

Introduction to DGE

[View on GitHub](#)

Approximate time: 75 minutes

Learning Objectives

- Exploring expression data using data visualization
- Using volcano plots to evaluate relationships between DEG statistics
- Plotting expression of significant genes using heatmaps

Visualizing the results

When we are working with large amounts of data it can be useful to display that information graphically to gain more insight. During this lesson, we will get you started with some basic and more advanced plots commonly used when exploring differential gene expression data, however, many of these plots can be helpful in visualizing other types of data as well.

We will be working with three different data objects we have already created in earlier lessons:

- Metadata for our samples (a dataframe): `meta`
- Normalized expression data for every gene in each of our samples (a matrix): `normalized_counts`
- Tibble versions of the DESeq2 results we generated in the last lesson: `res_tableOE_tb` and `res_tableKD_tb`

First, let's create a metadata tibble from the data frame (don't lose the row names!)

```
mov10_meta <- meta %>%  
  rownames_to_column(var="samplename") %>%  
  as_tibble()
```

Next, let's bring in a column with gene symbols to the `normalized_counts` object, so we can use them to label our plots. Ensembl IDs are great for many things, but the gene symbols are much more recognizable to us, as biologists.

```
# DESeq2 creates a matrix when you use the counts() function  
## First convert normalized_counts to a data frame and transfer the row names to a column  
normalized_counts <- counts(dds, normalized=T) %>%  
  data.frame() %>%  
  rownames_to_column(var="gene")
```

```
# Next, merge together (ensembl IDs) the normalized counts data frame with a subset
grch38annot <- tx2gene %>%
  dplyr::select(ensgene, symbol) %>%
  dplyr::distinct()

## This will bring in a column of gene symbols
normalized_counts <- merge(normalized_counts, grch38annot, by.x="gene", by.y="ensgene")

# Now create a tibble for the normalized counts
normalized_counts <- normalized_counts %>%
  as_tibble()

normalized_counts
```

NOTE: A possible alternative to the above:

```
normalized_counts <- counts(dds, normalized=T) %>%
  data.frame() %>%
  rownames_to_column(var="gene") %>%
  as_tibble() %>%
  left_join(grch38annot, by=c("gene" = "ensgene"))
```

Plotting significant DE genes

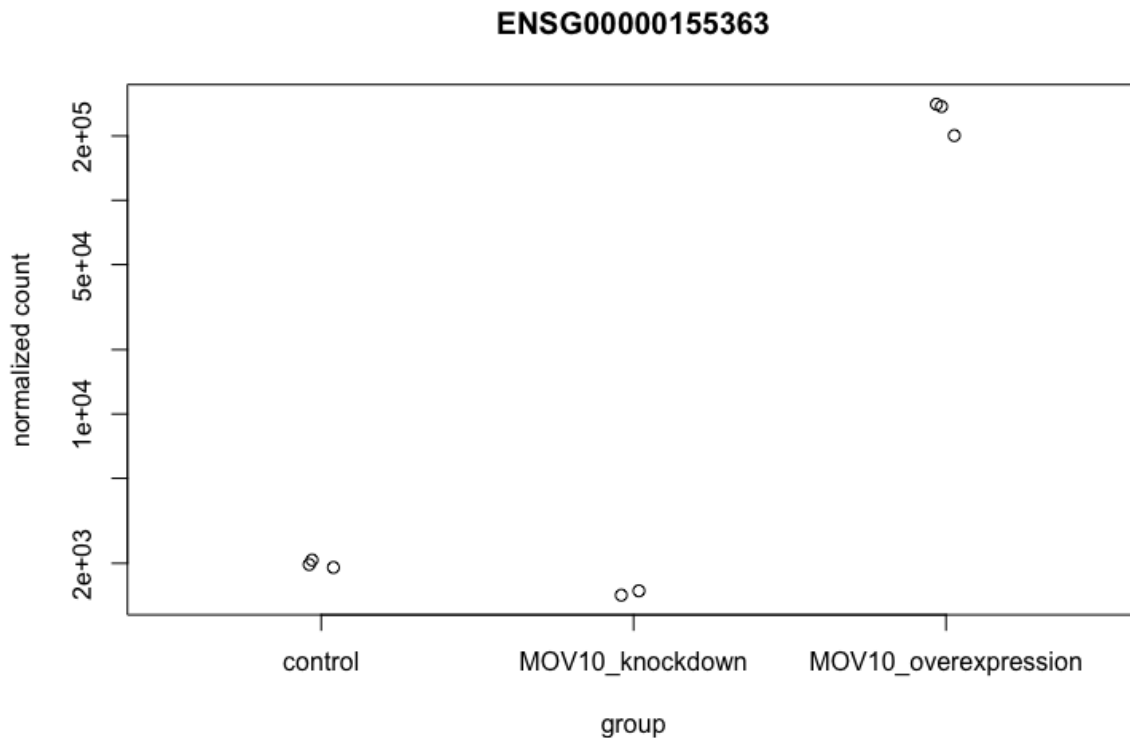
One way to visualize results would be to simply plot the expression data for a handful of genes. We could do that by picking out specific genes of interest or selecting a range of genes.

Using DESeq2 `plotCounts()` to plot expression of a single gene

To pick out a specific gene of interest to plot, for example MOV10, we can use the `plotCounts()` from DESeq2. `plotCounts()` requires that the gene specified matches the original input to DESeq2, which in our case was Ensembl IDs.

```
# Find the Ensembl ID of MOV10
grch38annot[grch38annot$symbol == "MOV10", "ensgene"]

# Plot expression for single gene
plotCounts(dds, gene="ENSG00000155363", intgroup="sampletype")
```



This DESeq2 function only allows for plotting the counts of a single gene at a time, and is not flexible regarding the appearance.

Using ggplot2 to plot expression of a single gene

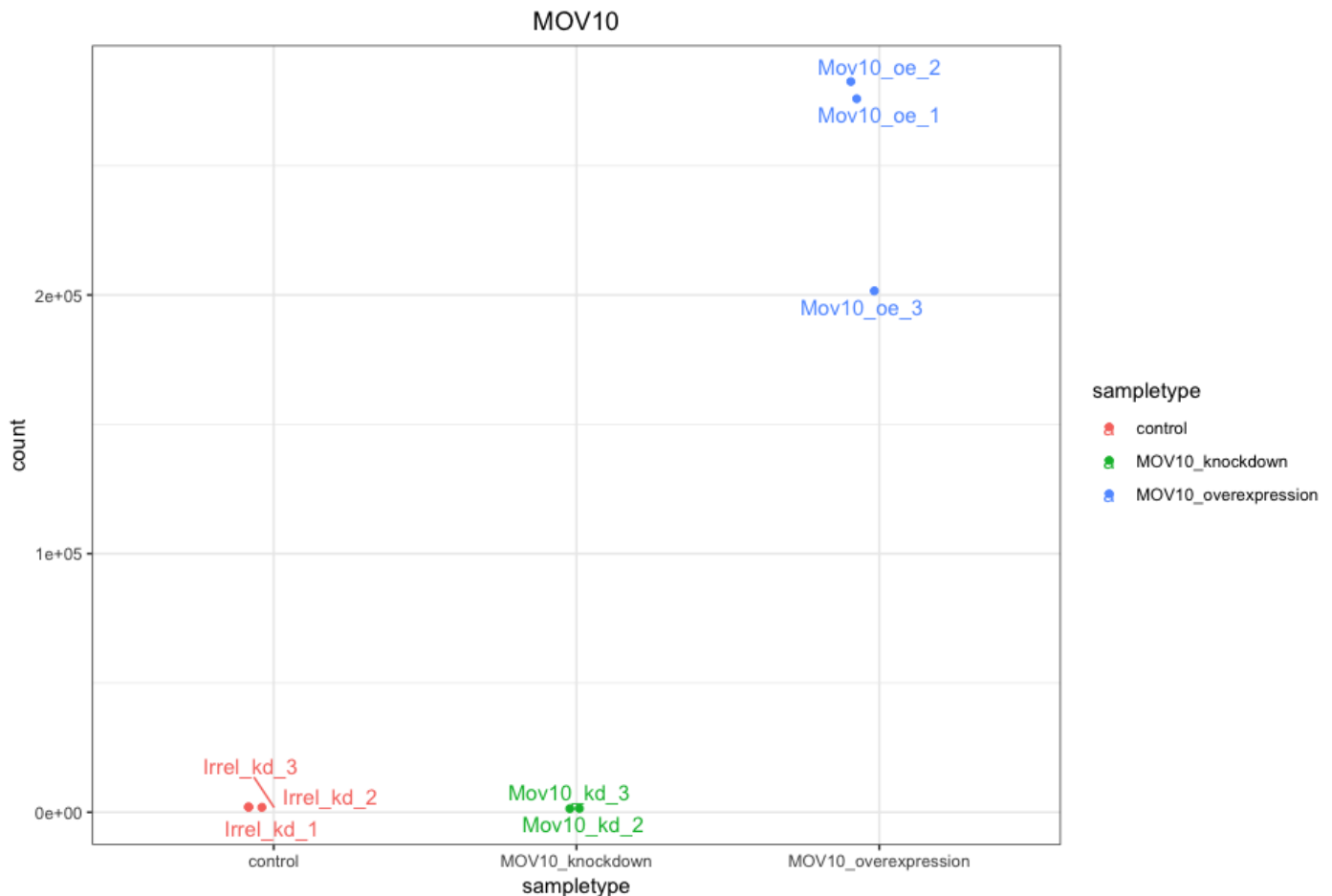
If you wish to change the appearance of this plot, we can save the output of `plotCounts()` to a variable specifying the `returnData=TRUE` argument, then use `ggplot()`:

```
# Save plotcounts to a data frame object
d <- plotCounts(dds, gene="ENSG00000155363", intgroup="samplotype", returnData=TRUE)

# What is the data output of plotCounts()?
d %>% View()

# Plot the MOV10 normalized counts, using the samplenames (rownames(d) as labels)
ggplot(d, aes(x = samplotype, y = count, color = samplotype)) +
  geom_point(position=position_jitter(w = 0.1, h = 0)) +
  geom_text_repel(aes(label = rownames(d))) +
  theme_bw() +
  ggtitle("MOV10") +
  theme(plot.title = element_text(hjust = 0.5))
```

Note that in the plot below (code above), we are using `geom_text_repel()` from the `ggrepel` package to label our individual points on the plot.



If you are interested in plotting the expression of multiple genes all together, please refer to [the short lesson linked here](#) where we demo this for the top 20 most significantly expressed genes.

Heatmap

In addition to plotting subsets, we could also extract the normalized values of *all* the significant genes and plot a heatmap of their expression using `pheatmap()`.

```
### Extract normalized expression for significant genes from the OE and control samples
norm_OEsig <- normalized_counts[,c(1:4,7:9)] %>%
  filter(gene %in% sigOE$gene)
```

Now let's draw the heatmap using `pheatmap`:

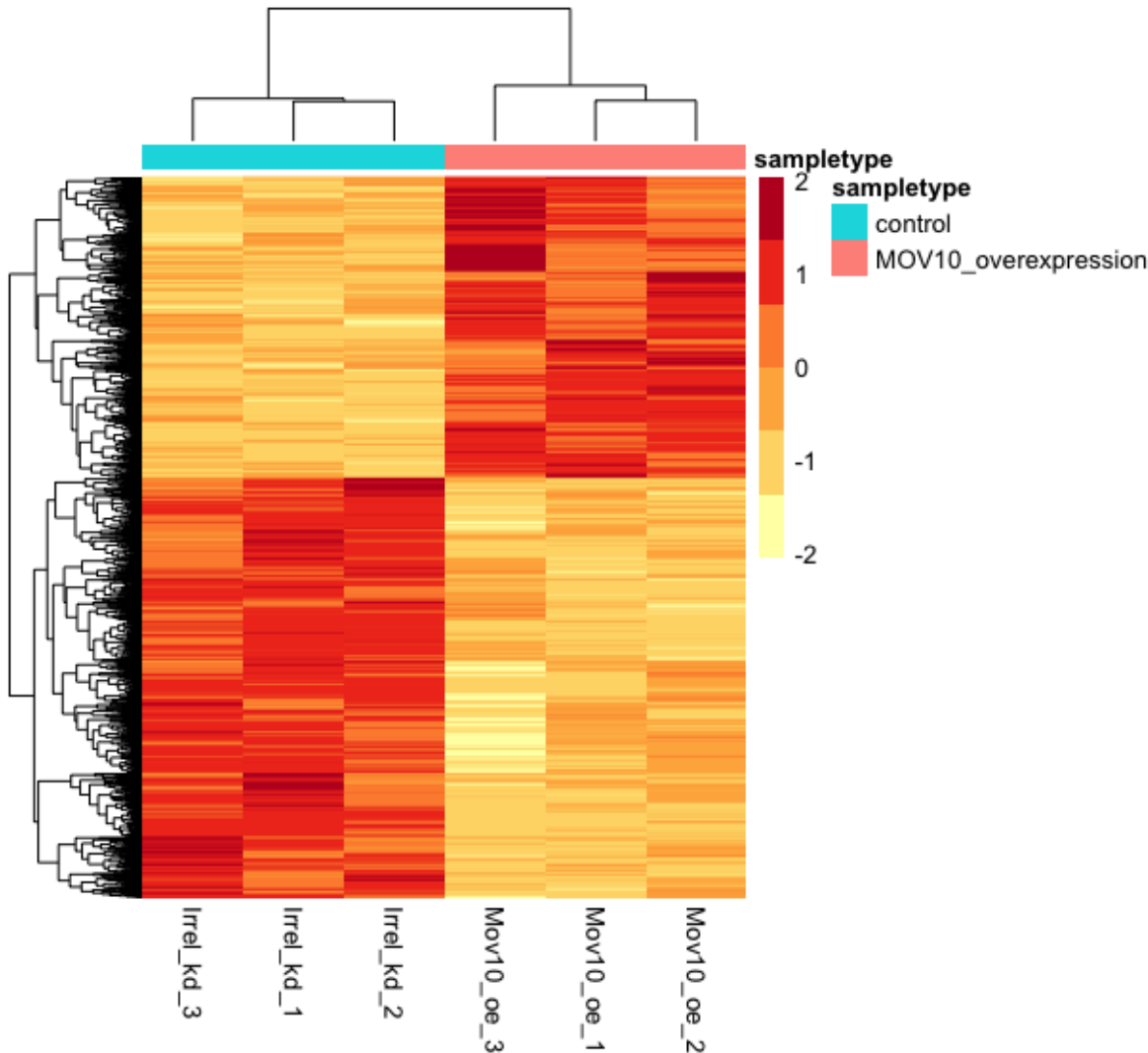
```
### Set a color palette
heat_colors <- brewer.pal(6, "YlOrRd")

### Run pheatmap using the metadata data frame for the annotation
pheatmap(norm_OEsig[2:7],
  color = heat_colors,
  cluster_rows = T,
  show_rownames = F,
```

```

annotation = meta,
border_color = NA,
fontsize = 10,
scale = "row",
fontsize_row = 10,
height = 20)

```



NOTE: There are several additional arguments we have included in the function for aesthetics. One important one is `scale="row"`, in which Z-scores are plotted, rather than the actual normalized count value.

Z-scores are computed on a gene-by-gene basis by subtracting the mean and then dividing by the standard deviation. The Z-scores are computed **after the clustering**, so that it only affects the graphical aesthetics and the color visualization is improved.

Volcano plot

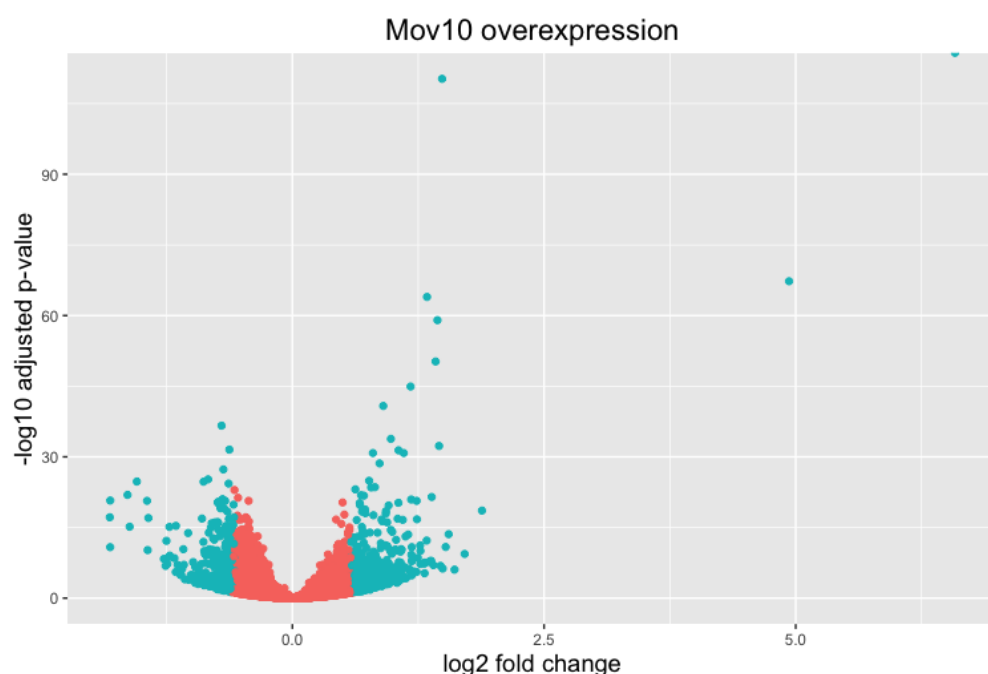
The above plot would be great to look at the expression levels of a good number of genes, but for more of a global view there are other plots we can draw. A commonly used one is a volcano plot; in which you have the log transformed adjusted p-values plotted on the y-axis and log2 fold change values on the x-axis.

To generate a volcano plot, we first need to have a column in our results data indicating whether or not the gene is considered differentially expressed based on p-adjusted values and we will include a log2fold change here.

```
## Obtain logical vector where TRUE values denote padj values < 0.05 and fold change
res_table0E_tb <- res_table0E_tb %>%
  mutate(threshold_0E = padj < 0.05 & abs(log2FoldChange) >= 0.58)
```

Now we can start plotting. The `geom_point` object is most applicable, as this is essentially a scatter plot:

```
## Volcano plot
ggplot(res_table0E_tb) +
  geom_point(aes(x = log2FoldChange, y = -log10(padj), colour = threshold_0E)) +
  ggtitle("Mov10 overexpression") +
  xlab("log2 fold change") +
  ylab("-log10 adjusted p-value") +
  #scale_y_continuous(limits = c(0,50)) +
  theme(legend.position = "none",
        plot.title = element_text(size = rel(1.5), hjust = 0.5),
        axis.title = element_text(size = rel(1.25)))
```



This is a great way to get an overall picture of what is going on, but what if we also wanted to know where the top 10 genes (lowest padj) in our DE list are located on this plot? We could label those dots with the gene name on the Volcano plot using `geom_text_repel()`.

First, we need to order the `res_tableOE` tibble by `padj`, and add an additional column to it, to include on those gene names we want to use to label the plot.

```
## Add all the gene symbols as a column from the grch38 table using bind_cols()
res_tableOE_tb <- bind_cols(res_tableOE_tb, symbol=grch38annot$symbol[match(res_tableOE_tb$gene_id, grch38annot$gene_id)])

## Create an empty column to indicate which genes to label
res_tableOE_tb <- res_tableOE_tb %>% mutate(gene_labels = "")

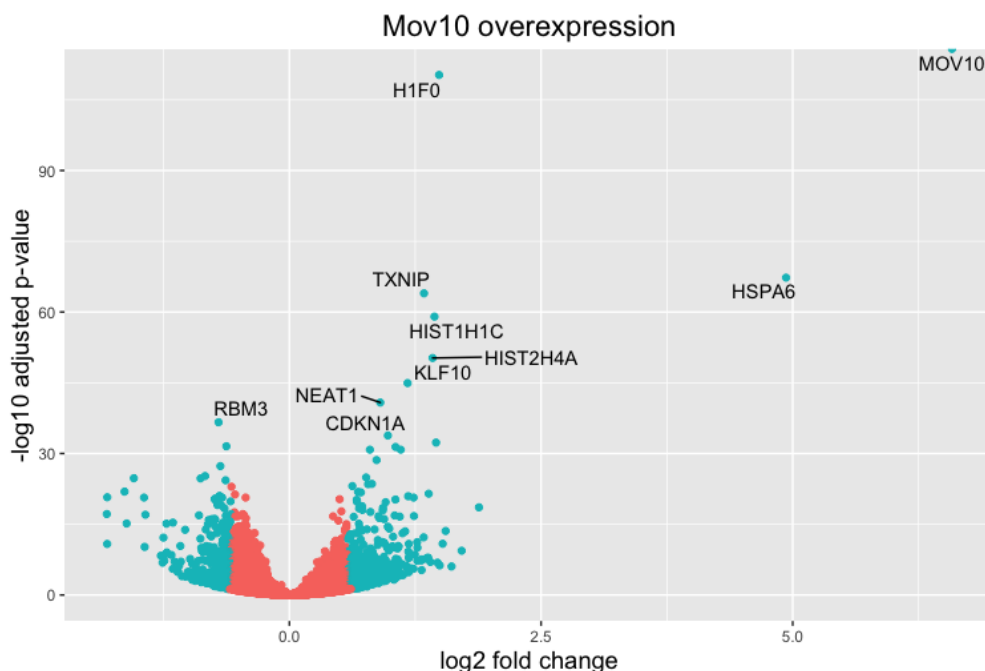
## Sort by padj values
res_tableOE_tb <- res_tableOE_tb %>% arrange(padj)

## Populate the gene_labels column with contents of the gene symbols column for the top 10 genes
res_tableOE_tb$gene_labels[1:10] <- as.character(res_tableOE_tb$symbol[1:10])

View(res_tableOE_tb)
```

Next, we plot it as before with an additional layer for `geom_text_repel()` wherein we can specify the column of gene labels we just created.

```
ggplot(res_tableOE_tb, aes(x = log2FoldChange, y = -log10(padj))) +
  geom_point(aes(colour = threshold_OE)) +
  geom_text_repel(aes(label = gene_labels)) +
  ggtitle("Mov10 overexpression") +
  xlab("log2 fold change") +
  ylab("-log10 adjusted p-value") +
  theme(legend.position = "none",
        plot.title = element_text(size = rel(1.5), hjust = 0.5),
        axis.title = element_text(size = rel(1.25)))
```



NOTE: If using the DESeq2 tool for differential expression analysis, the package 'DEGreport' can use the DESeq2 results output to make the top20 genes and the volcano plots generated above by writing a few lines of simple code. While you can customize the plots above, you may be interested in using the easier code. Below are examples of the code to create these plots:

```
DEGreport::degPlot(dds = dds, res = res, n = 20, xs = "type", group = "c")

DEGreport::degVolcano(
  data.frame(res[,c("log2FoldChange", "padj")]), # table - 2 columns
  plot_text = data.frame(res[1:10, c("log2FoldChange", "padj", "id")])) #

# Available in the newer version for R 3.4
DEGreport::degPlotWide(dds = dds, genes = row.names(res)[1:5], group = "c")
```

This lesson has been developed by members of the teaching team at the [Harvard Chan Bioinformatics Core \(HBC\)](#). These are open access materials distributed under the terms of the [Creative Commons Attribution license \(CC BY 4.0\)](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

- Materials and hands-on activities were adapted from [RNA-seq workflow](#) on the Bioconductor website

DGE_workshop_salmon_online is maintained by [hbctraining](#).

This page was generated by [GitHub Pages](#).