DESeq2

DESeq2 analysis

Load DESeq2 and the data to be analyzed

```
# load DESeq2
suppressPackageStartupMessages(library(DESeq2))

# load additional packages used for result exploration:
library(ggplot2)
library(RColorBrewer)

# because DESeq2 works with matrices and uses row names as identifiers we load our data accordingly
counts <- read.csv("~/work/cmm262-2020/Module_1/Data/tardbp_counts_only.csv",
                   header = TRUE,
                   row.names = 1
                   )
head(counts)
```

```
##                     NT_shRNA_hepg2_rep1 NT_shRNA_hepg2_rep2
## ENSG00000227232.5_2                  27                  42
## ENSG00000238009.6_6                   0                   4
## ENSG00000237683.5                    22                  20
## ENSG00000239906.1_5                   6                   2
## ENSG00000241860.6_5                  26                  32
## ENSG00000228463.4                    77                  69
##                     TARDBP_shRNA_hepg2_rep1 TARDBP_shRNA_hepg2_rep2
## ENSG00000227232.5_2                      40                      35
## ENSG00000238009.6_6                       4                       8
## ENSG00000237683.5                        16                      27
## ENSG00000239906.1_5                       3                       7
## ENSG00000241860.6_5                      35                      35
## ENSG00000228463.4                        63                      67
```

```
# we also need our condition identifiers so DESeq2 know what to compare against what
col_data <- read.csv("~/work/cmm262-2020/Module_1/Data/tardbp_conditions_for_deseq2.csv",
                     header = TRUE,
                     row.names = 1
                     )
head(col_data)
```

```
##                           condition
## NT_shRNA_hepg2_rep1         control
## NT_shRNA_hepg2_rep2         control
## TARDBP_shRNA_hepg2_rep1   knockdown
## TARDBP_shRNA_hepg2_rep2   knockdown
```

Using our featurecounts processed data we don“t have to do any data normalization since DESeq2 will do that for us. Instead we only have to format our data so DESeq2 can analyze it. More analysis details can be

found in the example page of DESeq2.

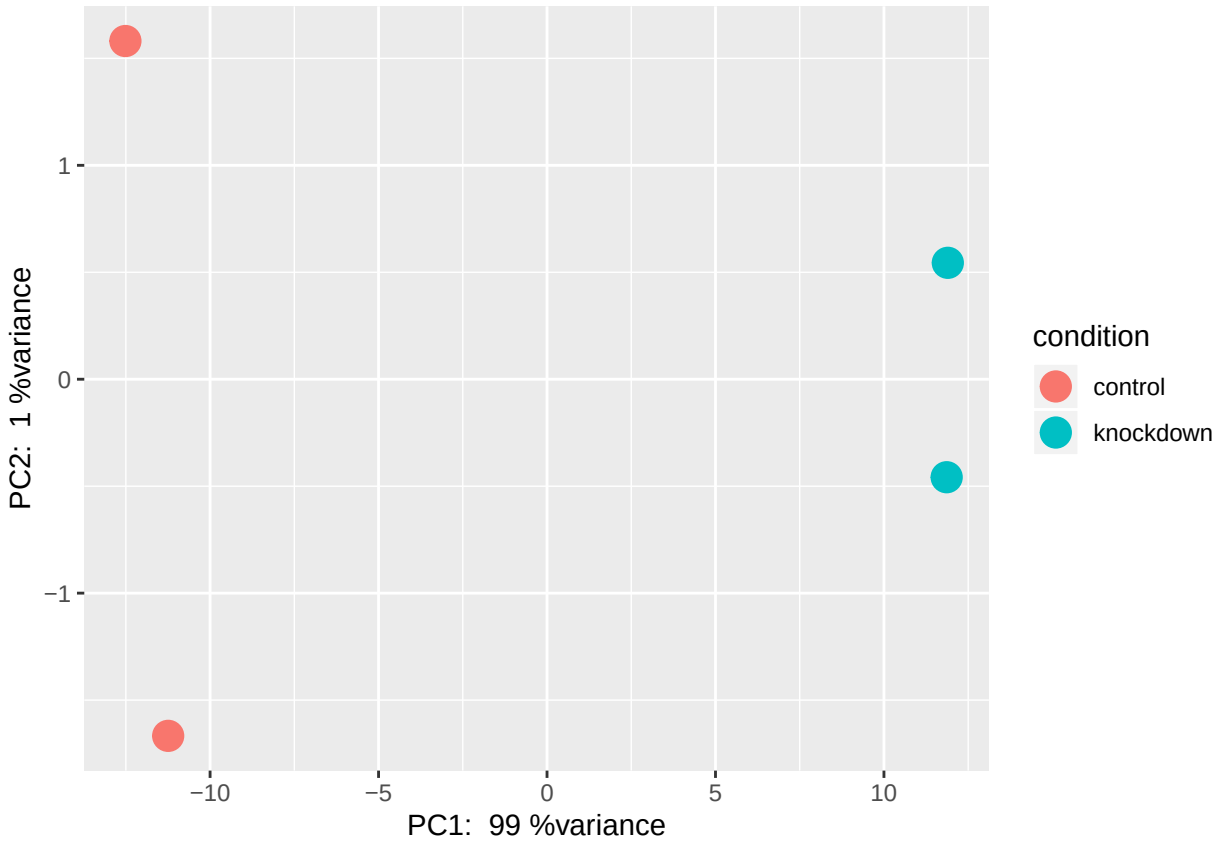
First, we will define experimental parameters. If using a counts matrix, we will use the following function, where countData is equal to our counts matrix ("counts"), colData is equal to our conditions, and design accounts for how we wish to model our effect (in this case, by the "condition" or treatment with a specific shRNA). The factor variable, in this case condition, needs to be columns of coldata:

```
dds <- DESeqDataSetFromMatrix(countData = counts,
                              colData = col_data,
                              design = ~ condition
                              )
dds
```

```
## class: DESeqDataSet
## dim: 19021 4
## metadata(1): version
## assays(1): counts
## rownames(19021): ENSG00000227232.5_2 ENSG00000238009.6_6 ...
##   ENSG00000210195.2 ENSG00000210196.2
## rowData names(0):
## colnames(4): NT_shRNA_hepg2_rep1 NT_shRNA_hepg2_rep2
##   TARDBP_shRNA_hepg2_rep1 TARDBP_shRNA_hepg2_rep2
## colData names(1): condition
```

Let's use some built in DESeq2 functions to do a little more QC. Let"s graph our individual samples using Principal Component Analysis (PCA). This allows us to assess overall variance within our experiment by defining principal components. This plot shows our samples in a 2D plane spanned by their first two principal components. This is useful for visualizing the overall effect of experimental covariates, in this case shRNA treatment, as well as batch effects that may confound findings. To address batch issues look into ComBat For a more thorough explanation of PCA, please refer to this notebook in the Tutorials folder.

```
# applying regularized log transformation (removes mean-variance dependence)
rld <- rlog(dds)

data <- plotPCA(rld, intgroup = "condition", returnData = TRUE )

percent_var <- round(100 * attr(data, "percentVar"))

ggplot(data, aes(x = PC1, y = PC2, color = condition)) +
  geom_point(size = 5) +
  xlab(paste("PC1: ", percent_var[1], "%variance")) +
  ylab(paste("PC2: ", percent_var[2], "%variance"))
```



As we can see, 99% of variance at the gene level is attibutable to PC1, along which our experimental conditions cluster separately. Thus, we have reasonably high confidence that out treatment is the main source of variance in our experiment.

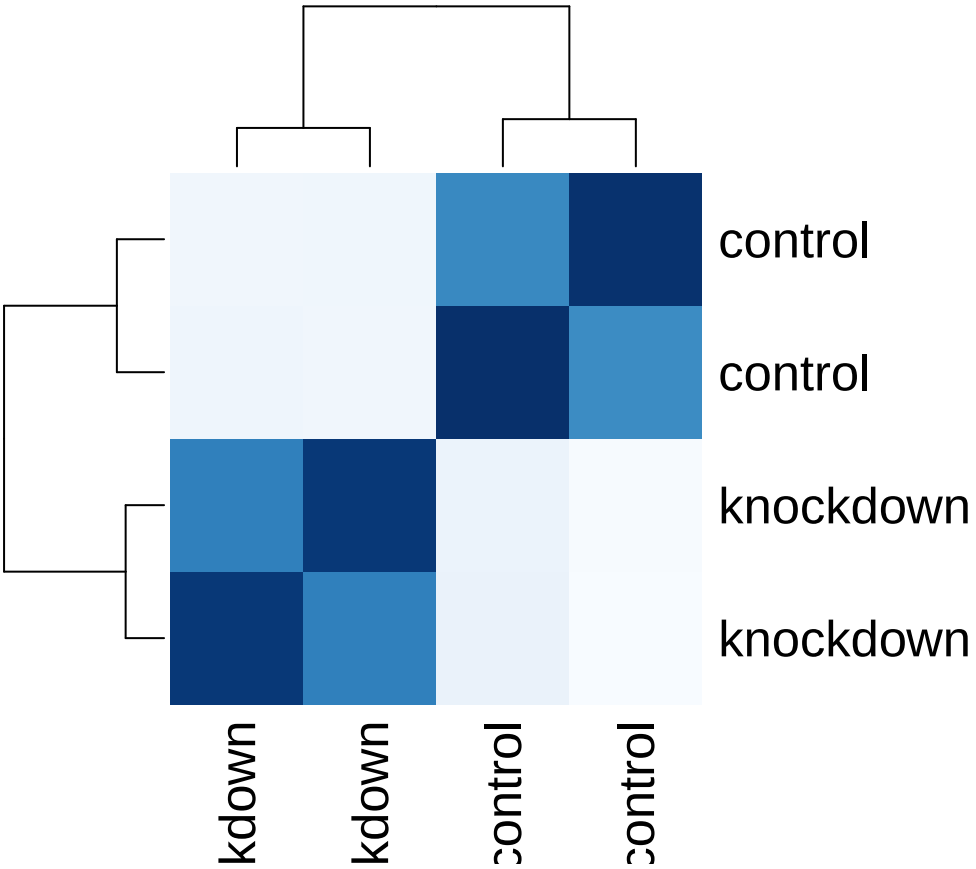
Similarly, this can also be visualized via heatmap, which summarizes sample-to-sample similarities via hierarchical clustering:

```
sample_dists <- dist(t(assay(rld)))

sample_dist_matrix <- as.matrix(sample_dists)

rownames(sample_dist_matrix) <- paste(rld$condition)

colnames(sample_dist_matrix) <- paste(rld$condition)

colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)

heatmap(sample_dist_matrix,
        clustering_distance_rows = sample_dists,
        clustering_distance_cols = sample_dists,
        col = colors
        )
```

```
## Warning in plot.window(...): "clustering_distance_rows" is not a graphical
## parameter
```

```
## Warning in plot.window(...): "clustering_distance_cols" is not a graphical
## parameter
```

```
## Warning in plot.xy(xy, type, ...): "clustering_distance_rows" is not a graphical
## parameter
## Warning in plot.xy(xy, type, ...): "clustering_distance_cols" is not a graphical
## parameter
## Warning in title(...): "clustering_distance_rows" is not a graphical parameter
## Warning in title(...): "clustering_distance_cols" is not a graphical parameter
```



We then execute DESeq2 on our dataset:

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

```
## estimating size factors
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

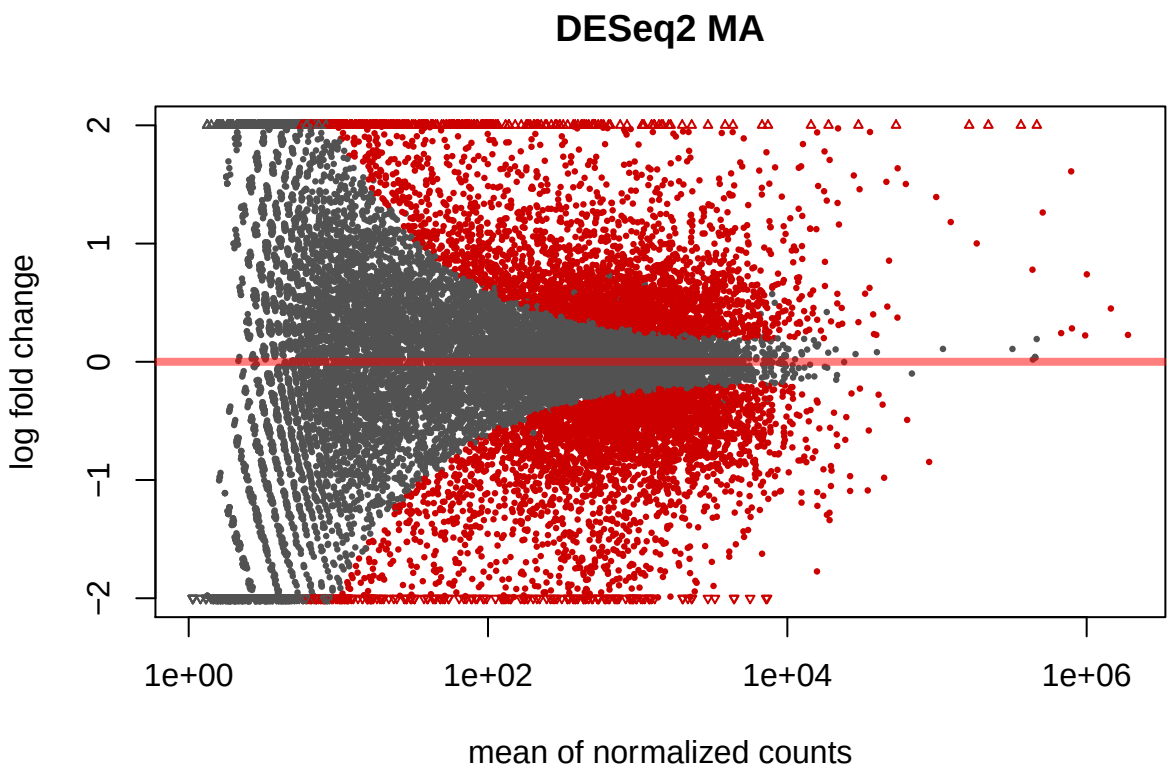
Taken directly from the documentation: results() extracts a result table from a DESeq analysis giving base means across samples, log2 fold changes, standard errors, test statistics, p-values and adjusted p-values

```
res <- results(dds_res)
```

The function plotMA allows us to plot the log2 fold changes over the mean of normalized counts for all the samples in dds. Points are colored red if the adjusted p value (alpha) is less than 0.1. Points which fall out of

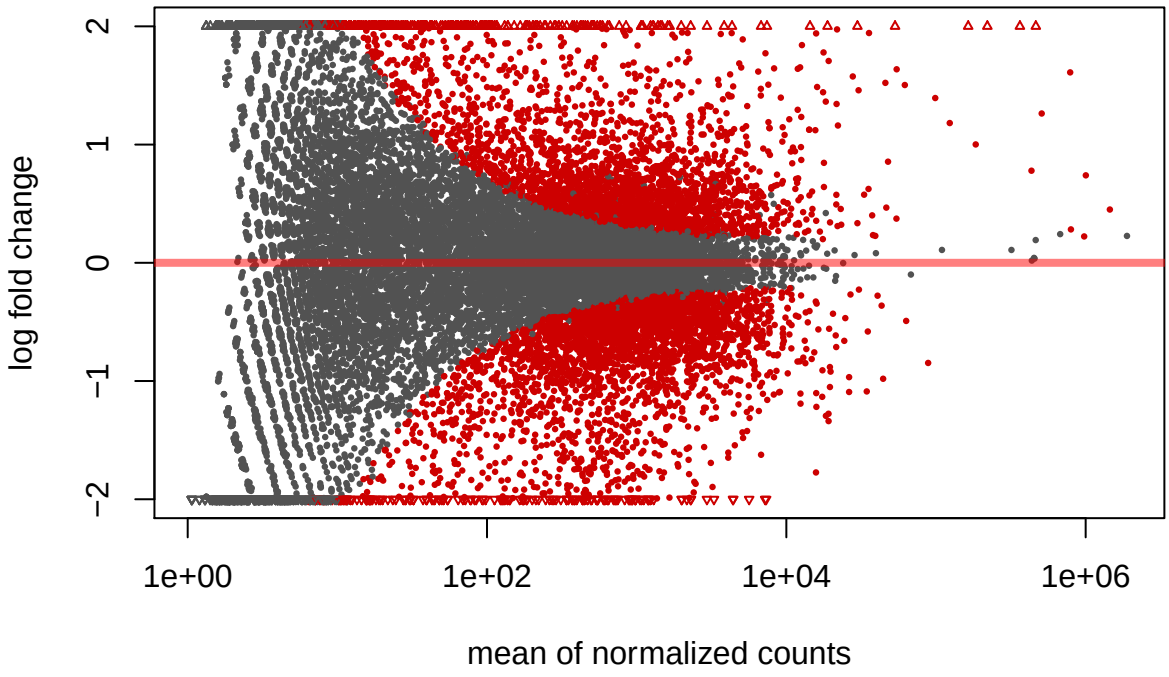
the window are plotted as open triangles pointing either up or down. The window can be widened using the ylim argument

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

plotMA(res,
       main = "DESeq2 MA",
       ylim = c(-2, 2)
       )
```



We can also get a more stringent view of our differentially expressed genes by rerunning result on dds with an additional argument alpha. This allows us to be more discerning with our adjusted p value threshold, in this case allowing us to decrease the cutoff to alpha=0.05

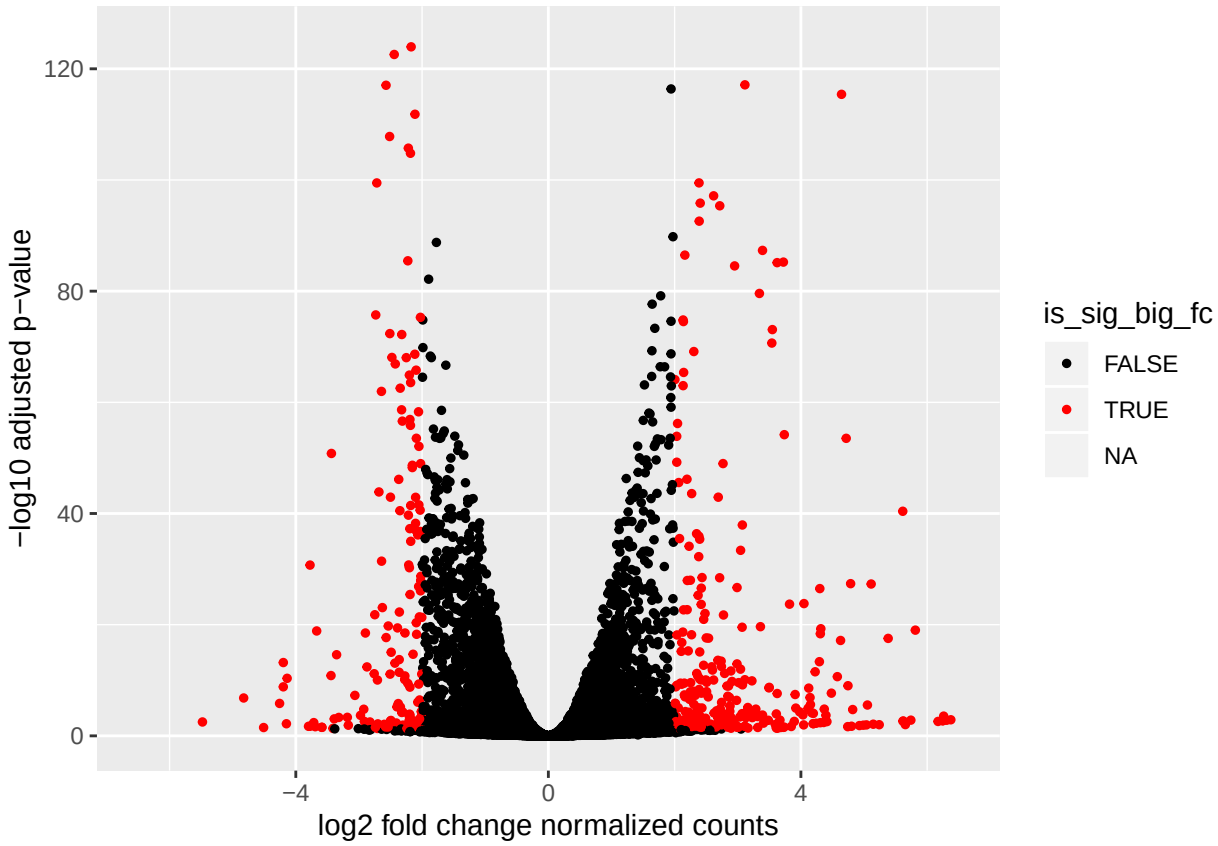
```
res <- results(dds_res, alpha = 0.05)
res_df <- as.data.frame(res)

plotMA(res,
       main = "DESeq2 MA alpha=0.05",
       ylim = c(-2, 2)
       )
```



We also care about the effect size, which in this case is the log2(fold change) of the gene. A good way to visualize this is with a volcano plot.

```
res_df$neg_log10_padj <- -log10(res_df$padj)
res_df$is_sig <- res_df$padj < 0.05
res_df$is_sig_big_fc <- res_df$is_sig & (res_df$log2FoldChange > 2 | res_df$log2FoldChange < -2)

ggplot(res_df, aes(x = log2FoldChange, y = neg_log10_padj, color = is_sig_big_fc)) +
      geom_point(size = 1) +
      scale_color_manual(values = c("black", "red")) +
      xlab("log2 fold change normalized counts") +
      ylab("-log10 adjusted p-value") +
      xlim(-6.5, 6.5) +
      ylim(0, 125)
```

```
## Warning: Removed 2596 rows containing missing values (geom_point).
```



Extract out just genes you consider significantly enriched.

```
sig_res_df_w_na <- res_df[res_df$is_sig_big_fc,]

# filter out genes with NA values in p-adjusted column
sig_res_df <- sig_res_df_w_na[-which(is.na(sig_res_df_w_na$padj)),]
```

OPTIONAL:

Convert ENSEMBL gene IDs to HSNC Gene Symbols

```
# Install biomaRt
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("biomaRt")
```

```
## Bioconductor version 3.10 (BiocManager 1.30.10), R 3.6.2 (2019-12-12)
```

```
## Installing package(s) 'biomaRt'
```

```
##
## The downloaded binary packages are in
##  /var/folders/nf/hh8b_kbd78j25lzr6k_ptf5w0000gn/T//RtmpWxOWKD/downloaded_packages
```

```
## Old packages: 'farver', 'prettyunits', 'stringi'
```

```
library("biomaRt")

mart <- useDataset("hsapiens_gene_ensembl", useMart("ensembl"))
ensembl_ids <- rownames(sig_res_df)
```

```
# Oops, our ensembl ids have versioning and are not the stable id.
# Let's strip the ensembl gene ids to their stable version.

ensembl_ids_stripped <- gsub("\\..*", "", ensembl_ids)

gene_list <- getBM(filters = "ensembl_gene_id",
                   attributes = c("ensembl_gene_id","hgnc_symbol"),
                   values = ensembl_ids_stripped,
                   mart = mart
                   )
```

```
## Cache found
```

```
# Not all the ensembl IDs mapped so we need to filter for just those IDs

rownames(sig_res_df) <- ensembl_ids_stripped
filtered_sig_res_df <- sig_res_df[gene_list$ensembl_gene_id,]

# Adjust row names to gene symbols
rownames(filtered_sig_res_df) <- make.names(gene_list$hgnc_symbol, unique=TRUE)

# Write this filtered data frame to a file
write.csv(filtered_sig_res_df,
          file = "~/work/cmm262-2020/Module_1/Data/TARDBP_sig_genes.csv"
          )
```