# Lab Journal - Gene Expression Analysis

Noise exposures causing hearing loss generate proteotoxic stress and activate the proteostasis network

**Student**: Vincent Talen

**Student number**: 389015

**Class**: BFV2

**Study**: Bio-Informatica

**Institute**: Institute for Life Science & Technology

**Teacher**: Marcel Kempenaar

**Date**: 2022-03-29

## Contents

## 1 R Setup

To correctly create this document and to be able to run all the R code a few settings have to be set and libraries imported. All the code chunks need to be visible in the end pdf so the echo is set to true and to render the pdf's faster whilst developing the cache is also set to true. Multiple packages are used in this project, all of them are placed in a vector which then can be used to load all the packages on a single line with lapply. For faster rendering times when using DESeq2 a higher core amount it specified.

```r
# Set code chunks visibility in pdf output to true
knitr::opts_chunk$set(echo = TRUE)
# Use cache
knitr::opts_chunk$set(cache = TRUE)

# Create vector with all packages that will be used
packages <- c("affy", "pander", "scales", "BiocParallel", "DESeq2", "pheatmap",
              "PoiClaClu", "ggplot2")
# Load each package in the vector with lapply
invisible(lapply(packages, library, character.only = TRUE))
# Drop the packages variable from memory since it will not be used again
remove(packages)
# Set cores to be used for when using Bioc DESeq2
register(MulticoreParam(6))
```

# 2 Exploratory Data Analysis

## 2.1 Preparing the data

To start this project out the data set will be loaded into the memory so the statistics can be performed on it. The first few lines will be shown to show what the contents of the data frame are like and dim() is used to show the length of the data set.

```
# Load in the data set into a data frame
myData <- read.table("GSE160639_RawReadCount.txt", header = T, sep = "\t")
# Change the int row names to the gene_id column and delete the gene_id column afterwards
row.names(myData) <- myData$gene_id
myData <- myData[-1]
# Show the first few lines of the data frame
head(myData)
```

```
##                      A1_70dB A2_70dB A3_70dB A4_70dB B1_94dB B2_94dB B3_94dB
## ENSMUSG00000064351  807021  844885  746507  600313  455027  512010  325579
## ENSMUSG00000069919  366445  395997  269558  284141  302984  427023  410969
## ENSMUSG00000052305  243346  341576  226918  233215  249790  301111  480944
## ENSMUSG00000015090  352912  323278  255015  234177  318835  364719  366617
## ENSMUSG00000064370  381672  361431  312361  262815  222349  253774  159885
## ENSMUSG00000001506  312965  305261  233933  193533  256679  279963  202360
##                      B4_94dB C1_105dB C2_105dB C4_105dB C5_105dB
## ENSMUSG00000064351  416803   745783   521567   444325   761621
## ENSMUSG00000069919  408831   390024   343923   474316   401705
## ENSMUSG00000052305  634849   320317   371479   321232   328205
## ENSMUSG00000015090  381912   307553   251633   333419   258235
## ENSMUSG00000064370  191874   360531   267048   265633   310993
## ENSMUSG00000001506  216174   340531   230902   196198   222483
```

```
# Print the amount of genes and columns
cat("Amount of genes:", dim(myData)[1], "\tColumns in dataframe", dim(myData)[2])
```

```
## Amount of genes: 35496   Columns in dataframe 12
```

By using the str() function the structure of the dataframe/dataset can be seen. Every column consists only of int values, so only the count data. (Not shown in PDF)

```
str(myData)
```

For future use the column indices are assigned to variables. This is so mistakes won't be made later on accidentally selecting a wrong column and to simply make it easier to use.

```
SPL_70dB <- 1:4
SPL_94dB <- 5:8
SPL_105dB <- 9:12
```

## 2.2 Data Summary

To get to know the data set more the data will be visualized in multiple ways. Seeing how everything is grouped and divided gives us a deeper understanding of what the quality of the data set is and what types of statistics will have to be performed.

First a simple summary performing 6-number-statistic on the data columns will be done, this gives a summary overview of each replication for all groups.

```
# Disable printing 'table continues' lines between split sections of the table
panderOptions("table.continues", "")
# Pretty print the summary of the data frame
pander::pander(summary(myData), split.tables = 80)
```

| A1_70dB | A2_70dB | A3_70dB | A4_70dB |
|---|---|---|---|
| Min. : 0.0 | Min. : 0 | Min. : 0.0 | Min. : 0.0 |
| 1st Qu.: 1.0 | 1st Qu.: 0 | 1st Qu.: 0.0 | 1st Qu.: 0.0 |
| Median : 21.0 | Median : 20 | Median : 16.0 | Median : 16.0 |
| Mean : 1161.2 | Mean : 1192 | Mean : 973.6 | Mean : 877.3 |
| 3rd Qu.: 909.2 | 3rd Qu.: 915 | 3rd Qu.: 749.2 | 3rd Qu.: 690.0 |
| Max. :807021.0 | Max. :844885 | Max. :746507.0 | Max. :600313.0 |

| B1_94dB | B2_94dB | B3_94dB | B4_94dB |
|---|---|---|---|
| Min. : 0.0 | Min. : 0 | Min. : 0 | Min. : 0 |
| 1st Qu.: 0.0 | 1st Qu.: 0 | 1st Qu.: 0 | 1st Qu.: 0 |
| Median : 16.0 | Median : 17 | Median : 15 | Median : 18 |
| Mean : 941.7 | Mean : 1058 | Mean : 910 | Mean : 1091 |
| 3rd Qu.: 730.0 | 3rd Qu.: 817 | 3rd Qu.: 692 | 3rd Qu.: 813 |
| Max. :455027.0 | Max. :512010 | Max. :480944 | Max. :634849 |

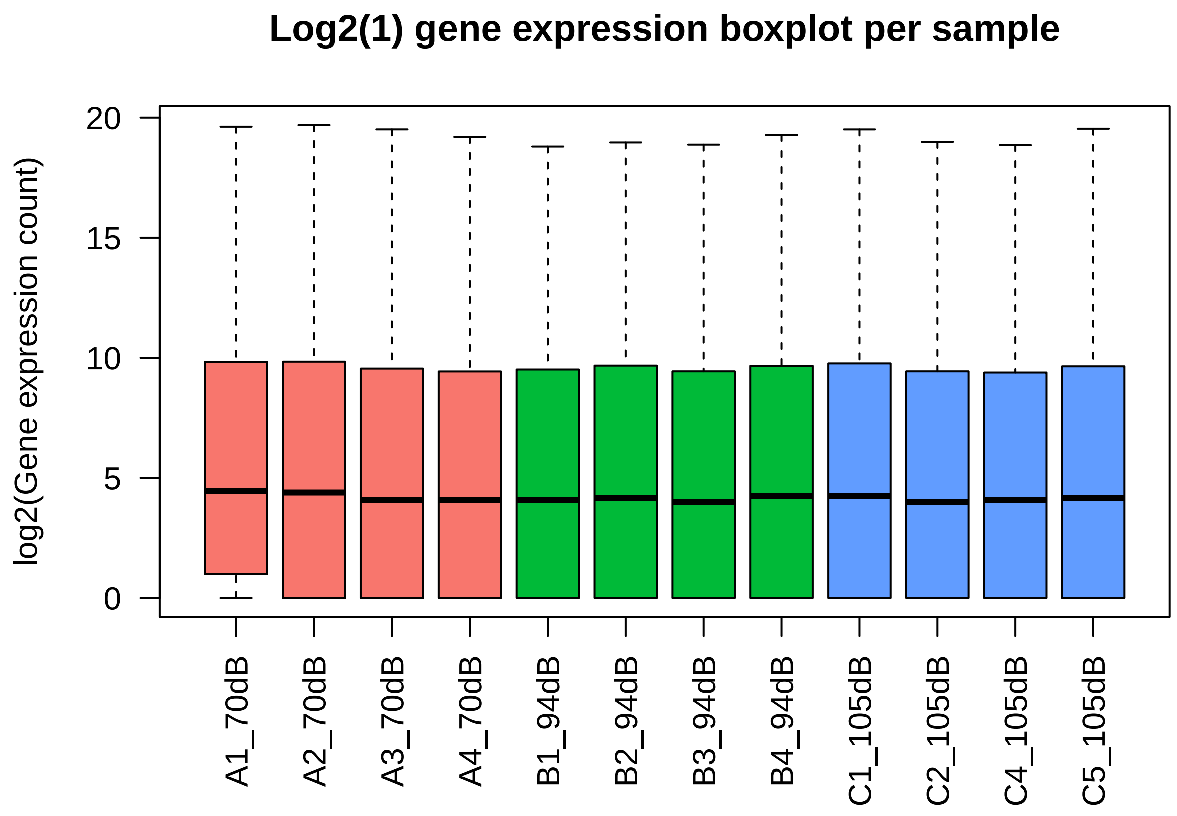
| C1_105dB | C2_105dB | C4_105dB | C5_105dB |
|---|---|---|---|
| Min. : 0 | Min. : 0.0 | Min. : 0.0 | Min. : 0 |
| 1st Qu.: 0 | 1st Qu.: 0.0 | 1st Qu.: 0.0 | 1st Qu.: 0 |
| Median : 18 | Median : 15.0 | Median : 16.0 | Median : 17 |
| Mean : 1134 | Mean : 913.6 | Mean : 903.1 | Mean : 1045 |
| 3rd Qu.: 870 | 3rd Qu.: 692.0 | 3rd Qu.: 669.0 | 3rd Qu.: 799 |
| Max. :745783 | Max. :521567.0 | Max. :474316.0 | Max. :761621 |

At first glance of the summary it can be seen that the expression at the lowest (70) dB genes seem to be expressed the most with the expression declining whilst the SPL increases. It also looks like there is a noticeable difference between each replication.

### 2.3 Boxplot

Now a boxplot will be made for each data column, visualizing this will help quickly spot irregularities after which further detailed statistics need to be performed to do any quality control. To increase the visibility a log2 transformation will be done to show changes more informative.

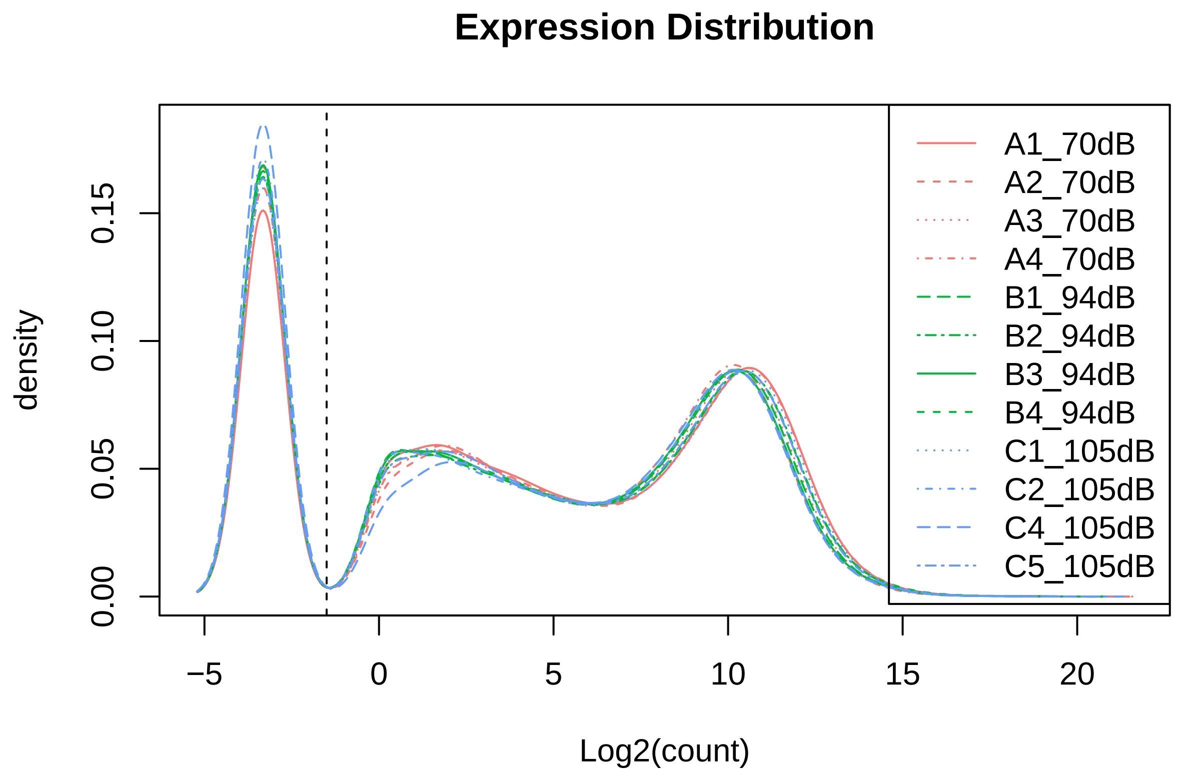
```r
# Create a color pallet variable to use in graphs
myColors = rep(hue_pal()(3), each = 4)
# Plot the boxplot with log2(1) scale.
boxplot(log2(myData + 1), col = myColors,
        outline = FALSE,las = 2, ylab = "log2(Gene expression count)",
        main="Log2(1) gene expression boxplot per sample")
```



## 2.4 Density plot

Another useful way of visualizing the data set is using a density plot, it shows a distribution of the log2-transformed count data for all samples which makes spotting problems easier.

```r
# Plot a density plot with log2(0.1) scale
# Use a different color for each group and line type for each replication
plotDensity(log2(myData + 0.1), col=myColors,
            lty=c(1:ncol(myData)), xlab="Log2(count)",
            main="Expression Distribution")

# Add a legend and a vertical line
legend("topright", names(myData), lty=c(1:ncol(myData)), col=myColors)
abline(v=-1.5, lwd=1, col="black", lty=2)
```



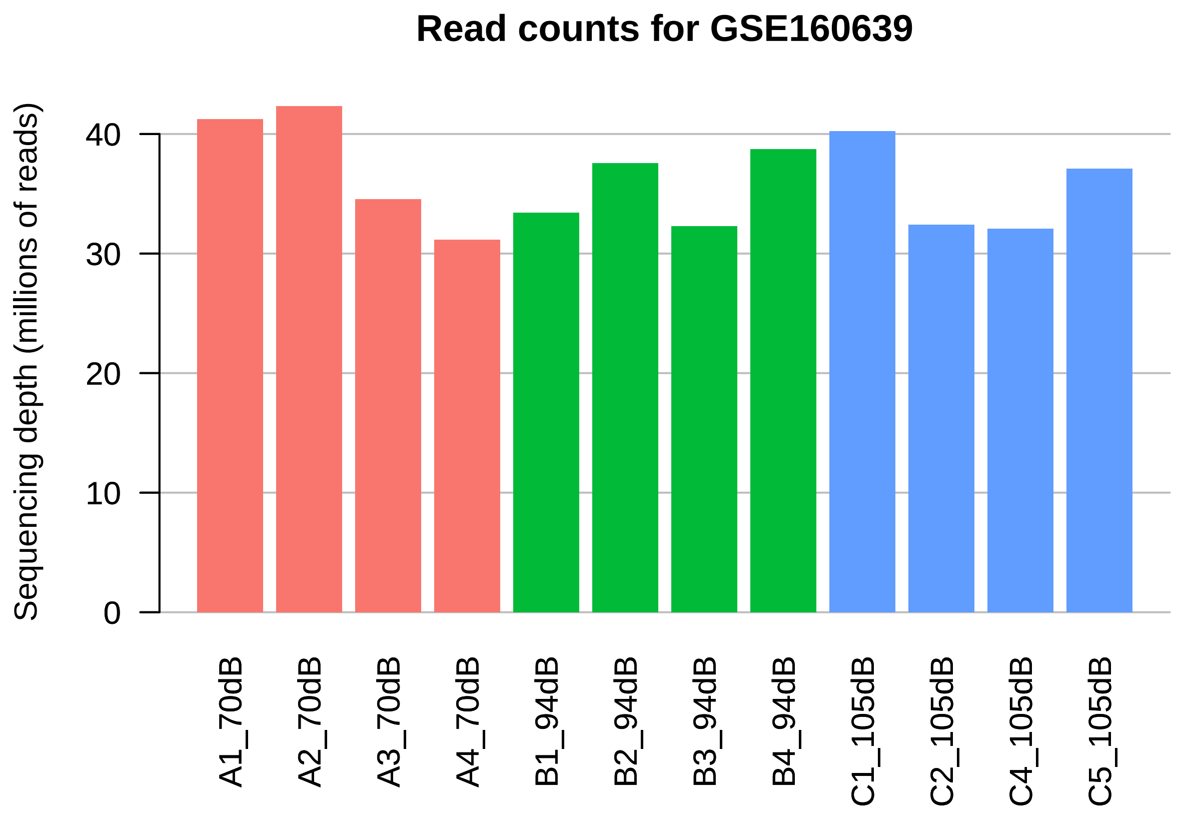
Looking at the density plot of the expression distribution it looks like all replications have a relatively similar amount of reads sequenced since all the peaks lie together.

## 2.5 Normalizing the data

### Checking read depth with barplot

Sometimes there seems to be a difference in expression between groups even if there actually isn't, if there is a big difference in count values within the samples of a group that could be due to the read depth of the samples. To see the read depth of all samples fast, the total number of mapped reads per sample can be shown in a bar plot.

```r
# Initialize plot
barplot(colSums(myData) / 1e6, las = 2, border = NA)

# Add horizontal axis
axis(2, tck = 1, lty = 1, col = "gray", labels = FALSE)

# Create the plot
par(new = TRUE)
barplot(colSums(myData) / 1e6, main = "Read counts for GSE160639",
        las = 2, col = myColors, border = NA,
        ylab = "Sequencing depth (millions of reads)")
```



Looking at the bar plot that was created it can be seen that all the samples have a good, similar, sequencing depth. Though all the read depths are high and similar the data will still need to be normalized, because there might still be unwanted variation disrupting our analysis.

**Performing the data normalization with a variance stabilizing transformation**

To normalize the data the *vsc* function of the DESeq2 library will be used. First a *DESeqDataSet*-object will need to be created since that's what is needed to use the function. There are multiple techniques to normalize data but for now only normalized data from a variance stabilizing transformation will be used.

```r
# Create DESeqDataSet object
ddsMat <- DESeqDataSetFromMatrix(countData = myData, design = ~ 1,
                                 colData = data.frame(samples = names(myData)))
# Perform normalization
rld.dds <- vst(ddsMat)
# 'Extract' normalized values
rld <- assay(rld.dds)
# Remove the 'Large DESeqTransform' object from memory
remove(rld.dds)

# Pretty print the head of the normalized data
panderOptions("table.continues", "")
pander::pander(head(rld), split.tables = 70)
```

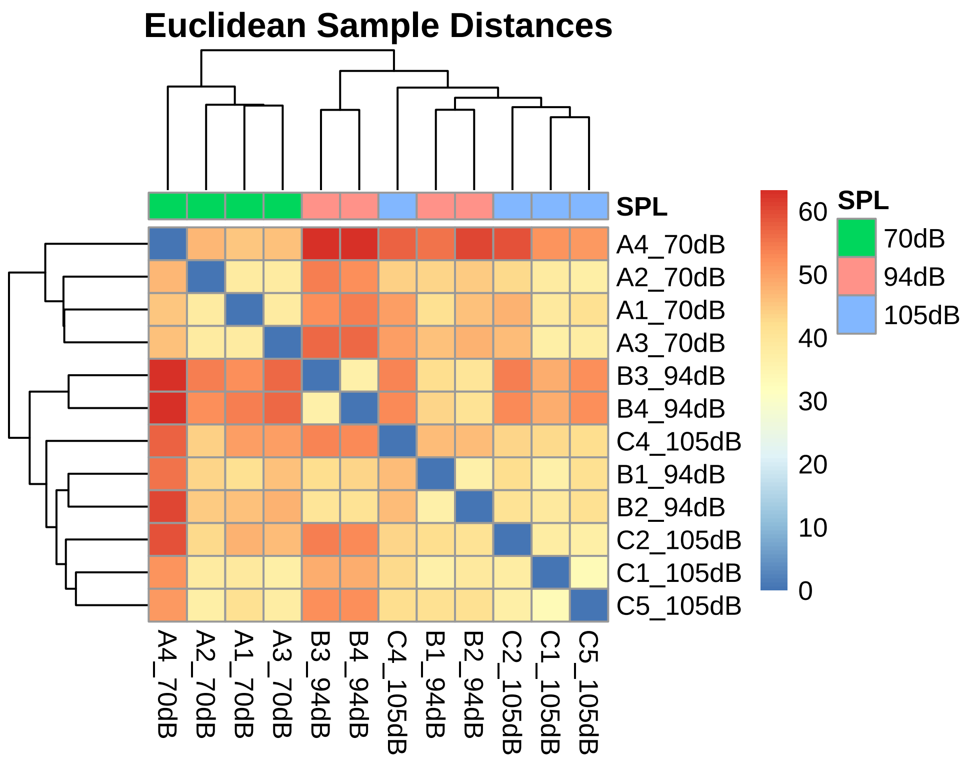
| | A1_70dB | A2_70dB | A3_70dB | A4_70dB | B1_94dB | B2_94dB | B3_94dB | B4_94dB | C1_105dB | C2_105dB | C4_105dB | C5_105dB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **ENSMUSG00000064351** | 19.36 | 19.44 | 19.54 | 19.37 | 18.87 | 18.9 | 18.49 | 18.61 | 19.34 | 19.15 | 18.96 | 19.49 |
| **ENSMUSG00000069919** | 18.22 | 18.35 | 18.07 | 18.29 | 18.28 | 18.63 | 18.83 | 18.59 | 18.41 | 18.55 | 19.05 | 18.56 |
| **ENSMUSG00000052305** | 17.63 | 18.14 | 17.83 | 18 | 18.01 | 18.13 | 19.05 | 19.22 | 18.12 | 18.66 | 18.49 | 18.27 |
| **ENSMUSG00000015090** | 18.17 | 18.06 | 17.99 | 18.01 | 18.36 | 18.41 | 18.66 | 18.49 | 18.07 | 18.1 | 18.55 | 17.93 |
| **ENSMUSG00000064370** | 18.28 | 18.22 | 18.29 | 18.18 | 17.84 | 17.88 | 17.47 | 17.49 | 18.3 | 18.18 | 18.22 | 18.2 |
| **ENSMUSG00000001506** | 18 | 17.98 | 17.87 | 17.73 | 18.04 | 18.02 | 17.81 | 17.67 | 18.21 | 17.97 | 17.78 | 17.71 |

Normalized data has now been obtained with values that are comparable, unlike the raw read counts. What looked like differences in expression in the count data could now with the normalized data look no different at all between the groups.

## 2.6 Creating a heatmap to see sample distances

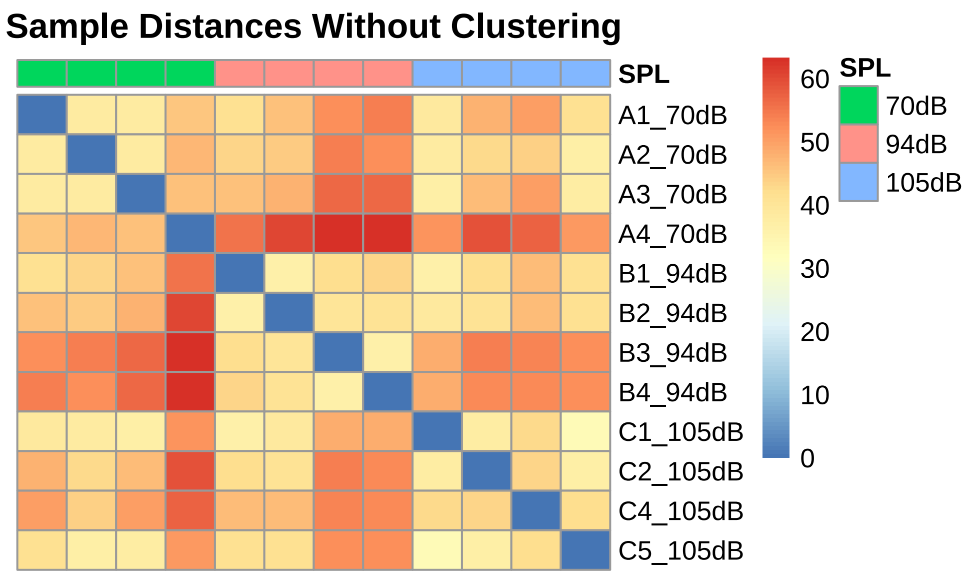
To see what the differences in expression between groups are, the data will be visualized with a heatmap. This heatmap will show the Euclidean distances between all samples. To make a heatmap distance calculations need to be performed, for each combination of samples a distance metric is calculated that will be used to check for variation within the sample groups. The normalized data will be transposed and only after that can be used to calculate the sample distances.

```
# Calculate basic distance metric (using euclidean distance)
sampledists <- dist(t(rld))

# Convert the 'dist' object into a matrix for creating a heatmap
sampleDistMatrix <- as.matrix(sampledists)

# The annotation is an extra layer that will be plotted above the heatmap columns
annotation <- data.frame(SPL = factor(rep(1:3, each = 4),
                                       labels = c("70dB", "94dB", "105dB")))

# Set the rownames of the annotation dataframe to the sample names
rownames(annotation) <- names(myData)

# Create heatmap
pheatmap(sampleDistMatrix, main = "Euclidean Sample Distances",
         annotation_col = annotation, show_colnames = TRUE,
         clustering_distance_rows = sampledists,
         clustering_distance_cols = sampledists)
```



The resulting heatmap is not as beautiful as desired and thus does not give that much insight in the data. This is probably mainly due to the fact that all the samples are from the exact same type of tissue and the only differentiating factor is the Sound Pressure Level the mice were exposed to. With the idea that it might be more beneficial to create a heatmap that doesn't change the clustering a second one was created.

```r
# Create second heatmap without changed clustering
pheatmap(sampleDistMatrix, main = "Sample Distances Without Clustering",
         annotation_col = annotation, show_colnames = FALSE,
         cluster_rows = FALSE, cluster_cols = FALSE)
```



In this second heatmap it can be better seen that there are trends between and within sample groups, almost creating borders between groups. Because there are a few samples with noticeable differentiating values from the other samples in their own group the first heatmap became less readable since there is only 1 factor that differs. One could also say that by looking at this heatmap sample A4 looks to be an outlier and might cause noise in statistics later on.

## 2.7 Using Multi-Dimensional Scaling to spot clustering

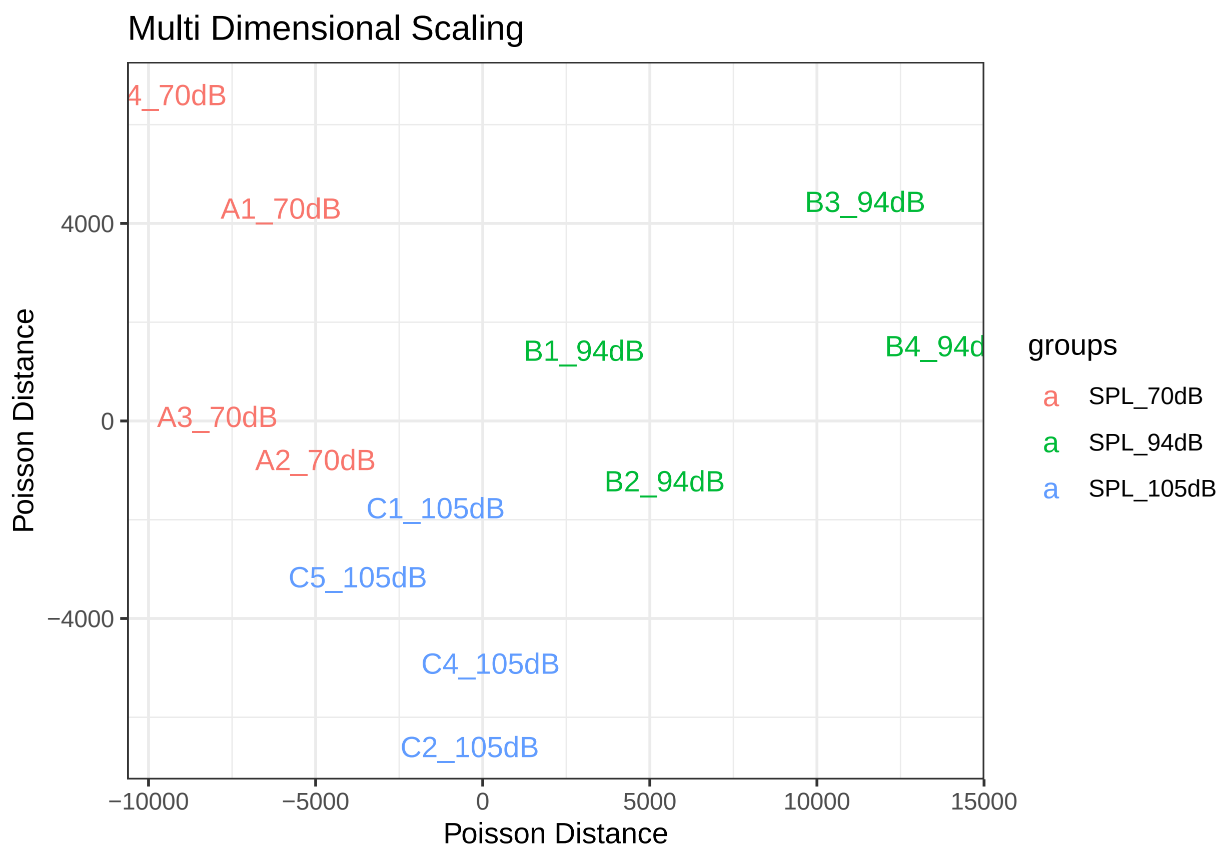
Another way to see how distant samples are from each other is by performing Multi-Dimensional Scaling (MDS), this creates a 2D plot where it is the hope that distinct clusters are formed. What's different with MDS is that a slightly different distance metric is used; Poisson instead of the previously used Euclidean with the dist() function. To get the *Poisson Distance* data the 'PoiClaClu' library is used, it uses the raw count data (keep in mind the 'ddsMat' DataSet still contains raw count data, only the 'rld' matrix is normalized).

```
# Note: uses the raw-count data, PoissonDistance performs normalization
# set by the 'type' parameter (uses DESeq)
dds <- assay(ddsMat)
poisd <- PoissonDistance(t(dds), type = "deseq")
# Extract the matrix with distances
samplePoisDistMatrix <- as.matrix(poisd$dd)

# Calculate the MDS and get the X- and Y-coordinates
mdsPoisData <- data.frame( cmdscale(samplePoisDistMatrix) )
# And set some better readable names for the columns
names(mdsPoisData) <- c('x_coord', 'y_coord')
mdsPoisData
```

```
##       x_coord     y_coord
## 1  -6039.3225  4308.88762
## 2  -5004.5860  -782.13899
## 3  -7939.7214    90.81125
## 4  -9463.8490  6607.43794
## 5   3032.6580  1431.82351
## 6   5442.7042 -1212.37390
## 7  11440.5382  4445.34593
## 8  13835.0550  1522.68025
## 9  -1409.1580 -1757.52593
## 10  -389.4577 -6594.13393
## 11   233.9949 -4905.50780
## 12 -3738.8558 -3155.30595
```

Now the MDS coordinates used for plotting the distances using Poisson Distance are calculated they'll be plotted with the ggplot library.

```r
# Separate the annotation factor (as the variable name is used as label)
groups <- factor(rep(1:3, each=4),
                 labels = c("SPL_70dB", "SPL_94dB", "SPL_105dB"))
coldata <- names(myData)

# Create the plot using ggplot
ggplot(mdsPoisData, aes(x_coord, y_coord, color = groups, label = coldata)) + theme_bw() +
       geom_text(size = 4) + ggtitle('Multi Dimensional Scaling') +
       labs(x = "Poisson Distance", y = "Poisson Distance")
```



The three different groups seem to cluster pretty nicely in the generated plot, there is a noticeable seperation between the different SPL groups' coordinates.

Based on all of the exploratory data analysis that was done it looks like all the samples are of good enough quality so that there does not need to be any further data cleaning. This marks the end of this chapter and now the 'real' analysis will start.

# 3 Discovering Differentialy Expressed Genes (DEGs)

Tekst

## 3.1 Pre-processing

Tekst