Introduction to Bulk RNAseq data analysis

Visualisation of Differential Expression Results

Last modified: 01 Oct 2024

Contents

Visualisation

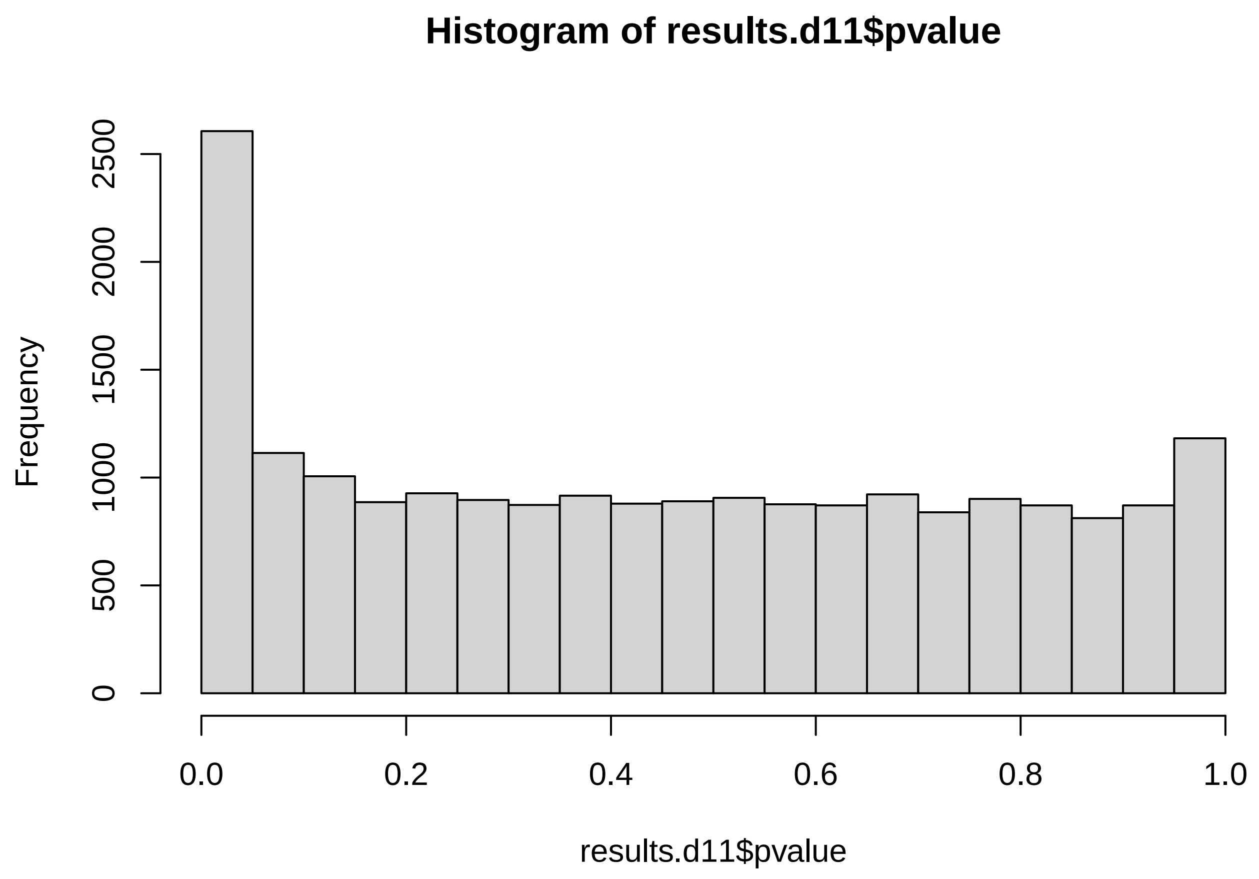
In this section we will consider various ways in which we can visualise the results of our differential expression analysis. We will use the `DESeq2` results from the interaction model for Infected vs Uninfected at day 11.

```
library(DESeq2)
library(tidyverse)

results.d11 <- readRDS("RObjects/DESeqResults.interaction_d11.rds")
```

Histogram of p-values

A quick and easy "sanity check" for our DE results is to generate a p-value histogram. What we should see is a high bar at `0 - 0.05` and then a roughly uniform tail to the right of this. There is a nice explanation of other possible patterns in the histogram and what to do when you see them in this post.

```
hist(results.d11$pvalue)
```



Shrinking the log2FoldChange

`DESeq2` provides a function called `lfcShrink` that shrinks log-Fold Change (LFC) estimates towards zero using an empirical Bayes procedure. The reason for doing this is that there is high variance in the LFC estimates when counts are low and this results in lowly expressed genes appearing to show greater differences between groups than highly expressed genes. The `lfcShrink` method compensates for this and allows better visualisation and ranking of genes. There a few different shrinkage methods available, we will use the `ashr` method.

To run the shrinkage algorithm we will also need the `DESeq2 Dataset` object that was used to generate the results. We will also want to add gene symbols to the shrunk results for easier interpretation.

```
ddsObj <- readRDS("RObjects/DESeqDataSet.interaction.rds")
annot <- readRDS("RObjects/Ensembl_annotations.rds")

shrink.11 <- lfcShrink(ddsObj,
                          res = results.d11,
                          type = "ashr")
```

```
## using 'ashr' for LFC shrinkage. If used in published research, please cite:
##     Stephens, M. (2016) False discovery rates: a new deal. Biostatistics, 18:2.
##     https://doi.org/10.1093/biostatistics/kxw041
```

```
shrinkTab.11 <- as.data.frame(shrink.11) %>%
    rownames_to_column("GeneID") %>%
    left_join(annot, "GeneID")
```

Note that it is important to provide both the `DESeq2` object and the `DESeq2 results` object to the `lfcShrink` function. It is not necessary to provide the `DESeq2 results` object, we can instead just provide a contrast to create the shrunk results for, however, if we do this the function will recalculate the p-values and padj, but this time the alpha for independent filtering will be set to the default of 0.1 and so the adjusted p-values will be different.

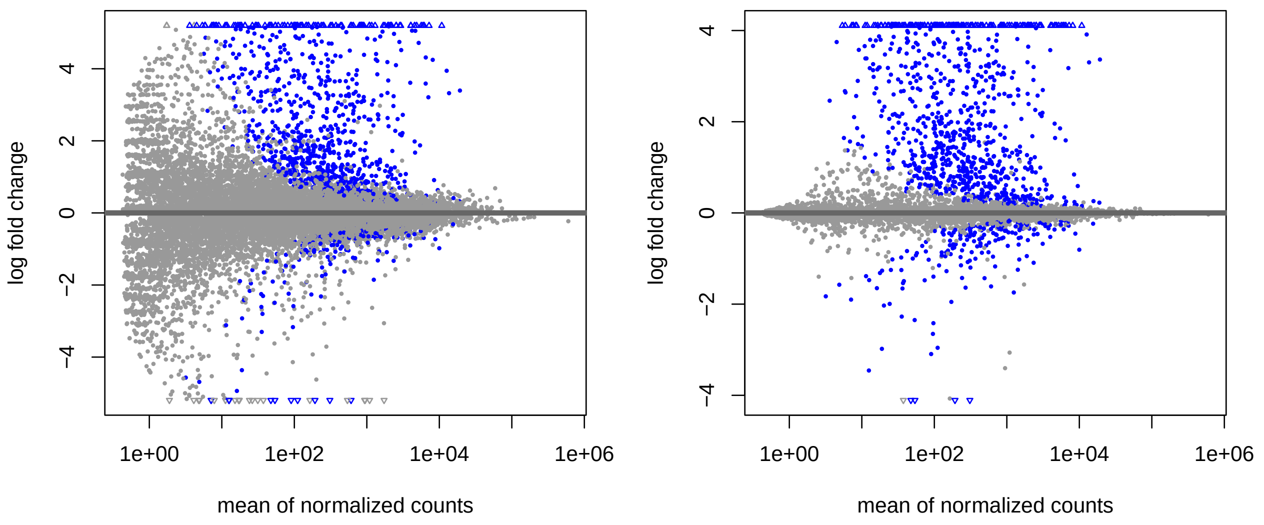
Let's save the shrunk results to a file so we can use them later.

```
saveRDS(shrinkTab.33, file="results/Shrunk_Results.d33.rds")
```

MA plots

MA plots are a common way to visualize the results of a differential analysis. We met them briefly during the DESeq2 session. This plot shows the log-Fold Change for each gene against its average expression across all samples in the two conditions being contrasted. `DESeq2` has a handy function for plotting this. Let's use it too compare the shrunk and un-shrunk fold changes.

```
par(mfrow=c(1,2))
plotMA(results.d11, alpha=0.05)
plotMA(shrink.11, alpha=0.05)
```



The DESeq2 in `plotMA` function is fine for a quick look, but these inbuilt functions aren't easy to customise, make changes to the way it looks or add things such as gene labels. For this we would recommend using the ggplot package.

Volcano Plots

Another common visualisation is the *volcano plot* which displays a measure of significance on the y-axis and fold-change on the x-axis. We will use ggplot to create this.

A Brief Introduction to `ggplot2`

The `ggplot2` package has emerged as an attractive alternative to the traditional plots provided by base R. A full overview of all capabilities of the package is available from the cheatsheet.

In brief:-

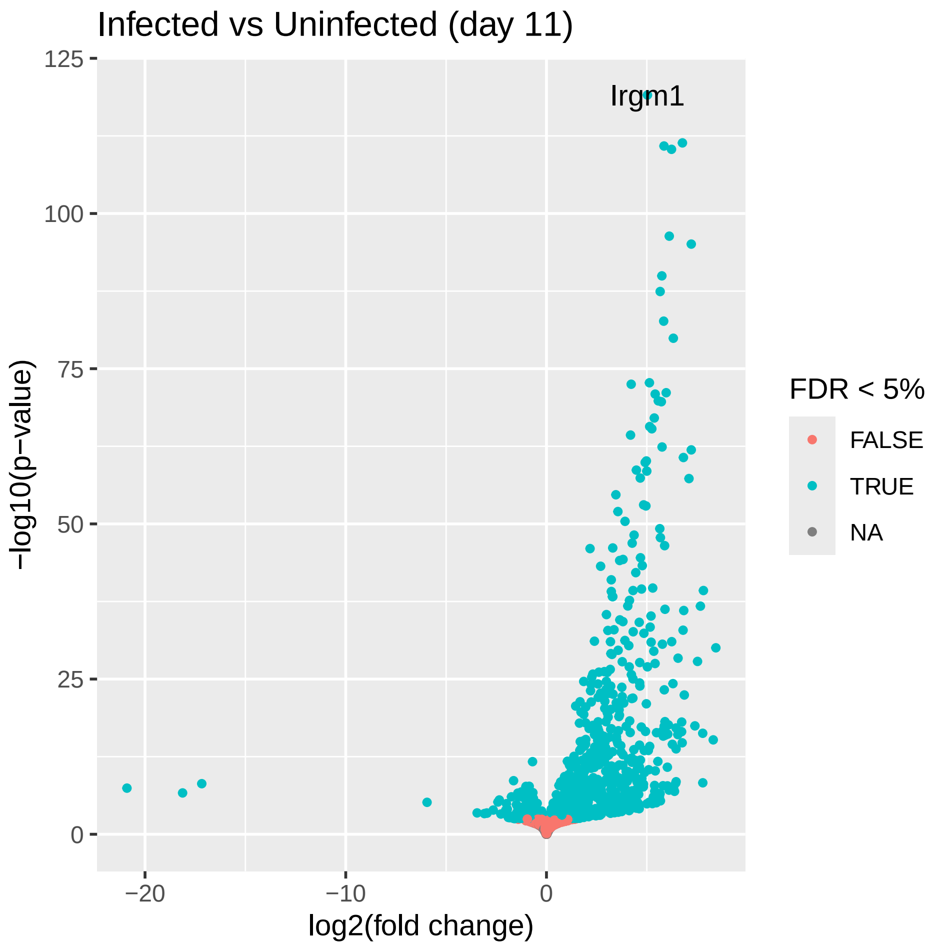
- `shrinkTab.11` is our data frame containing the variables we wish to plot

- `aes` creates a mapping between the variables in our data frame to the ***aes***thetic properties of the plot:
  - the x-axis will be mapped to `log2FoldChange`
  - the y-axis will be mapped to the `-log10(pvalue)`
- `geom_point` specifies the particular type of plot we want (in this case a scatter plot)
- `geom_text` allows us to add labels to some or all of the points
  - see the cheatsheet for other plot types

The real advantage of `ggplot2` is the ability to change the appearance of our plot by mapping other variables to aspects of the plot. For example, we could colour the points based on the sample group. To do this we can add metadata from the `sampleinfo` table to the data. The colours are automatically chosen by `ggplot2`, but we can specify particular values. For the volcano plot we will colour according whether the gene has a pvalue below 0.05. We use a `-log10` transformation for the y-axis; it's commonly used for p-values as it means that more significant genes have a higher scale.

```
ggplot(shrinkTab.11, aes(x = log2FoldChange, y = -log10(pvalue))) +
    geom_point(aes(colour = padj < 0.05), size=1) +
    geom_text(data = ~top_n(.x, 1, wt = -padj), aes(label = Symbol)) +
    labs(x = "log2(fold change)", y = "-log10(p-value)", colour = "FDR < 5%",
         title = "Infected vs Uninfected (day 11)")
```

```
## Warning: Removed 47 rows containing missing values or values outside the scale
## range (`geom_point()`).
```



Exercise 1 - Volcano plot for 33 days

We just made the volcano plot for the 11 days contrast, you will make the one for the 33 days contrast.

First load in the results for the 33 days contrast.

```
results.d33 <- readRDS("RObjects/DESeqResults.interaction_d33.rds")
```

(a) Shrink the results for the 33 days contrast and add the annotation.

```
## using 'ashr' for LFC shrinkage. If used in published research, please cite:
##     Stephens, M. (2016) False discovery rates: a new deal. Biostatistics, 18:2.
##     https://doi.org/10.1093/biostatistics/kxw041
```

(b) Create a plot with points coloured by padj < 0.05 similar to how we did in the first volcano plot

(c) Compare these two volcano plots, what differences can you see between the two contrasts?

Exercise 2 - MA plot for day 33 with ggplot2

For this exercise create an MA plot for day 33 like the ones we plotted with `plotMA` from **DESeq2** but this time using ggplot2.

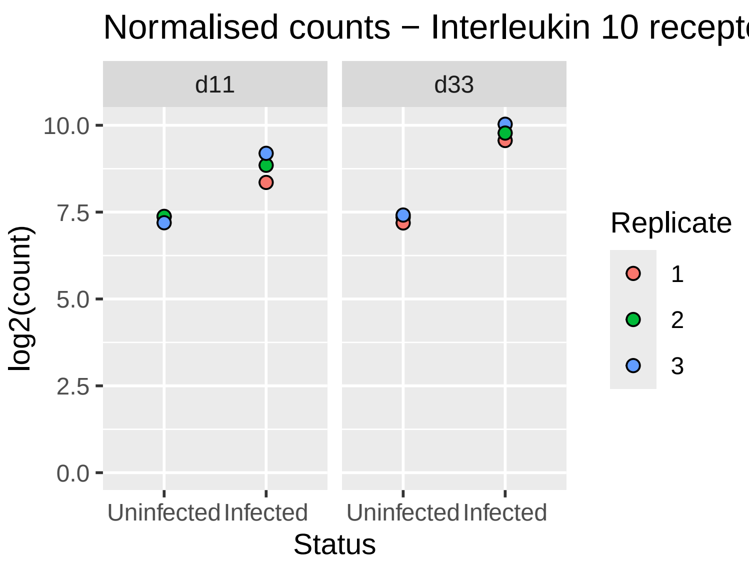
The x-axis should be the log2 of the mean gene expression across all samples, and the y-axis should be the log2 of the fold change between Infected and Uninfected.

Strip Charts for gene expression

Before following up on the DE genes with further lab work, a recommended *sanity check* is to have a look at the expression levels of the individual samples for the genes of interest. We can quickly look at grouped expression by using `plotCounts` function of `DESeq2` to retrieve the normalised expression values from the `ddsObj` object and then plotting with `ggplot2`.

We are going investigate the Il10ra gene:

```
geneID <- filter(shrinkTab.11, Symbol=="Il10ra") %>% pull(GeneID)

plotCounts(ddsObj,
           gene = geneID,
           intgroup = c("TimePoint", "Status", "Replicate"),
           returnData = T) %>%
    ggplot(aes(x=Status, y=log2(count))) +
      geom_point(aes(fill=Replicate), shape=21, size=2) +
      facet_wrap(~TimePoint) +
      expand_limits(y=0) +
      labs(title = "Normalised counts - Interleukin 10 receptor, alpha")
```

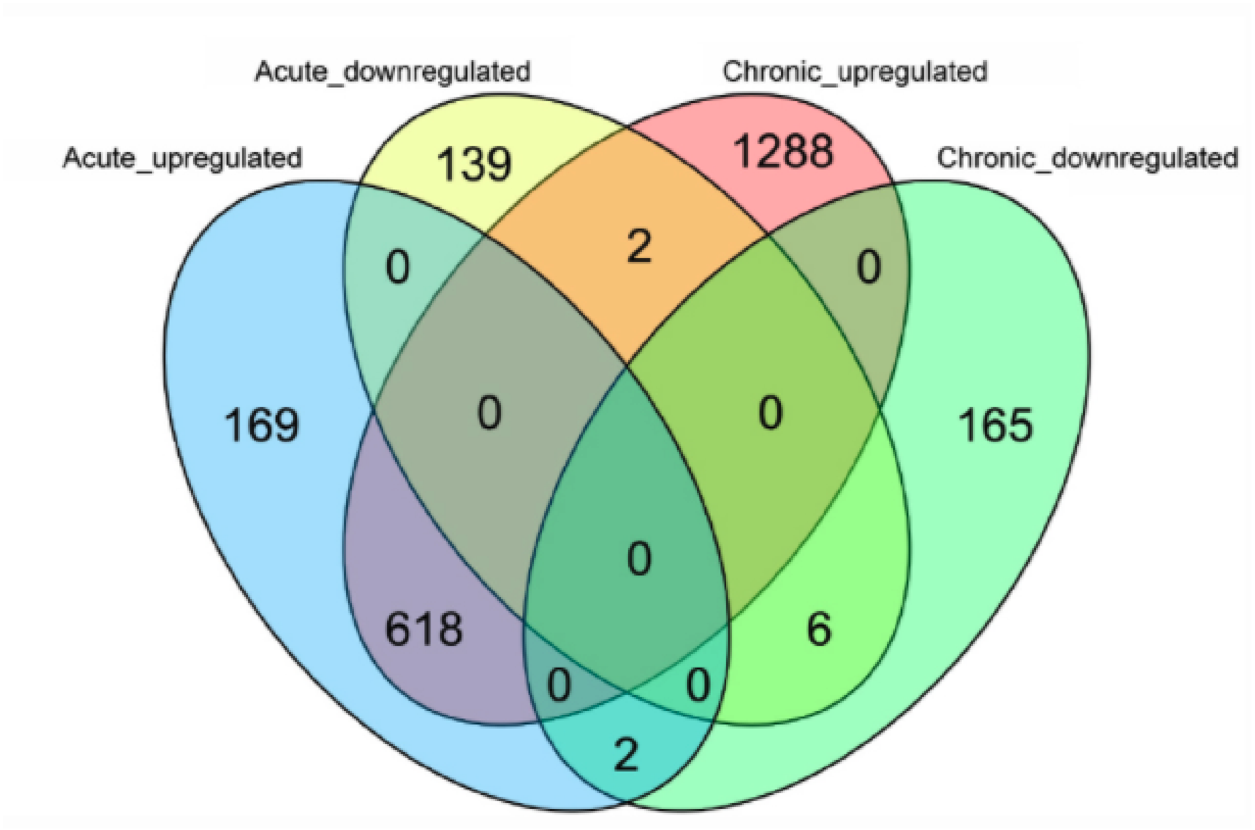


Exercise 3

For this exercise create another strip chart for the gene Jchain.

Venn Diagram of differentially expressed genes

In the paper you may notice they have presented a Venn diagram of the results.



We will recreate it with our analysis. To do this we are using the package `ggvenn` which is an extension to ggplot from Linlin Yan.

```
library(ggvenn)
```

We want to plot four “sets” on the venn diagram:

- Significantly up-regulated on day 11
- Significantly down-regulated on day 11
- Significantly up-regulated on day 33
- Significantly down-regulated on day 33

Each comprising genes at that are statistically significant at a 5% FDR level for the respective contrast.

There are two ways of providing the data to `ggvenn`. The first is to provide a table with features (genes) in the rows and the sets (contrasts) in the columns, and `TRUE` or `FALSE` in the cells to indicate whether the features is in that set. For our data the table would look like this:

```
## # A tibble: 20,091 x 5
##    Geneid        Upregulated_11 Downregulated_11 Upregulated_33 Downregulated_33
##    <chr>         <lgl>          <lgl>            <lgl>          <lgl>
##  1 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
##  2 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
##  3 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
##  4 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
##  5 ENSMUSG00000~ FALSE          FALSE            FALSE          TRUE
##  6 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
##  7 ENSMUSG00000~ TRUE           FALSE            TRUE           FALSE
##  8 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
##  9 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
## 10 ENSMUSG00000~ FALSE          FALSE            FALSE          FALSE
## # i 20,081 more rows
```

The second option is to provide a list with one element for each set. Each element is then a vector of the features in that set. For our data this would look like this:

```
## List of 4
##  $ Upregulated_11  : chr [1:878] "ENSMUSG00000000078" "ENSMUSG00000000204" "ENSMUSG00000000275" "ENSM
##  $ Downregulated_11: chr [1:194] "ENSMUSG00000000320" "ENSMUSG00000000811" "ENSMUSG00000001864" "ENSM
##  $ Upregulated_33  : chr [1:1866] "ENSMUSG00000000078" "ENSMUSG00000000154" "ENSMUSG00000000204" "ENS
##  $ Downregulated_33: chr [1:916] "ENSMUSG00000000056" "ENSMUSG00000000125" "ENSMUSG00000000202" "ENSM
```

We will use the list option as the code for builing the list is more concise.

The code for building each of the four vectors of gene ids is basically the same with a couple of minor changes for each set, rather the repeating the code we can create a function to do this for us.

To build up the function, first, let’s see how we would do this for the genes on day 11.

```
Upregulated_d11 <- shrinkTab.11 %>%
    filter(padj < 0.05) %>%
    filter(log2FoldChange > 0) %>%
    pull("GeneID")
head(Upregulated_d11)
```

```
## [1] "ENSMUSG00000000078" "ENSMUSG00000000204" "ENSMUSG00000000275"
## [4] "ENSMUSG00000000290" "ENSMUSG00000000317" "ENSMUSG00000000386"
```

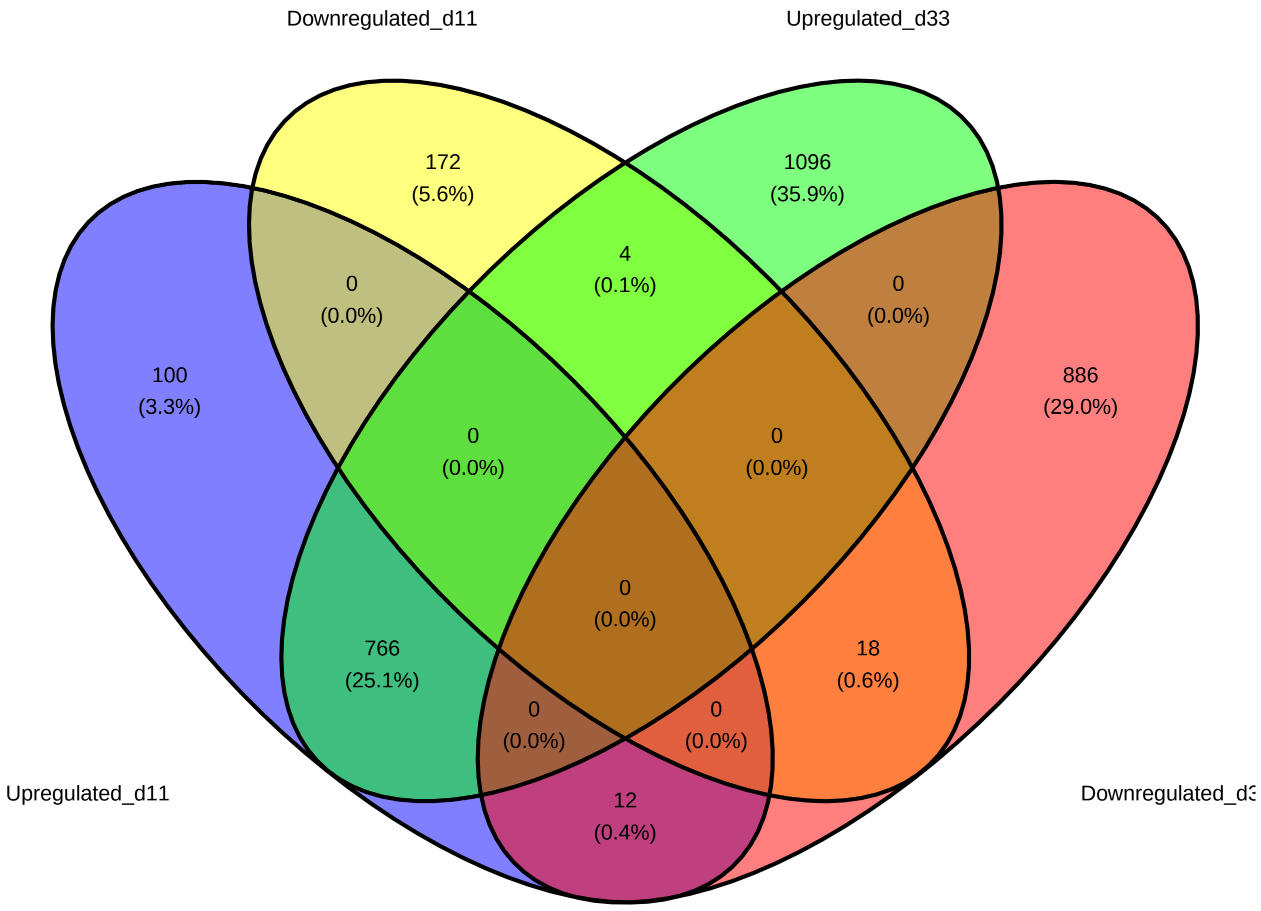
The function we will create is just a generalisation of this code. We want to be able to do the same operation using different tables (day 11 and day 33), and we also need to be able to get the up- or down-regulated genes. We can do this by passing the table and the direction as arguments to the function. To change the direction of the regulation, we can leave the boolean filter as `log2FoldChange > 0` and multiply the `log2FoldChange` by 1 or -1 depending on the direction we want.

```
getGenes <- function(shrTab, direction) {
    sign <- ifelse(direction == "up", 1, -1)
    shrTab %>%
        filter(padj < 0.05) %>%
        filter(sign * log2FoldChange > 0) %>%
        pull("GeneID")
}

vennList <- list(Upregulated_d11 = getGenes(shrinkTab.11, "up"),
                 Downregulated_d11 = getGenes(shrinkTab.11, "down"),
                 Upregulated_d33 = getGenes(shrinkTab.33, "up"),
                 Downregulated_d33 = getGenes(shrinkTab.33, "down"))

str(vennList)
```

```
## List of 4
##  $ Upregulated_d11  : chr [1:878] "ENSMUSG00000000078" "ENSMUSG00000000204" "ENSMUSG00000000275" "ENS
##  $ Downregulated_d11: chr [1:194] "ENSMUSG00000000320" "ENSMUSG00000000811" "ENSMUSG00000001864" "ENS
##  $ Upregulated_d33  : chr [1:1866] "ENSMUSG00000000078" "ENSMUSG00000000154" "ENSMUSG00000000204" "EN
##  $ Downregulated_d33: chr [1:916] "ENSMUSG00000000056" "ENSMUSG00000000125" "ENSMUSG00000000202" "ENS
```

Now we just pass the list to the `ggvenn` function.

```
ggvenn(vennList, set_name_size = 4)
```



Heatmap of gene expression

Another common way to visualise differential gene expression results it to plot a heatmap of the normalised counts. There are many R packages that can be used to achieve this, we're going to use the package `pheatmap`.

```
library(pheatmap)
```

We can't (an don't want to) plot the entire data set, let's just select the top 300 by false discovery rate (`padj`). We'll want to use normalised expression values, so we'll use the `vst` function.

```
# get the top genes
sigGenes <- shrinkTab.11 %>%
    top_n(300, wt = -padj) %>%
    pull("GeneID")

# filter the data for the top 300 by padj
plotDat <- vst(ddsObj)[sigGenes,] %>%
  assay()
```

The range expression values for different genes can vary widely. Some genes will have very high expression. Our heatmap is going to be coloured according to gene expression. If we used a colour scale from 0 (no expression) to the maximum expression, the scale will be dominated by our most extreme genes and it will be difficult to discern any difference between most of the genes.

To overcome this we will z-scale the counts for each gene across the samples. This scaling method results in values for each gene that show the number of standard deviations the gene expression is from the mean for that gene across all the samples - the mean will be '0', '1' means 1 standard deviation higher than the mean, '-1' means 1 standard deviation lower than the mean. Depending on the package that you are using, you may have to do this yourself before plotting the heatmap, but `pheatmap` has a built-in function to do this.

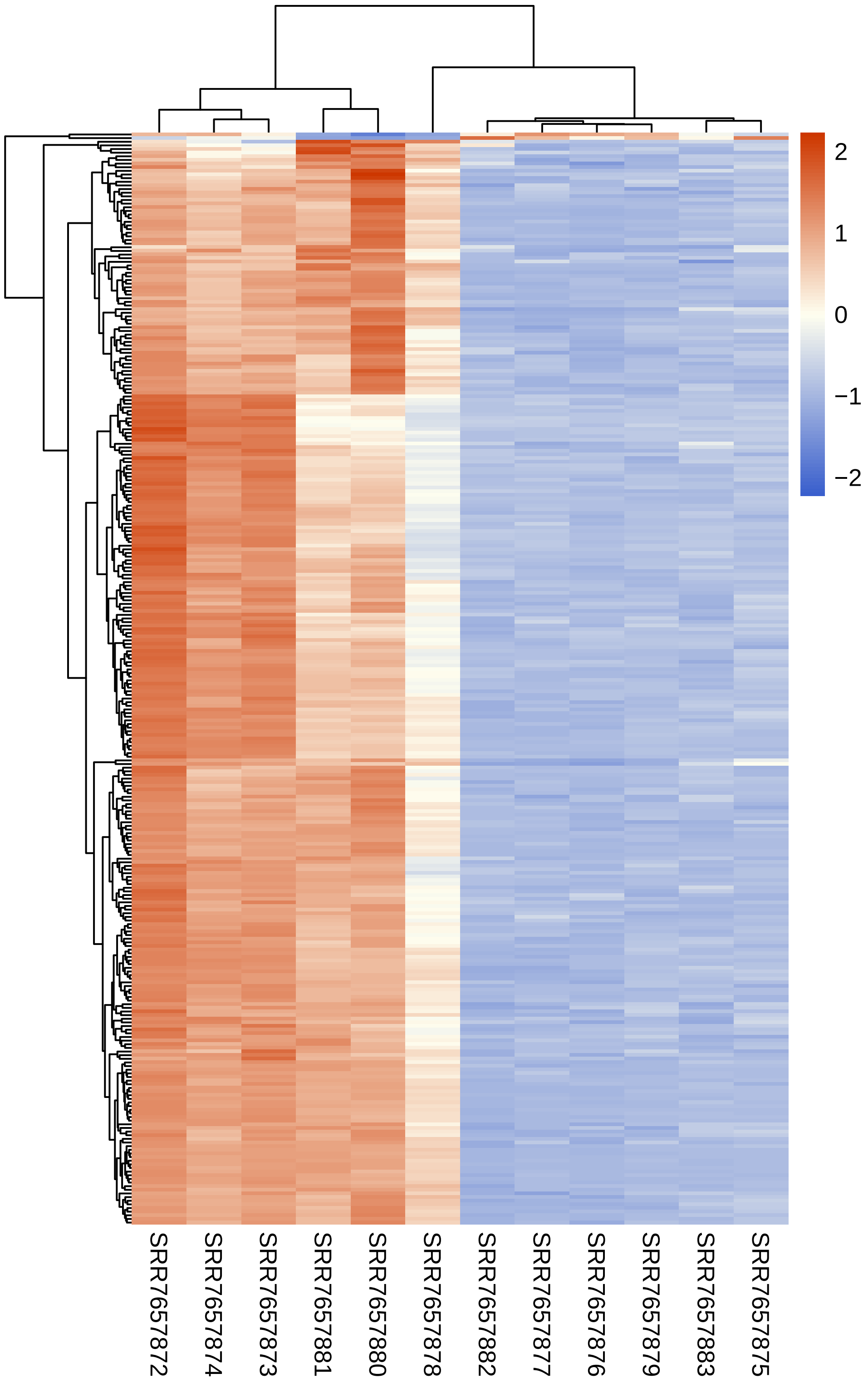
We will also create a colour palette for the heatmap. We will use a blue-white-red palette with blue for low expression, white for the mean and red for high. We just need to provide three colours and use the function `colorRampPalette` to create a function that will interpolate between these colours.

```
colours <- c("royalblue3", "ivory", "orangered3")
hmPalette <- colorRampPalette(colours)(100)
hmPalette
```

```
##   [1] "#3A5FCD" "#3D62CD" "#4165CE" "#4568CF" "#496BCF" "#4D6FD0" "#5172D1"
##   [8] "#5575D1" "#5978D2" "#5D7CD3" "#617FD4" "#6582D4" "#6985D5" "#6D89D6"
##  [15] "#718CD6" "#758FD7" "#7992D8" "#7D95D9" "#8199D9" "#859CDA" "#899FDB"
##  [22] "#8DA2DB" "#91A6DC" "#95A9DD" "#99ACDD" "#9DAFDE" "#A1B3DF" "#A5B6E0"
##  [29] "#A9B9E0" "#ADBCE1" "#B1BFE2" "#B5C3E2" "#B9C6E3" "#BDC9E4" "#C1CCE5"
##  [36] "#C5D0E5" "#C9D3E6" "#CDD6E7" "#D1D9E7" "#D5DDE8" "#D9E0E9" "#DDE3E9"
##  [43] "#E1E6EA" "#E5E9EB" "#E9EDEC" "#EDF0EC" "#F1F3ED" "#F5F6EE" "#F9FAEE"
##  [50] "#FDFDEF" "#FEFCED" "#FDF8E8" "#FCF4E3" "#FBF0DF" "#FAECDA" "#F9E8D5"
##  [57] "#F8E4D0" "#F7E0CB" "#F6DCC6" "#F5D8C1" "#F4D4BD" "#F3D0B8" "#F2CCB3"
##  [64] "#F1C8AE" "#F0C4A9" "#EFC0A4" "#EEBC9F" "#EDB89B" "#ECB496" "#EBB091"
##  [71] "#EAAC8C" "#E9A887" "#E8A482" "#E7A07E" "#E69C79" "#E59774" "#E4936F"
##  [78] "#E38F6A" "#E28B65" "#E18760" "#E0835C" "#DF7F57" "#DE7B52" "#DD774D"
##  [85] "#DC7348" "#DB6F43" "#DA6B3F" "#D9673A" "#D86335" "#D75F30" "#D65B2B"
##  [92] "#D55726" "#D45321" "#D34F1D" "#D24B18" "#D14713" "#D0430E" "#CF3F09"
##  [99] "#CE3B04" "#CD3700"
```

As well as scaling the gene expression values, `pheatmap` will also cluster the genes and samples. We could turn this off if we want to.

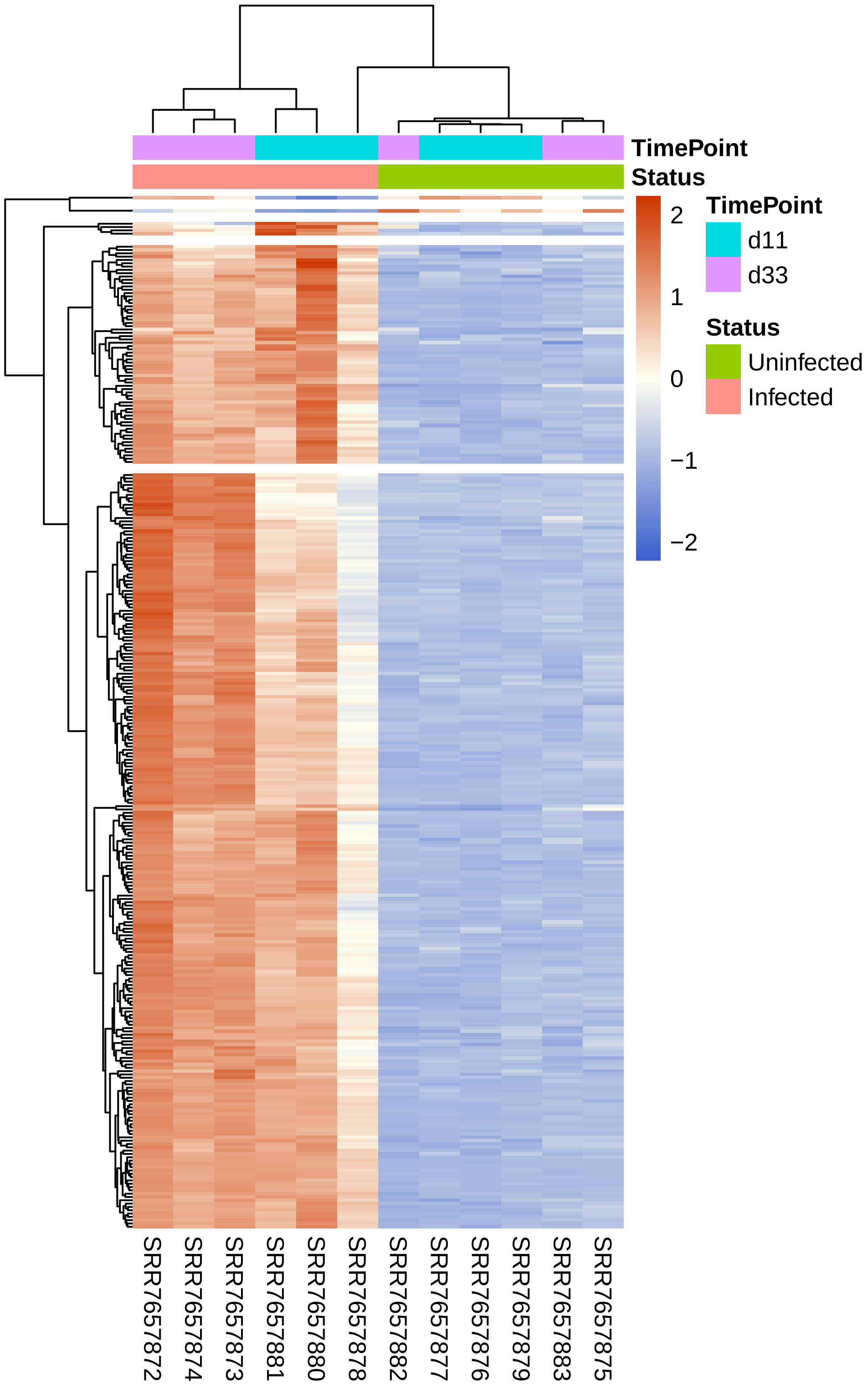
```
pheatmap(plotDat,
        cluster_rows = TRUE,
        cluster_cols = TRUE,
        scale = "row",
        show_rownames = FALSE,
        color = hmPalette)
```



We can also split the heat map into clusters and add some annotation in the form of coloured bars at the top of the heatmap to show the status and timepoint of the samples.

To add the annotation we simple need to create a data frame with the annotation.

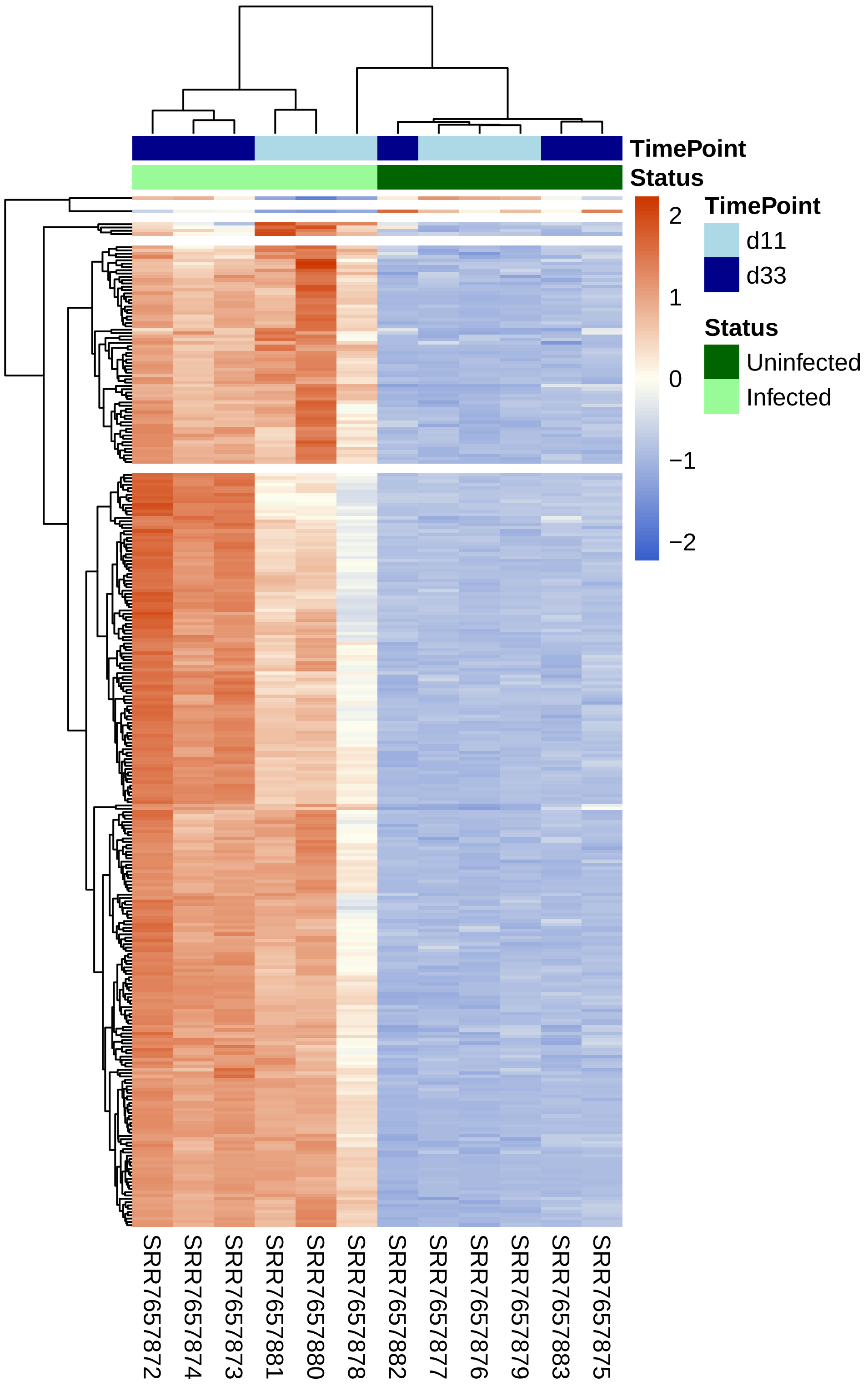
```r
annot_df <- colData(ddsObj) %>%
  as.data.frame() %>%
  select(Status, TimePoint)

pheatmap(plotDat,
         cluster_rows = TRUE,
         cluster_cols = TRUE,
         scale = "row",
         show_rownames = FALSE,
         color = hmPalette,
         cutree_rows = 5,
         annotation_col = annot_df)
```



`pheatmap` has automatically selected colours for the annotation, but we can specify our own colours if we want.

```
annot_col <- list(Status = c("Uninfected" = "darkgreen",
                             "Infected" = "palegreen"),
                  TimePoint = c("d11" = "lightblue",
```

```r
                                "d33" = "darkblue"))

pheatmap(plotDat,
         cluster_rows = TRUE,
         cluster_cols = TRUE,
         scale = "row",
         show_rownames = FALSE,
         color = hmPalette,
         cutree_rows = 5,
         annotation_col = annot_df,
         annotation_colors = annot_col)
```


TimePoint
Status
2
1
0
−1
−2
TimePoint
d11
d33
Status
Uninfected
Infected
SRR7657872
SRR7657874
SRR7657873
SRR7657881
SRR7657880
SRR7657878
SRR7657882
SRR7657877
SRR7657876
SRR7657879
SRR7657883
SRR7657875