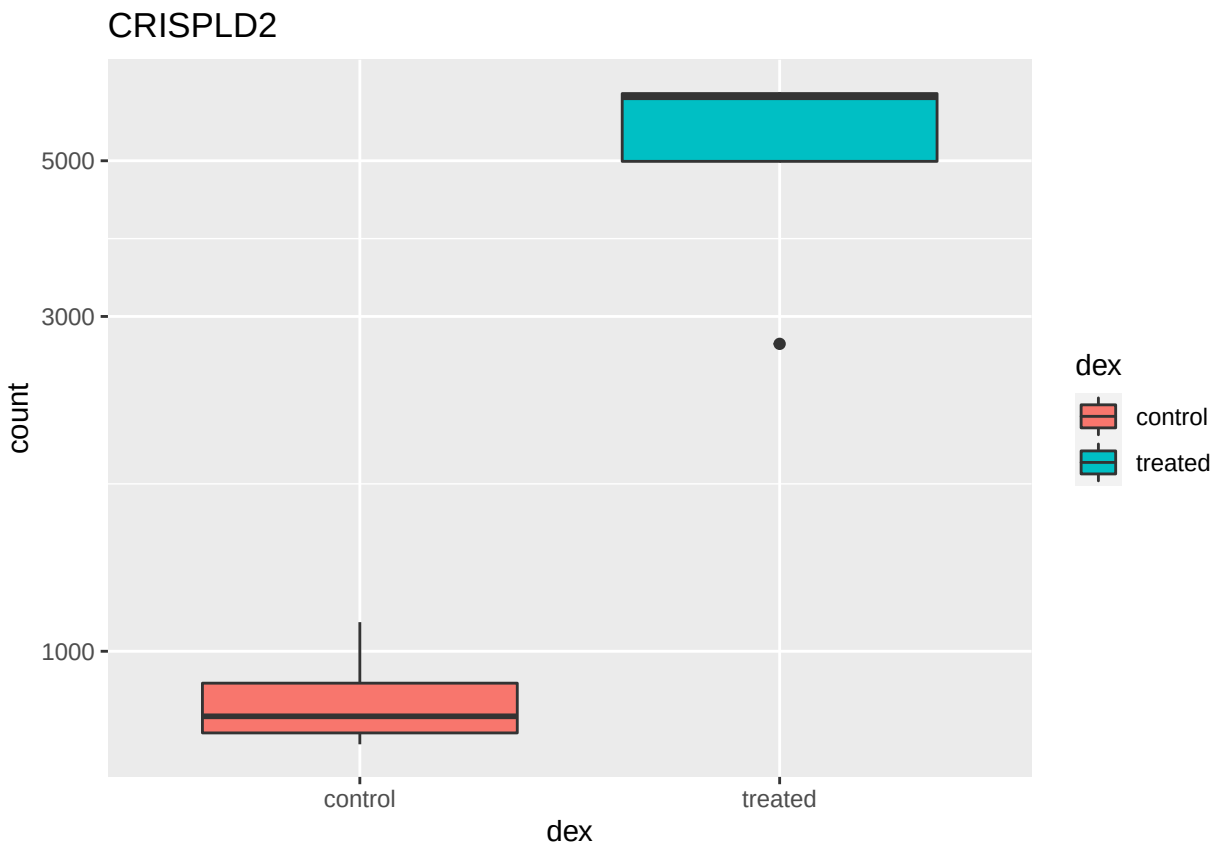
We can mow use this returned object to plot a boxplot with the base graphics function boxplot().

```
boxplot(count ~ dex , data=d)
```



Since the returned object is a data.frame it is also all setup for ggplot2 based plotting. For instance:

```
library(ggplot2)
ggplot(d, aes(dex, count, fill=dex)) +
  geom_boxplot() +
  scale_y_log10() +
  ggtitle("CRISPLD2")
```



Session Information

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] gageData_2.32.0             gage_2.44.0
##  [3] pathview_1.34.0             EnhancedVolcano_1.12.0
```

```
##  [5] ggrepel_0.9.1               org.Hs.eg.db_3.14.0
##  [7] AnnotationDbi_1.56.2        DESeq2_1.34.0
##  [9] SummarizedExperiment_1.24.0 Biobase_2.54.0
## [11] MatrixGenerics_1.6.0        matrixStats_0.61.0
## [13] GenomicRanges_1.46.1        GenomeInfoDb_1.30.1
## [15] IRanges_2.28.0              S4Vectors_0.32.3
## [17] BiocGenerics_0.40.0         ggplot2_3.3.5
## [19] dplyr_1.0.8
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-7           bit64_4.0.5            ash_1.0-15
##  [4] RColorBrewer_1.1-2     httr_1.4.2             Rgraphviz_2.38.0
##  [7] tools_4.1.2            utf8_1.2.2             R6_2.5.1
## [10] vipor_0.4.5            KernSmooth_2.23-20     DBI_1.1.2
## [13] colorspace_2.0-2       withr_2.4.3            tidyselect_1.1.1
## [16] ggrastr_1.0.1          ggalt_0.4.0            bit_4.0.4
## [19] compiler_4.1.2         extrafontdb_1.0        graph_1.72.0
## [22] cli_3.1.1              DelayedArray_0.20.0    labeling_0.4.2
## [25] KEGGgraph_1.54.0       scales_1.1.1           proj4_1.0-11
## [28] genefilter_1.76.0      stringr_1.4.0          digest_0.6.29
## [31] rmarkdown_2.11         XVector_0.34.0         pkgconfig_2.0.3
## [34] htmltools_0.5.2        extrafont_0.17         fastmap_1.1.0
## [37] highr_0.9              maps_3.4.0             rlang_1.0.1
## [40] rstudioapi_0.13        RSQLite_2.2.10         farver_2.1.0
## [43] generics_0.1.2         BiocParallel_1.28.3    RCurl_1.98-1.6
## [46] magrittr_2.0.2         GO.db_3.14.0           GenomeInfoDbData_1.2.7
## [49] Matrix_1.4-0           Rcpp_1.0.8             ggbeeswarm_0.6.0
## [52] munsell_0.5.0          fansi_1.0.2            lifecycle_1.0.1
## [55] stringi_1.7.6          yaml_2.2.2             MASS_7.3-55
## [58] zlibbioc_1.40.0        grid_4.1.2             blob_1.2.2
## [61] parallel_4.1.2         crayon_1.5.0           lattice_0.20-45
## [64] Biostrings_2.62.0      splines_4.1.2          annotate_1.72.0
## [67] KEGGREST_1.34.0        locfit_1.5-9.4         knitr_1.37
## [70] pillar_1.7.0           geneplotter_1.72.0     XML_3.99-0.8
## [73] glue_1.6.1             evaluate_0.14          png_0.1-7
## [76] vctrs_0.3.8            Rttf2pt1_1.3.10        gtable_0.3.0
## [79] purrr_0.3.4            assertthat_0.2.1       cachem_1.0.6
## [82] xfun_0.29              xtable_1.8-4           survival_3.2-13
## [85] tibble_3.1.6           beeswarm_0.4.0         memoise_2.0.1
## [88] ellipsis_0.3.2
```