Zebrafish Dataset Practical 1 With Plots

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

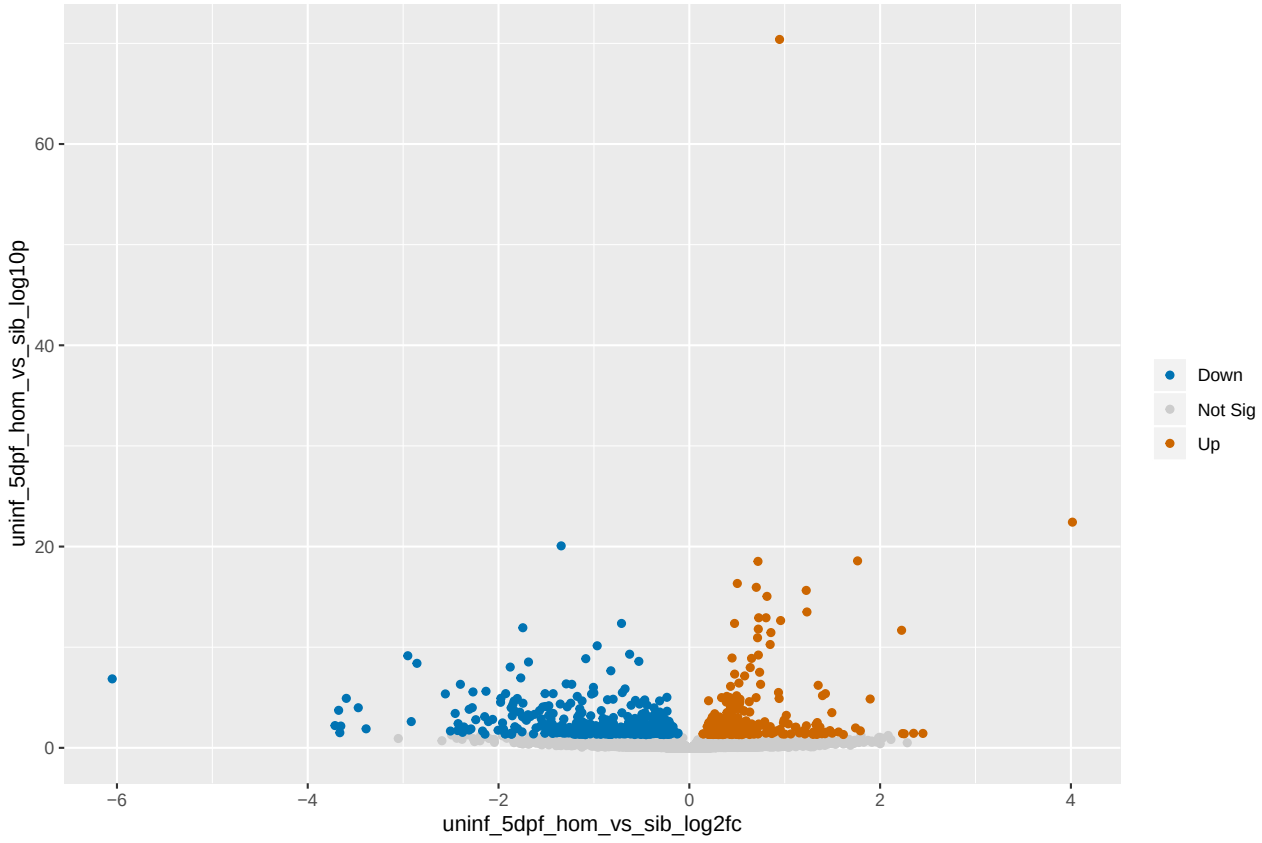
Prepare the data for making a volcano plot. We need to convert the adjusted p-values to -$\log_{10}$(adjusted p-value). Also, we are going to make a new column that marks whether a gene is significantly different or not and another one that shows whether the significant genes are up or down.

```
# 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 -$\log_{10}$(p-value) against the $\log_2$(fold change) for each gene, with the genes coloured by whether the gene is up or down or not significant.

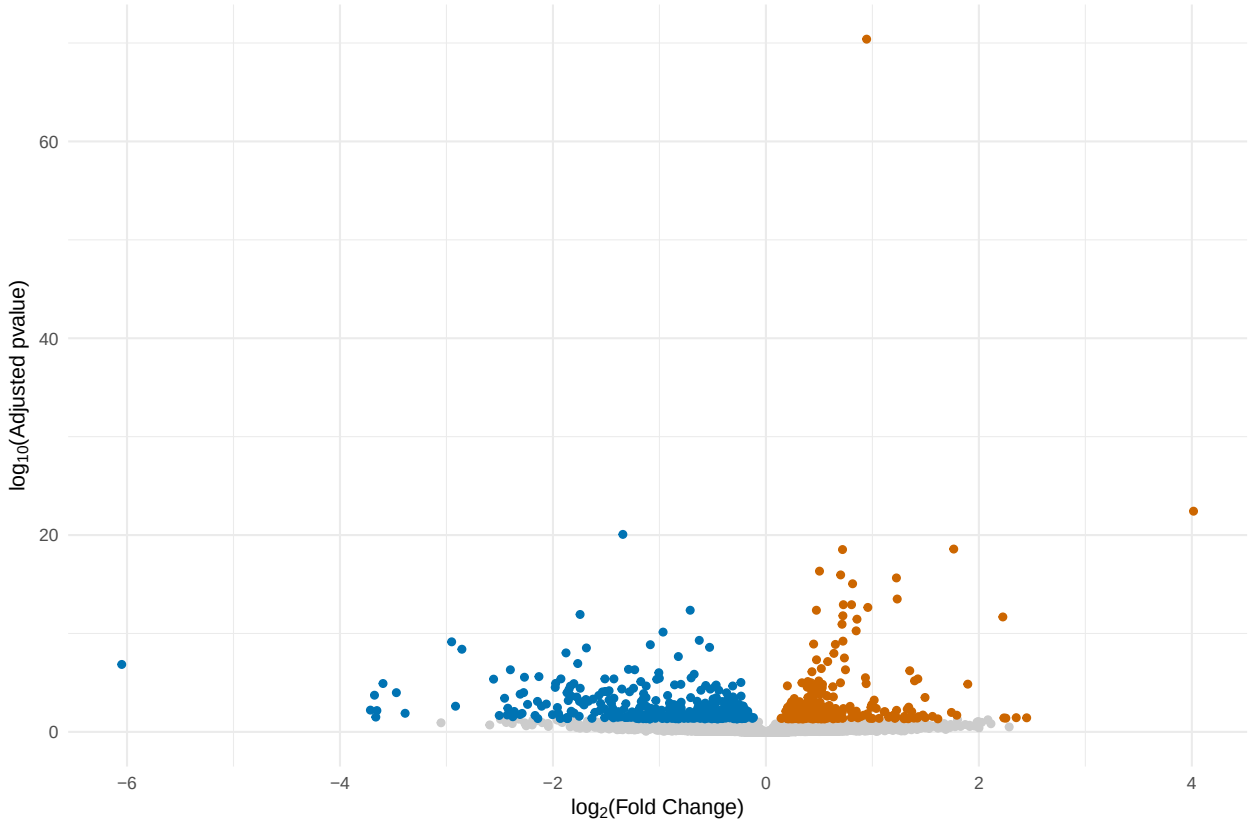
```
# plot adjusted pvalue against log2 fold change
# with up coloured in orange and down coloured in blue
volcano_plot <-
  ggplot(data = deseq_results,
  # This sets the data for the plot
        aes(x = uninf_5dpf_hom_vs_sib_log2fc, y = uninf_5dpf_hom_vs_sib_log10p,
            colour = uninf_5dpf_hom_vs_sib_up_or_down)) +
  # 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.

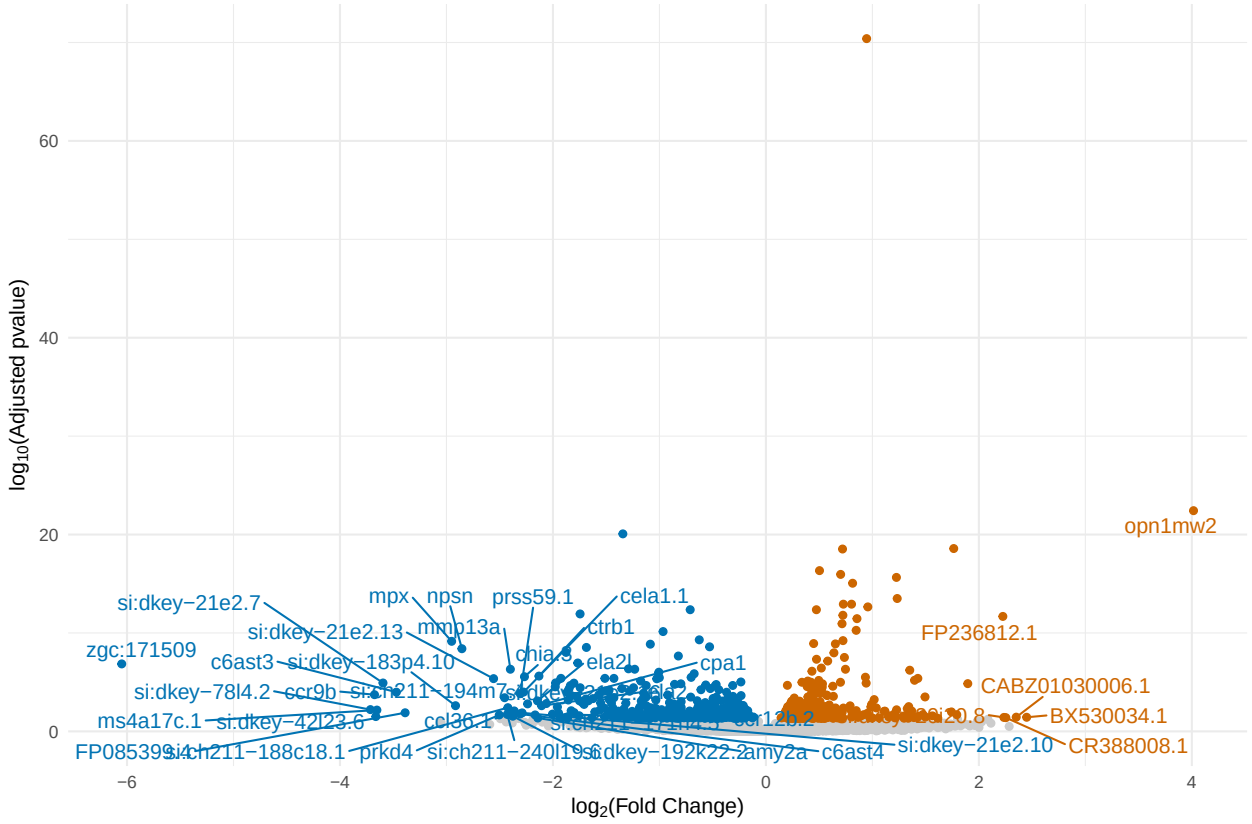
Next we can add labels to some of the points which meet certain criteria. For example we can filter the data based on $\log_2$(fold change) and adjusted p-value.

```
# 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.

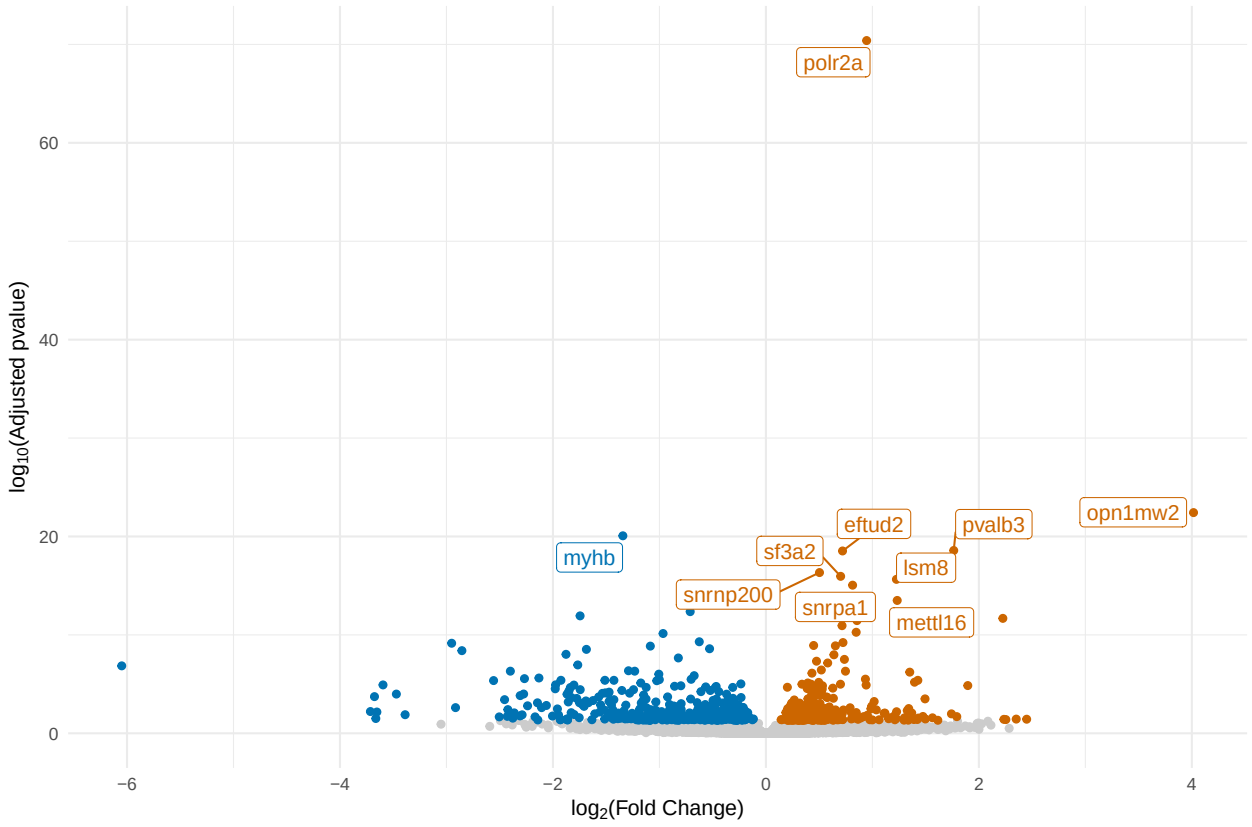
```
# load the ggrepel package
library(ggrepel)
labelled_plot <- volcano_plot +
  geom_text_repel(data = biggest_changers, aes(label = Name))
  # add text to the plot using the biggest changers data frame
  # and use the Name column as the label
```

```
  # 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.

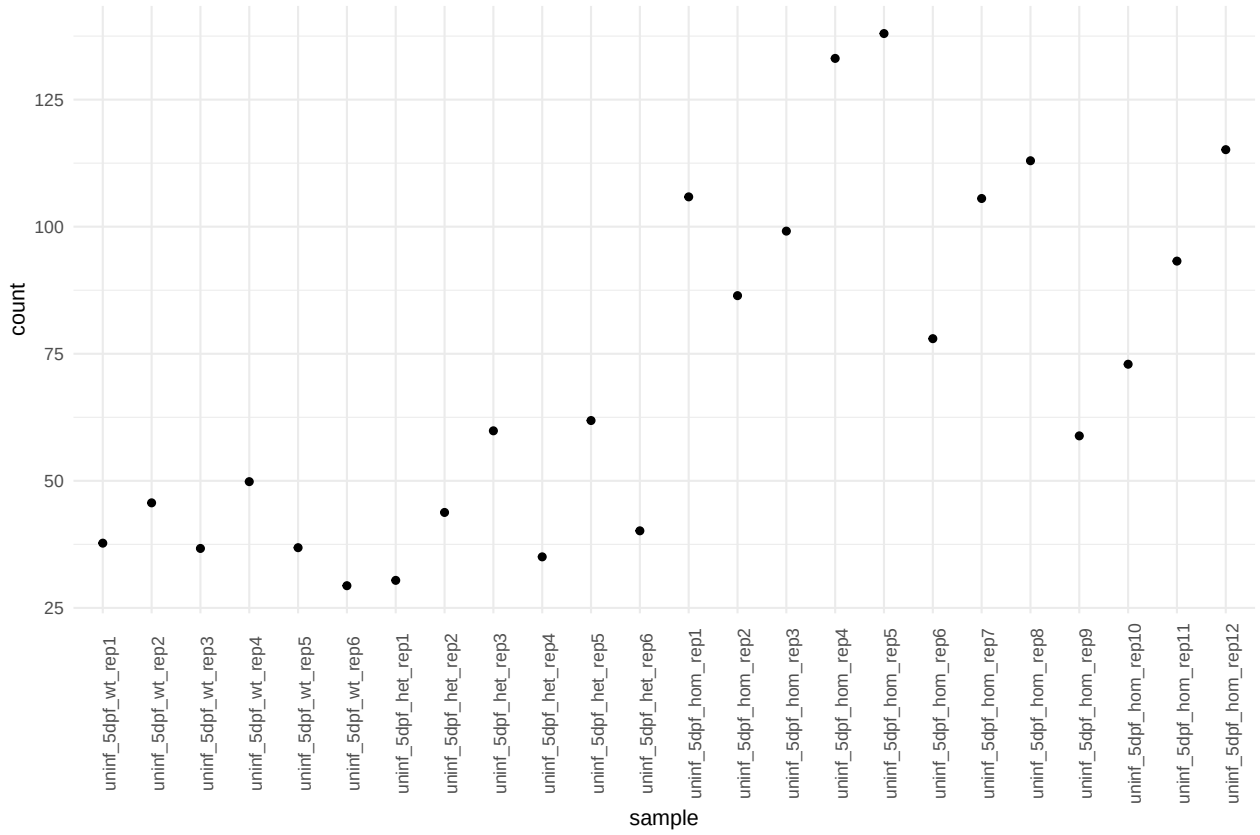
```
normalised_counts <-
  select(deseq_results, "GeneID", matches('uninf_5dpf_.*_normalised')) %>%
  rename_all(list(~ str_replace(., "_normalised_count", "")))

# 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)))
```

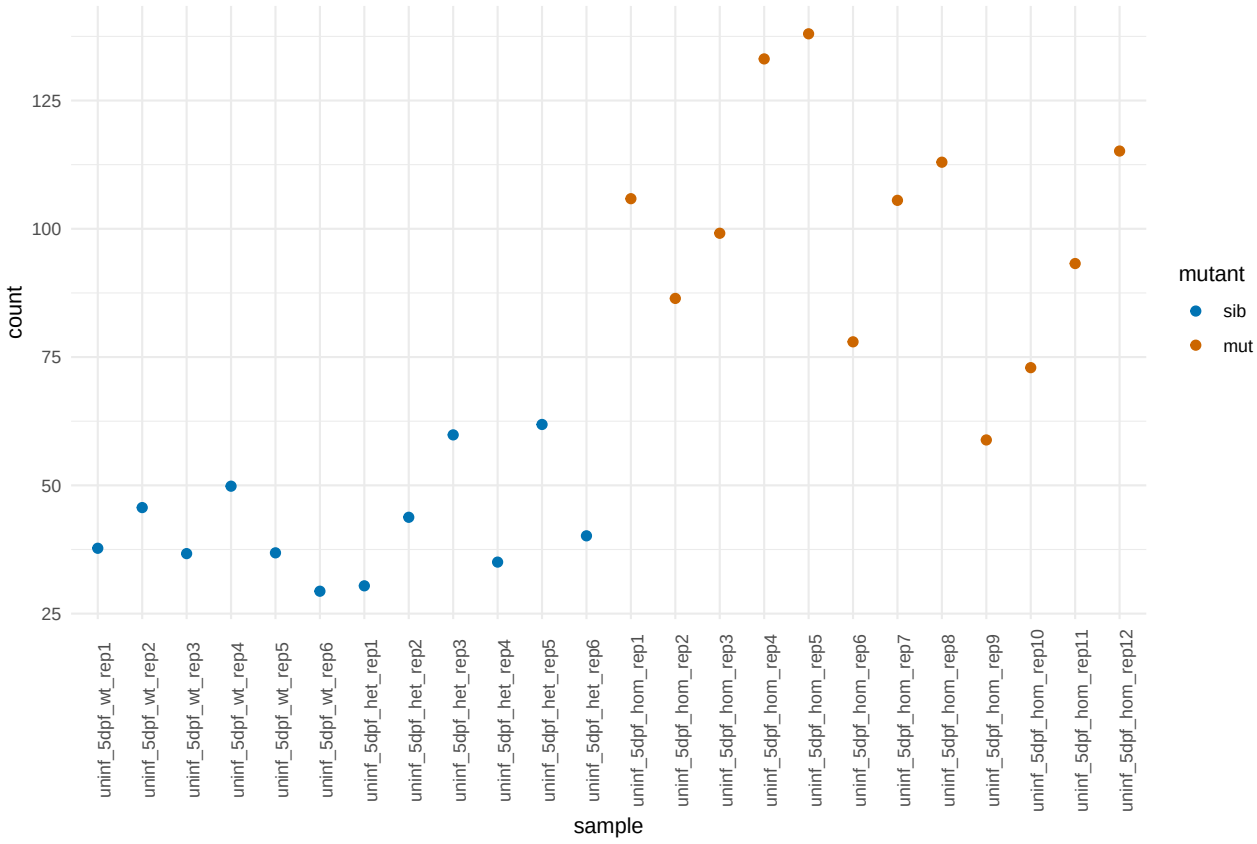
To see what it looks like now, try `head(counts_for_gene)`. To see the counts for each individual sample we can plot sample on the x-axis and count on the y, like this:

```
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))
```



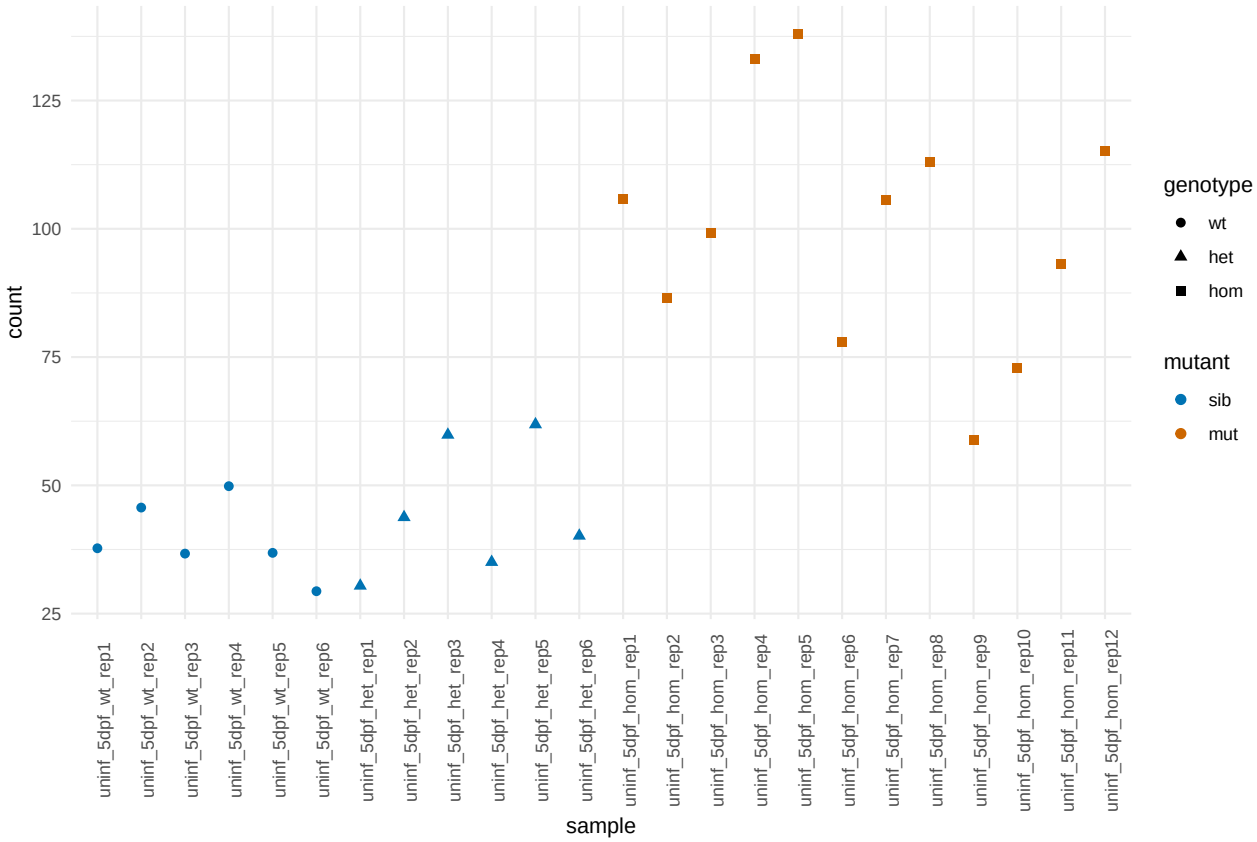
We can customise the plot to make it look nicer by using the information in the `sample_info_uninf_5dpf` object.

```
# 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))
```



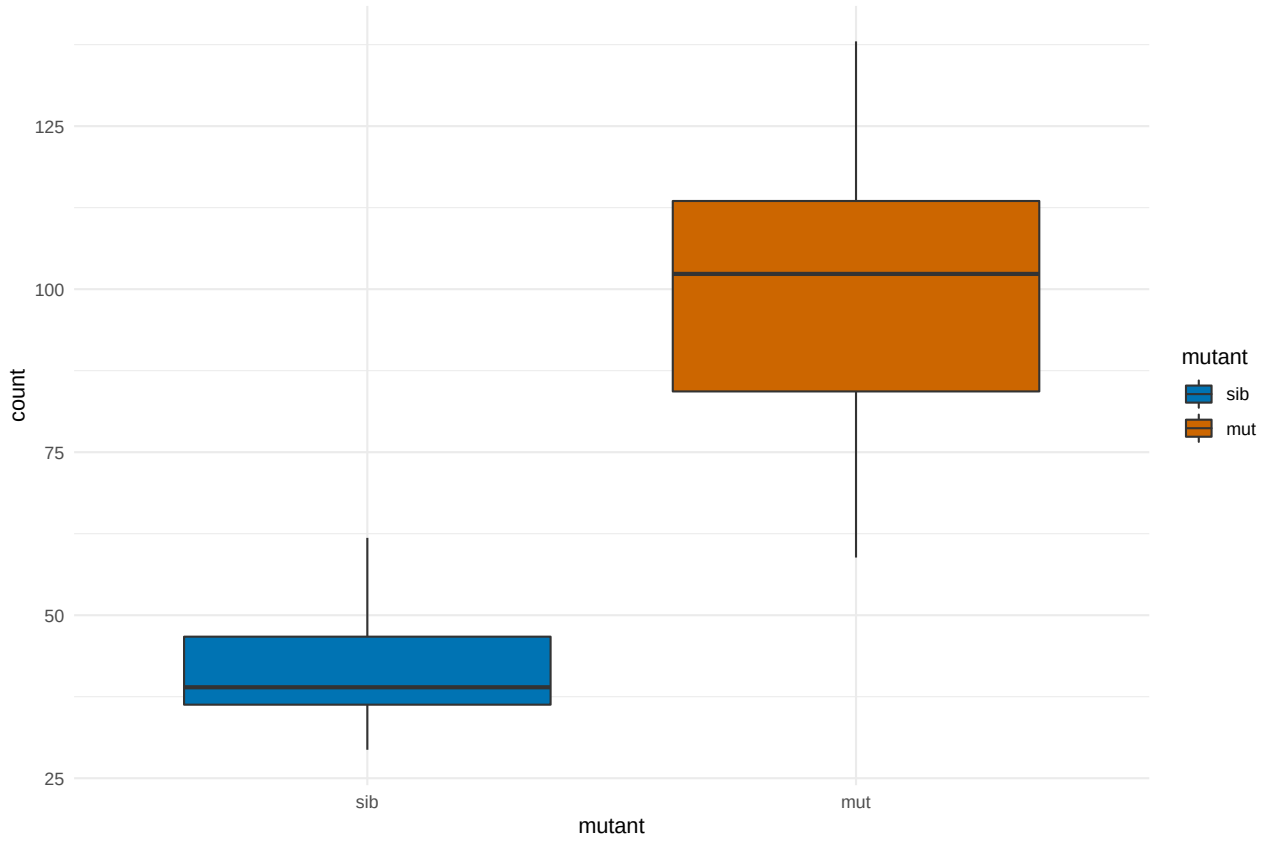
We could also have the shape of the points represent the actual genotypes, in case we want to check whether the hets are similar to the wild types.

```
# 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))
```



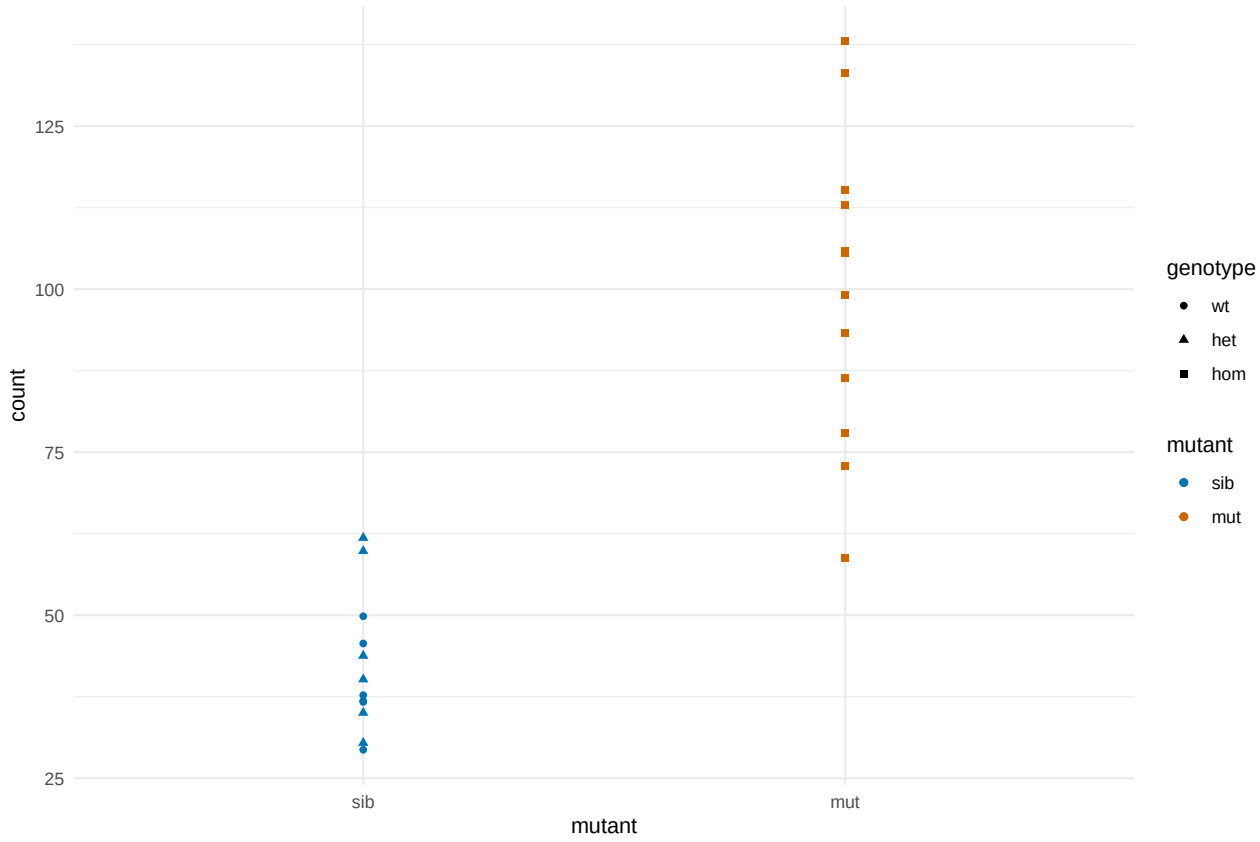
If we have a large number of samples a boxplot might be more appropriate.

```
# 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)
```



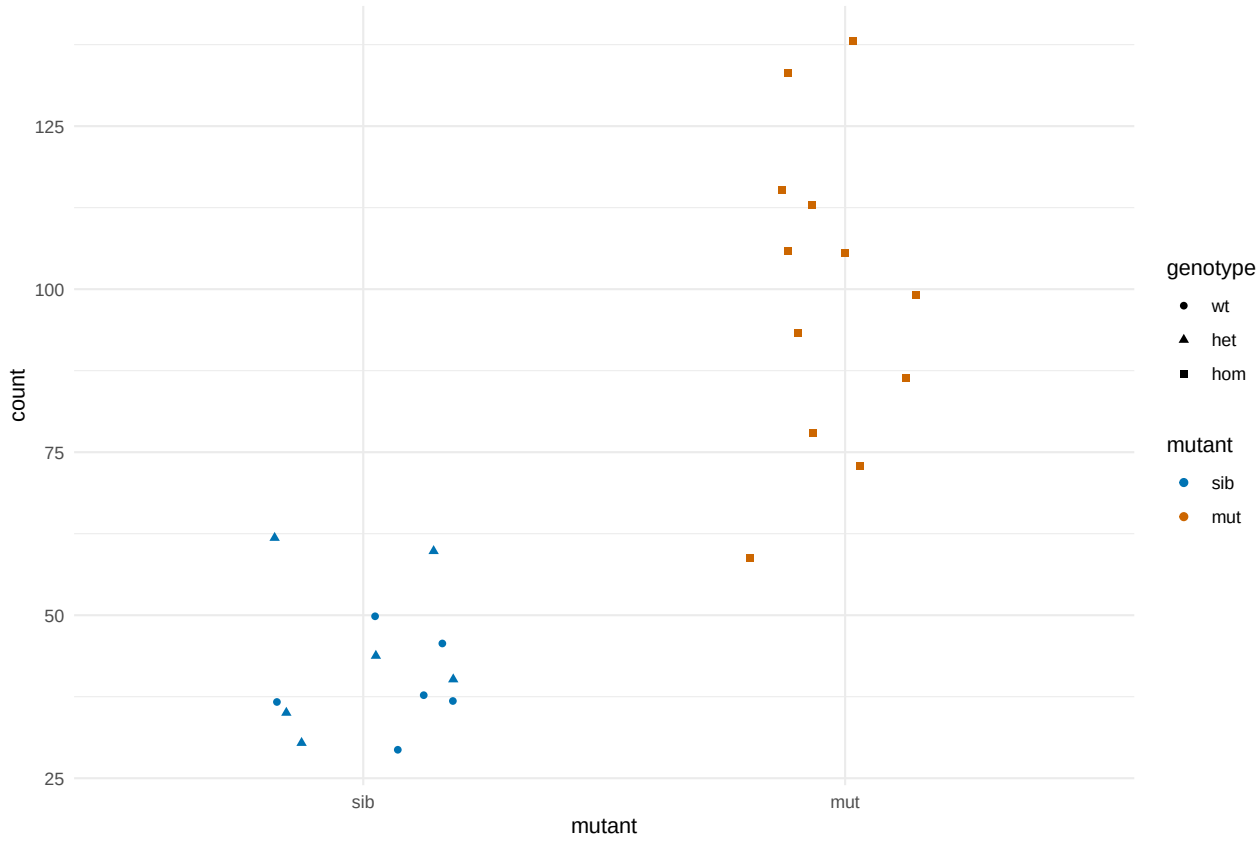
Or we could plot the points grouped by mutant status rather than by sample.

```
# 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.

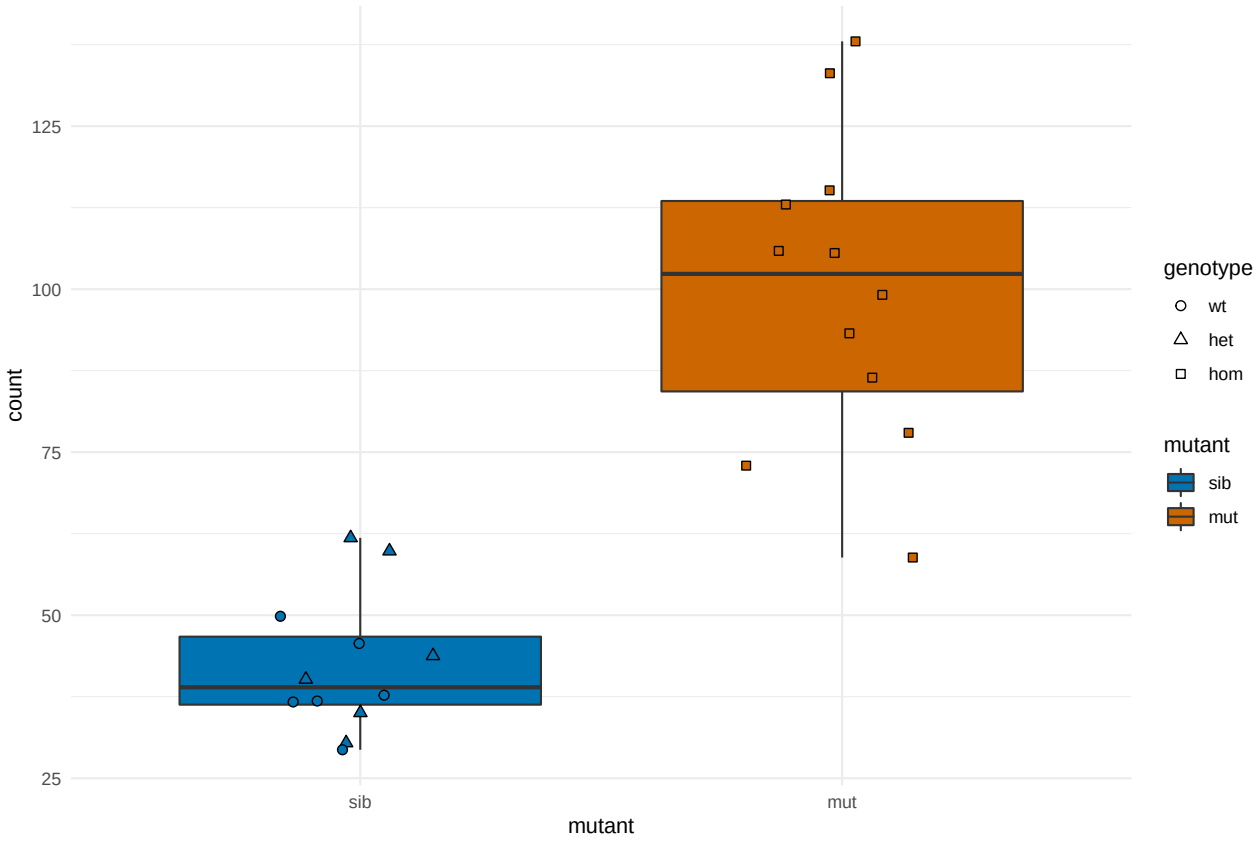
```
# jitter points to prevent overplotting
set.seed(163754)
points_jittered <- ggplot(data = counts_plus_sample_info_uninf_5dpf) +
  geom_point( aes(x = mutant, y = count,
                  shape = genotype, colour = mutant),
              position = position_jitter(width = 0.2, height = 0)) +
  theme_minimal() +
  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)
}
```

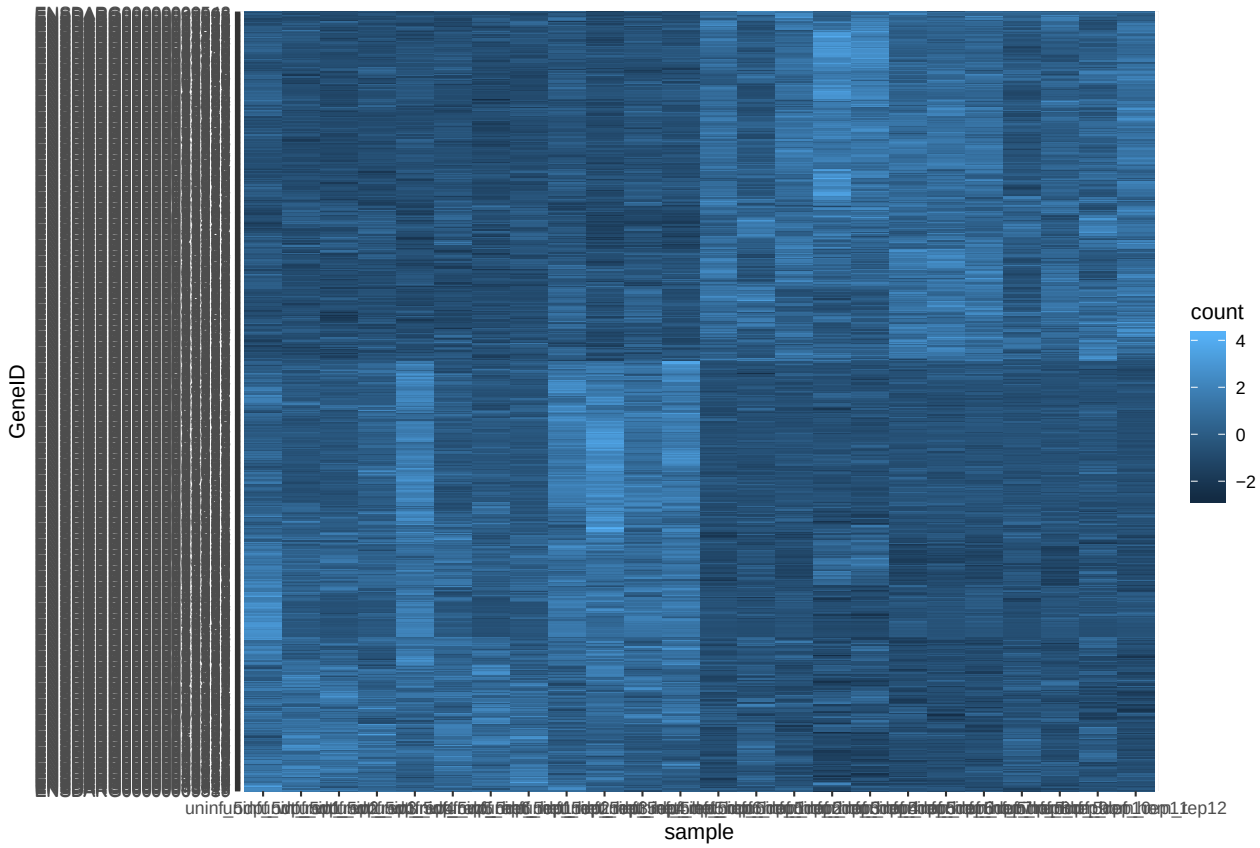
```
# cluster rows
sig_normalised_counts_scaled_clustered <- cluster(sig_normalised_counts_scaled)
```

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).

```
# 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)))
```

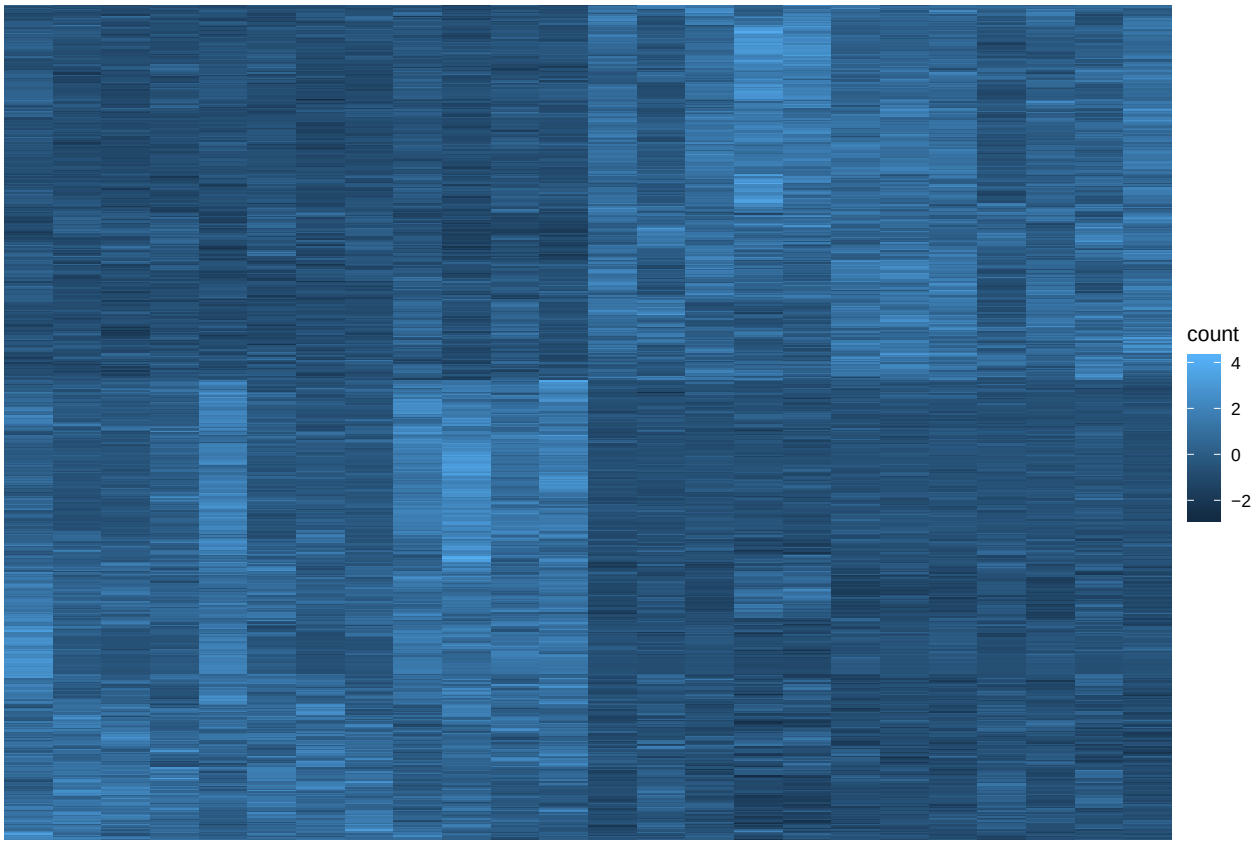
```
# transform from wide to long
sig_scaled_clustered_long <-
  pivot_longer(sig_normalised_counts_scaled_clustered, -GeneID,
               names_to = 'sample', values_to = 'count')
```

```
# 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
```



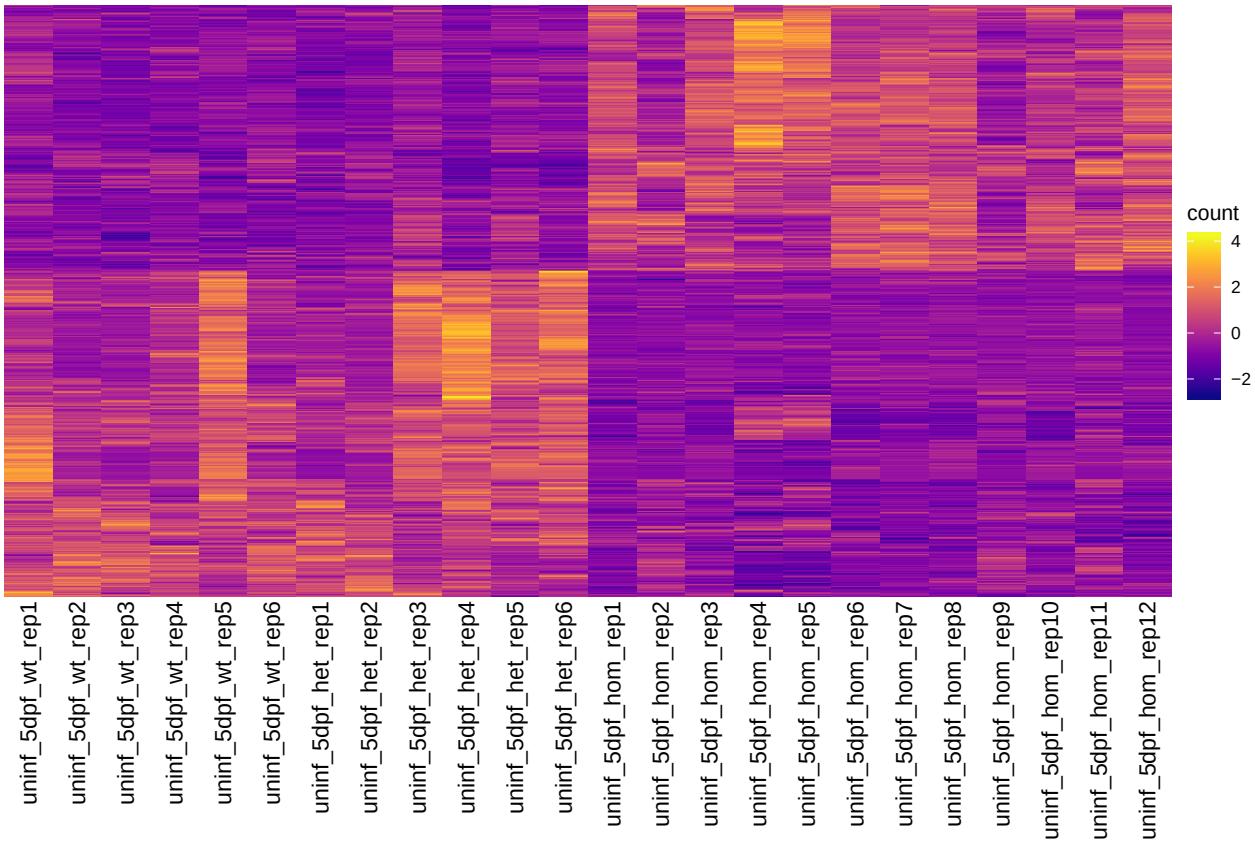
Each gene ID appears as a y-axis label and they cannot be read. Also the sample names aren't legible either. We can remove both sets of labels with `theme_void()`.

```
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).

```
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))
```


count
4
2
0
−2
uninf_5dpf_wt_rep1
uninf_5dpf_wt_rep2
uninf_5dpf_wt_rep3
uninf_5dpf_wt_rep4
uninf_5dpf_wt_rep5
uninf_5dpf_wt_rep6
uninf_5dpf_het_rep1
uninf_5dpf_het_rep2
uninf_5dpf_het_rep3
uninf_5dpf_het_rep4
uninf_5dpf_het_rep5
uninf_5dpf_het_rep6
uninf_5dpf_hom_rep1
uninf_5dpf_hom_rep2
uninf_5dpf_hom_rep3
uninf_5dpf_hom_rep4
uninf_5dpf_hom_rep5
uninf_5dpf_hom_rep6
uninf_5dpf_hom_rep7
uninf_5dpf_hom_rep8
uninf_5dpf_hom_rep9
uninf_5dpf_hom_rep10
uninf_5dpf_hom_rep11
uninf_5dpf_hom_rep12