# 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 four required files (`Amp.counts.tsv`, `Amp.samples.tsv`, `Oxy.counts.tsv` and `Oxy.samples.tsv`) in your home directory.

To begin, here are a couple of 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 (adjusted p-value < 0.05) for the amphetamine and oxycodone treatments. How do these numbers change if you reduce the adjusted p-value threshold to 0.005 or even 0.0005?
2. Using `awk`, create two new files that just contain the subset of significantly differentially expressed genes (adjusted p-value < 0.05). Keep these two files as you'll need them later in the week. Also, using cut, create two new files that just contain the Ensembl IDs of the significantly differentially expressed genes. Again, keep these two files for later.

The rest of the practical uses R.
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 <- 'Amp.counts.tsv'
```

```
# load data
deseq_results <-
  read_tsv(deseq_results_file,
           col_types = cols(Chr = col_character(),
                            Strand = col_character()))
```

Here are some functions for inspecting a data.frame.
`head` shows the top 6 rows. If the object is a tibble, only the columns that fit on the width of the page are shown.

`glimpse` shows the data frame transposed so that the columns become rows. This makes it possible to see all of the columns if they don't fit on one page width.

`View` opens up a new viewer window which display the data like a spreadsheet.

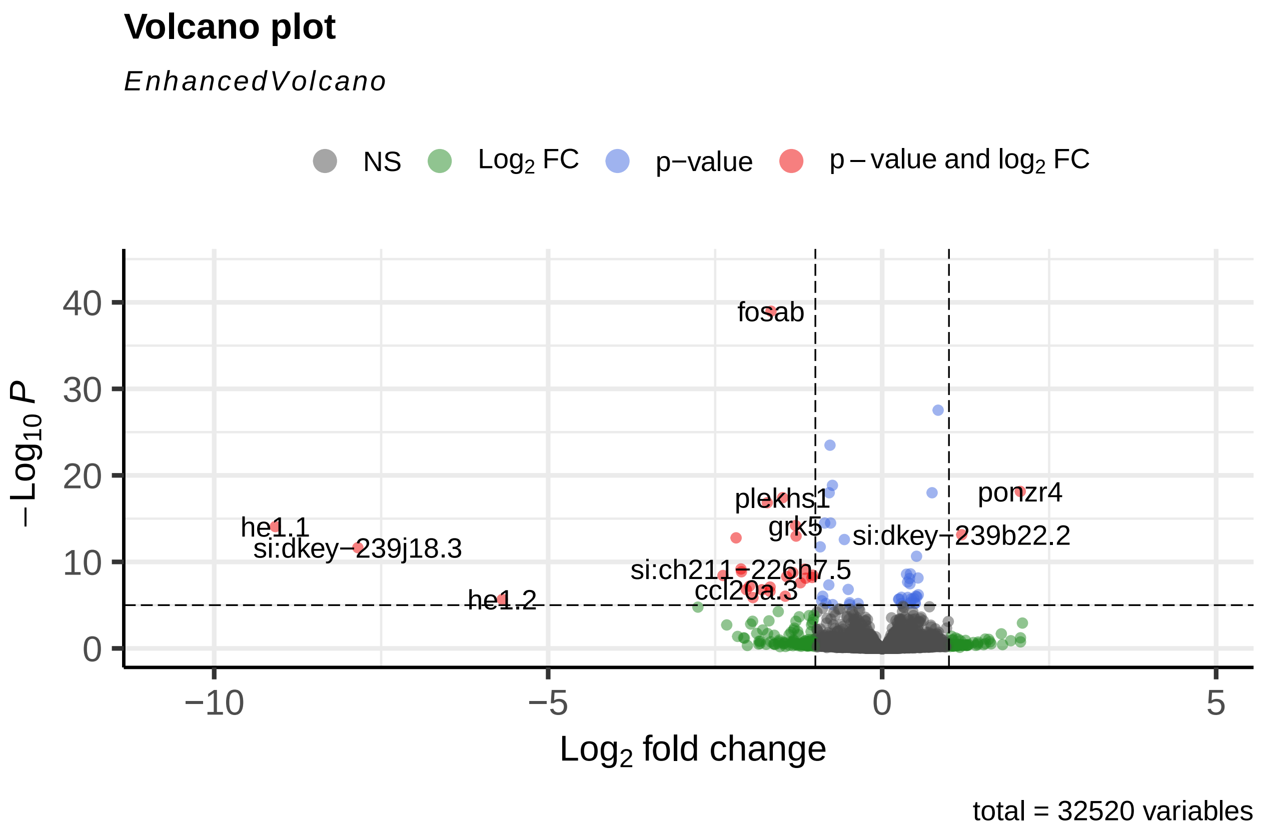
Try them out.

```
head(deseq_results)

glimpse(deseq_results)

View(deseq_results)
```

## Volcano plot

`EnhancedVolcano` is an R package for making volcano plots. The main function, `Enhanced-Volcano()`, expects a data frame with a column of $\log_2$(fold change) and one for adjusted p value. The names of the genes are supplied as a separate vector.

```r
library(EnhancedVolcano)

EnhancedVolcano(
  deseq_results, # results data frame
  lab = deseq_results$Name,
  x = 'log2fc', # column name of log2 fold change
  y = 'adjp' # column name of adjusted pvalue
)
```



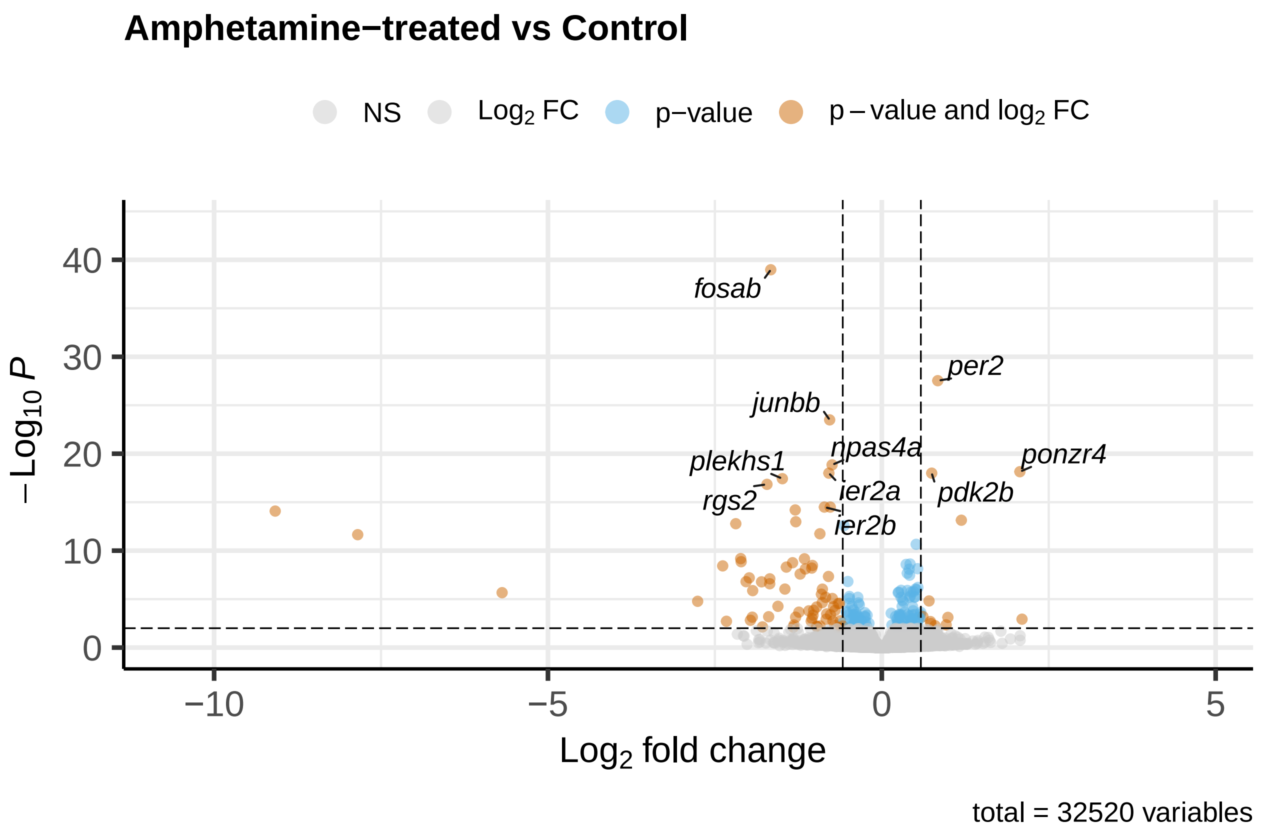
### Exercises

The `EnhancedVolcano()` function has many ways to customise the plot. Read the documentation (`?EnhancedVolcano`) and re-plot the volcano plot with these changes.

1. Change the colours of the different categories (NS, Log2FC etc.)
2. Change the p-value cut-off to 0.01
3. Change the log2[Fold Change] cut-off to `log2(1.5)`.
4. Label the 10 genes with smallest p values and change the font face to italic.

**Solutions**

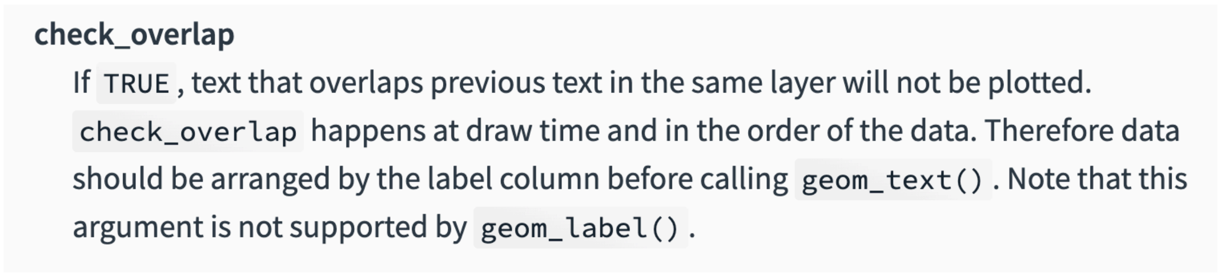
```
# get the gene names of the top 10 genes by p value
genes_to_label <- deseq_results %>%
  arrange(adjp) %>%
  pull(Name) %>%
  head(10)

EnhancedVolcano(
  deseq_results,
  lab = deseq_results$Name,
  selectLab = genes_to_label,
  drawConnectors = TRUE,
  arrowheads = FALSE,
  min.segment.length = 0.1,
  x = 'log2fc',
  y = 'adjp',
  pCutoff = 1e-02,
  FCcutoff = log2(1.5),
  labFace = "italic",
  col = c("grey80", "grey80", "#59B3E6", "#CC6600"),
  title = "Amphetamine-treated vs Control",
  subtitle = NULL
)
```



**Arrangement of text labels**

`EnhancedVolcano` adds text labels for genes, either ones above the log2fc and pvalue cut-offs, or labels supplied to the `selectLab` argument.

if the `drawConnectors` argument is set to FALSE, `EnhancedVolcano` uses `geom_text`/`geom_label`. This means that the labels are plotted directly on top of the points. Also, if `geom_text` is being used, the `check_overlap` argument is set to TRUE. This means that if any of the text labels overlap previously plotted labels they will not be plotted. This only applies to `geom_text`. If `geom_label` is used, the labels will just be plotted on top of each other.



**check_overlap**

If `TRUE`, text that overlaps previous text in the same layer will not be plotted. `check_overlap` happens at draw time and in the order of the data. Therefore data should be arranged by the label column before calling `geom_text()`. Note that this argument is not supported by `geom_label()`.

However, if `drawConnectors` is set to TRUE, the `geom_text_repel`/`geom_label_repel` functions are used. These try to arrange the labels so that they don't overlap points and don't overlap each other. They do this by adding random amounts of jitter to the labels and checking for overlaps.

This should mean that the labels avoid the points and each other. However, `EnhancedVolcano` subsets the data it gives to `geom_(text/label)_repel` to just the points to be labelled. That means `geom_(text/label)_repel` doesn't know about any of the other points and so can't avoid them.

For an example of using `ggplot2` to create a volcano plot, see the **worked examples** section.

## Heatmap

The `pheatmap` package can be used to create heatmaps.

```
library(pheatmap)
```

First you need to create a matrix of values to plot as a heatmap.
Using `filter` and `select`, subset the results to differentially expressed genes (adjp < 0.05) and select the normalised count columns. Save to an object called `sig_counts`

```
sig_counts <- deseq_results %>%
  # filter to DE genes
  filter(adjp < 0.05) %>%
  # select normalised count columns
  select(contains(' normalised count'))
```

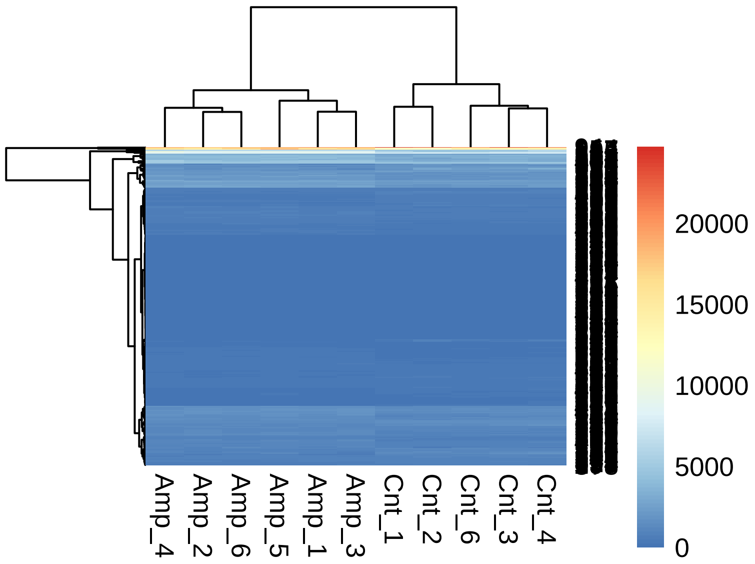
The column names all contain the string ' normalised count'. The column names get used as x-axis labels on the heatmap, so we need to remove it. We can set the column names of the data frame with the `colnames` function and use the `str_replace` function from the `stringr` package to remove the ending.
`str_replace` takes 3 arguments.

1. A vector of strings to do the substitution on
2. A pattern to look for
3. A string to replace it with

```
# This substitutes the string " normalised count" with
# the empty string ""
colnames(sig_counts) <-
  str_replace(colnames(sig_counts), " normalised count", "")
```

Plot a heatmap of the normalised counts using the `pheatmap` function

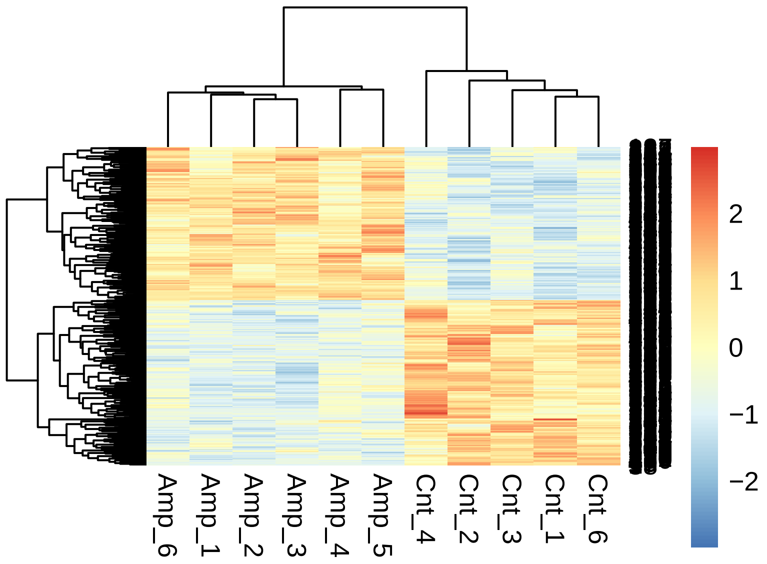
```
pheatmap(sig_counts)
```



This initial plot has some issues. First, the counts need scaling. At the moment, the colour scale is dominated by the small number of very highly expressed genes. Scaling is done by mean centering and scaling the counts by the standard deviation for each row ($\frac{x-\overline{x}}{\sigma}$, Z-score).

`pheatmap` has an option `scale`, which can either scale the values by column or row or both

```
pheatmap(sig_counts,
         scale = "row")
```

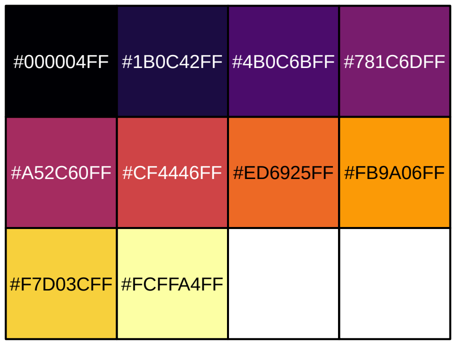


The rows and columns in the heatmap are automatically clustered and a tree for each is drawn.

The default colour scheme makes it difficult to see values in the middle of the range. Let's change the colour palette to one from the `viridis` package.

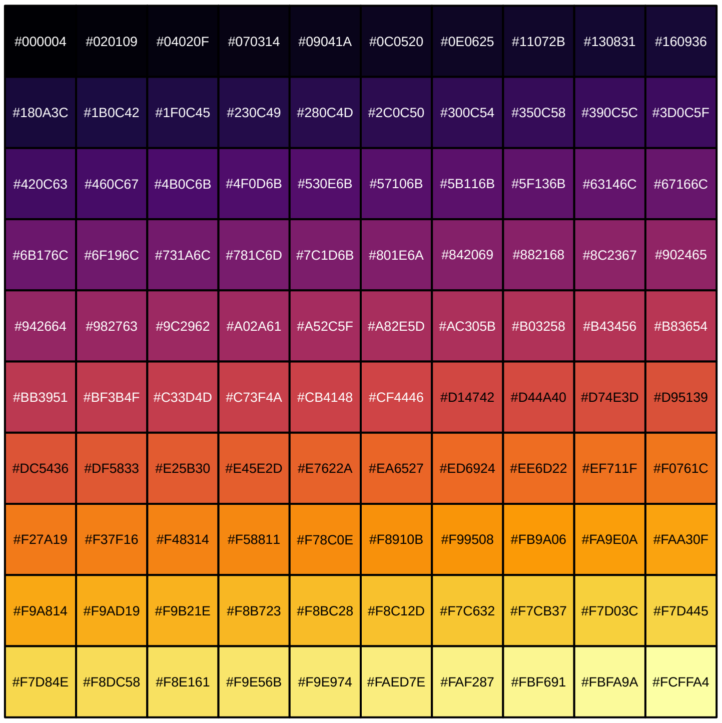
The `inferno` function, from the `viridis` package, returns a vector of n colours (10 in this case) from the inferno colour scale.

```
library(viridis)
scales::show_col(inferno(10), cex_label = 0.6)
```



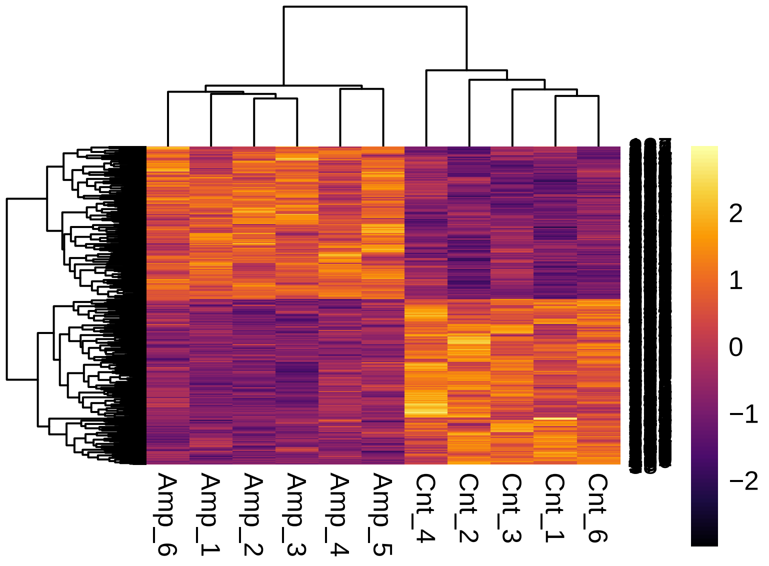
The `colorRampPalette` function returns a function to interpolate more colours between those supplied to create a smooth colour gradient.

```
scales::show_col(colorRampPalette(inferno(10))(100),
                 cex_label = 0.4)
```



We can add this to the heatmap using the `color` argument.

```
pheatmap(sig_counts,
         scale = "row",
         color = colorRampPalette(inferno(10))(100))
```



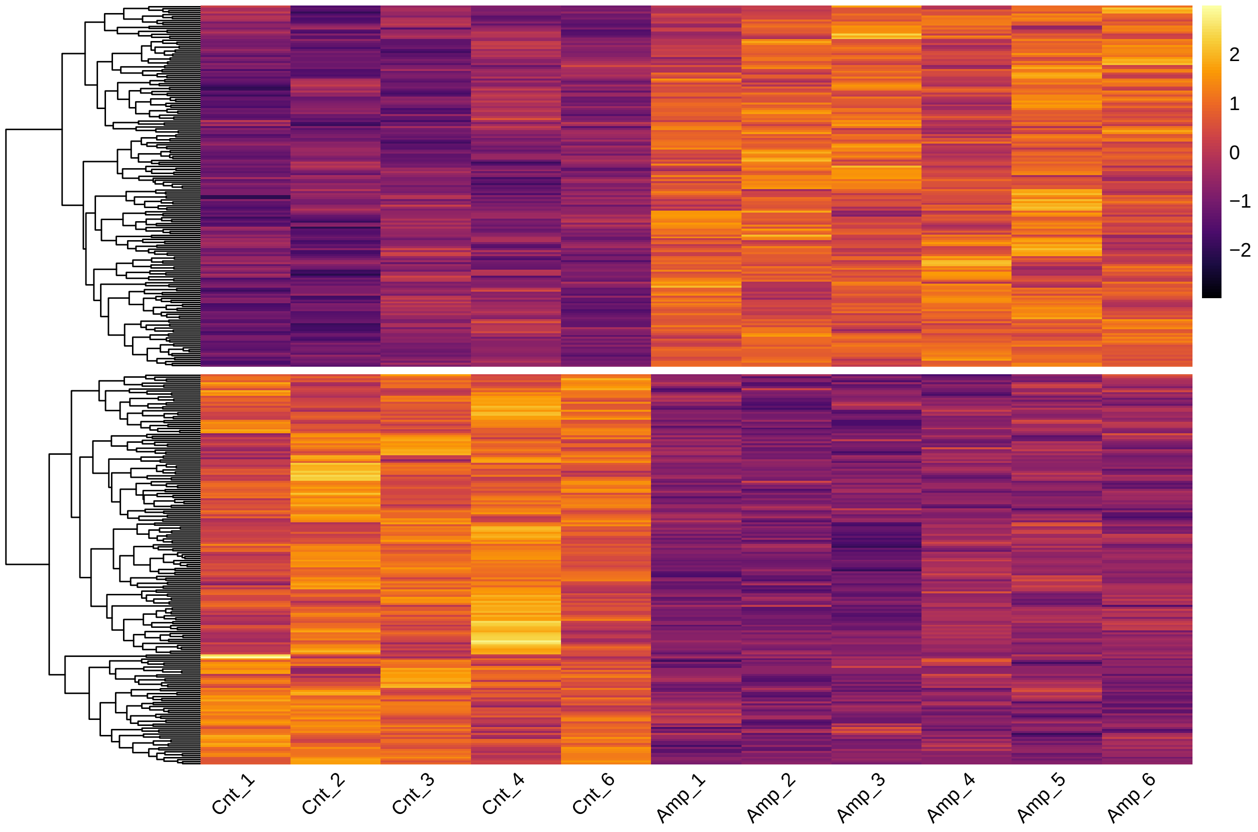
**Exercises**

Read the documentation for `pheatmap` and re-plot the heatmap with these changes.

1. Turn off plotting the gene names. There are too many genes in the heatmap to be able to read individual labels.
2. Rotate the column labels.
3. Turn off the column clustering
4. Split the heatmap in half based on the row clustering.
5. Give more room to the gene clustering tree.

**Solutions**

```
pheatmap(sig_counts,
         scale = "row",
         cluster_cols = FALSE,
         color = colorRampPalette(inferno(10))(100),
         show_rownames = FALSE,
         angle_col = 45,
         cutree_rows = 2,
         treeheight_row = 100)
```



## Count plot

To plot the normalised counts for each sample for a gene, we need a table of the sample info. The samples file has columns for the sample name and drug treatment for each sample.

```
sample_info_file <- 'Amp.samples.tsv'
```

```
sample_info <-
  read_tsv(sample_info_file,
           col_names = c('sample', 'treatment')) %>%
  # set the order of sample by the order in which they appear
  # and set levels of treatment explicitly
  mutate(sample = fct_inorder(sample),
         treatment = factor(treatment,
                            levels = c('Cnt', 'Amp')))
```

To produce a count plot, we select the `Gene` and `*normalised count` columns, make the data tidy and join in the sample information. The inner_join function from `dplyr` joins two data frames together by matching values in common columns. In this case, we are going to join the two on the `sample` column.

To make the sample names match those in the sample_info, we need to remove " normalised count" from the column names.

```
normalised_counts <-
  select(deseq_results, Gene, contains('normalised')) %>%
  # make data tidy
  pivot_longer(cols = -Gene,
               names_to = "sample",
               values_to = "count") %>%
  # rename columns by removing " normalised count"
  # and set levels of sample using sample info
  mutate(sample = str_replace(sample, " normalised count", ""),
         sample = factor(sample,
                         levels = levels(sample_info$sample))) %>%
  # join in sample information
  inner_join(sample_info)
```

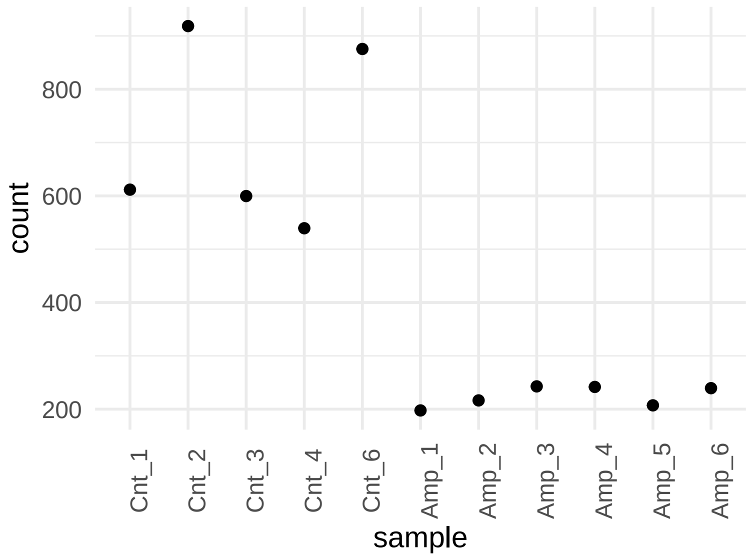
Then we filter to get the counts for a specific gene.

```
# get a specific gene
# ENSDARG00000031683 == fosab
counts_for_gene <- filter(normalised_counts,
                          Gene == "ENSDARG00000031683")
```

To see what the `counts_for_gene` object looks like, try `head(counts_for_gene)`.

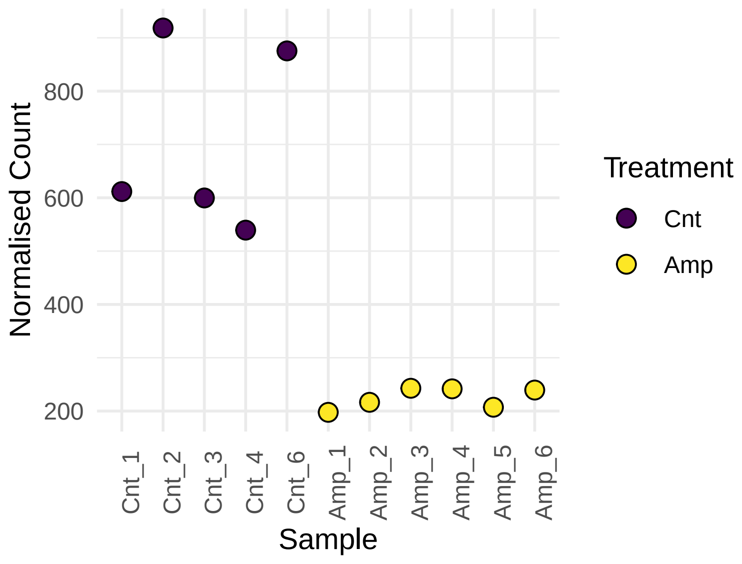
To see the counts for each individual sample we can plot sample on the x-axis and count on the y, like this:

```
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))

print(basic_count_plot)
```



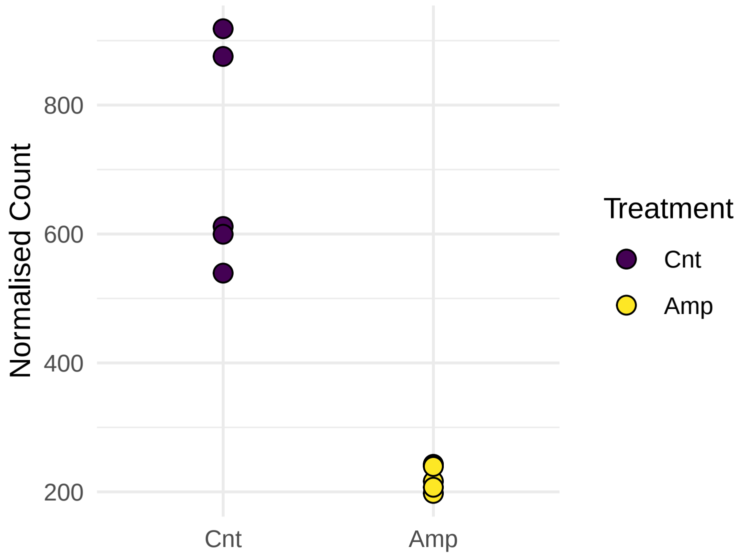
We can customise the plot to make it look nicer by colouring the points by the treatment variable.

```
# plot as points in different colours
sample_count_plot_coloured <- ggplot(data = counts_for_gene) +
  # fill points by treatment variable
  geom_point( aes(x = sample, y = count, fill = treatment),
              # shape names can be used as well as numbers
              size = 3, shape = 'circle filled') +
  # fill using the viridis scale
  scale_fill_viridis_d() +
  # tidy up the labels
  labs(x = "Sample", y = "Normalised Count", fill = "Treatment") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90))

print(sample_count_plot_coloured)
```



With this plot, we can see the normalised count value for each individual sample, but with lots of samples this will become unwieldy. Another option is to group the points by the `treatment` variable.

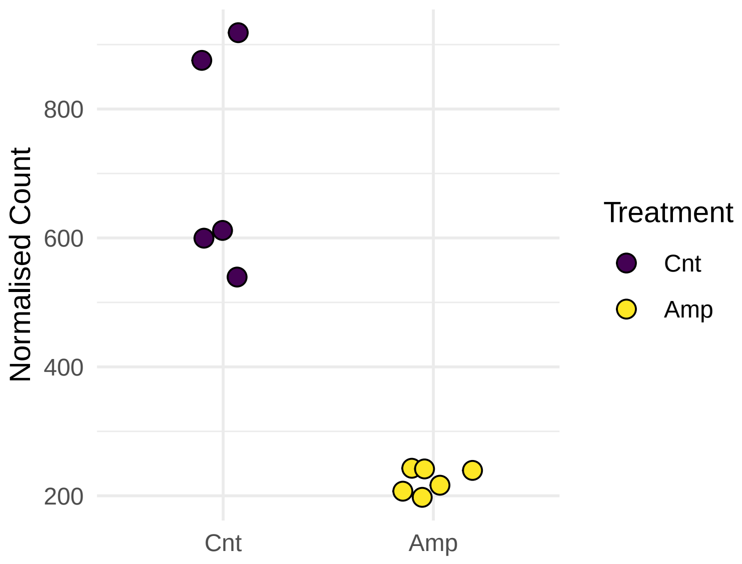
```
# plot points by treatment status
points_by_treatment <- ggplot(data = counts_for_gene) +
  geom_point( aes(x = treatment, y = count, fill = treatment),
              size = 3, shape = 'circle filled') +
  scale_fill_viridis_d() +
  labs(y = "Normalised Count", fill = "Treatment") +
  theme_minimal() +
  # this removes the x-axis title
  theme(axis.title.x = element_blank())

print(points_by_treatment)
```



The points for each treatment group 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 to spread the points out.

The `position` argument of `geom_point` is used to adjust the position of the points.
The `position_jitter` function adds a small value to both the x and y values.
Use the `width` and `height` arguments to control how large the spread of values is.
The `seed` argument makes the jitter reproducible.

```r
# jitter points to prevent overplotting
points_jittered <- ggplot(data = counts_for_gene) +
  geom_point( aes(x = treatment, y = count, fill = treatment),
              size = 3, shape = 'circle filled',
              position = position_jitter(width = 0.2,
                                         height = 0,
                                         seed = 16354)) +
  scale_fill_viridis_d() +
  labs(y = "Normalised Count", fill = "Treatment") +
  theme_minimal() +
  theme(axis.title.x = element_blank())

print(points_jittered)
```

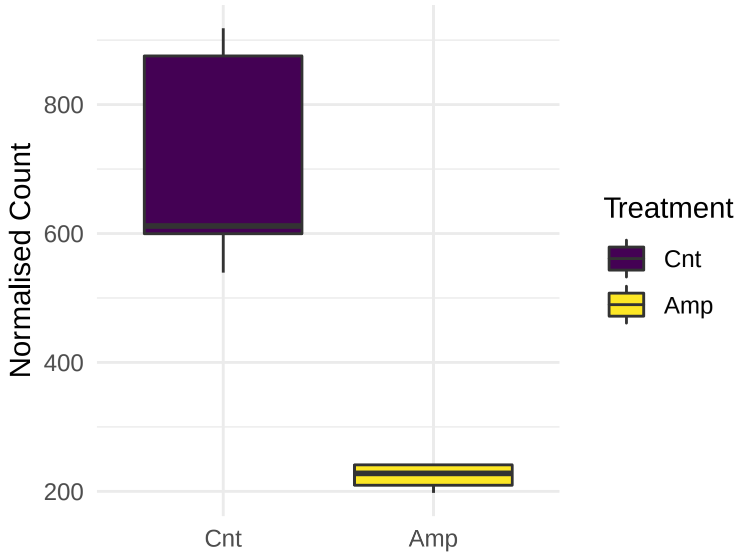


**Exercises**

1. Try plotting a boxplot grouped by `treatment` instead of points for each sample.

**Solutions**

```
# boxplot
basic_boxplot <- ggplot(data = counts_for_gene) +
  geom_boxplot( aes(x = treatment, y = count, fill = treatment)) +
  scale_fill_viridis_d() +
  labs(y = "Normalised Count", fill = "Treatment") +
  theme_minimal() +
  theme(axis.title.x = element_blank())

print(basic_boxplot)
```



## Worked Examples

### Volcano Plot

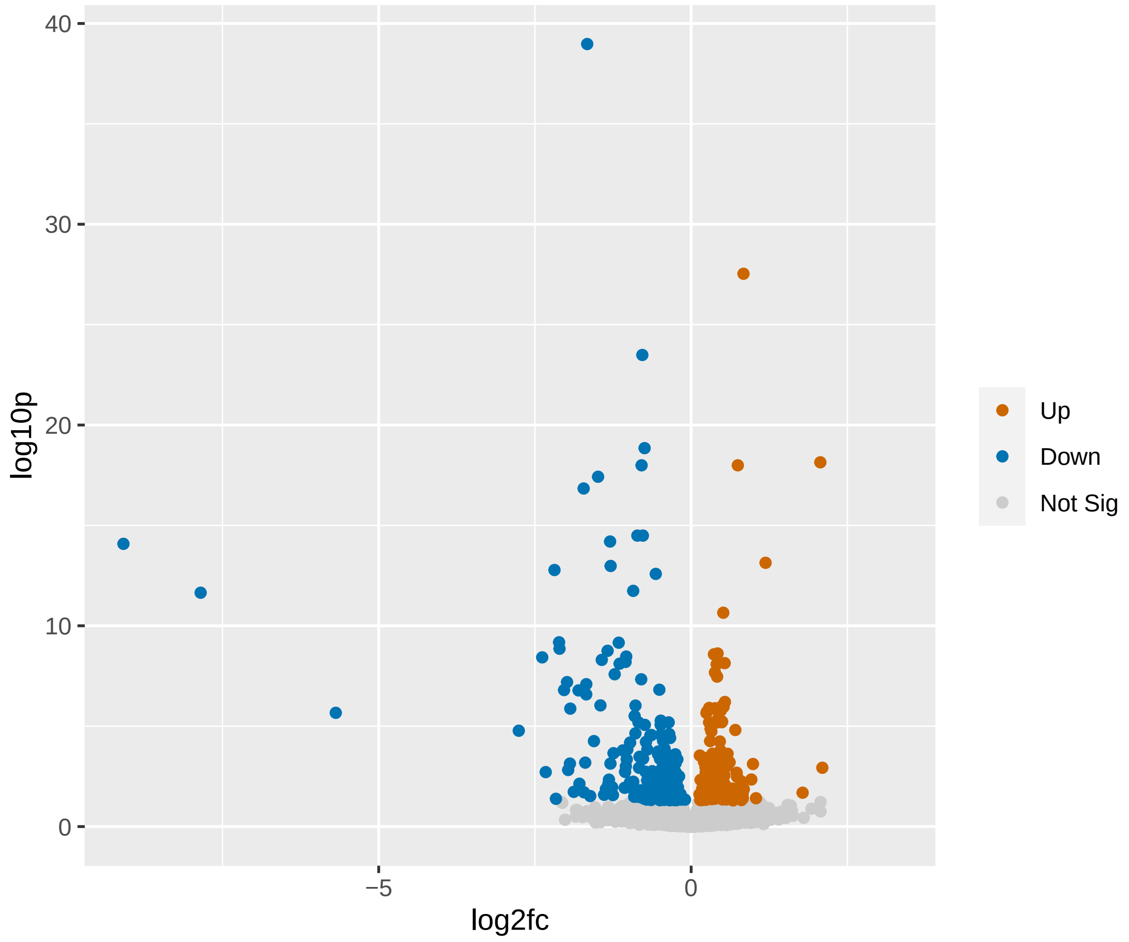
Prepare the data for making a volcano plot. We need to convert the adjusted p-values to -$\log_{10}$(adjusted p-value). Also, we are going to make a new column that marks whether a gene is significantly different or not and another column that shows whether genes are up or down or not differentially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

```
# make -log10p column
deseq_results <-
  mutate(deseq_results, log10p = -log10(adjp),
         sig = !is.na(adjp) & adjp < 0.05,
         up_down = case_when(
           sig & log2fc > 0 ~ 'Up',
           sig & log2fc < 0 ~ 'Down',
           TRUE ~ 'Not Sig'
         )) %>%
  arrange(desc(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 <-
  # This sets the data for the plot
  # and specifies we want to plot log10pval against log2fc
  # and colour it by the up_down column
  # these aesthetics will apply to any other geoms added
  ggplot(data = deseq_results,
         aes(x = log2fc, y = log10p,
             colour = up_down)) +
  # this says we want to plot the data as points
  geom_point() +
  # and this explicitly sets the colours for the 3 categories
  # and removes the legend title
  scale_colour_manual(name = "",
    values = c(Up = '#cc6600', Down = '#0073b3',
               `Not Sig` = "grey80"))
```

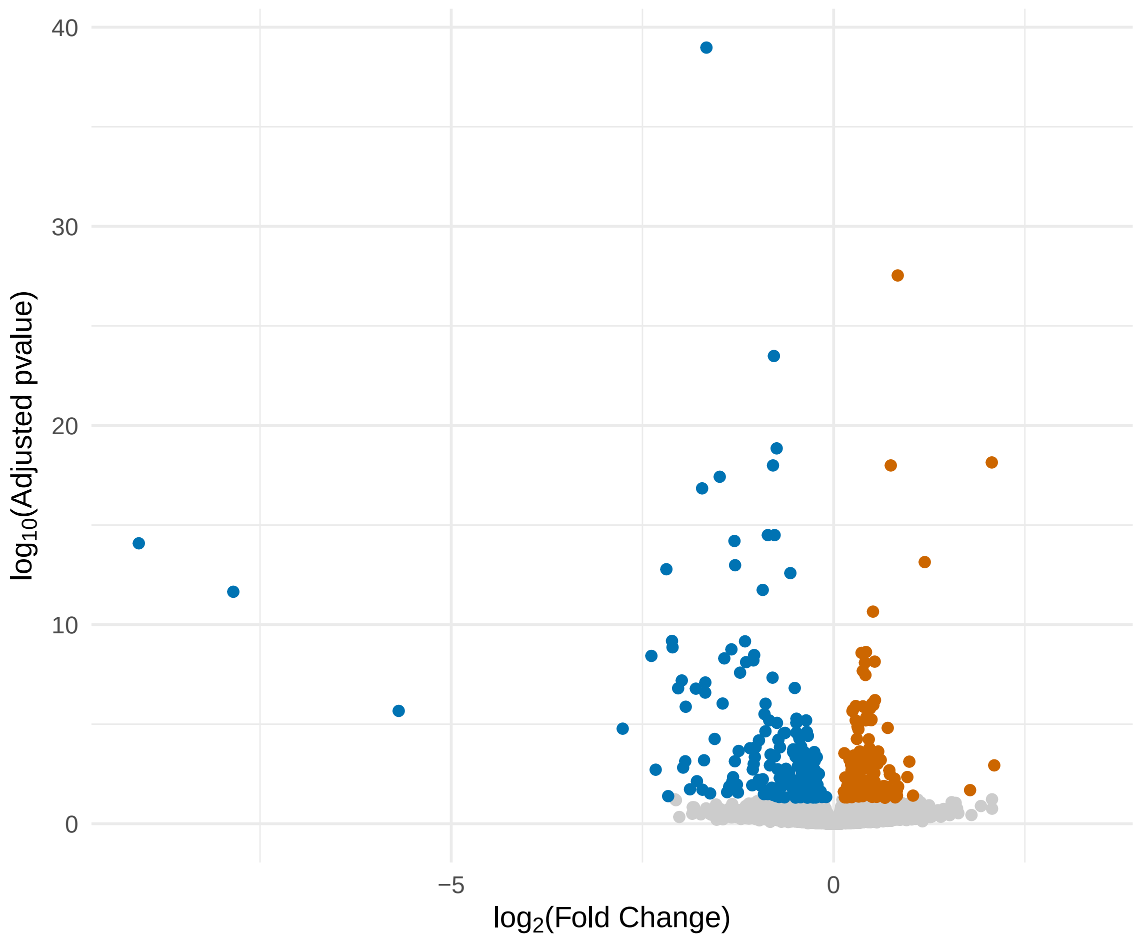
To print the ggplot object use `print(volcano_plot)`.



Now that we have a basic plot we can add things to the same plot object using the + operator and save the new plot object

For example, we can take the volcano plot and remove the legend and make better axis titles. Lastly we can change the grey background.

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

print(volcano_plot)
```



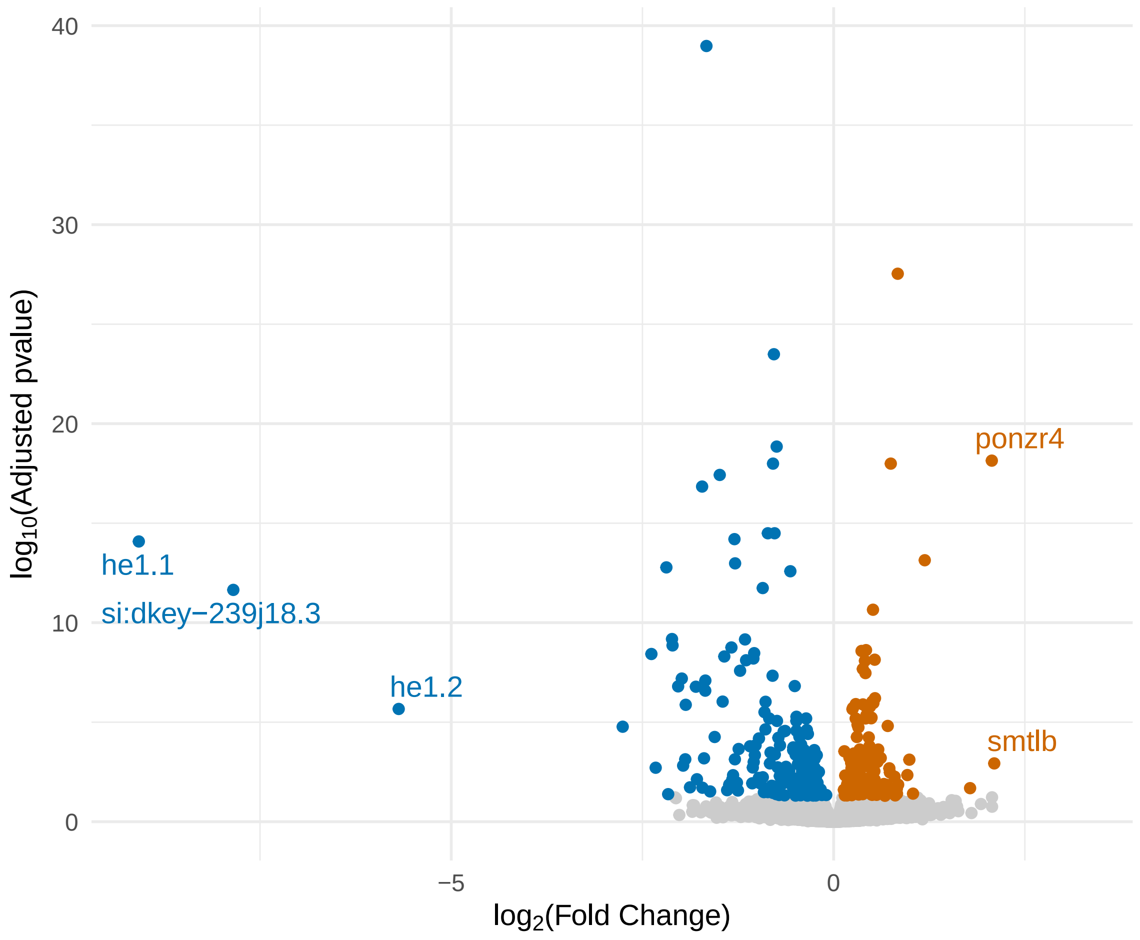
The code above shows how ggplot objects can be built up by creating a basic object and then progressively adding more to it.

Next we can add labels to some of the points which meet certain criteria. Let's label the biggest changers.

We will add the labels using the `ggrepel` https://github.com/slowkow/ggrepel package. This is a package designed to position point labels on plots by avoiding the points and other labels, so that all the labels are legible.
So that `ggrepel` knows where all the points are we need to use the whole data frame. But, we only want a few labels, so we need to make a new column that is an empty string for any points we don't want to label and has the gene name for the ones we do want to label.

```
# create new column for names of genes we want to label
deseq_results <- deseq_results %>%
  mutate(gene_label = case_when(
    up_down == 'Down' & log2fc < -3 ~ Name,
    up_down == 'Up' & log2fc > 2 ~ Name,
    TRUE ~ ""
  ))
```

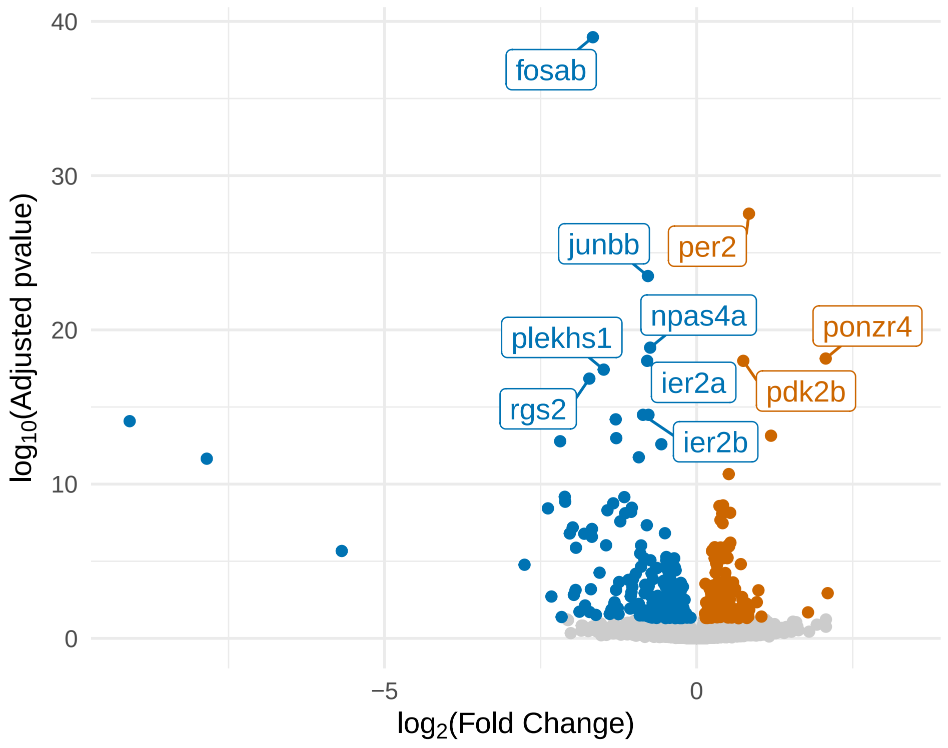
```
# load the ggrepel package
library(ggrepel)
# add geom_text_repel
# since the x, y and colour aesthetics are defined in the
# ggplot call only one more aesthetic (label) is required
labelled_plot <-
  ggplot(data = deseq_results,
         aes(x = log2fc, y = log10p,
             colour = up_down)) +
  geom_point() +
  geom_text_repel(aes(label = gene_label)) +
  scale_colour_manual(name = "",
    values = c(Up = '#cc6600', Down = '#0073b3',
               `Not Sig` = "grey80")) +
  guides(colour = "none") +
  labs(x = expr(log[2]*'(Fold Change)'),
       y = expr(log[10]*'(Adjusted pvalue)')) +
  theme_minimal()
print(labelled_plot)
```



One thing to notice here is that I've had to recreate the entire plot object rather than adding to it. That's because the plot object saves a copy of the original data frame.  So any changes you make to the data frame will not be reflected in the plot object.

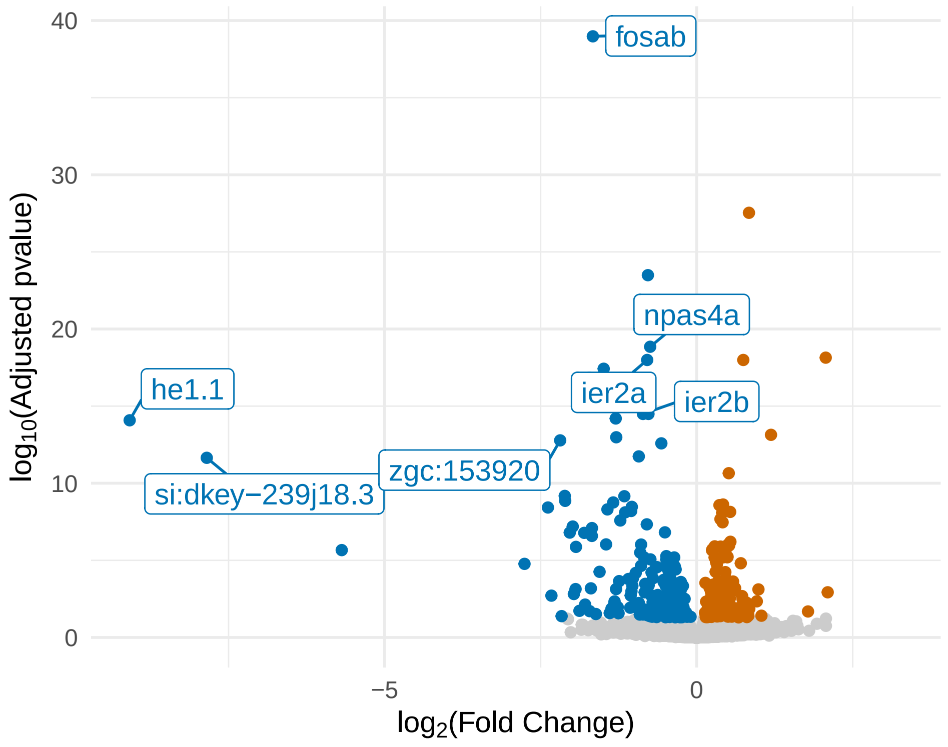
Or we could do just the top 10 genes by adjusted p-value.

```
# get the top 10 genes by adjusted pvalue
top_10_genes <-
  arrange(deseq_results, adjp) %>%
  # sort by adjusted pvalue and get Gene ID
  head(10) %>% pull('Gene')

# remake the gene_label column
deseq_results <- deseq_results %>%
  mutate(gene_label = case_when(
    # this tests whether the Gene ID exists
    # in the top_10_genes vector
    Gene %in% top_10_genes ~ Name,
    TRUE ~ ""
  ))
# this uses geom_label_repel() which draws boxes behind the text
top_10_plot <- ggplot(data = deseq_results,
  aes(x = log2fc, y = log10p, colour = up_down)) +
  geom_point() +
  geom_label_repel(aes(label = gene_label), seed = 765,
                   min.segment.length = 0) +
  scale_colour_manual(values = c(Up = '#cc6600', Down = '#0073b3',
                               `Not Sig` = "grey80")) +
  guides(colour = "none") +
  labs(x = expr(log[2]*'(Fold Change)'),
       y = expr(log[10]*'(Adjusted pvalue)')) +
  theme_minimal()
print(top_10_plot)
```



Or an arbitrary vector of gene ids.

```
# start with a vector of gene ids
gene_ids <- c("ENSDARG00000031683", "ENSDARG00000055752",
              "ENSDARG00000099195", "ENSDARG00000086881",
              "ENSDARG00000023656", "ENSDARG00000070041",
              "ENSDARG00000089806")

# remake the gene_label column
deseq_results <- deseq_results %>%
  mutate(gene_label = case_when(
    Gene %in% gene_ids ~ Name,
    TRUE ~ ""
  ))

specific_genes_volcano_plot <- ggplot(data = deseq_results,
    aes(x = log2fc, y = log10p, colour = up_down)) +
  geom_point() +
  geom_label_repel(aes(label = gene_label), seed = 765,
                   min.segment.length = 0) +
  scale_colour_manual(values = c(Up = '#cc6600',
              Down = '#0073b3', `Not Sig` = "grey80")) +
  guides(colour = "none") +
  labs(x = expr(log[2]*'(Fold Change)'),
       y = expr(log[10]*'(Adjusted pvalue)')) +
  theme_minimal()

print(specific_genes_volcano_plot)
```



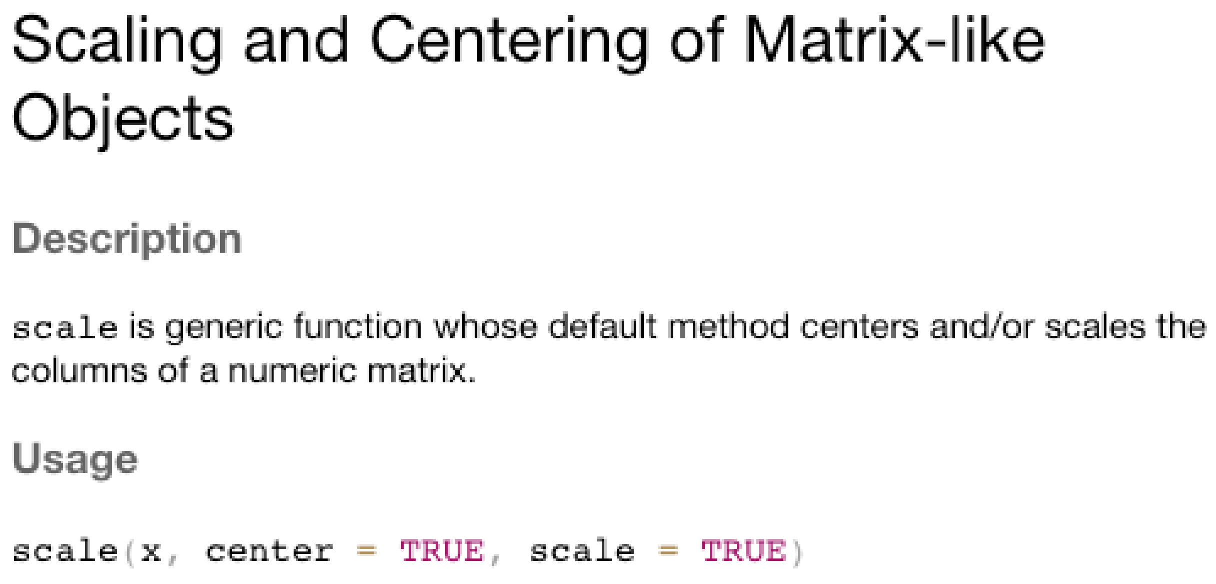
## Heatmap

For this we're going to use a subset of the data so that it is easy to see what is going on.

```r
# filter to DE genes and take the first 5 genes and 6 sample columns
sig_counts <- deseq_results %>%
  filter(adjp < 0.05) %>%
  arrange(adjp) %>%
  slice_head(n = 5) %>%
  select(Gene, contains(' normalised count')) %>%
  rename_with(.fn = function(x){
    sub(" normalised count", "", x) } ) %>%
  select(Gene, Cnt_1, Cnt_2, Cnt_3, Amp_1, Amp_2, Amp_3)

kable(sig_counts)
```

| Gene | Cnt_1 | Cnt_2 | Cnt_3 | Amp_1 | Amp_2 | Amp_3 |
|---|---:|---:|---:|---:|---:|---:|
| ENSDARG00000031683 | 611.66318 | 918.44271 | 599.64194 | 197.61987 | 216.40538 | 242.72234 |
| ENSDARG00000034503 | 633.70510 | 715.18080 | 597.14343 | 1079.09638 | 1330.19823 | 1266.16969 |
| ENSDARG00000104773 | 875.06409 | 1003.13517 | 787.86288 | 512.89249 | 457.62790 | 545.64747 |
| ENSDARG00000055752 | 751.62935 | 703.88847 | 687.08972 | 397.99722 | 402.03753 | 442.44269 |
| ENSDARG00000087440 | 17.63353 | 11.29233 | 16.65672 | 68.93716 | 57.57574 | 74.53678 |

First we need to mean centre and scale the counts for each gene. The `scale` function will do this, but it works on columns rather than rows.



# Scaling and Centering of Matrix-like Objects

### Description

`scale` is generic function whose default method centers and/or scales the columns of a numeric matrix.

### Usage

```r
scale(x, center = TRUE, scale = TRUE)
```

So, we have to transpose the matrix of counts, so that the columns represent the genes and the rows are samples. We can do this using the `t` function.
The `scale` function expects a matrix where all the columns are numeric, so we have to remove the Gene column. I've added the gene ids to the rownames of the matrix with `set_rownames`

```
sig_counts %>%
  select(-Gene) %>% # remove Gene column
  as.matrix() %>% # turn into a matrix
  # set the rownames to Gene
  magrittr::set_rownames(sig_counts$Gene) %>%
  t()
```

| | ENSDARG00000031683 | ENSDARG00000034503 | ENSDARG00000104773 | ENSDARG00000055752 |
|---|---|---|---|---|
| Cnt_1 | 611.6632 | 633.7051 | 875.0641 | 751.6294 |
| Cnt_2 | 918.4427 | 715.1808 | 1003.1352 | 703.8885 |
| Cnt_3 | 599.6419 | 597.1434 | 787.8629 | 687.0897 |
| Amp_1 | 197.6199 | 1079.0964 | 512.8925 | 397.9972 |
| Amp_2 | 216.4054 | 1330.1982 | 457.6279 | 402.0375 |
| Amp_3 | 242.7223 | 1266.1697 | 545.6475 | 442.4427 |

Once the matrix is transposed we can apply the `scale` function.

```
sig_counts %>%
  select(-Gene) %>%
  as.matrix() %>%
  magrittr::set_rownames(sig_counts$Gene) %>%
  t() %>% scale()
```

| | ENSDARG00000031683 | ENSDARG00000034503 | ENSDARG00000104773 | ENSDARG00000055752 |
|---|---|---|---|---|
| Cnt_1 | 0.5033128 | -0.9227744 | 0.7997079 | 1.1263122 |
| Cnt_2 | 1.5519304 | -0.6748157 | 1.3750149 | 0.8394540 |
| Cnt_3 | 0.4622225 | -1.0340442 | 0.4079922 | 0.7385163 |
| Amp_1 | -0.9119481 | 0.4327053 | -0.8271999 | -0.9985388 |
| Amp_2 | -0.8477364 | 1.1968950 | -1.0754535 | -0.9742620 |
| Amp_3 | -0.7577812 | 1.0020340 | -0.6800616 | -0.7314816 |

And transpose back using `t` again.

```
sig_counts %>%
  select(-Gene) %>%
  as.matrix() %>%
  magrittr::set_rownames(sig_counts$Gene) %>%
  t() %>% scale() %>% t()
```

| | Cnt_1 | Cnt_2 | Cnt_3 | Amp_1 | Amp_2 |
|---|---|---|---|---|---|
| ENSDARG00000031683 | 0.5033128 | 1.5519304 | 0.4622225 | -0.9119481 | -0.8477364 |
| ENSDARG00000034503 | -0.9227744 | -0.6748157 | -1.0340442 | 0.4327053 | 1.1968950 |
| ENSDARG00000104773 | 0.7997079 | 1.3750149 | 0.4079922 | -0.8271999 | -1.0754535 |
| ENSDARG00000055752 | 1.1263122 | 0.8394540 | 0.7385163 | -0.9985388 | -0.9742620 |
| ENSDARG00000087440 | -0.8097571 | -1.0285228 | -0.8434562 | 0.9601709 | 0.5682125 |

To cluster the matrix I have written a small function that takes a matrix and clusters the rows using the `hclust` function.

```
# function to cluster the rows of a data frame
cluster <- function(mat) {
  # create a distance matrix of pairwise distances between each gene
  distance_matrix <- dist(mat)
  # cluster based on the distance matrix
  clustering <- hclust(distance_matrix)
  # reorder the original matrix based on the clustering
  mat_ordered <- mat[ clustering$order, ]
  return(mat_ordered)
}
```

```
sig_counts %>%
  select(-Gene) %>%
  as.matrix() %>%
  magrittr::set_rownames(sig_counts$Gene) %>%
  t() %>% scale() %>% t() %>%
  cluster()
```

| | Cnt_1 | Cnt_2 | Cnt_3 | Amp_1 | Amp_2 |
|---|---|---|---|---|---|
| ENSDARG00000034503 | -0.9227744 | -0.6748157 | -1.0340442 | 0.4327053 | 1.1968950 |
| ENSDARG00000087440 | -0.8097571 | -1.0285228 | -0.8434562 | 0.9601709 | 0.5682125 |
| ENSDARG00000055752 | 1.1263122 | 0.8394540 | 0.7385163 | -0.9985388 | -0.9742620 |
| ENSDARG00000031683 | 0.5033128 | 1.5519304 | 0.4622225 | -0.9119481 | -0.8477364 |
| ENSDARG00000104773 | 0.7997079 | 1.3750149 | 0.4079922 | -0.8271999 | -1.0754535 |

After clustering the matrix, we can make the matrix back into a tibble and pivot it to make it compatible with ggplot. To set the order of the Genes and samples we can use the `fct_inorder` function from the `forcats` package.

```
# scale, cluster and pivot
counts_scaled_clustered <- sig_counts %>%
  select(-Gene) %>%
  as.matrix() %>%
  magrittr::set_rownames(sig_counts$Gene) %>%
  t() %>% scale() %>% t() %>%
  cluster() %>%
  as_tibble(rownames = 'Gene') %>% # make it back into a tibble
  pivot_longer(-Gene, names_to = 'sample') %>% # pivot longer for plottin
  mutate(Gene = fct_inorder(Gene),
         sample = fct_inorder(sample))
```

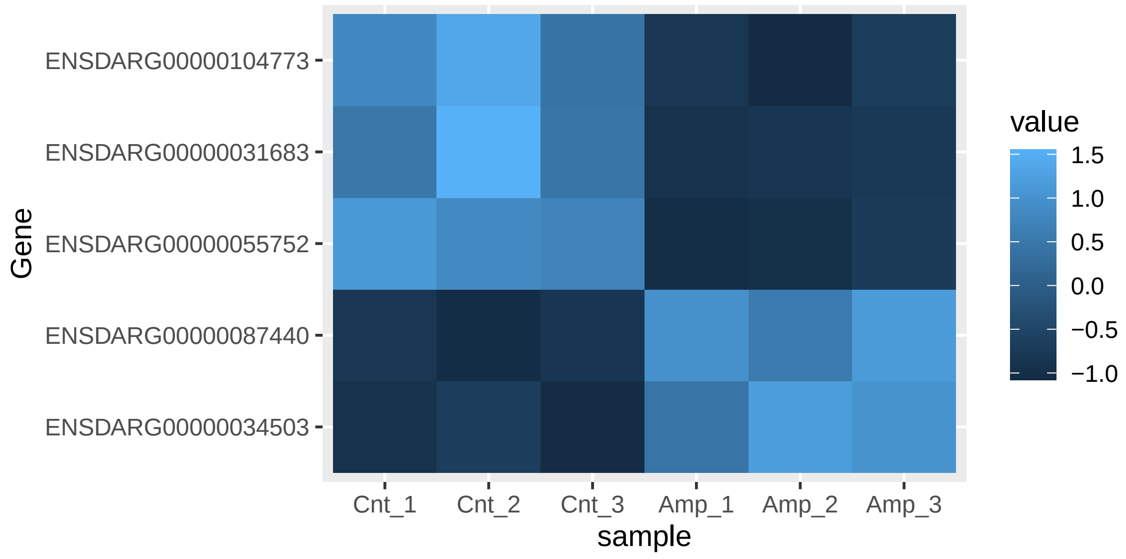
Now we have the data in the right format. To make a heatmap, we use the samples as the x-axis values, the genes as the y-axis values and the scaled counts as the fill value mapped to colour.
Each level of `samples` gets an integer value based on it's position in the factor and the same with `Gene`. Something to remember is that the first level of `Gene` is assigned the value 1, so it is plotted at the bottom of the y-axis, whereas the last level is plotted at the top.

```
counts_scaled_clustered %>%
  mutate(Gene_num = as.numeric(Gene),
         sample_num = as.numeric(sample)) %>%
  select(Gene, Gene_num, sample, sample_num) %>%
  head(10) %>%
  kable()
```

| Gene | Gene_num | sample | sample_num |
|---|---:|---|---:|
| ENSDARG00000034503 | 1 | Cnt_1 | 1 |
| ENSDARG00000034503 | 1 | Cnt_2 | 2 |
| ENSDARG00000034503 | 1 | Cnt_3 | 3 |
| ENSDARG00000034503 | 1 | Amp_1 | 4 |
| ENSDARG00000034503 | 1 | Amp_2 | 5 |
| ENSDARG00000034503 | 1 | Amp_3 | 6 |
| ENSDARG00000087440 | 2 | Cnt_1 | 1 |
| ENSDARG00000087440 | 2 | Cnt_2 | 2 |
| ENSDARG00000087440 | 2 | Cnt_3 | 3 |
| ENSDARG00000087440 | 2 | Amp_1 | 4 |

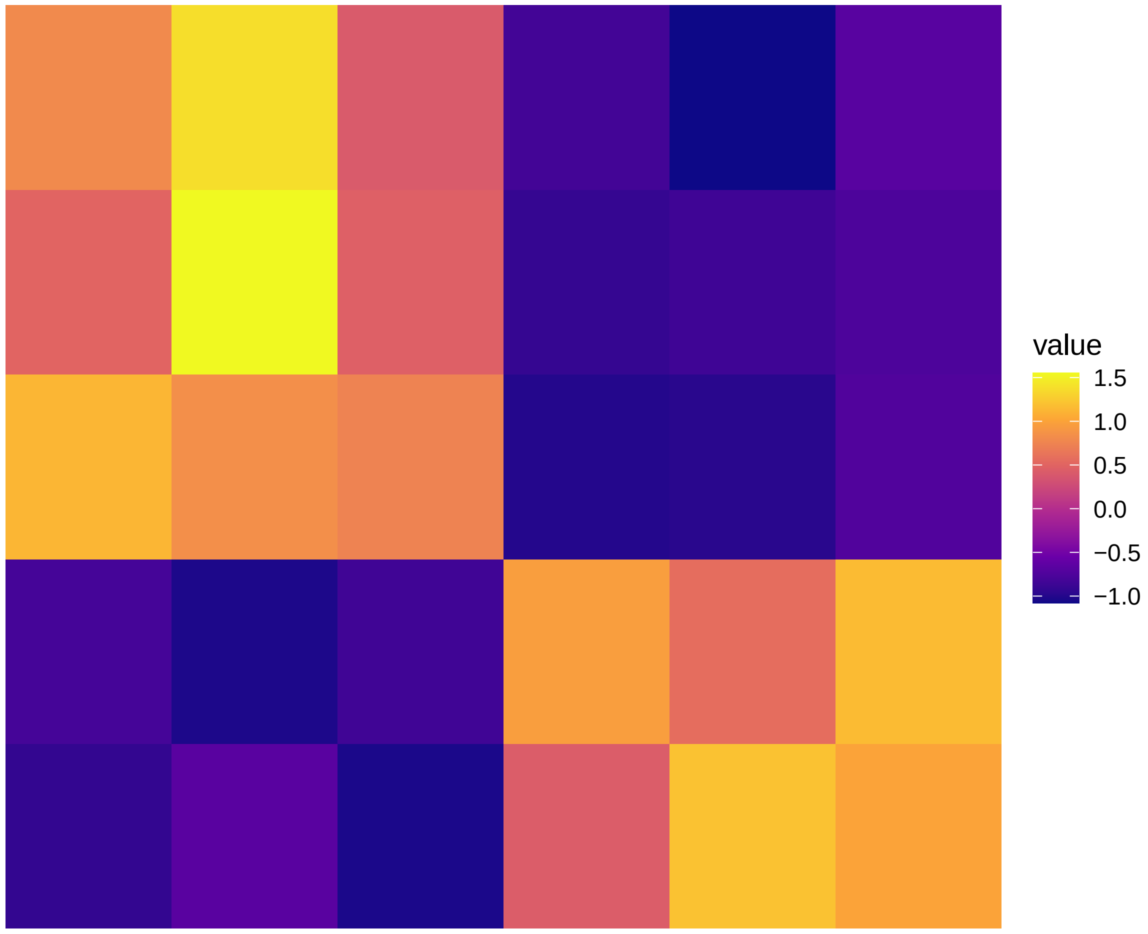
To plot coloured squares/rectangles we can use `geom_tile`.

```
ggplot(data = counts_scaled_clustered,
       aes(x = sample, y = Gene)) +
  geom_tile(aes(fill = value))
```



To make this look better, ggplot provides the viridis colour scales which we use here with scale_fill_viridis_c.
theme_void gets rid of all gridlines and axis ticks, text and titles.

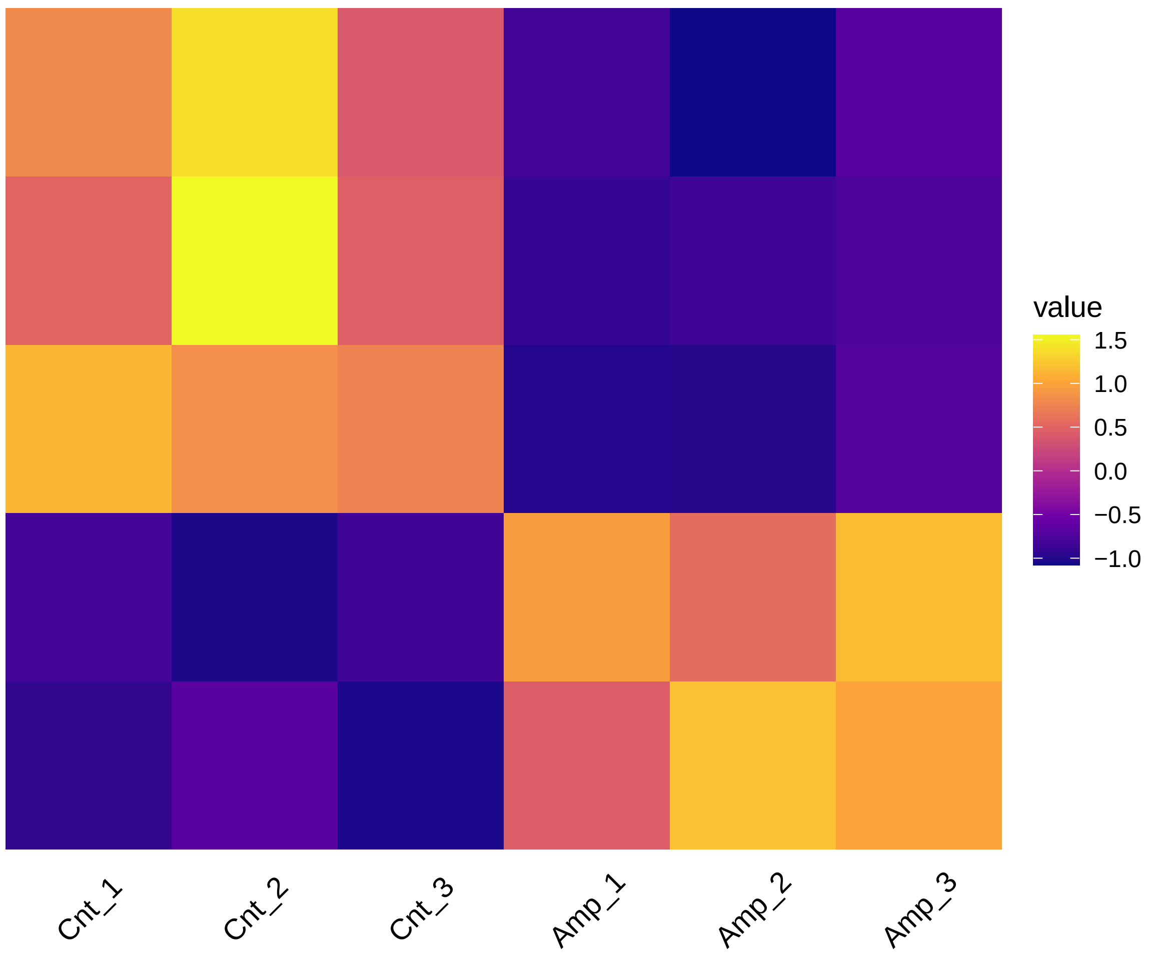
```
ggplot(data = counts_scaled_clustered,
       aes(x = sample, y = Gene)) +
  geom_tile(aes(fill = value)) +
  scale_fill_viridis_c(option = 'plasma') +
  theme_void()
```



If you have a very large heatmap to plot, it may be worth using geom_raster which runs faster by making all the tiles the same size.

If you want individual sample names on the x-axis, you can override just that bit of theme_void by adding + theme(axis.text.x = element_text(colour = "black")) after theme_void.

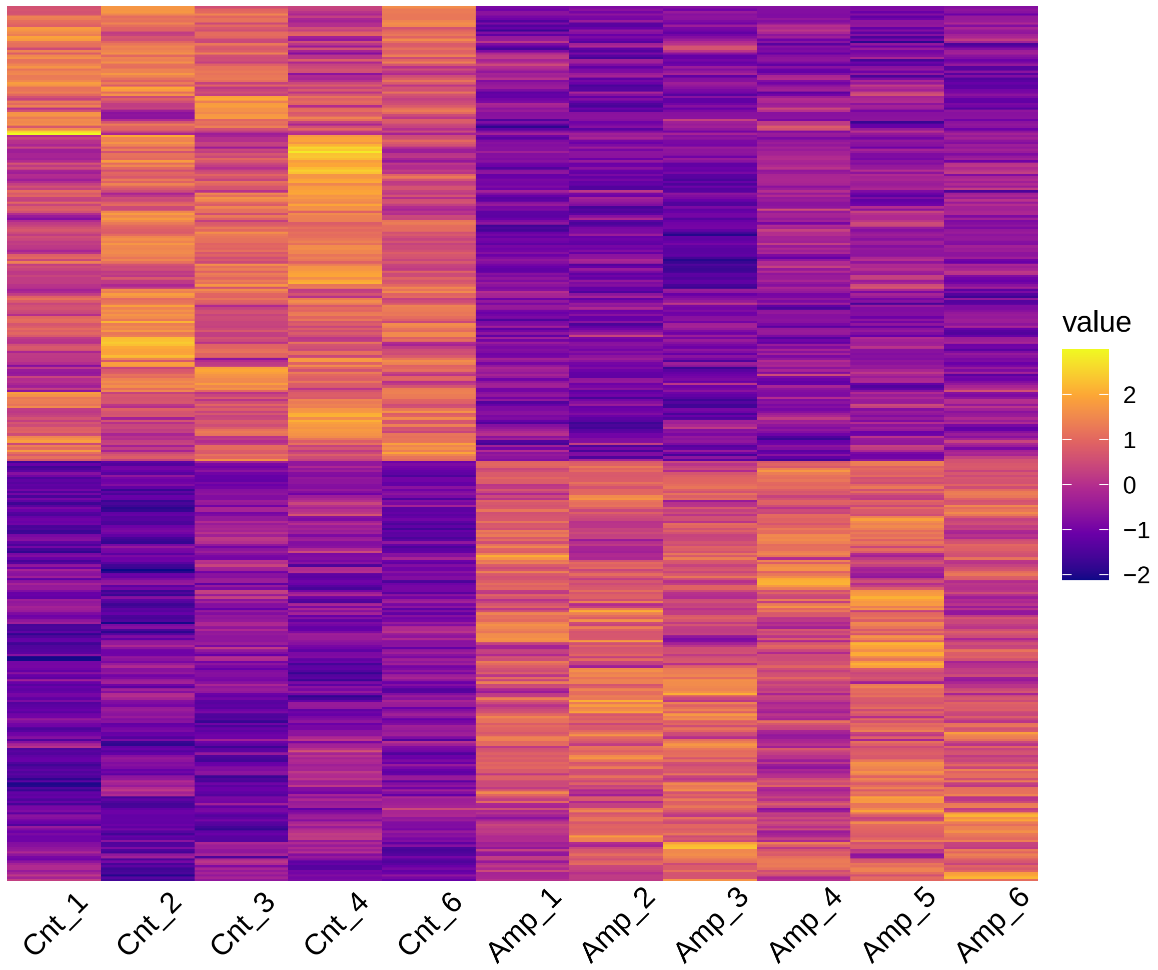
```
ggplot(data = counts_scaled_clustered,
       aes(x = sample, y = Gene)) +
  geom_tile(aes(fill = value)) +
  scale_fill_viridis_c(option = 'plasma') +
  theme_void() +
  theme(axis.text.x = element_text(colour = "black", angle = 45))
```



Now, let's redo with all the data.

```
# get DE genes
sig_counts <- deseq_results %>%
  filter(adjp < 0.05) %>%
  select(Gene, contains(' normalised count')) %>%
  select(Gene, contains('Cnt'), contains('Amp')) %>%
  rename_with(.fn = function(x){
    sub(" normalised count", "", x) } )

# scale, cluster and pivot
counts_scaled_clustered <- sig_counts %>%
  select(-Gene) %>%
  as.matrix() %>%
  magrittr::set_rownames(sig_counts$Gene) %>%
  t() %>% scale() %>% t() %>%
  cluster() %>%
  as_tibble(rownames = 'Gene') %>%
  pivot_longer(-Gene, names_to = 'sample') %>%
  mutate(Gene = fct_inorder(Gene),
         sample = fct_inorder(sample))
```

```
ggplot(data = counts_scaled_clustered,
       aes(x = sample, y = Gene)) +
  geom_tile(aes(fill = value)) +
  scale_fill_viridis_c(option = 'plasma') +
  theme_void() +
  theme(axis.text.x = element_text(colour = "black", angle = 45))
```


value
Cnt_1
Cnt_2
Cnt_3
Cnt_4
Cnt_6
Amp_1
Amp_2
Amp_3
Amp_4
Amp_5
Amp_6