SP1 Analysis

Orfeas Gkourlias

3/10/2022

3. Exploratory Data Analysis

The data will be need to be extracted from the count file first. Since it's in .tsv format, the seperator is going to be tab based. The inclusion of headers adds an X to the sequence ID's, because R is unable to make headers out of just integers. To counteract this, the colnames will be manually added.

3.1 Loading the data

```
file <- c("..\\data\\GSE152262_RNAseq_Raw_Counts.tsv")
# Raw_Data will be the primary dataframe that gets worked on.
raw_data <- read.table(file, sep = '\t', header = TRUE, row.names = 1)

# The first two and single last columns are the case samples.
# Control samples are indicated with con.
colnames(raw_data) <- c("case24275", "case24277", "con4279", "con4280", "con4280a", "case24281")

# Rearranging the columns so that the first three are the case samples.
raw_data <- raw_data[, c(1,2,6,3,4,5)]

# Showing the first five rows as an example.
raw_data[1:5,]
```

```
##                 case24275 case24277 case24281 con4279 con4280 con4280a
## ENSG00000000003        23        30         8      11      43       31
## ENSG00000000005         0         0         0       0       0        2
## ENSG00000000419       778       910      1051     838     911     1113
## ENSG00000000457       378       438       389     441     772      738
## ENSG00000000460        44        51        28      58      61       65
```

```
# Showing the dimension and structure of the raw_data dataframe.
dim(raw_data)
```

```
## [1] 58307     6
```

```
str(raw_data)
```

```
## 'data.frame':    58307 obs. of  6 variables:
##  $ case24275: int  23 0 778 378 44 14575 30 54 213 546 ...
##  $ case24277: int  30 0 910 438 51 21109 23 89 206 589 ...
##  $ case24281: int  8 0 1051 389 28 27759 68 50 180 561 ...
##  $ con4279  : int  11 0 838 441 58 7164 94 105 333 452 ...
##  $ con4280  : int  43 0 911 772 61 11710 151 77 419 407 ...
##  $ con4280a : int  31 2 1113 738 65 11846 148 69 384 373 ...
```

The data is now loaded in as a data frame. Every row shows the raw counts of a specific gene being expressed. 4275, 4277 and 4281 are the variant types. The datatypes are correct in this case_log2. There should only be integers included, except for the gene names.

Now that the data has been properly loaded, objects can be made to differentiate the control_log2 and case_log2 counts. Before separating the groups, it'll be useful to apply a log2 function to our data. This makes it so that the data is more informative and tidier, because of outliers and the big range being worked with.

```r
# Transforming the read data of every columns to the log2 value
# 1 is added to every column to make sure there are no log2(0) values.
raw_data_log2 <- log2(raw_data + 1)

# Dividing the case and controls columns into seperate dataframes for later use.
case <- raw_data[,c(1:3)]
control <- raw_data[,c(3:6)]

# Applying the same division, but with the log values for plotting purposes.
case_log2 <- raw_data_log2[,c(1:3)]
control_log2 <- raw_data_log2[,c(4:6)]

# Displaying the first rows of divided dataframes.
case_log2[1,]
```

```
##                 case24275 case24277 case24281
## ENSG00000000003  4.584963  4.954196  3.169925
```

```r
control_log2[1,]
```

```
##                  con4279  con4280 con4280a
## ENSG00000000003 3.584963 5.459432        5
```

The control_log2 and case_log2 data is now stored in different variables, as shown above.

Visualizing using boxplot and density plot

More insight on the data can be gained by plotting and summarizing it. Every column will first be summarized. Following that, the mean values will be compared in a boxplot.

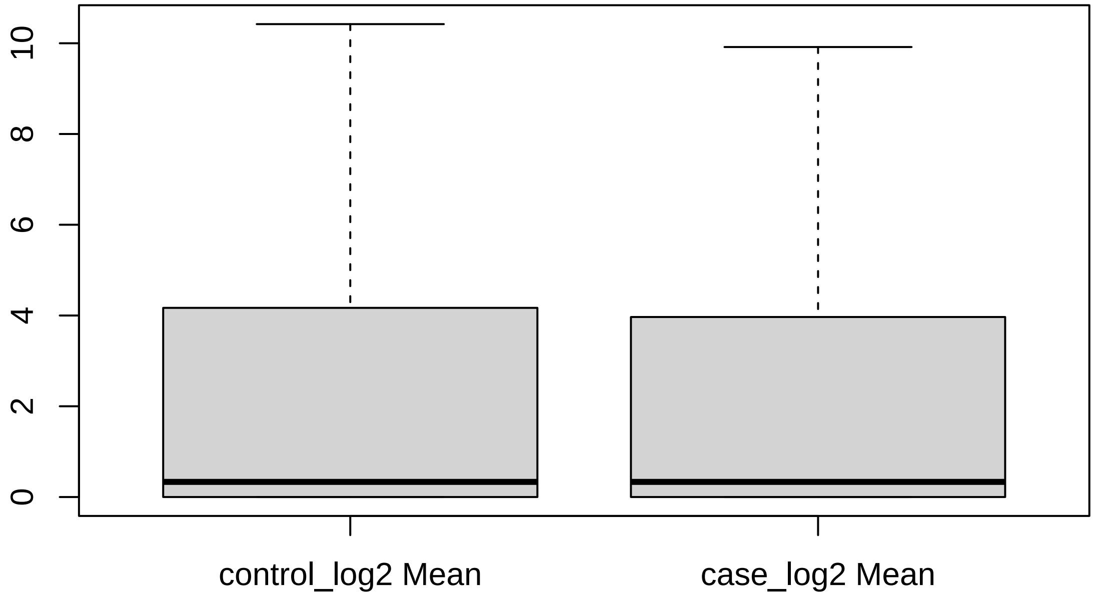
```r
# Applying a summary on all the log2 data.
summary(raw_data_log2)
```

```
##    case24275        case24277        case24281         con4279
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 0.000   Median : 0.000   Median : 0.000   Median : 0.000
##  Mean   : 2.424   Mean   : 2.552   Mean   : 2.405   Mean   : 2.524
##  3rd Qu.: 3.907   3rd Qu.: 4.248   3rd Qu.: 3.907   3rd Qu.: 4.170
##  Max.   :23.704   Max.   :23.582   Max.   :23.642   Max.   :23.549
##     con4280          con4280a
##  Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 0.000   Median : 0.000
##  Mean   : 2.579   Mean   : 2.571
##  3rd Qu.: 4.248   3rd Qu.: 4.170
##  Max.   :23.675   Max.   :24.155
```

```r
# Getting the mean values of both controlled and case sample expression values
# For every gene. This might be useful later.
case_log2$mean = apply(X = case_log2[1:3], MARGIN = 1, FUN = mean)
control_log2$mean = apply(X = control_log2[1:3], MARGIN = 1, FUN = mean)

# Doing the same to the raw data frames.
case$mean = apply(X = case[1:3], MARGIN = 1, FUN = mean)
control$mean = apply(X = control[1:3], MARGIN = 1, FUN = mean)

# Plotting the log2 data.
boxplot(control_log2$mean, case_log2$m, outline = FALSE, names = c("control_log2 Mean", "case_log2 Mean
```



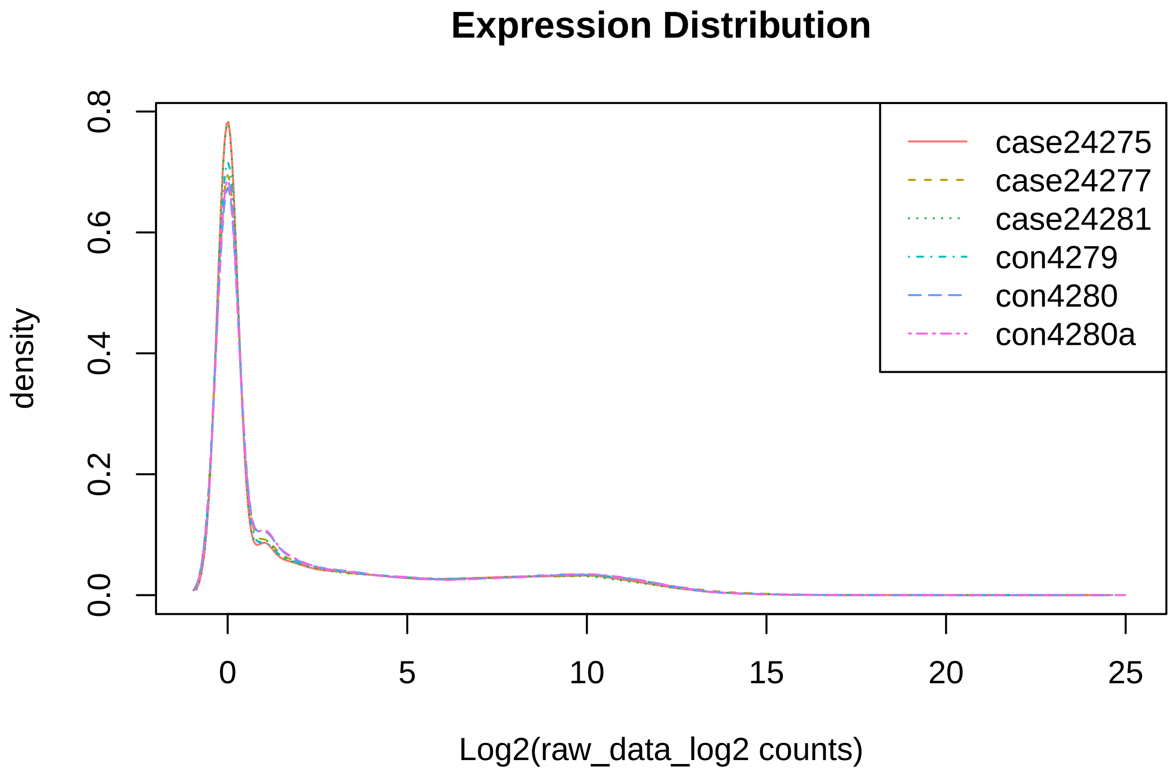
These boxplots are not yet very informative. The only thing that can be seen from them is that the case_log2s have a slightly lower expression level on average

Maybe a density plot allows for a more informative figure.

```r
# Creating the recurring colors that will be used for the columns
myColors <- hue_pal()(6)

# Density plotting the log2 data, using the colors created above.
plotDensity(raw_data_log2, col=rep(myColors, each=1), lty=c(1:ncol(raw_data_log2)),
            main = "Expression Distribution", xlab = "Log2(raw_data_log2 counts)")

# Adding a legend for clarity.
legend('topright', names(raw_data_log2), lty=c(1:ncol(raw_data_log2)),
       col=rep(myColors, each=1))
```



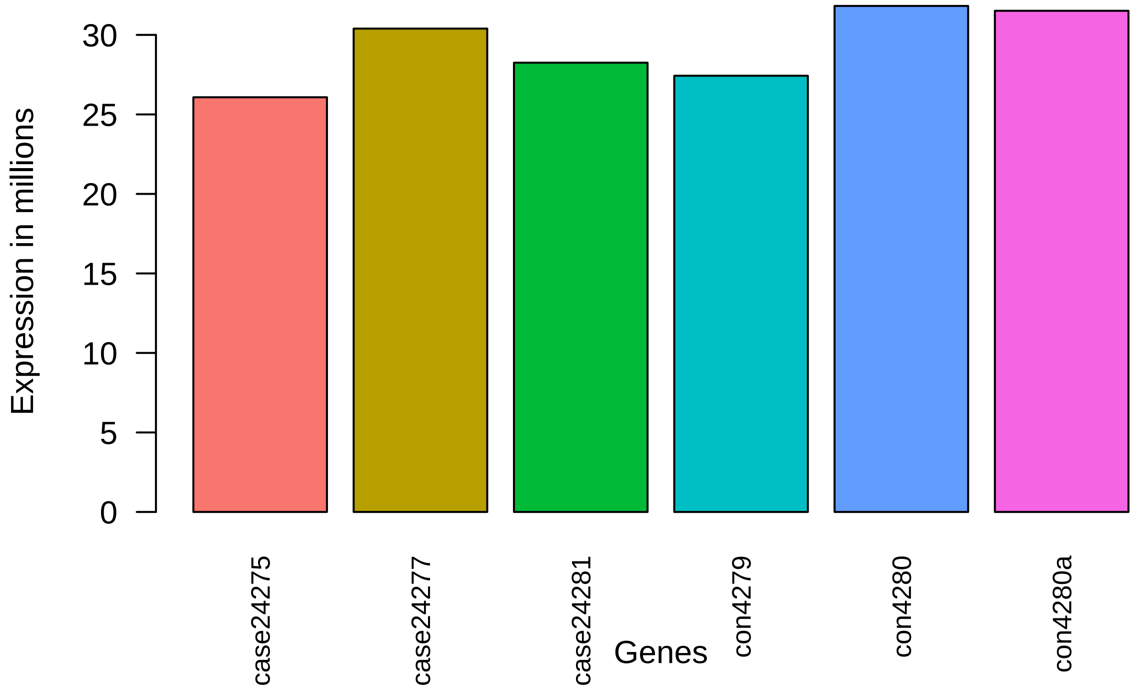
As can be seen in the plot, the highest amount of expressions, besides 0, seem to be around 10.

3.4 Visualizing using heatmap and MDS

Before continuing with this step, the data will have to be normalized. There are 5 which rows are not actual genes. They will be removed. After that, a barplot will be generated to show whether theres a difference in expression in millions, using col sums.

```
# There are 5 rows which do not count for actual genes.
# These rows are currently not relevant, but shouldn't be present from now on.
remove_rows <- c("__not_aligned", "__no_feature", "__no_feature",
                 "__alignment_not_unique", "__too_low_aQual", "__ambiguous")
raw_data <- raw_data[!(row.names(raw_data) %in% remove_rows),]

# Bar plotting the new data. Division by 1e6 shows the values in millions.
barplot(colSums(raw_data) / 1e6, las = 2, cex.names = 0.8, col = myColors, xlab = "Genes", ylab = "Expr
```



Judging by that figure, the control group seems to have a higher average expression when summarised on all genes.

Now the DESeq2 library will be used to normalize the data. the VST function within this package is the next function. The data will first have to make a Summarized Experiment object, which is done first.

```
# Creating the dds Matrix, so that it ca nbe used in the vst function
(ddsMat <- DESeqDataSetFromMatrix(countData = raw_data,
                                  colData = data.frame(samples=names(raw_data)),
                                  design = ~ 1))
```

```
## class: DESeqDataSet
## dim: 58302 6
## metadata(1): version
## assays(1): counts
## rownames(58302): ENSG00000000003 ENSG00000000005 ... ENSG00000284747
##   ENSG00000284748
## rowData names(0):
## colnames(6): case24275 case24277 ... con4280 con4280a
## colData names(1): samples
```

```
# Applying vst and saving it into the rld.dds object.
rld.dds <- vst(ddsMat)

# Applying assay on that object then saving it into rld.
rld <- assay(rld.dds)
```

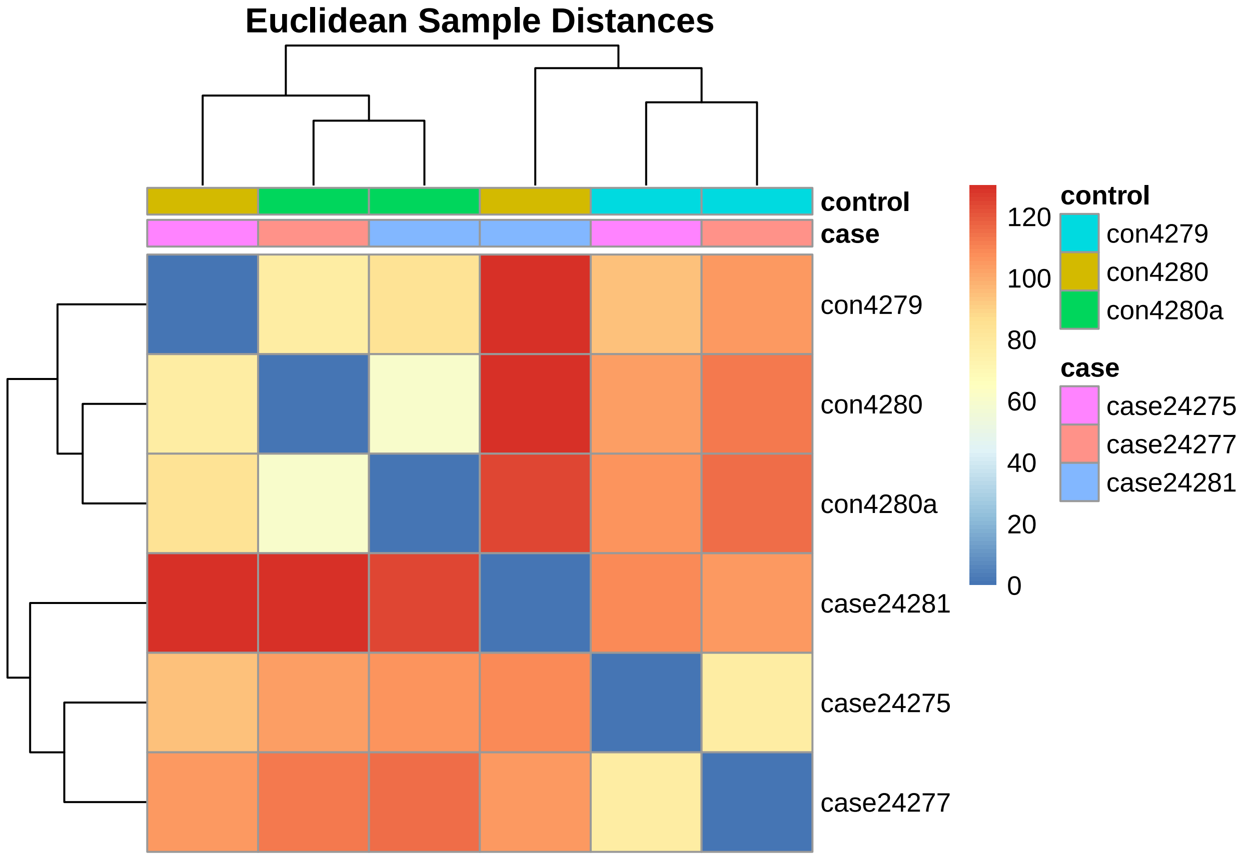
Distance calculation may now be performed on the normalized data. The matrix will first have to be transposed. After distance calculations have been performed, a heatmap may be constructed.

```
# To create the heatmap, distances first get calculated & stored in a matrix
sampledists <- dist( t( rld ))
sampleDistMatrix <- as.matrix(sampledists)

# Annotation dataframe gets created for heatmap.
annotation <- data.frame(case = factor(rep(1:3, each = 1),
                                       labels = c("case24275", "case24277", "case24281")),
                         control = factor(rep(rep(1:3, each = 2), 1),
                                          labels = c("con4279", "con4280", "con4280a")))

# Rownames for the annotation get taken from raw_data.
rownames(annotation) <- names(raw_data)

# Heatmap function gets called on the matrix and annotation objects.
pheatmap(sampleDistMatrix, show_colnames = FALSE,
         annotation_col = annotation,
         clustering_distance_rows = sampledists,
         clustering_distance_cols = sampledists,
         main = "Euclidean Sample Distances")
```



The resulting heatmap shows where the large differences in expression are located.

The distances can also be shown using a 2d-plot, by performing multi dimensional scaling.

```
# Creating the objects required by ggplot for mds.
dds <- assay(ddsMat)
poisd <- PoissonDistance( t(dds), type = "deseq")
```

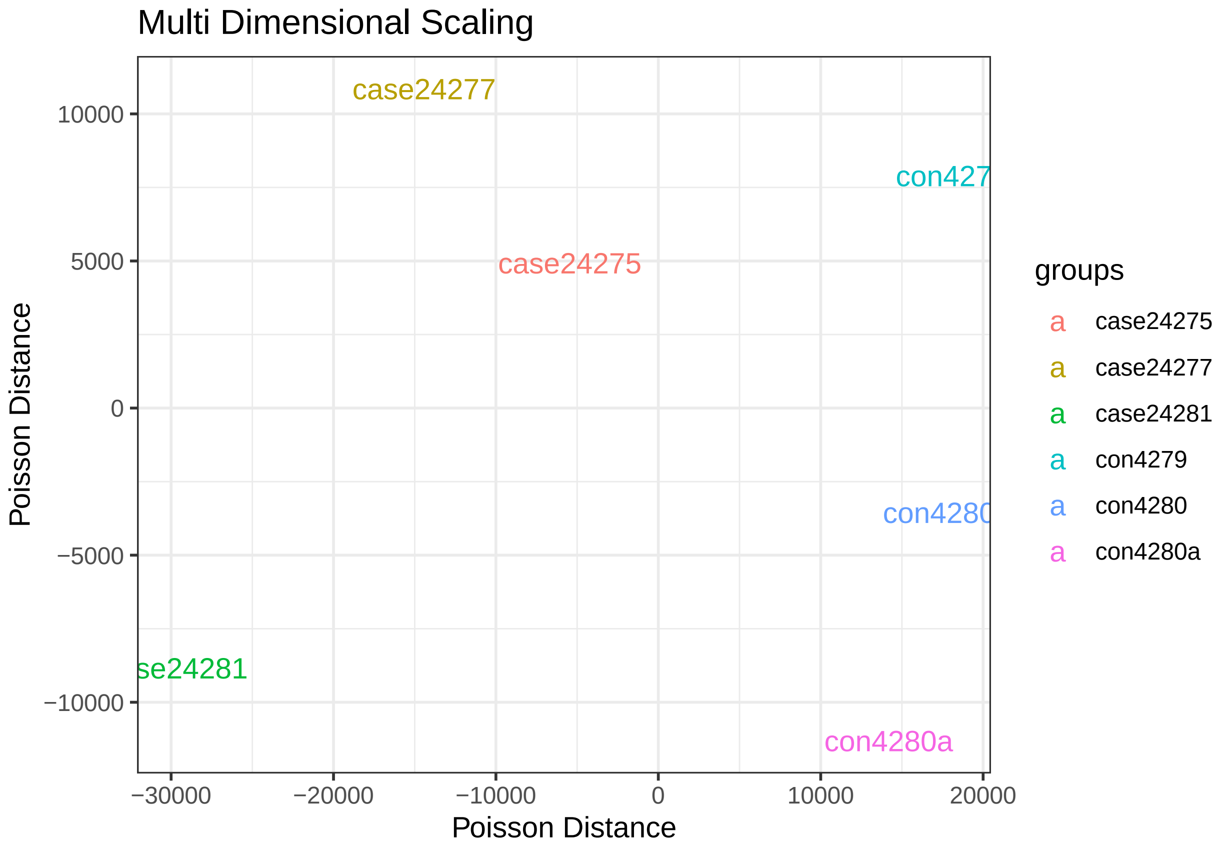
```
samplePoisDistMatrix <- as.matrix(poisd$dd)
mdsPoisData <- data.frame( cmdscale(samplePoisDistMatrix) )

# Creating names for the coords.
names(mdsPoisData) <- c('x_coord', 'y_coord')

# Getting the columns as factors.
groups <- factor(rep(1:6, each=1),
                 labels = names(raw_data))

# Column name extraction.
coldata <- names(raw_data)

# Plotting the distance data in a 2d plot with ggplot.
ggplot(mdsPoisData, aes(x_coord, y_coord, color = groups, label = coldata)) +
  geom_text(size = 4) +
  ggtitle('Multi Dimensional Scaling') +
  labs(x = "Poisson Distance", y = "Poisson Distance") +
  theme_bw()
```



# 3.5 Cleaning Data After examination of the case and control groups, there shouldn't be any samples removed. This would also not be possible, because at least 3 samples are required per group.

4 Discovering Differentialy Expressed Genes (DEGs)

Proceeding all the insight gained from plotting the data, it may now all be analysed in R. The purpose being is the discovery of DEGs, differentialy expressed genes. The earlier plots showed that there will most likely be plenty of those. Before performing the analysis steps, the data will need to go through a pre-processing phase.

4.1 Pre-processing