Introduction to Bulk RNAseq data analysis
Initial exploration of RNA-seq data

Last modified: 21 Apr 2021

Contents

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

- look at the effects of variance and how to mitigate this with data transformation

- do some initial exploration of the raw count data using principle component analysis

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"))
sampleinfo %>%
  arrange(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 <- str_c("salmon/", sampleinfo$SampleName, "/quant.sf")
files <- set_names(files, sampleinfo$SampleName)

tx2gene <- read_tsv("references/tx2gene.tsv")
```

```
##
## -- Column specification -------------------------------------------------
## cols(
##   TxID = col_character(),
##   GeneID = col_character()
## )
```

```
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] 20.381 0 1.966 1.059 0.949 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:35896] "ENSMUSG00000000001" "ENSMUSG00000000003" "ENSMUSG00000000028" "ENSMUSG00000
##   .. ..$ : 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" "ENSMUSG00000
##   .. ..$ : chr [1:12] "SRR7657878" "SRR7657881" "SRR7657880" "SRR7657874" ...
##  $ length           : num [1:35896, 1:12] 2905 541 1884 2100 480 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:35896] "ENSMUSG00000000001" "ENSMUSG00000000003" "ENSMUSG00000000028" "ENSMUSG00000
##   .. ..$ : chr [1:12] "SRR7657878" "SRR7657881" "SRR7657880" "SRR7657874" ...
##  $ countsFromAbundance: chr "no"
```

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

```
##                     SRR7657878 SRR7657881 SRR7657880 SRR7657874 SRR7657882
## ENSMUSG00000000001    1039.000   1005.888    892.000    917.360   1136.691
## ENSMUSG00000000003       0.000      0.000      0.000      0.000      0.000
## ENSMUSG00000000028      65.000     74.000     72.000     44.000     45.999
## ENSMUSG00000000037      39.000     47.000     29.001     54.001     67.000
## ENSMUSG00000000049       8.000      9.000      4.000      4.000      4.000
```

```
## ENSMUSG00000000056    2163.468   2067.819   2006.924   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     52.000     55.999     58.000
## ENSMUSG00000000037      62.000     69.000     35.000     60.000     21.001
## ENSMUSG00000000049       9.001      6.000     10.000      4.000      8.000
## ENSMUSG00000000056    1412.733   2154.230   2121.740   1962.000   2274.701
##                       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")
```

Exercise 1

> We have loaded in the raw counts here. These are what we need for the differential expression analysis. For other investigations we might want counts normalised to library size. `tximport` allows us to import "transcript per million" (TPM) scaled counts instead.
>
> 1. Create a new object called `tpm` that contains length scaled TPM counts. You will need to add an extra argument to the command. Use the help page to determine how you need to change the code: `?tximport`.

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. It also makes use of a new symbol `%>%`, called the "pipe," which makes the code a bit tidier.

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.

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

Filtering the genes

For many analysis methods it is advisable to filter out as many genes as possible before the analysis to decrease the impact of multiple testing correction on false discovery rates. This is normally done by filtering out genes with low numbers of reads and thus likely to be uninformative.

With `DESeq2` this is however not necessary as it applies `independent filtering` during the analysis. On the other hand, some filtering for genes that are very lowly expressed does reduce the size of the data matrix, meaning that less memory is required and processing steps are carried out faster. Furthermore, 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.

We will keep all genes where the total number of reads across all samples is greater than 5.

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

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

```r
# 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
```

```r
# 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:

- Raw counts range is very large
- Variance increases with mean gene expression, this has impact on assessing the relationships.

```r
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.   :652317    Max.   :590722    Max.   :435515    Max.   :444447
##   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.   :699342    Max.   :418059    Max.   :613857    Max.   :757857
##   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 :   316    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.   :722647    Max.   :652247    Max.   :616070    Max.   :625798
```

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


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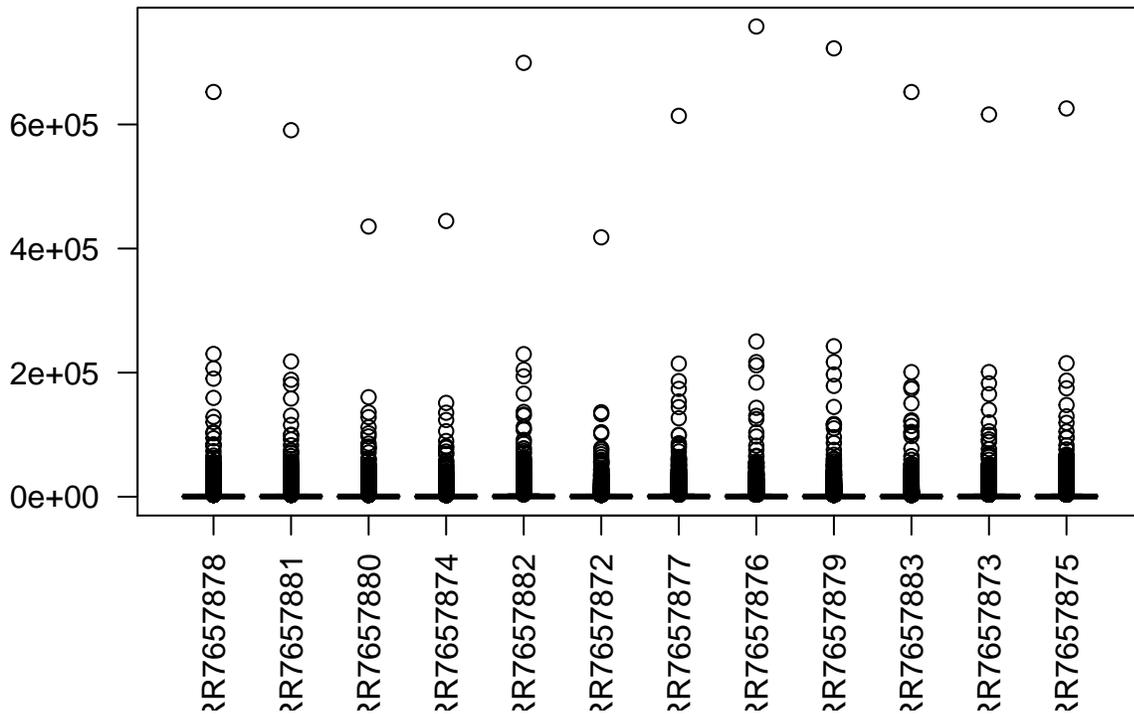
```r
# Raw counts mean expression Vs standard Deviation (SD)
plot(rowMeans(filtCounts), rowSds(filtCounts),
    main='Raw counts: sd vs mean',
    xlim=c(0,10000),
    ylim=c(0,5000))
```

Raw counts: sd vs mean
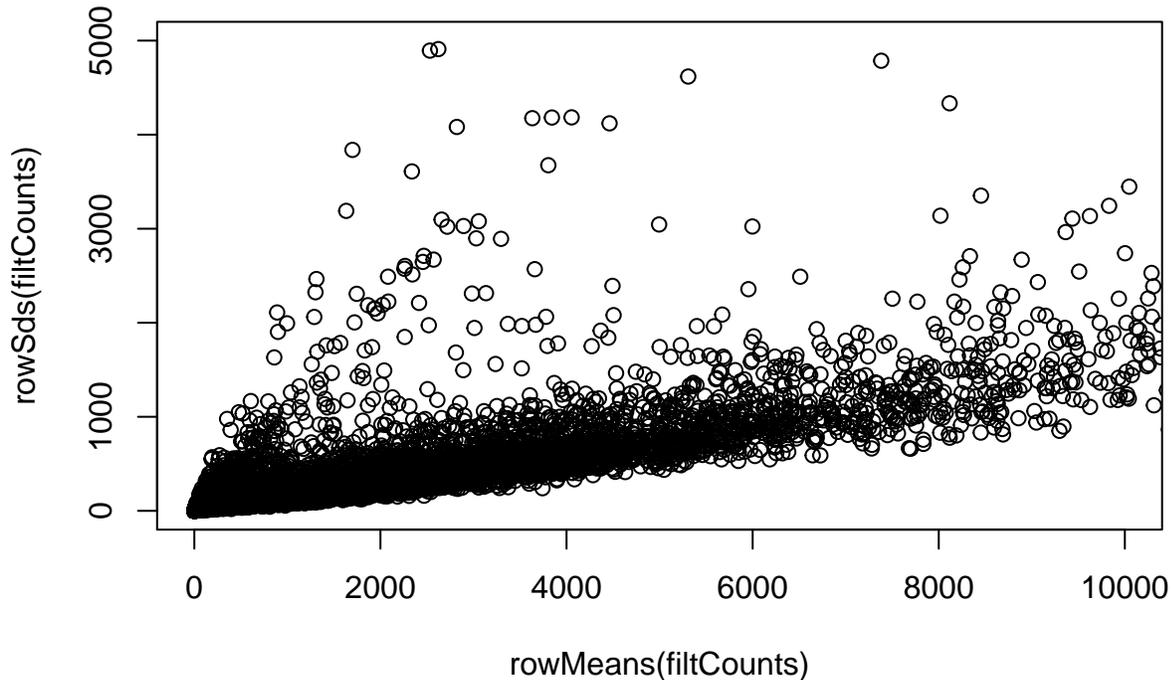
Data transformation

To avoid problems posed by raw counts, they can be transformed. Several transformation methods exist to limit the dependence of variance on mean gene expression:

- Simple log2 transformation
- VST : variance stabilizing transformation
- rlog : regularized log transformation

log2 transformation

Because some genes are not expressed (detected) in some samples, their count are 0. As log2(0) returns -Inf in R which triggers errors by some functions, we add 1 to every count value to create 'pseudocounts.' The lowest value then is 1, or 0 on the log2 scale ($\log2(1) = 0$).

```r
# Get log2 counts
logcounts <- log2(filtCounts + 1)
# summary(logcounts[,1]) # summary for first column
# summary(logcounts) # summary for each column
```

We will check the distribution of read counts using a boxplot and add some colour to see if there is any difference between sample groups.

```r
# make a colour vector
statusCols <- str_replace_all(sampleinfo$Status, c(Infected="red", Uninfected="orange"))

# Check distributions of samples using boxplots
boxplot(logcounts,
        xlab="",
```

```
        ylab="Log2(Counts)",
        las=2,
        col=statusCols,
        main="Log2(Counts)")
# Let's add a blue horizontal line that corresponds to the median
abline(h=median(logcounts), col="blue")
```

Log2(Counts)
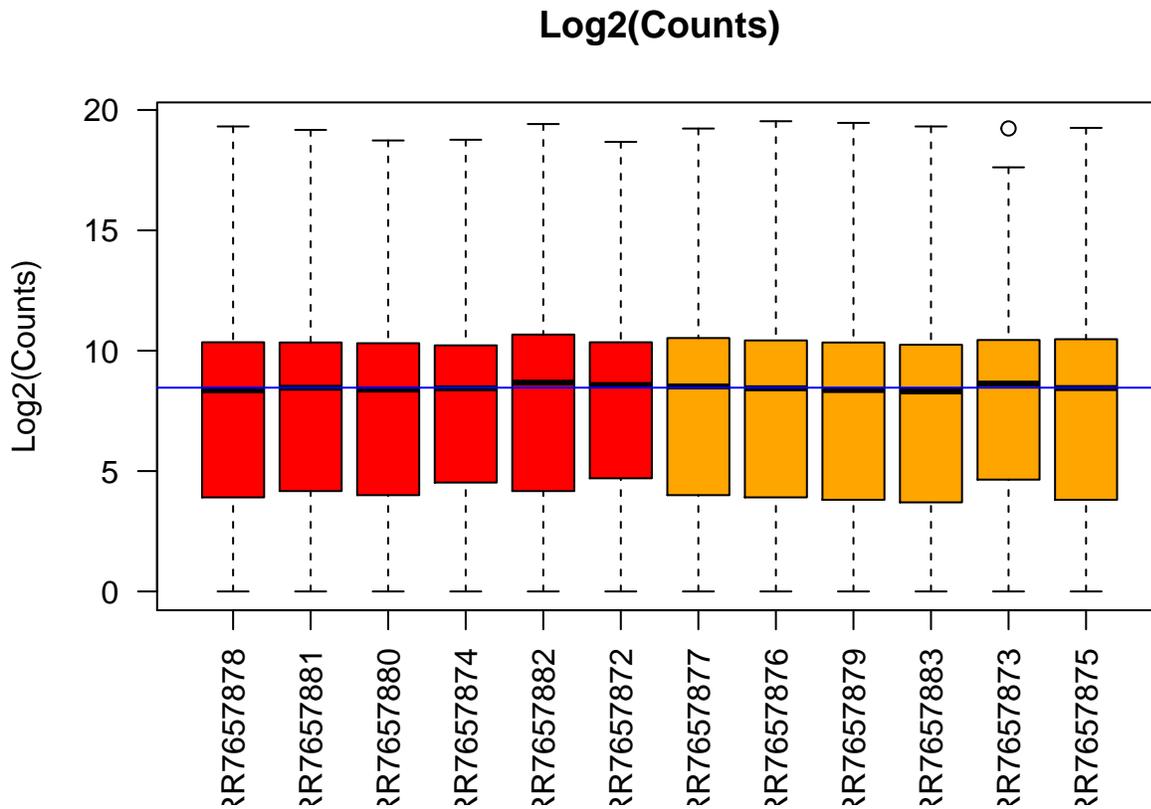
From the boxplots we see that overall the density distributions of raw log-counts are not identical but still not very different. If a sample is really far above or below the blue horizontal line (overall median) we may need to investigate that sample further.

```
# Log2 counts standard deviation (sd) vs mean expression
plot(rowMeans(logcounts), rowSds(logcounts),
     main='Log2 Counts: sd vs mean')
```

Log2 Counts: sd vs mean
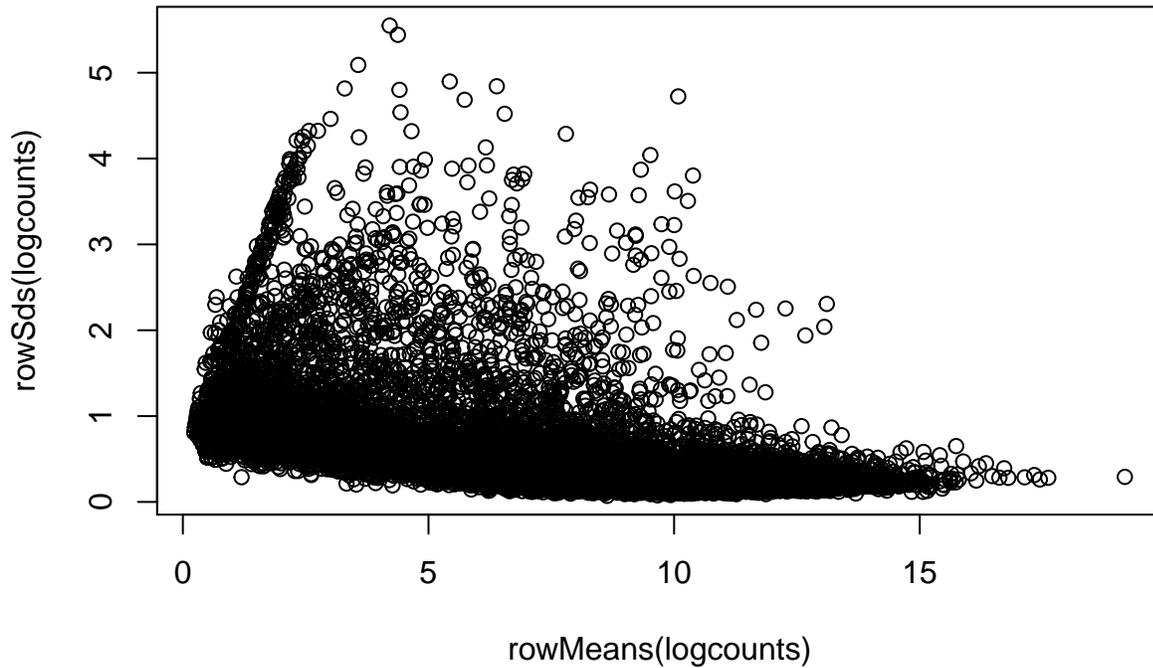
In contrast to raw counts, with log2 transformed counts lowly expressed genes show higher variation.

VST : variance stabilizing transformation

Variance stabilizing transformation (VST) aims at generating a matrix of values for which variance is constant across the range of mean values, especially for low mean.

The `vst` function computes the fitted dispersion-mean relation, derives the transformation to apply and accounts for library size.

```r
vst_counts <- vst(filtCounts)

# Check distributions of samples using boxplots
boxplot(vst_counts,
        xlab="",
        ylab="VST counts",
        las=2,
        col=statusCols)
# Let's add a blue horizontal line that corresponds to the median
abline(h=median(vst_counts), col="blue")
```
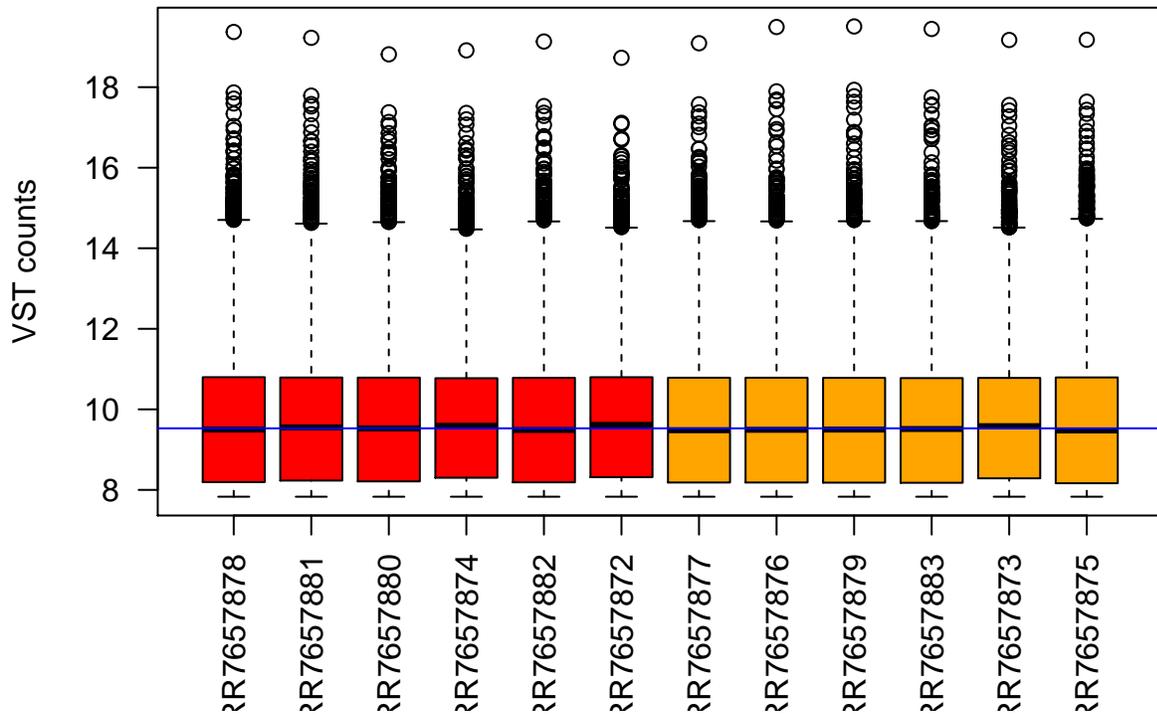
```
# VST counts standard deviation (sd) vs mean expression
plot(rowMeans(vst_counts), rowSds(vst_counts),
     main='VST counts: sd vs mean')
```

VST counts: sd vs mean
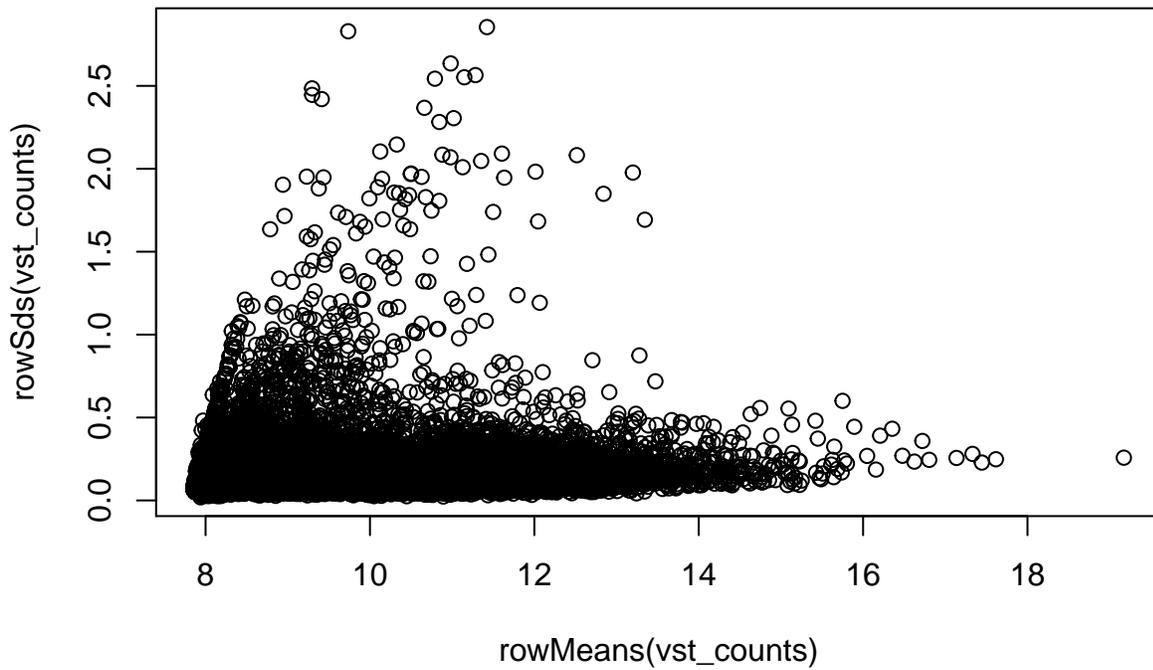
Exercise 2

1. Use the `DESeq2` function `rlog` to transform the count data. This function also normalises for library size.
2. Plot the count distribution boxplots with this data
   How has this affected the count distributions?

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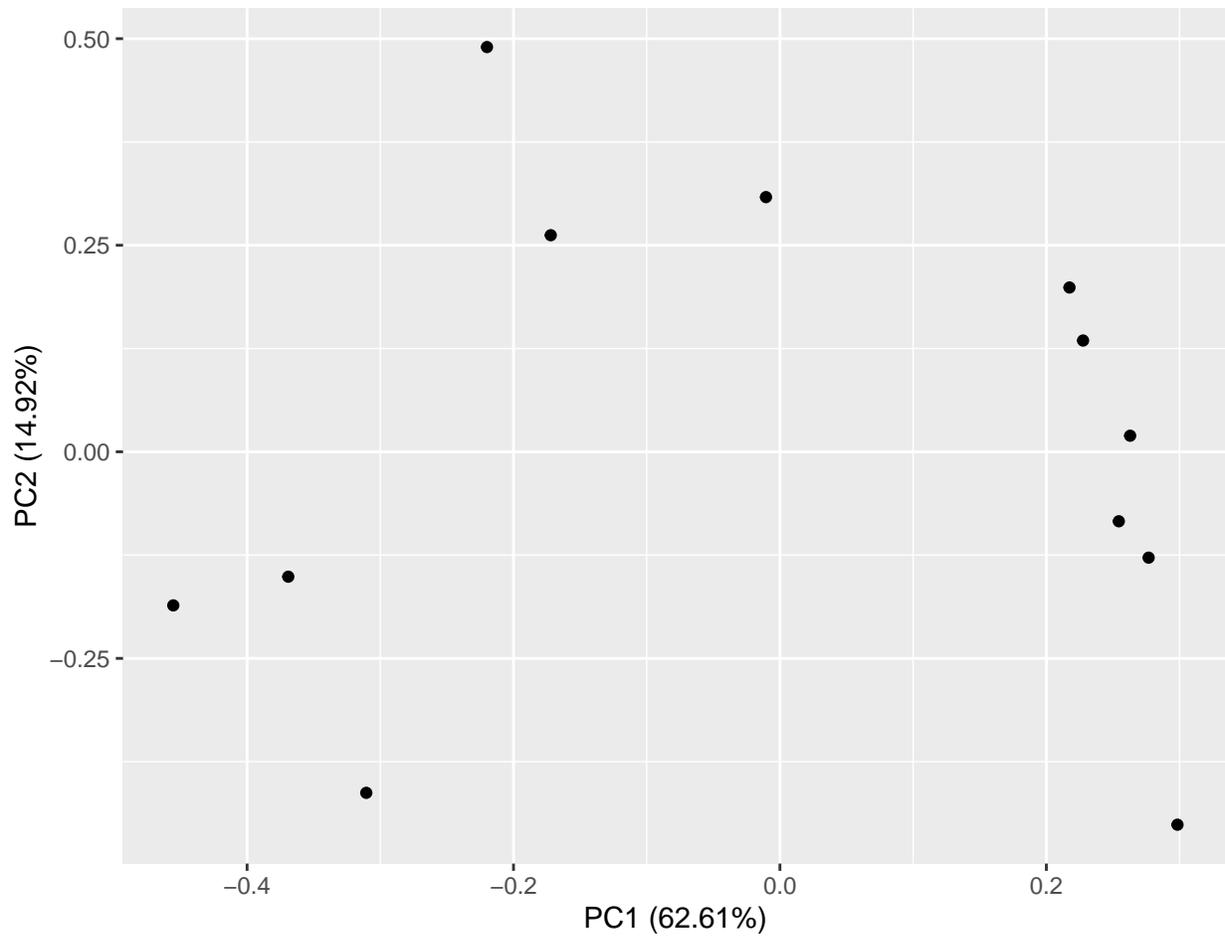
We can use colour and shape to identify the Cell Type and the Status of each sample.

```
autoplot(pcDat,
         data = sampleinfo,
         colour="Status",
         shape="TimePoint",
         size=5)
```
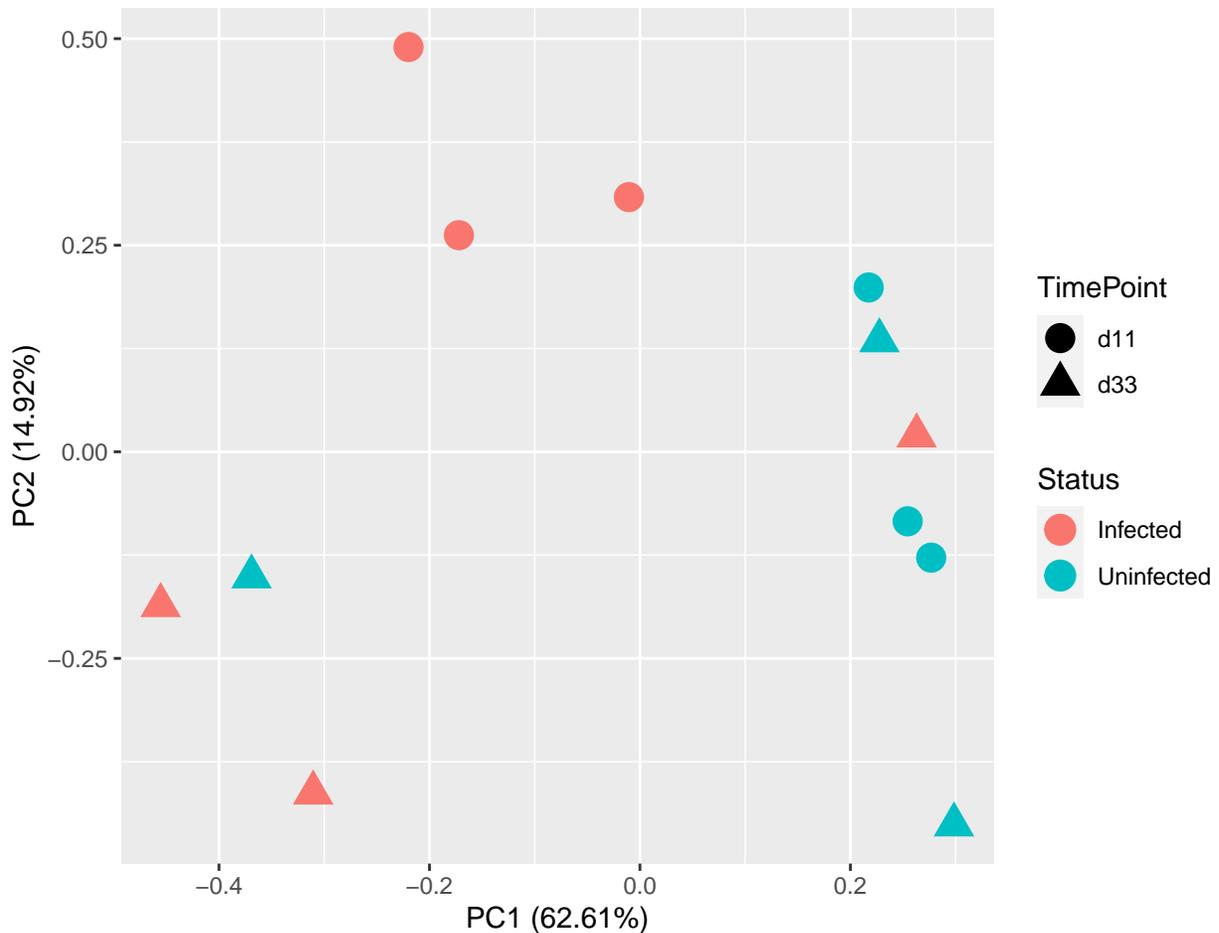
Exercise 3

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)
```
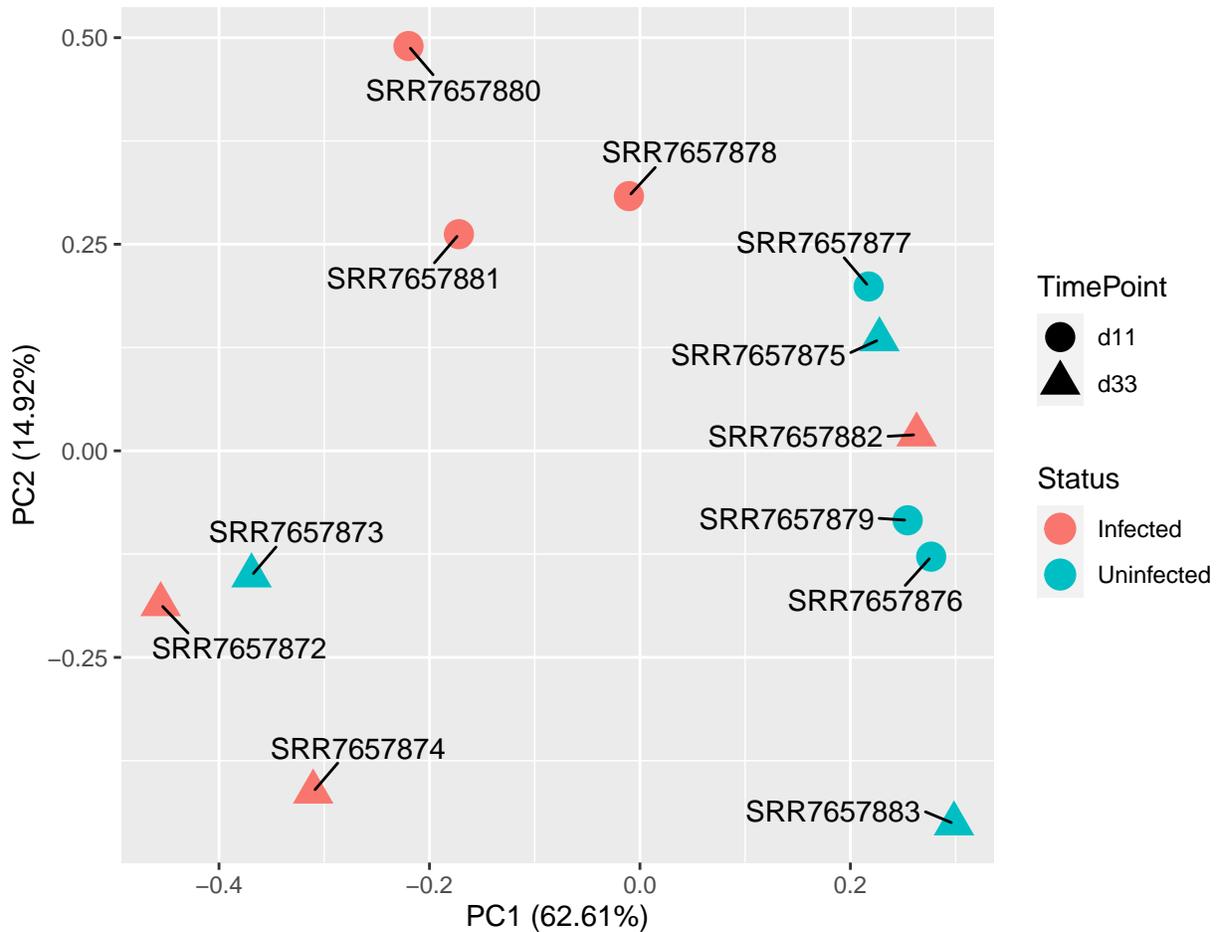

The mislabelled samples are *SRR7657882*, which is labelled as *Test* but should be *Control*, and *SRR7657873*, which is labelled as *Control* but should be *Test*. Let's fix the sample sheet.

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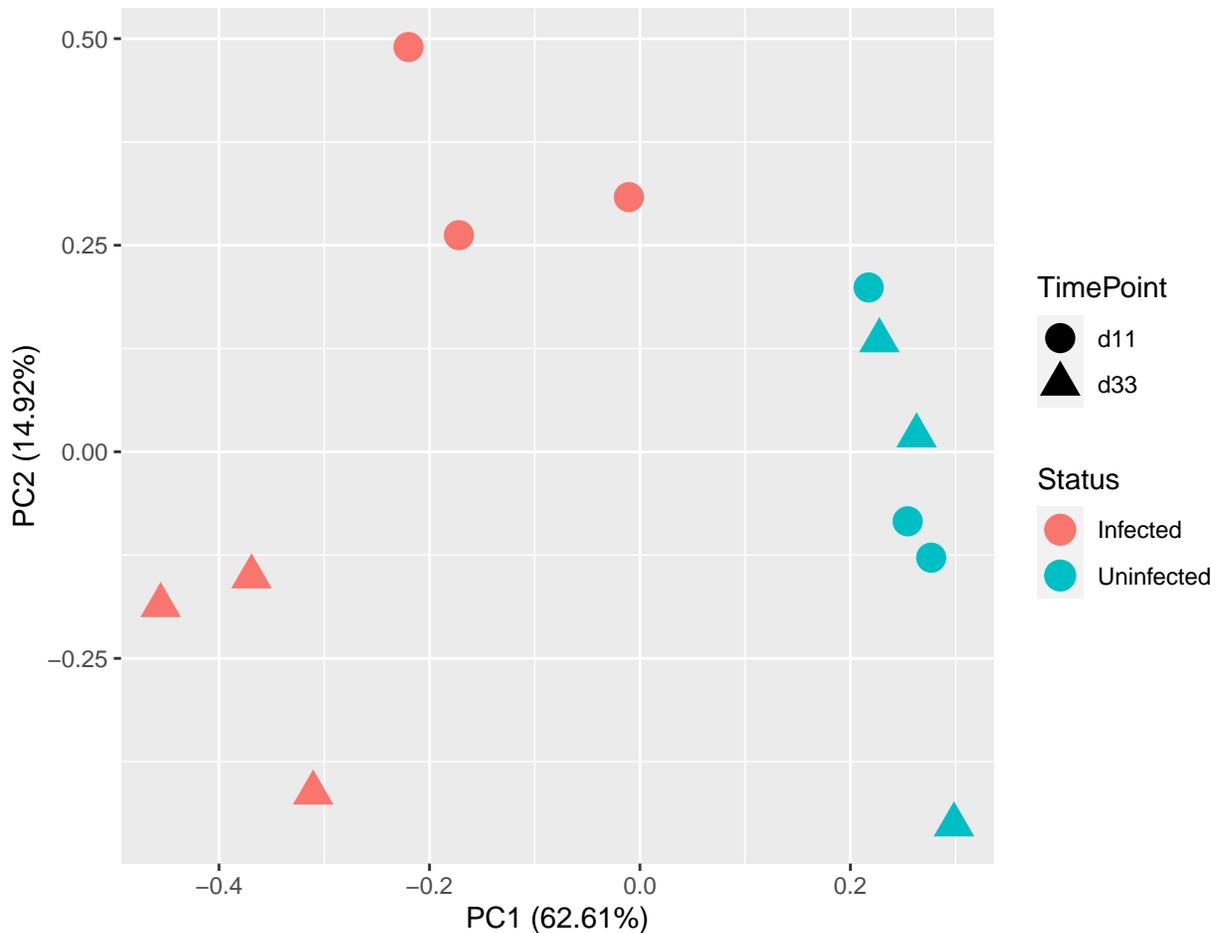
Replicate samples from the same group cluster together in the plot, while samples from different groups form separate clusters. This indicates that the differences between groups are larger than those within groups. The biological signal of interest is stronger than the noise (biological and technical)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

Also, there appears to be a strong difference between days 11 and 33 post infection for the test group, but the day 11 and day 33 samples for the controls 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.

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.