Introduction to Bulk RNAseq data analysis

Initial exploration of RNA-seq data

Last modified: 26 Sep 2024

Contents

Introduction **1**

Data import **2**
A brief description of the data set 2
Reading in the sample metadata 2
Reading in the count data 3

Prepare count matrix **4**
Create a raw counts matrix for data exploration 4
Filtering the genes 4

Count distribution and Data transformations **5**
Raw counts 5
Data transformation 7

Principal Component Analysis **9**
Hierachical clustering 13

References **15**

Introduction

In this section we will begin the process of analyzing the RNAseq data in R. In the next section we will use DESeq2 for differential analysis. A detailed analysis workflow, recommended by the authors of DESeq2 can be found on the Bionconductor website.

Before embarking on the main analysis of the data, it is essential to do some exploration of the raw data. We want to assess the patterns and characteristics of the data and compare these to what we expect from mRNAseq data and assess the data based on our knowledge of the experimental design. The primary means of data explorations are summary statistics and visualisations. In this session we will primarily concentrate on assessing if the patterns in the raw data conform to what we know about the experimental design. This is essential to identify problems such as batch effects, outlier samples and sample swaps.

Due to time constraints we are not able to cover all the ways we might do this, so additional information on initial data exploration are available in the supplementary materials.

In this session we will:

- import our counts into R

- filter out unwanted genes

- transform the data to mitigate the effects of variance
- do some initial exploration of the raw count data using principle component analysis and hierarchical clustering

Data import

First, let's load all the packages we will need to analyse the data.

```
library(tximport)
library(DESeq2)
library(tidyverse)
```

A brief description of the data set

The data for this tutorial comes from the paper Transcriptomic Profiling of Mouse Brain During Acute and Chronic Infections by *Toxoplasma gondii* Oocysts (Hu et al. 2020). The raw data (sequence reads) can be downloaded from the NCBI Short Read Archive under project number **PRJNA483261**.

Please see extended material for instructions on downloading raw files from SRA.

This study examines changes in the gene expression profile in mouse brain in response to infection with the protozoan *Toxoplasma gondii*. The authors performed transcriptome analysis on samples from infected and uninfected mice at two time points, 11 days post infection and 33 days post infection. For each sample group there are 3 biological replicates. This effectively makes this a two factor study with two groups in each factor:

- Status: Infected/Uninfected
- Time Point: 11 dpi/33 dpi

Reading in the sample metadata

The SampleInfo.txt file contains basic information about the samples that we will need for the analysis today: name, cell type, status.

```
# Read the sample information into a data frame
sampleinfo <- read_tsv("data/samplesheet.tsv", col_types = c("cccc"))
arrange(sampleinfo, Status, TimePoint, Replicate)
```

```
## # A tibble: 12 x 4
##    SampleName Replicate Status     TimePoint
##    <chr>      <chr>     <chr>      <chr>
##  1 SRR7657878 1         Infected   d11
##  2 SRR7657881 2         Infected   d11
##  3 SRR7657880 3         Infected   d11
##  4 SRR7657874 1         Infected   d33
##  5 SRR7657882 2         Infected   d33
##  6 SRR7657872 3         Infected   d33
##  7 SRR7657877 1         Uninfected d11
##  8 SRR7657876 2         Uninfected d11
##  9 SRR7657879 3         Uninfected d11
## 10 SRR7657883 1         Uninfected d33
## 11 SRR7657873 2         Uninfected d33
## 12 SRR7657875 3         Uninfected d33
```

Reading in the count data

Salmon (Patro 2017) was used to quantify gene expression from raw reads against the Ensembl transcriptome GRCm38 version 102 (as described in the previous session).

First we need to read the data into R from the `quant.sf` files under the *salmon* directory. To do this we use the `tximport` function. We need to create a named vector in which the values are the paths to the `quant.sf` files and the names are sample names that we want in the column headers - these should match the sample names in our `sampleinfo` table.

The Salmon quantification results are per transcript, we'll want to summarise to gene level. To this we need a table that relates transcript IDs to gene IDs.

```
files <- file.path("salmon", sampleinfo$SampleName, "quant.sf")
files <- set_names(files, sampleinfo$SampleName)
tx2gene <- read_tsv("references/tx2gene.tsv")
```

```
## Rows: 119414 Columns: 2
## -- Column specification ----------------------------------------------------------------------
## Delimiter: "\t"
## chr (2): TxID, GeneID
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
txi <- tximport(files, type = "salmon", tx2gene = tx2gene)
```

```
## reading in files with read_tsv
## 1 2 3 4 5 6 7 8 9 10 11 12
## summarizing abundance
## summarizing counts
## summarizing length
```

```
str(txi)
```

```
## List of 4
##  $ abundance          : num [1:35896, 1:12] 20.39 0 1.97 1.06 0.95 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:35896] "ENSMUSG00000000001" "ENSMUSG00000000003" "ENSMUSG00000000028" "ENSMUSG0000
##   .. ..$ : chr [1:12] "SRR7657878" "SRR7657881" "SRR7657880" "SRR7657874" ...
##  $ counts             : num [1:35896, 1:12] 1039 0 65 39 8 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:35896] "ENSMUSG00000000001" "ENSMUSG00000000003" "ENSMUSG00000000028" "ENSMUSG0000
##   .. ..$ : chr [1:12] "SRR7657878" "SRR7657881" "SRR7657880" "SRR7657874" ...
##  $ length             : num [1:35896, 1:12] 2903 541 1883 2098 480 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:35896] "ENSMUSG00000000001" "ENSMUSG00000000003" "ENSMUSG00000000028" "ENSMUSG0000
##   .. ..$ : chr [1:12] "SRR7657878" "SRR7657881" "SRR7657880" "SRR7657874" ...
##  $ countsFromAbundance: chr "no"
```

```
head(txi$counts)
```

```
##                    SRR7657878 SRR7657881 SRR7657880 SRR7657874 SRR7657882
## ENSMUSG00000000001   1039.000   1005.889    892.000    917.360   1136.690
## ENSMUSG00000000003      0.000      0.000      0.000      0.000      0.000
## ENSMUSG00000000028     65.000     73.999     72.000     44.000     46.000
## ENSMUSG00000000037     39.000     47.000     29.000     53.999     67.000
## ENSMUSG00000000049      8.000      9.000      4.000      4.000      4.000
## ENSMUSG00000000056   2163.469   2067.819   2006.925   1351.675   2367.801
##                      SRR7657872 SRR7657877 SRR7657876 SRR7657879 SRR7657883
## ENSMUSG00000000001   1259.000   1351.221   1110.999   1067.634   1134.522
## ENSMUSG00000000003      0.000      0.000      0.000      0.000      0.000
## ENSMUSG00000000028     60.000     35.000     52.001     56.000     58.000
## ENSMUSG00000000037     62.000     69.000     34.999     60.000     20.999
## ENSMUSG00000000049      9.000      6.000     10.000      4.000      8.000
## ENSMUSG00000000056   1412.733   2154.230   2121.740   1962.000   2274.702
##                      SRR7657873 SRR7657875
## ENSMUSG00000000001   1272.003   1065.000
## ENSMUSG00000000003      0.000      0.000
## ENSMUSG00000000028     75.000     54.000
## ENSMUSG00000000037     50.000     28.000
## ENSMUSG00000000049      6.000      9.000
## ENSMUSG00000000056   1693.000   2260.046
```

Save the `txi` object for use in later sessions.

```
saveRDS(txi, file = "salmon_outputs/txi.rds")
```

A quick intro to dplyr

One of the most complex aspects of learning to work with data in `R` is getting to grips with subsetting and manipulating data tables. The package `dplyr` (Wickham et al. 2018) was developed to make this process more intuitive than it is using standard base `R` processes.

In particular we will use the commands:

- `select` to select columns from a table
- `filter` to filter rows based on the contents of a column in the table
- `rename` to rename columns

We will encounter a few more `dplyr` commands during the course, we will explain their use as we come to them.

If you are familiar with R but not `dplyr` or `tidyverse` then we have a very brief introduction here. A more detailed introduction can be found in our online R course

Prepare count matrix

Create a raw counts matrix for data exploration

DESeq2 will use the txi object directly but we will need a counts matrix to do the data exploration.

```
rawCounts <- round(txi$counts, 0)
```

Filtering the genes

Many, if not most, of the genes in our annotation will not have been detected at meaningful levels in our samples - very low counts are most likely technical noise rather than biology. For the purposes of visualization it is important to remove the genes that are not expressed in order to avoid them dominating the patterns that we observe.

The level at which you filter at this stage will not effect the differential expression analysis. The cutoff used for filtering is a balance between removing noise and keeping biologically relevant information. A common approach is to remove genes that have less than a certain number of reads across all samples. The exact level is arbitrary and will depend to some extent on nature of the dataset (overall read depth per sample, number of samples, balance of read depth between samples etc). We will keep all genes where the total number of reads across all samples is greater than 5.

```
# check dimension of count matrix
dim(rawCounts)
```

```
## [1] 35896    12
```

```
# for each gene, compute total count and compare to threshold
# keeping outcome in vector of 'logicals' (ie TRUE or FALSE, or NA)
keep <- rowSums(rawCounts) > 5
# summary of test outcome: number of genes in each class:
table(keep, useNA = "always")
```

```
## keep
## FALSE  TRUE  <NA>
## 15805 20091     0
```

```
# subset genes where test was TRUE
filtCounts <- rawCounts[keep,]
# check dimension of new count matrix
dim(filtCounts)
```

```
## [1] 20091    12
```

Count distribution and Data transformations

Differential expression calculations with DESeq2 uses raw read counts as input, but for visualization purposes we use transformed counts.

Raw counts

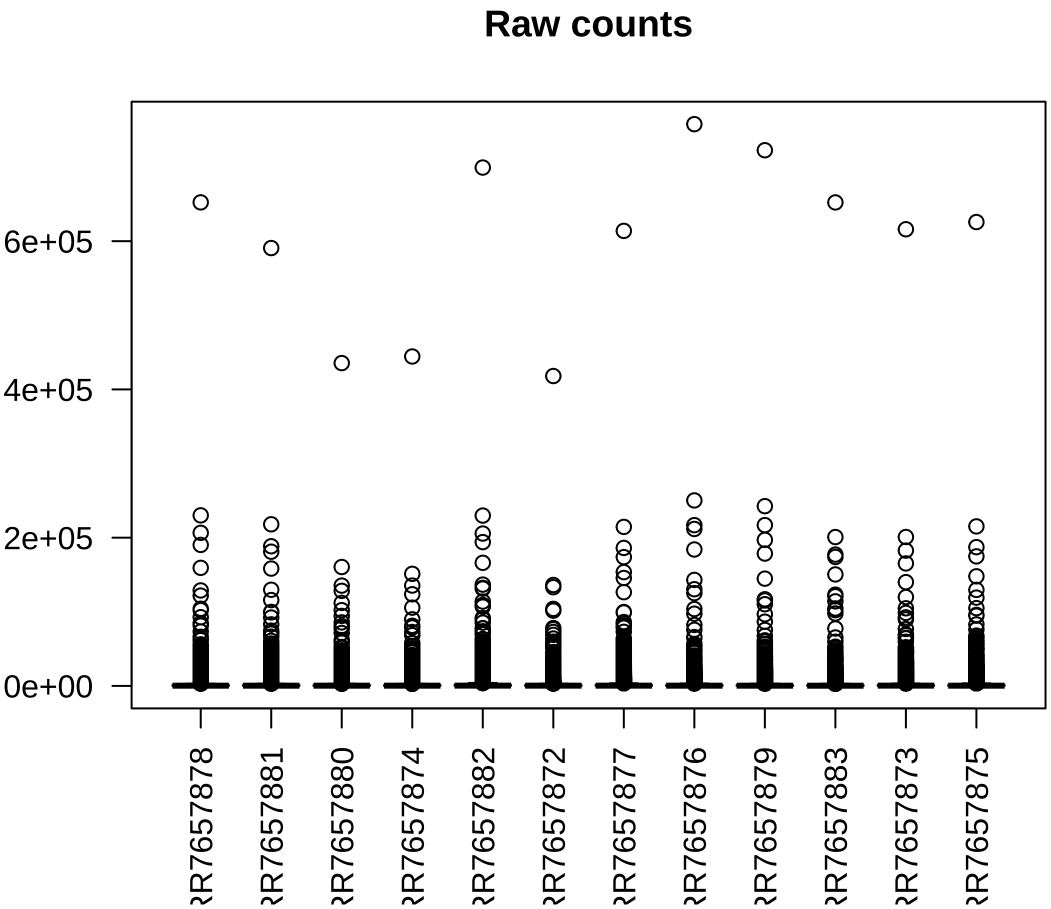
Why not raw counts? Two issues:

- The range of values in raw counts is very large with many small values and a few genes with very large values. This can make it difficult to see patterns in the data.

```
summary(filtCounts)
```

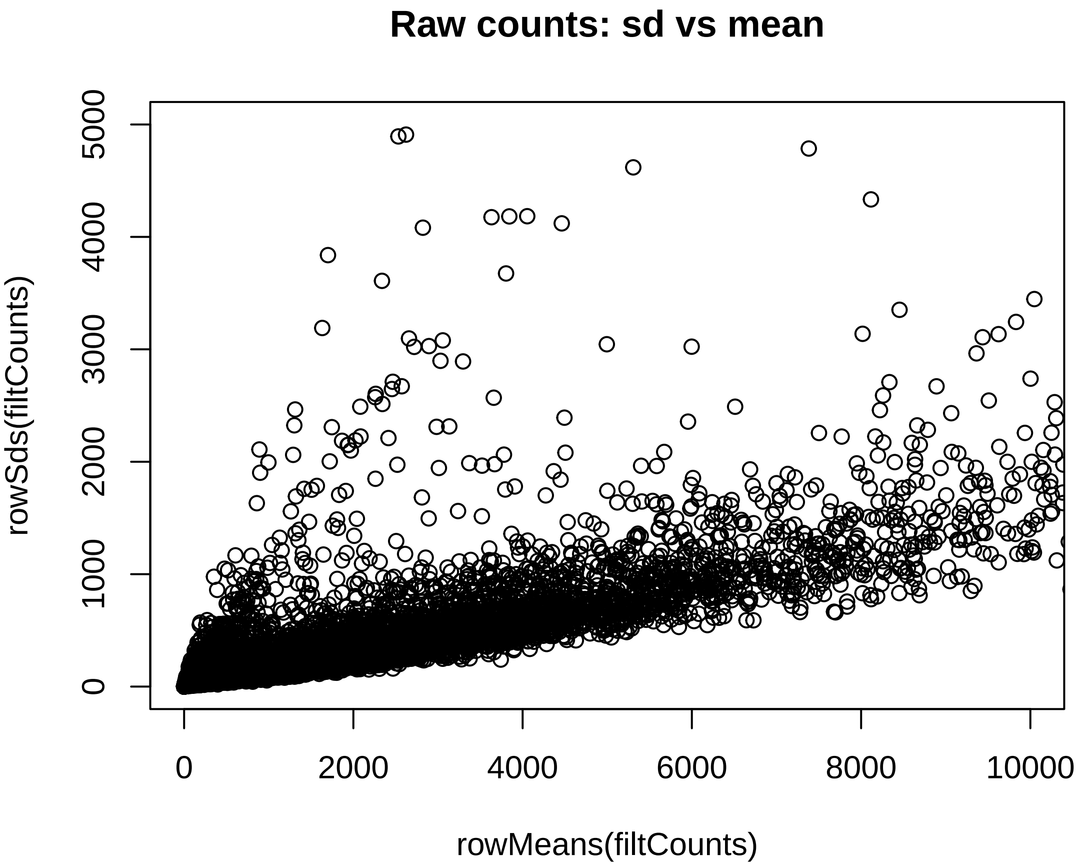
```
##    SRR7657878       SRR7657881       SRR7657880       SRR7657874
##  Min.   :     0   Min.   :     0   Min.   :     0   Min.   :     0
##  1st Qu.:    14   1st Qu.:    17   1st Qu.:    15   1st Qu.:    22
##  Median :   327   Median :   351   Median :   333   Median :   346
##  Mean   :  1387   Mean   :  1346   Mean   :  1330   Mean   :  1200
##  3rd Qu.:  1305   3rd Qu.:  1297   3rd Qu.:  1268   3rd Qu.:  1193
##  Max.   :652318   Max.   :590723   Max.   :435516   Max.   :444448
##    SRR7657882       SRR7657872       SRR7657877       SRR7657876
##  Min.   :     0   Min.   :     0   Min.   :     0   Min.   :     0
##  1st Qu.:    17   1st Qu.:    25   1st Qu.:    15   1st Qu.:    14
##  Median :   407   Median :   380   Median :   365   Median :   346
##  Mean   :  1696   Mean   :  1286   Mean   :  1536   Mean   :  1441
##  3rd Qu.:  1628   3rd Qu.:  1304   3rd Qu.:  1473   3rd Qu.:  1376
##  Max.   :699333   Max.   :418060   Max.   :613859   Max.   :757858
##    SRR7657879       SRR7657883       SRR7657873       SRR7657875
##  Min.   :     0   Min.   :     0   Min.   :     0   Min.   :     0
##  1st Qu.:    13   1st Qu.:    12   1st Qu.:    24   1st Qu.:    13
##  Median :   329   Median :   315   Median :   396   Median :   348
##  Mean   :  1363   Mean   :  1279   Mean   :  1430   Mean   :  1505
##  3rd Qu.:  1296   3rd Qu.:  1215   3rd Qu.:  1392   3rd Qu.:  1424
##  Max.   :722648   Max.   :652247   Max.   :616071   Max.   :625800
```

```
# few outliers affect distribution visualization
boxplot(filtCounts, main = 'Raw counts', las = 2)
```



- Variance increases with mean gene expression, this has impact on assessing the relationships, e.g. by clustering.

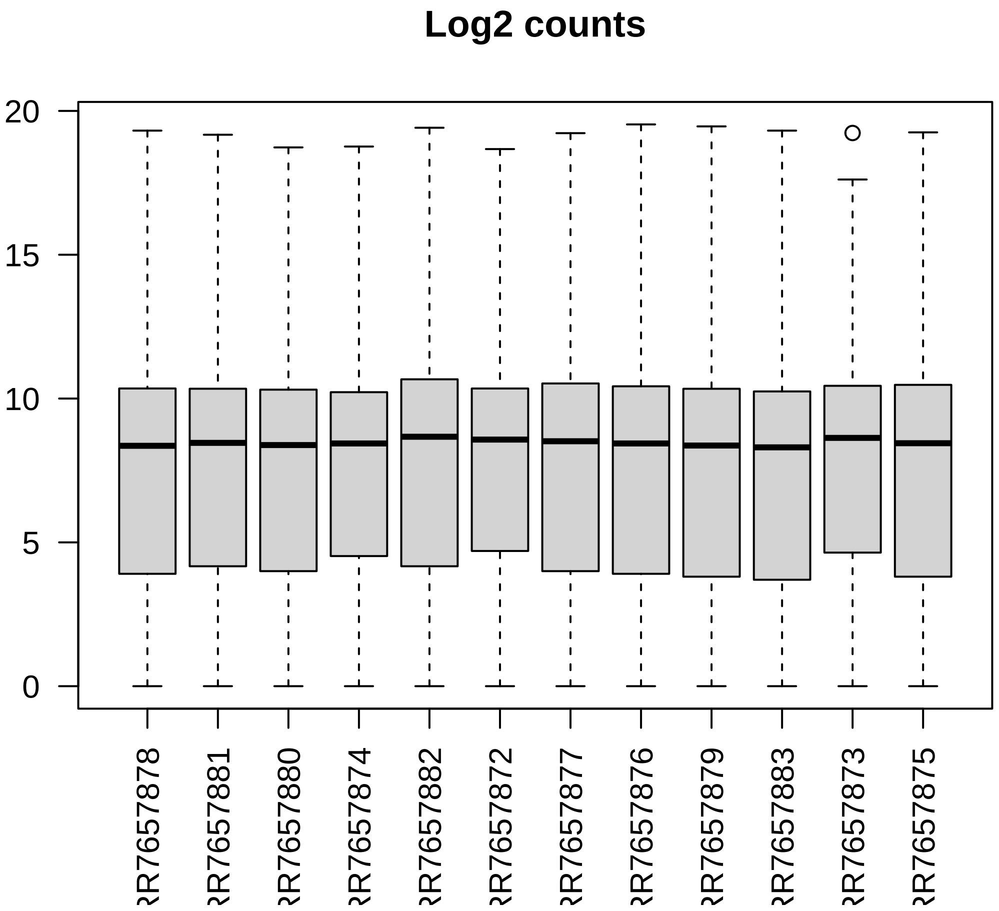
```
# Raw counts mean expression Vs standard Deviation (SD)
plot(rowMeans(filtCounts), rowSds(filtCounts),
     main = 'Raw counts: sd vs mean',
     xlim = c(0, 10000),
     ylim = c(0, 5000))
```



Data transformation

To avoid problems posed by raw counts, they can be transformed. A simple log2 transformation can be used to overcome the issue of the range of values. Note, when using a log transformation, it is important to add a small "pseudocount" to the data to avoid taking the log of zero.

```
logCounts <- log2(filtCounts + 1)
boxplot(logCounts, main = 'Log2 counts', las = 2)
```



However, this transformation does not account for the variance-mean relationship. DESeq2 provides two additional functions for transforming the data:

- `VST` : variance stabilizing transformation
- `rlog` : regularized log transformation

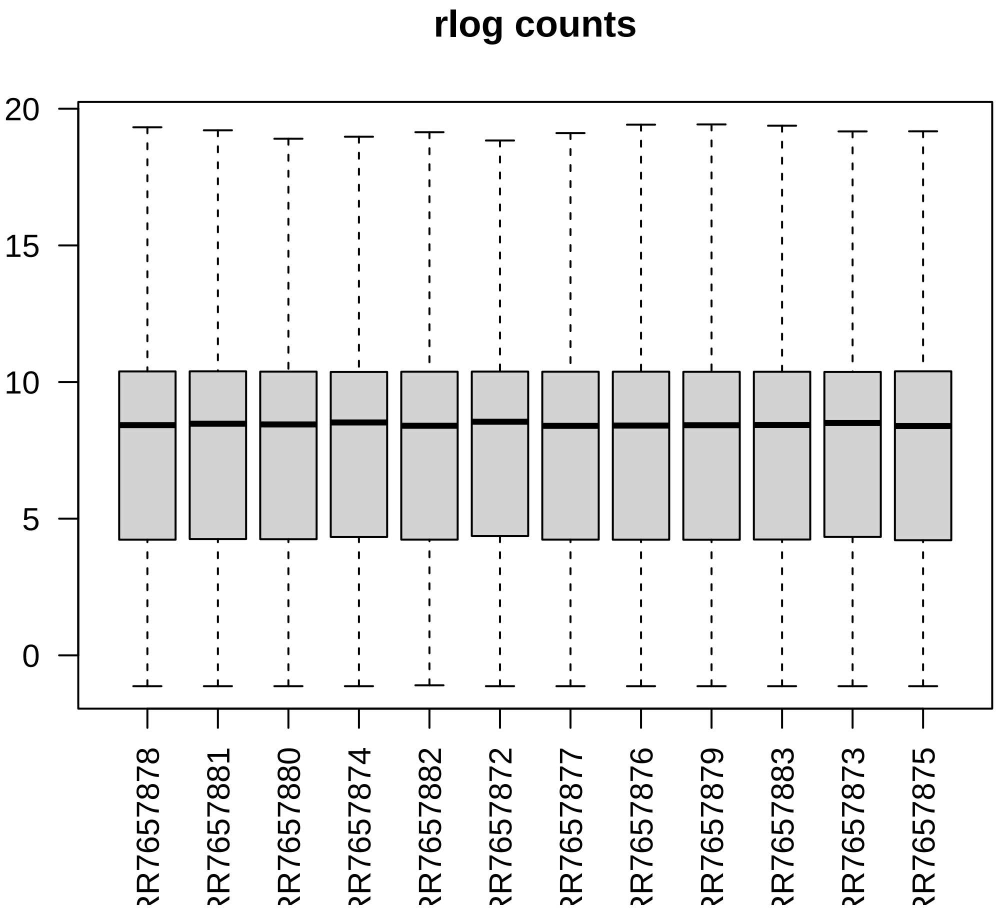
As well as log2 transforming the data, both transformations produce data which has been normalized with respect to library size and deal with the mean-variance relationship. The effects of the two transformations are similar. `rlog` is preferred when there is a large difference in library size between samples, however, it is considerably slower than `VST` and is not recommended for large datasets. For more information on the differences between the two transformations see the paper and the DESeq2 vignette.

Our data set is small, so we will use `rlog` for the transformation.

```
rlogcounts <- rlog(filtCounts)
```

```
## converting counts to integer mode
```

```
boxplot(rlogcounts, main = 'rlog counts', las = 2)
```



Principal Component Analysis

A principal component analysis (PCA) is an example of an unsupervised analysis, where we don't specify the grouping of the samples. If the experiment is well controlled and has worked well, we should find that replicate samples cluster closely, whilst the greatest sources of variation in the data should be between treatments/sample groups. It is also an incredibly useful tool for checking for outliers and batch effects.

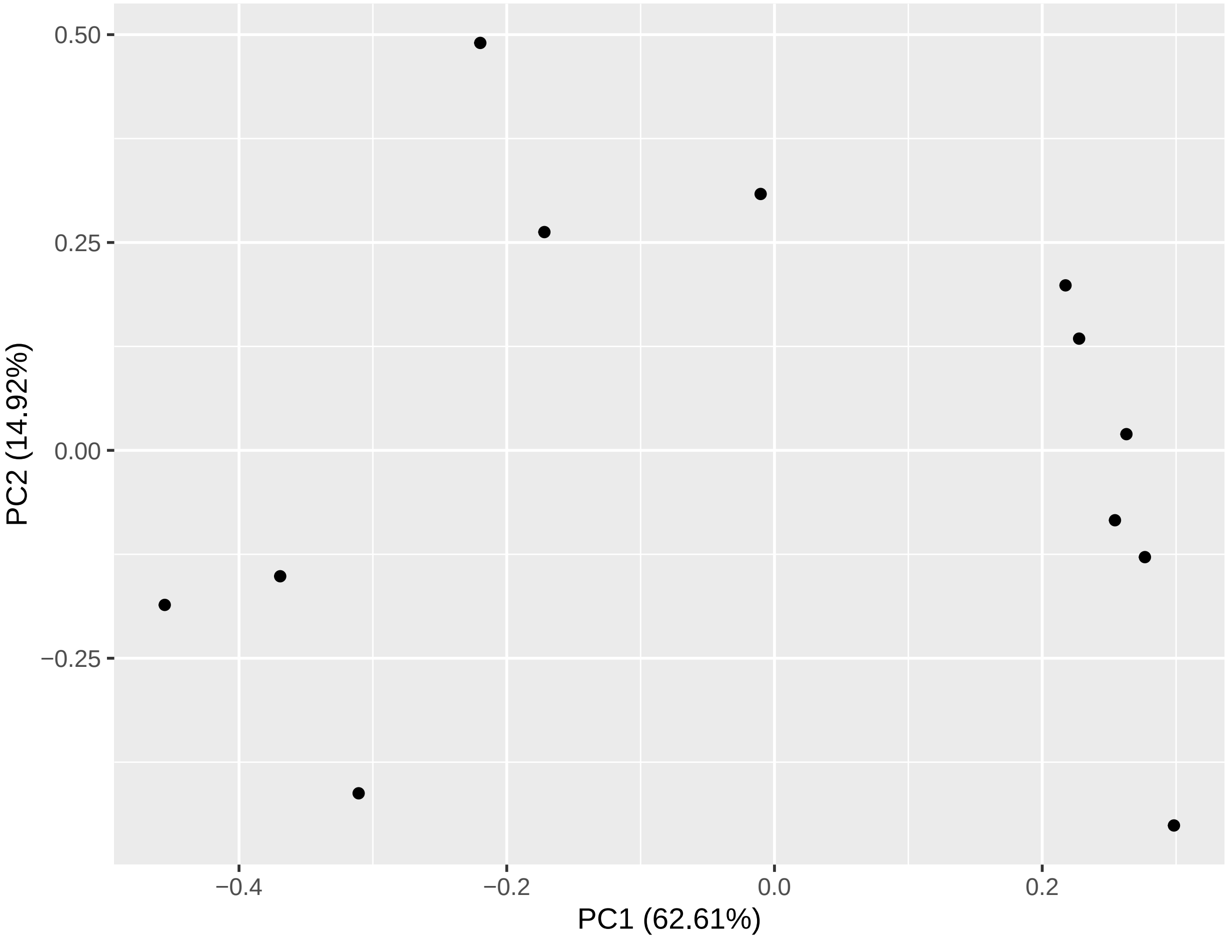
To run the PCA we should first normalise our data for library size and transform to a log scale. DESeq2 provides two separate commands to do this (`vst` and `rlog`). Here we will use the command `rlog`. `rlog` performs a log2 scale transformation in a way that compensates for differences between samples for genes with low read count and also normalizes between samples for library size.

You can read more about `rlog`, its alternative `vst` and the comparison between the two here.

To plot the PCA results we will use the `autoplot` function from the `ggfortify` package (Tang, Horikoshi, and Li 2016). `ggfortify` is built on top of `ggplot2` and is able to recognise common statistical objects such as PCA results or linear model results and automatically generate summary plot of the results in an appropriate manner.

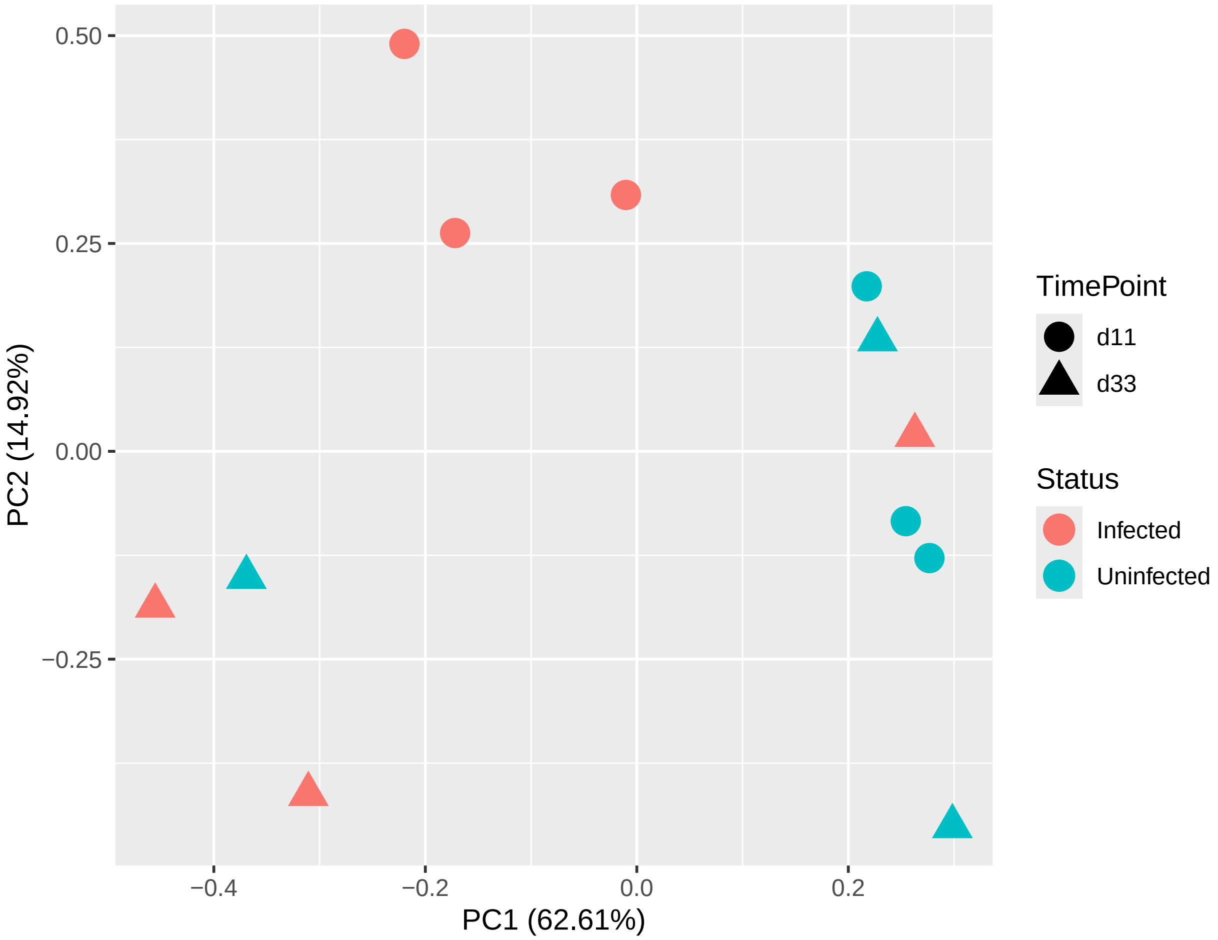
```
library(ggfortify)

rlogcounts <- rlog(filtCounts)

# run PCA
pcDat <- prcomp(t(rlogcounts))
# plot PCA
```

```
autoplot(pcDat)
```



We can use colour and shape to identify the Cell Type and the Status of each sample.

```
autoplot(pcDat,
         data = sampleinfo,
         colour = "Status",
         shape = "TimePoint",
         size = 5)
```



Exercise

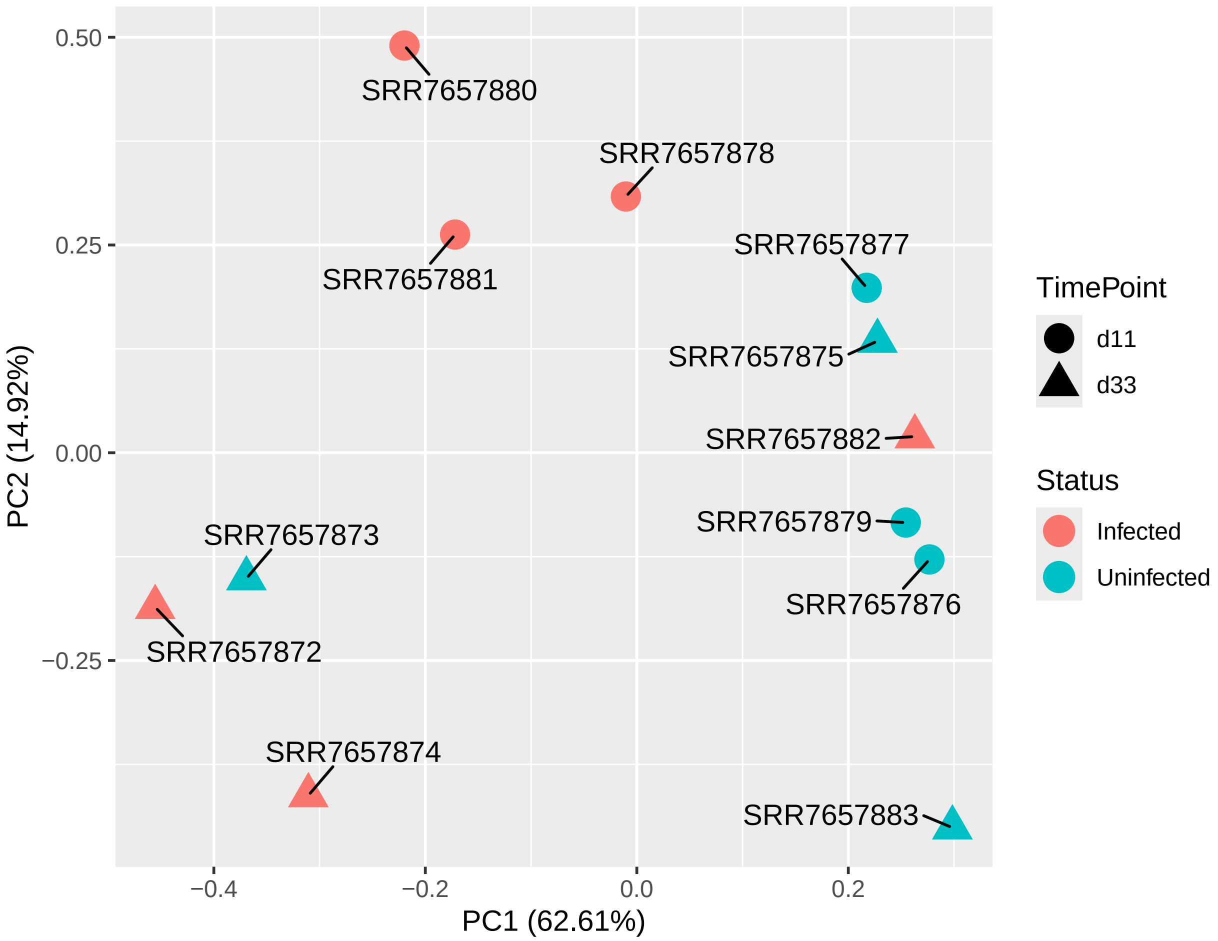
The plot we have generated shows us the first two principle components. This shows us the relationship between the samples according to the two greatest sources of variation. Sometime, particularly with more complex experiments with more than two experimental factors, or where there might be confounding factors, it is helpful to look at more principle components.

1. Redraw the plot, but this time plot the 2nd principle component on the x-axis and the 3rd prinicple component on the y axis. To find out how to do the consult the help page for the `prcomp` data method for the `autoplot` function: `?autoplot.prcomp`.

Discussion: What do the PCA plots tell us about our samples?

Let's identify these samples. The package `ggrepel` allows us to add text to the plot, but ensures that points that are close together don't have their labels overlapping (they *repel* each other).

```
library(ggrepel)

# setting shape to FALSE causes the plot to default to using the labels instead of points
autoplot(pcDat,
         data = sampleinfo,
         colour = "Status",
         shape = "TimePoint",
         size = 5) +
    geom_text_repel(aes(x = PC1, y = PC2, label = SampleName), box.padding = 0.8)
```



The mislabelled samples are *SRR7657882*, which is labelled as *Infected* but should be *Uninfected*, and *SRR7657873*, which is labelled as *Uninfected* but should be *Infected*. Let's fix the sample sheet.

We're going to use another `dplyr` command `mutate`.

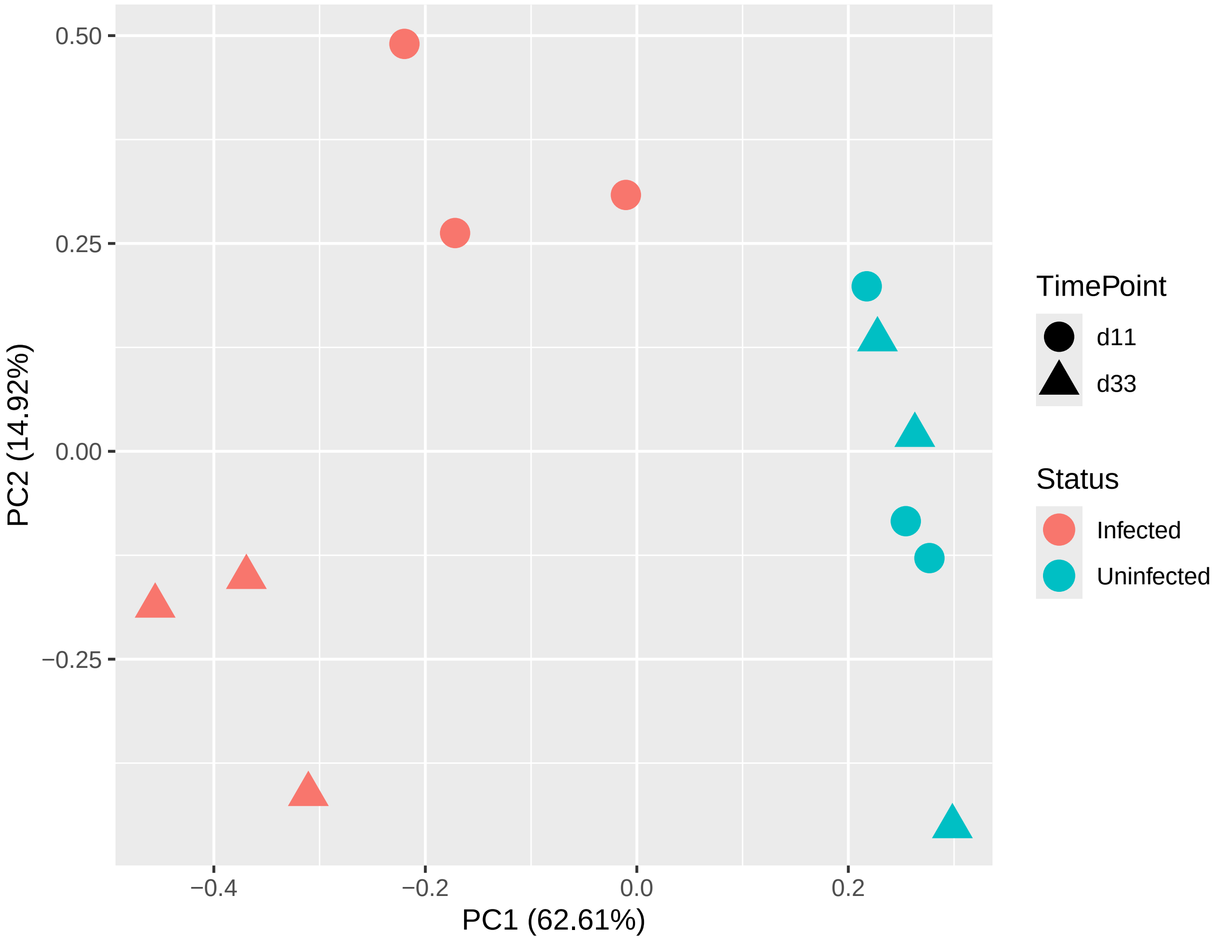
```
sampleinfo <- mutate(sampleinfo, Status = case_when(
                                           SampleName=="SRR7657882" ~ "Uninfected",
                                           SampleName=="SRR7657873" ~ "Infected",
                                           TRUE ~ Status))
```

. . . and export it so that we have the correct version for later use.

```
write_tsv(sampleinfo, "results/SampleInfo_Corrected.txt")
```

Let's look at the PCA now.

```
autoplot(pcDat,
         data = sampleinfo,
         colour = "Status",
         shape = "TimePoint",
         size = 5)
```



Replicate samples from the same group cluster together in the plot, while samples from different groups form separate clusters. This indicates that the differences between groups are larger than those within groups. The biological signal of interest is stronger than the noise (biological and technical) and can be detected.

Also, there appears to be a strong difference between days 11 and 33 post infection for the infected group, but the day 11 and day 33 samples for the uninfected are mixed together.

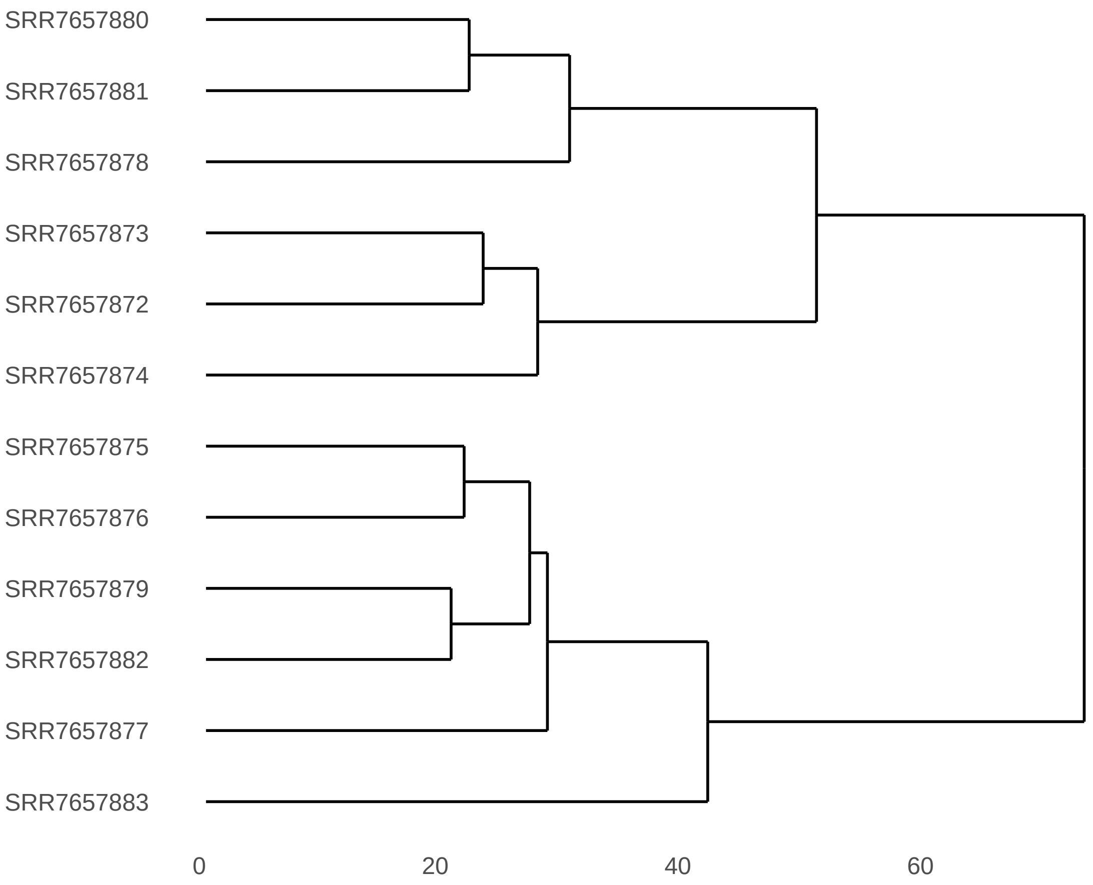
Clustering in the PCA plot can be used to motivate changes to the design matrix in light of potential batch effects. For example, imagine that the first replicate of each group was prepared at a separate time from the second replicate. If the PCA plot showed separation of samples by time, it might be worthwhile including time in the downstream analysis to account for the time-based effect.

Hierachical clustering

Earlier, we used principle component analysis to assess sources of variation in the data set and the relationship between the samples. Another method for looking at the relationship between the samples can be to run hierarchical clustering based on the Euclidean distance between the samples. Hierarchical clustering can often provide a clearer view of the clustering of the different sample groups than other methods such as PCA.

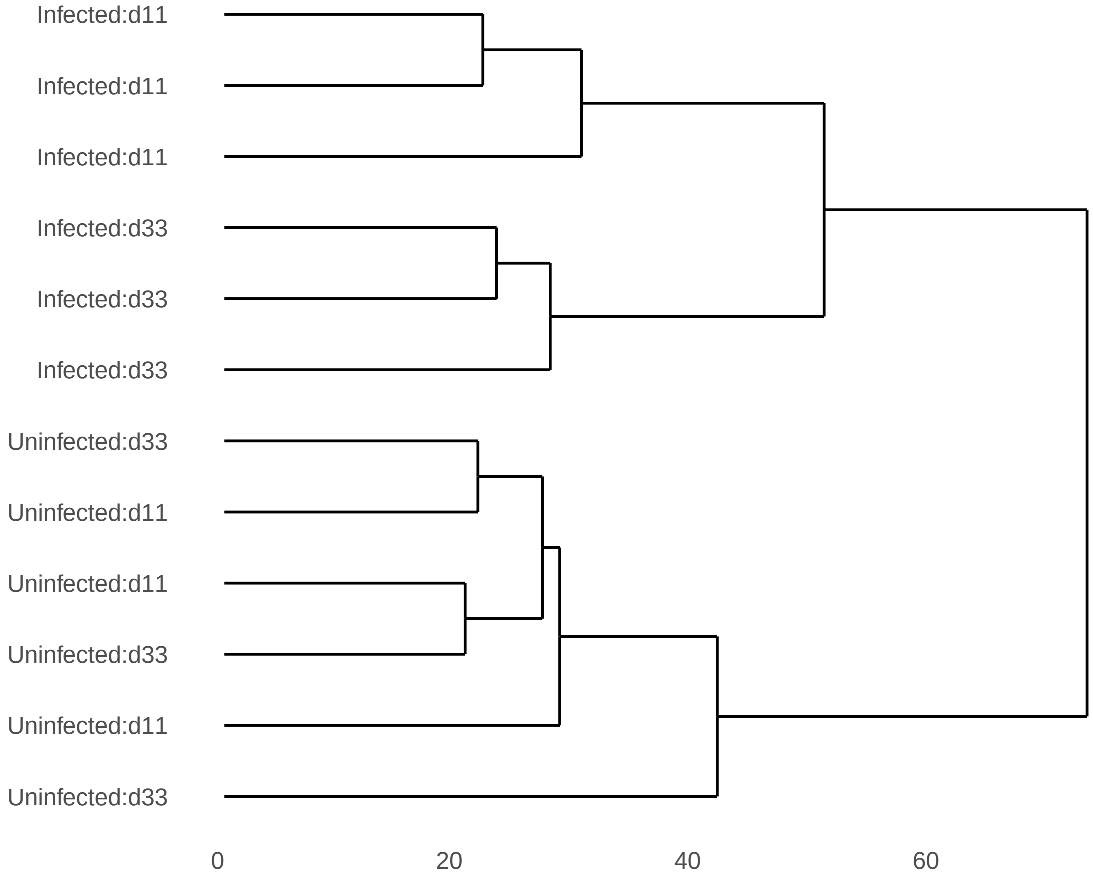
We will use the package `ggdendro` to plot the clustering results using the function `ggdendrogram`.

```
library(ggdendro)
hclDat <-  t(rlogcounts) %>%
   dist(method = "euclidean") %>%
   hclust()
ggdendrogram(hclDat, rotate = TRUE)
```



We really need to add some information about the sample groups. The simplest way to do this would be to replace the `labels` in the `hclust` object. Conveniently the labels are stored in the `hclust` object in the same order as the columns in our counts matrix, and therefore the same as the order of the rows in our sample meta data table. We can just substitute in columns from the metadata.

```r
hclDat2 <- hclDat
hclDat2$labels <- str_c(sampleinfo$Status, ":", sampleinfo$TimePoint)
ggdendrogram(hclDat2, rotate = TRUE)
```



We can see from this that the infected and uninfected samples cluster separately and that day 11 and day 33 samples cluster separately for infected samples, but not for uninfected samples.

References

Hu, Rui-Si, Jun-Jun He, Hany M. Elsheikha, Yang Zou, Muhammad Ehsan, Qiao-Ni Ma, Xing-Quan Zhu, and Wei Cong. 2020. "Transcriptomic Profiling of Mouse Brain During Acute and Chronic Infections by Toxoplasma Gondii Oocysts." *Frontiers in Microbiology* 11: 2529. https://doi.org/10.3389/fmicb.2020.570903.

Patro, Duggal, R. 2017. "Salmon Provides Fast and Bias-Aware Quantification of Transcript Expression." *Nature Methods* 14: 417–19. https://doi.org/10.1038/nmeth.4197.

Tang, Yuan, Masaaki Horikoshi, and Wenxuan Li. 2016. "Ggfortify: Unified Interface to Visualize Statistical Result of Popular r Packages." *The R Journal* 8. https://journal.r-project.org/.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2018. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.