Zebrafish Dataset Practical 1

Before you start, make sure you’ve read the document that describes the zebrafish dataset we’re using in this practical. And make sure you’ve put the required files (deseq2-results.tsv and samples.tsv) in your home directory.

To begin, here are three exercises that require using the command line in Terminal:

1. Using the awk and wc commands, work out how many genes are significantly differentially expressed (i.e. adjusted p-value < 0.05) for each of the four comparisons in deseq2-results.tsv.
2. Using the cut command, make four separate files, each of which contains just one of the comparisons. So each file will contain Ensembl ID, p-value, adjusted p-value, log2 fold change, chromosome, start, end, strand, biotype, name, description, 24 count columns and 24 normalised count columns. Keep these four files because you’ll need them for later exercises.
3. Filter the four files you have made using awk so that they only contain the significantly differentially expressed genes. Keep these four files as well.

The rest of this practical uses R, so open RStudio and load the tidyverse packages:

library(tidyverse)

Read in the DESeq2 results file:

# assign the results file name to a variable  
deseq\_results\_file <- 'deseq2-results.tsv'  
  
# load data  
deseq\_results <- read\_tsv(deseq\_results\_file)

## Volcano plot

Prepare the data for making a volcano plot. We need to convert the adjusted p-values to -log10(adjusted p-value). Also, we are going to make a new column that marks whether a gene is significantly different or not and another one that shows whether the significant genes are up or down.

# make -log10p column  
deseq\_results <-   
 mutate(deseq\_results,   
 uninf\_5dpf\_hom\_vs\_sib\_log10p = -log10(uninf\_5dpf\_hom\_vs\_sib\_adjp))  
# make significant column  
deseq\_results <-   
 mutate(deseq\_results,   
 uninf\_5dpf\_hom\_vs\_sib\_sig =   
 !is.na(uninf\_5dpf\_hom\_vs\_sib\_adjp) &   
 uninf\_5dpf\_hom\_vs\_sib\_adjp < 0.05 )  
# make factor for up and down genes  
deseq\_results <-   
 mutate(deseq\_results,   
 uninf\_5dpf\_hom\_vs\_sib\_up\_or\_down =   
 ifelse(uninf\_5dpf\_hom\_vs\_sib\_sig &   
 uninf\_5dpf\_hom\_vs\_sib\_log2fc > 0, 'Up',  
 ifelse(uninf\_5dpf\_hom\_vs\_sib\_sig &   
 uninf\_5dpf\_hom\_vs\_sib\_log2fc < 0, 'Down',  
 'Not Sig') ) )  
# sort by Adjusted pvalue for uninf\_5dpf\_hom\_vs\_sib comparison  
deseq\_results <- arrange(deseq\_results, desc(uninf\_5dpf\_hom\_vs\_sib\_adjp))

The basic volcano plot shows the -log10(p-value) against the log2(fold change) for each gene, with the genes coloured by whether the gene is up or down or not significant.

# plot adjusted pvalue against log2 fold change  
# with up coloured in orange and down coloured in blue  
volcano\_plot <-   
 ggplot(data = deseq\_results,   
 # This sets the data for the plot  
 aes(x = uninf\_5dpf\_hom\_vs\_sib\_log2fc, y = uninf\_5dpf\_hom\_vs\_sib\_log10p,   
 colour = uninf\_5dpf\_hom\_vs\_sib\_up\_or\_down)) +  
 # and specifies we want to plot log10pval against log2fc and   
 # colour it by the up\_or\_down column  
 # these aesthetics will apply to any other geoms added  
 geom\_point() +  
 # this says we want to plot the data as points  
 scale\_colour\_manual(name = "", values = c(Up = '#cc6600', Down = '#0073b3',  
 `Not Sig` = "grey80"))  
 # and this explicitly sets the colours for the 3 categories

To print the ggplot object use print(volcano\_plot). We can also change the grey background and have better axis titles. This also shows how to remove the legend if you want to.

volcano\_plot <- volcano\_plot +  
 # this says take the previous plot object and add to it  
 theme\_minimal() +  
 # a new theme that changes the grey background  
 labs(x = expr(log[2]\*'(Fold Change)'), y = expr(log[10]\*'(Adjusted pvalue)')) +  
 # better axis titles  
 guides(colour = "none")  
 # and remove the legend

The code above shows how ggplot objects can be built up by creating a basic object and then progressively adding more to it.

Next we can add labels to some of the points which meet certain criteria. For example we can filter the data based on log2(fold change) and adjusted p-value.

# data to highlight specific genes  
biggest\_changers <-   
 filter(deseq\_results,   
 # filter the deseq\_results object  
 abs(uninf\_5dpf\_hom\_vs\_sib\_log2fc) > 2,   
 # for rows with an absolute log2fc greater than 2  
 uninf\_5dpf\_hom\_vs\_sib\_adjp < 0.05 )  
 # and an adjusted pvalue less than 0.05

Now we add the labels using the ggrepel <https://github.com/slowkow/ggrepel> package. This is a package designed to better position point labels on plots so that all the labels are legible.

# load the ggrepel package  
library(ggrepel)  
labelled\_plot <- volcano\_plot +  
 geom\_text\_repel(data = biggest\_changers, aes(label = Name))  
 # add text to the plot using the biggest changers data frame  
 # and use the Name column as the label  
 # the x, y and column aesthetics are inherited from the original ggplot object

Or we could do just the top 10 genes by adjusted p-value.

# get the top 10 genes by adjusted pvalue  
top\_10\_genes <-   
 arrange(deseq\_results, uninf\_5dpf\_hom\_vs\_sib\_adjp) %>%   
 # sort by adjusted pvalue  
 head(10)  
 # and take the top 10 rows  
  
top\_10\_plot <- volcano\_plot +  
 geom\_label\_repel(data = top\_10\_genes, aes(label = Name))  
 # add labels using the top10 data.

### Extra exercises

1. Try replotting the volcano plot without polr2a. You’ll need to make a data.frame without polr2a in it using filter.
2. What does the plot look like if you don’t sort the data by adjusted p-value first? Rerun the code to load the data and create the -log10p, sig and up\_or\_down columns but don’t sort by adjusted p-value.

## Count Plot

To plot the normalised counts for each sample for a gene it helps to have a table of the sample info. This samples file has different properties of the condition (stage, treatment and genotype) in separate columns.

sample\_info\_file <- 'samples.tsv'  
sample\_info <- read\_tsv(sample\_info\_file)  
# set sample column name  
names(sample\_info)[1] <- 'sample'  
  
# split condition into three separate columns  
sample\_info <- separate(sample\_info, condition,   
 into = c('treatment', 'stage', 'genotype'), sep = "\_")  
  
# set levels of treatment, stage and genotype  
sample\_info$treatment <- fct\_relevel(sample\_info$treatment, "uninf", "inf")  
sample\_info$stage <- fct\_relevel(sample\_info$stage, '3dpf', '5dpf', '7dpf')  
sample\_info$genotype <- fct\_relevel(sample\_info$genotype, 'wt', 'het', 'hom')  
  
# get uninf\_5dpf samples in order wt, het, hom  
# rbind binds rows together to make a new data frame  
sample\_info\_uninf\_5dpf <- rbind(  
 filter(sample\_info, treatment == "uninf", stage == "5dpf", genotype == "wt"),  
 filter(sample\_info, treatment == "uninf", stage == "5dpf", genotype == "het"),  
 filter(sample\_info, treatment == "uninf", stage == "5dpf", genotype == "hom")  
)  
  
# set levels  
sample\_info\_uninf\_5dpf$sample <-   
 fct\_relevel(sample\_info\_uninf\_5dpf$sample,   
 as.character(sample\_info\_uninf\_5dpf$sample) )  
  
# make a new column of sibs and mutants, rather than wt, hets, homs  
sample\_info\_uninf\_5dpf <-   
 mutate(sample\_info\_uninf\_5dpf,   
 mutant = str\_replace(genotype, "wt", "sib")) %>%  
 mutate(mutant = str\_replace(mutant, "het", "sib")) %>%  
 mutate(mutant = str\_replace(mutant, "hom", "mut"))  
# set levels  
sample\_info\_uninf\_5dpf$mutant <-   
 fct\_relevel(sample\_info\_uninf\_5dpf$mutant, 'sib', 'mut')

To produce a count plot we subset the normalised\_counts object to the counts for a single gene. Then we transform the data.frame so that every observation is in its own row for ggplot.

normalised\_counts <-   
 select(deseq\_results, "GeneID", matches('uninf\_5dpf\_.\*\_normalised')) %>%  
 rename\_all(list(~ str\_replace(., "\_normalised\_count", "")))  
  
# get a specific gene  
counts\_for\_gene <- filter(normalised\_counts, GeneID == "ENSDARG00000055838")

# transform from wide to long  
counts\_for\_gene <- pivot\_longer(counts\_for\_gene, -GeneID,   
 names\_to = 'sample', values\_to = 'count')

# set levels of sample variable  
counts\_for\_gene$sample <- factor(counts\_for\_gene$sample,  
 levels = c(unique(counts\_for\_gene$sample)))

To see what it looks like now, try head(counts\_for\_gene). To see the counts for each individual sample we can plot sample on the x-axis and count on the y, like this:

basic\_count\_plot <- ggplot(data = counts\_for\_gene) +  
 geom\_point( aes(x = sample, y = count) ) +   
 theme\_minimal() +  
 theme(axis.text.x = element\_text(angle = 90))

We can customise the plot to make it look nicer by using the information in the sample\_info\_uninf\_5dpf object.

# join counts to sample data  
counts\_plus\_sample\_info\_uninf\_5dpf <- merge(sample\_info\_uninf\_5dpf, counts\_for\_gene)  
  
# set up a colour-blind friendly colour palette  
colour\_blind\_palette <-   
 c( 'blue' = rgb(0,0.45,0.7),  
 'vermillion' = rgb(0.8, 0.4, 0),  
 'blue\_green' = rgb(0, 0.6, 0.5),  
 'yellow' = rgb(0.95, 0.9, 0.25),  
 'sky\_blue' = rgb(0.35, 0.7, 0.9),  
 'orange' = rgb(0.9, 0.6, 0),  
 'purple' = rgb(0.8, 0.6, 0.7),  
 'black' = rgb(0, 0, 0) )  
colour\_palette <- unname(colour\_blind\_palette)  
  
# plot as points in different colours  
sample\_count\_plot\_coloured <- ggplot(data = counts\_plus\_sample\_info\_uninf\_5dpf) +  
 geom\_point( aes(x = sample, y = count,   
 colour = mutant), size = 2 ) +   
 theme\_minimal() +  
 scale\_colour\_manual(values = colour\_palette) +  
 theme(axis.text.x = element\_text(angle = 90))

We could also have the shape of the points represent the actual genotypes, in case we want to check whether the hets are similar to the wild types.

# plot with genotype as shape of points  
sample\_count\_plot\_colour\_shape <-   
 ggplot(data = counts\_plus\_sample\_info\_uninf\_5dpf) +  
 geom\_point( aes(x = sample, y = count,   
 shape = genotype, colour = mutant),  
 size = 2) +   
 theme\_minimal() +  
 scale\_colour\_manual(values = colour\_palette) +  
 theme(axis.text.x = element\_text(angle = 90))

If we have a large number of samples a boxplot might be more appropriate.

# boxplot   
basic\_boxplot <- ggplot(data = counts\_plus\_sample\_info\_uninf\_5dpf) +  
 geom\_boxplot( aes(x = mutant, y = count, fill = mutant)) +   
 theme\_minimal() +  
 scale\_fill\_manual(values = colour\_palette)

Or we could plot the points grouped by mutant status rather than by sample.

# plot points by mutant status  
points\_by\_mutant <- ggplot(data = counts\_plus\_sample\_info\_uninf\_5dpf) +  
 geom\_point( aes(x = mutant, y = count,   
 shape = genotype, colour = mutant) ) +   
 theme\_minimal() +  
 scale\_colour\_manual(values = colour\_palette)

The points for each mutant status appear at the same x position and may plot on top of each other. To avoid this we can add a random shift left or right.

# jitter points to prevent overplotting  
set.seed(163754)  
points\_jittered <- ggplot(data = counts\_plus\_sample\_info\_uninf\_5dpf) +  
 geom\_point( aes(x = mutant, y = count,   
 shape = genotype, colour = mutant),  
 position = position\_jitter(width = 0.2, height = 0)) +   
 theme\_minimal() +  
 scale\_colour\_manual(values = colour\_palette)

Or we could plot the points on top of the boxplot. For this we have to change the shape of the points with scale\_shape\_manual(values = c(21,24,22)) and use the fill aesthetic rather than colour. Otherwise the orange points will not show up on the orange background of the boxplot.

# add to boxplot  
set.seed(172)  
boxplot\_points\_jittered <-   
 ggplot(data = counts\_plus\_sample\_info\_uninf\_5dpf) +  
 geom\_boxplot( aes(x = mutant, y = count, fill = mutant),  
 outlier.shape = NA ) +   
 geom\_point( aes(x = mutant, y = count,   
 shape = genotype, fill = mutant),  
 size = 2,  
 position = position\_jitter(width = 0.2, height = 0)) +   
 theme\_minimal() +  
 scale\_fill\_manual(values = colour\_palette,  
 guide = guide\_legend(override.aes = list(shape = NA))) +   
 scale\_shape\_manual(values = c(21,24,22))

What happens if you don’t use outlier.shape = NA?

## Heatmap

To plot a heatmap of the significantly differentially expressed genes we need to subset the deseq\_results data frame, like this:

sig\_results <- filter(deseq\_results, uninf\_5dpf\_hom\_vs\_sib\_sig) %>%  
 arrange(uninf\_5dpf\_hom\_vs\_sib\_adjp)

We need to scale the normalised counts somehow or the colour scale will be affected by the most highly expressed genes. We centre the data by substracting the mean value for each gene and scale by dividing by the standard deviation for each gene using the scale function. scale works on columns and our data is by row so we need to transpose it (twice!).

rownames(sig\_results) <- sig\_results$GeneID  
sig\_normalised\_counts <- select(sig\_results, matches('uninf\_5dpf\_.\*\_normalised')) %>%  
 rename\_all(list(~ str\_replace(., "\_normalised\_count", "")))  
  
# scale  
sig\_normalised\_counts\_scaled <- as.data.frame(t( scale( t(sig\_normalised\_counts) ) ))

Then we cluster the rows of the data so that similar genes are grouped together.

# function to cluster the rows of a data frame  
cluster <- function(df) {  
 df <- as.data.frame(df)  
 distance\_matrix <- dist(df)  
 clustering <- hclust(distance\_matrix)  
 df\_ordered <- df[ clustering$order, ]  
 return(df\_ordered)  
}  
  
# cluster rows  
sig\_normalised\_counts\_scaled\_clustered <- cluster(sig\_normalised\_counts\_scaled)

Then we make the data long, make sure the levels of the factors are set correctly and plot the heatmap with geom\_raster ([geom\_raster](https://ggplot2.tidyverse.org/reference/geom_tile.html)).

# add a Gene ID column and set levels  
sig\_normalised\_counts\_scaled\_clustered$GeneID <-   
 factor(rownames(sig\_normalised\_counts\_scaled\_clustered),  
 levels = rev(rownames(sig\_normalised\_counts\_scaled\_clustered)))

# transform from wide to long  
sig\_scaled\_clustered\_long <-   
 pivot\_longer(sig\_normalised\_counts\_scaled\_clustered, -GeneID,   
 names\_to = 'sample', values\_to = 'count')

# set levels of sample  
sig\_scaled\_clustered\_long$sample <-   
 factor(sig\_scaled\_clustered\_long$sample,  
 levels = unique(sig\_scaled\_clustered\_long$sample))  
  
# create heatmap  
clustered\_heatmap <-   
 ggplot(data = sig\_scaled\_clustered\_long) +   
 geom\_raster( aes(x = sample, y = GeneID, fill = count) )  
# the data is sig\_scaled\_clustered\_long  
# The aesthetics map the x-axis to sample, the y-axis to GeneID and   
# the fill colour of the heatmap tiles to count

Each gene ID appears as a y-axis label and they cannot be read. Also the sample names aren’t legible either. We can remove both sets of labels with theme\_void().

clustered\_heatmap <-   
 ggplot(data = sig\_scaled\_clustered\_long) +   
 geom\_raster( aes(x = sample, y = GeneID, fill = count) ) +  
 theme\_void()

Then, if we want, we can add back the x-axis labels and rotate them to make them legible. We’ve also changed the colour scheme using the viridis package ([viridis](https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html)).

library(viridis)  
clustered\_heatmap <-   
 ggplot(data = sig\_scaled\_clustered\_long) +   
 geom\_raster( aes(x = sample, y = GeneID, fill = count) ) +  
 scale\_fill\_viridis(option = 'plasma') +   
 theme\_void() +   
 theme(axis.text.x =   
 element\_text(colour = "black", size = 12 , angle = 90,  
 hjust = 1, vjust = 0.5))