Gene expression relating to macrophages for trauma repair

Project for statistical assessment of a gene expression file with R studio

L T Stein

2/14/2023

Contents

Summary

Summary about this project

Introduction

Introduction about this project

1. Loading data into R

The gene expression file in .tsv format is loaded with read.table(), since the data includes tab separated data *sep* = '\t' is specified. Also a header is included so that must be notified with *header* = *TRUE* . Finally three rows are printed using pander() and head().

```
count_data <- read.table(paste0("/homes/ltstein/Kwartaal_7/Thema_opdracht/GSE215801_exp_counts.tsv/",
                         "GSE215801_exp_counts.tsv"), sep = '\t', header = TRUE)
colnames(count_data) <- c("Untreated_1", "IL4-treated_1", "PGE2-treated_1",
                          "IFNg-treated_1", "Untreated_2", "IL4-treated_2",
                          "PGE2-treated_2", "IFNg-treated_2", "Untreated_3",
                          "IL4-treated_3", "PGE2-treated_3", "IFNg-treated_3")
```

Table 1: Table continues below

	Untreated_1	IL4-treated_1	PGE2-treated_1
ENSG00000223972.5	34	40	53

	IFNg-treated_1	Untreated_2
ENSG00000223972.5	97	1

Table 1.1: Expression data of gene counts from .tsv file

In the table above the expression data is displayed of the .tsv file *"GSE215801_exp_counts.tsv"* .
The x-axis is divided into twelve samples and three batches, test groups. Additionally on the y-axis all the gene names are displayed.

In the code block below dim() prints the dimensions of the dataset and str() the structure.

```
structure <- str(count_data)
```

```
## 'data.frame':    24365 obs. of  12 variables:
##  $ Untreated_1   : int  34 320 10 11 0 5 7 57 37 88 ...
##  $ IL4-treated_1 : int  40 788 16 17 7 1 12 59 65 119 ...
##  $ PGE2-treated_1: int  53 733 31 36 1 8 14 71 69 104 ...
##  $ IFNg-treated_1: int  97 1177 31 38 5 12 23 74 99 1051 ...
##  $ Untreated_2   : int  1 205 3 0 0 3 6 36 5 97 ...
##  $ IL4-treated_2 : int  4 506 1 0 0 4 0 76 6 95 ...
##  $ PGE2-treated_2: int  3 543 1 3 1 11 7 72 11 80 ...
##  $ IFNg-treated_2: int  2 328 0 0 0 0 3 32 16 139 ...
##  $ Untreated_3   : int  4 448 9 0 1 4 0 60 16 92 ...
##  $ IL4-treated_3 : int  36 2933 39 0 45 19 114 241 107 241 ...
##  $ PGE2-treated_3: int  42 496 15 38 1 1 11 59 20 107 ...
##  $ IFNg-treated_3: int  3 362 0 0 0 3 0 22 16 136 ...
```

```
cat(structure)
```

```
dimension <- dim(count_data)
cat("dimensions =", dimension)
```

```
## dimensions = 24365 12
```

In the first code block above the structure and datatypes are displayed to confirm the data is correct . Also the second code block the dimensions of the x-as and y-axis are displayed from the dataset.

Case and control is assigned using indexing. This assigning makes a clearer difference between the experimental group and test group

```
case1 <- c(2,6,10)
case2 <- c(3,7,11)
case3 <- c(4,8,12)
control <- c(1,5,9)
```

In the code above case is assigned to the tested groups and control is assigned to the control group

2.1 Boxplot

Two boxplots are made of the data using the 'boxplot()' function. Since the raw data of the first boxplot shows a range which is too wide, it doesn't offer a clear view of the data. In the second boxplot a log-transformation is performed on the data using the 'log2()' function, that narrows down the range.

The replication groups are differentiated using the 'rep()' function. A vector with three colors is passed to rep, and rep is told that per four items the colors switch.

```
boxplot(count_data, las = 2, par(mar=c(8,8,4,2)), main = "Gene expression of twelve samples",
        xlab = "", ylab = "")
mtext("Samples", line = 0.7)
title(ylab = "Gene counts (n)", line = 5)
```

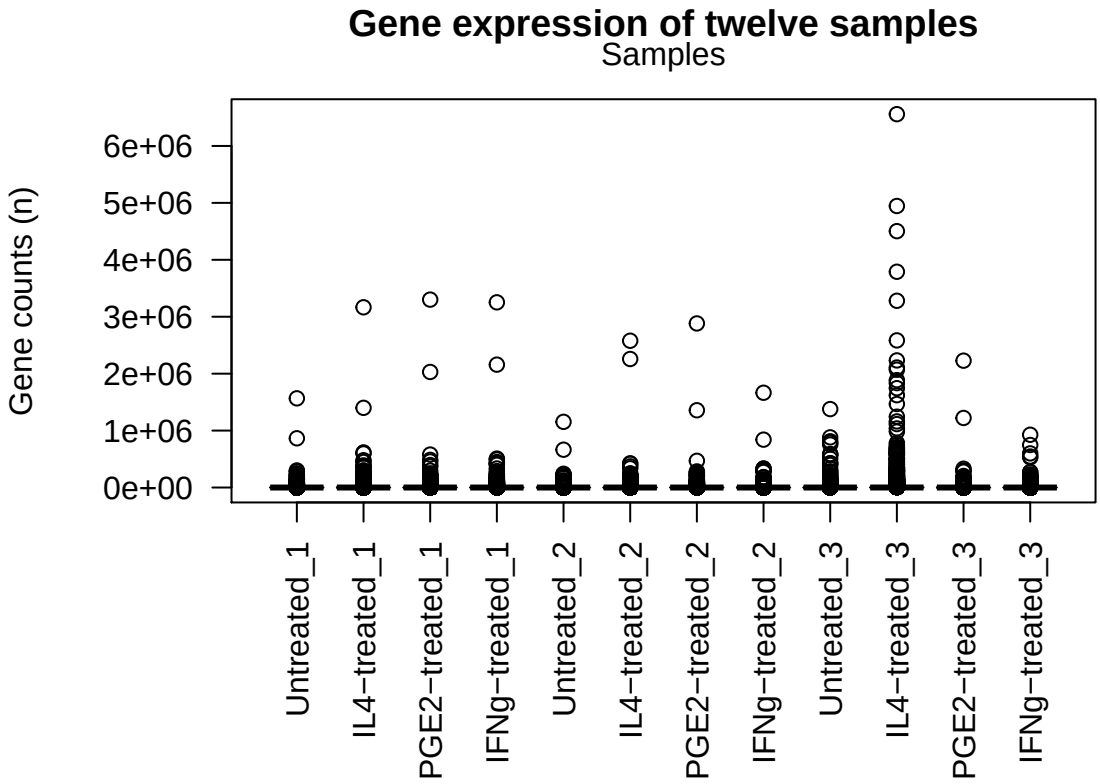


Figure 1: Boxplot of gene expression of twelve samples

In the figure above a boxplot is performed on the raw data. On the x-axis lie all the samples and on the y-axis are the gene counts.

```
color_place_holder <- rep(c("orange", "cyan", "purple"), each = 4)


boxplot(log2(count_data + 1), las = 2, par(mar=c(8,8,4,2)),
        main = "Gene expression of twelve samples that are log-transformed",
        xlab = "", ylab = "", col= color_place_holder)
mtext("Samples", line = 0.7)
title(ylab = "Gene counts (n)", line = 5)
```

In the figure above a boxplot is performed on the raw data, using a log-transformation. On the x-axis lie all the samples and on the y-axis are the gene counts.

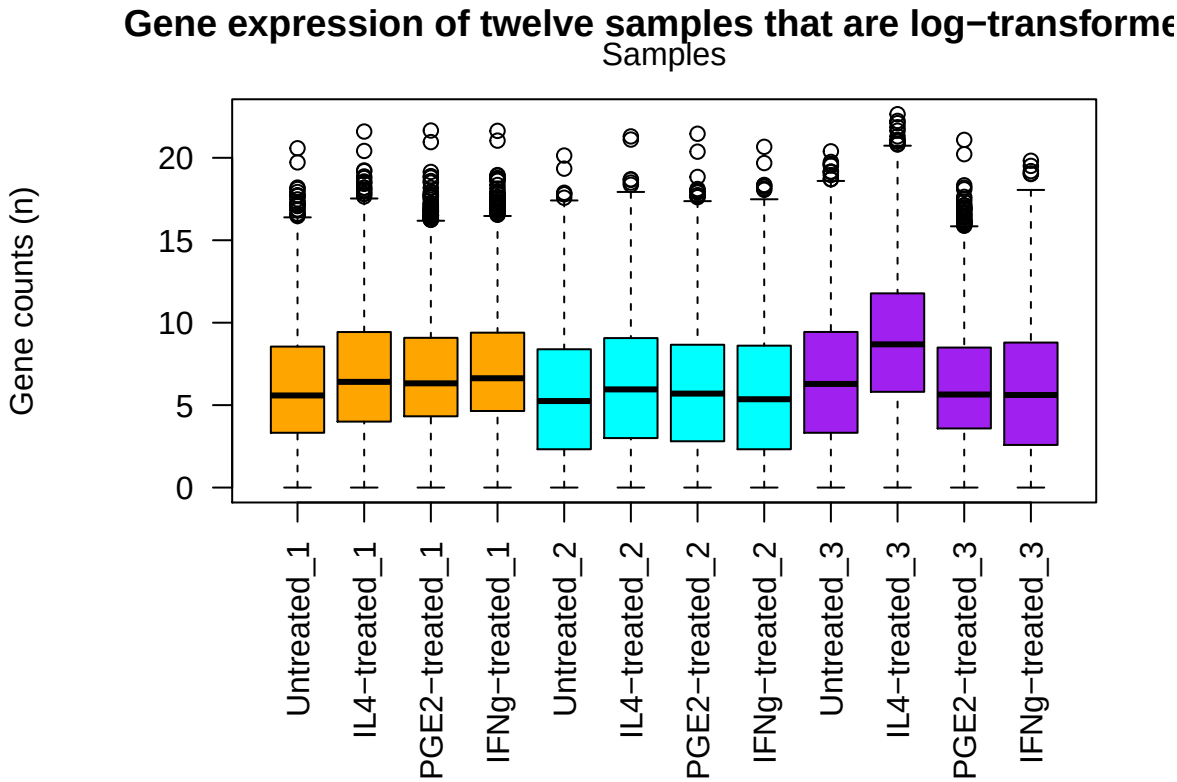


Figure 2: Boxplot of log-transformed gene expression of twelve samples

2.2 Density plot

The figure below is plotted using the 'plotDensity()' function from the ´affy' library. The legend is made with the 'legend()' function for all columns and the colors are specified using the same method as used in the boxplots above.

Finally a vertical line is added using 'abline()' to indicate that the values from the peak on its left are all inactive genes.

```
plotDensity(log2(count_data + 0.1), col = color_place_holder,
            main = "Density plot of gene expression data using log-transformation",
            xlab = "log2(count_data)")
legend('topright', names(count_data), lty=c(1:ncol(count_data)),
       col= color_place_holder, cex = 0.95)
abline(v=-1.5, lwd=1, col='red', lty=2)
```

In the figure above a density plot is made from the expression data. On the right side of the red line are all active genes and on its left side are therefore all inactive genes.

The replication group are divided into the same 3 colors as in the boxplots, and each sample received an individual line.

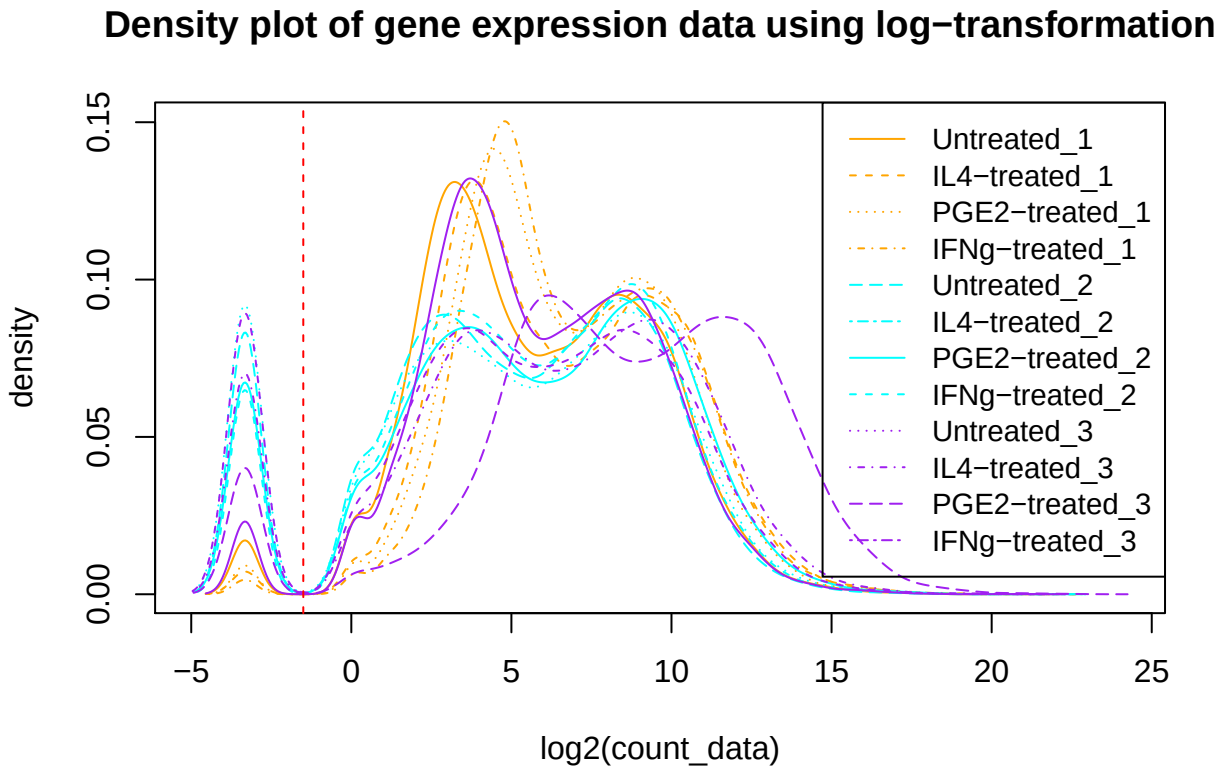
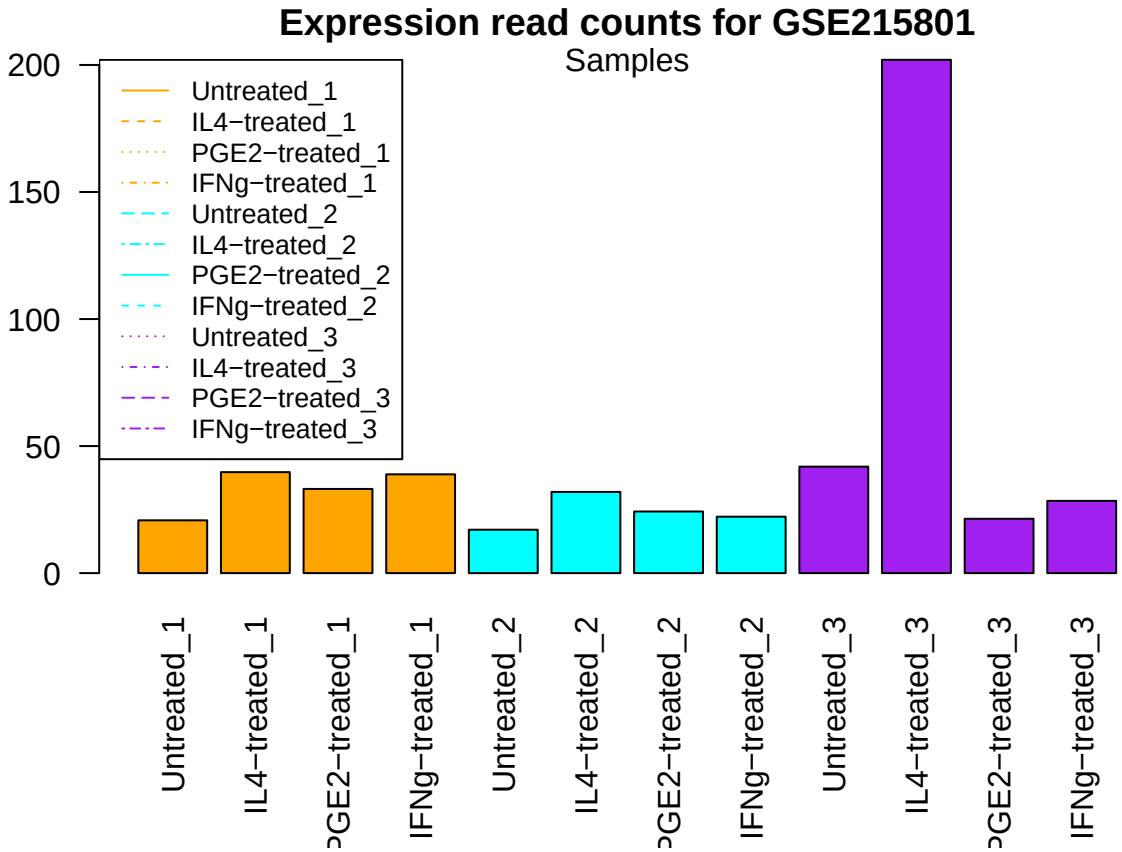


Figure 3: Density plot of gene expression data using log-transformation

3.1 Normalization

```
par(mar=c(7,3,2,2))
barplot((colSums(count_data) / 1e6),
        ylab = "Sequencing depth (times million)",
        xlab  = "",
        main = "Expression read counts for GSE215801",
        col = color_place_holder,
        las = 2)
mtext("Samples", line = -0.5)
legend('topleft', names(count_data), lty=c(1:ncol(count_data)),
       col= color_place_holder, cex = 0.80)
```



In the figure above a barplot is made of the sequencing depth for each sample. On the x-axis are all samples with divided into three replication groups as shown with the colors orange

In the code block below a DESeqDataSet is made with the DESeqDataSetFromMatrix() function from the 'DESeq2' library

```
(ddsMat <- DESeqDataSetFromMatrix(countData = count_data,
                                  colData = data.frame(samples = names(count_data)),
                                  design = ~ 1))
```

```
## class: DESeqDataSet
## dim: 24365 12
## metadata(1): version
## assays(1): counts
## rownames(24365): ENSG00000223972.5 ENSG00000227232.5 ...
##   ENSG00000278817.1 ENSG00000277196.4
## rowData names(0):
## colnames(12): Untreated_1 IL4-treated_1 ... PGE2-treated_3
##   IFNg-treated_3
## colData names(1): samples
```

The data is normalized using the *variance stabilizing transformation* method. In R this method takes shape in the vst() function with the *DESeqDataSetFromMatrix* as parameter. Later all the normalized values are extracted using the assay() function.

Table 3: Table continues below

	Untreated_1	IL4-treated_1	PGE2-treated_1
ENSG00000223972.5	6.436	6.087	6.478

Table 4: Table continues below

	IFNg-treated_1	Untreated_2	IL4-treated_2
ENSG00000223972.5	6.85	4.837	5.078

Table 5: Table continues below

	PGE2-treated_2	IFNg-treated_2	Untreated_3
ENSG00000223972.5	5.067	4.971	4.975

	IL4-treated_3	PGE2-treated_3	IFNg-treated_3
ENSG00000223972.5	5.149	6.676	5.023

Table 2.1: Normalized data of gene counts from .tsv file

In table 2 normalized data is presented for each sample for one gene.

3.2 Distance calculation

Distances between samples are calculated using the *dist()* function. The *t (transpose)* function within flips the data to a matrix format. After calculating all distances between samples that data format is converted to an actual matrix object using function *as.matrix()* . Finally the first row is tabelized.

```
# Calculating distances between samples
distance_data <- dist( t ( normalized.data  ))

# convert distance matrix data to an actual matrix object
DistMatrix <- as.matrix(distance_data)

# Tabelize group data
pander(DistMatrix[1:3, 1:4])
```

Table 7: Table continues below

	Untreated_1	IL4-treated_1	PGE2-treated_1
Untreated_1	0	72.51	89.01
IL4-treated_1	72.51	0	103.2
PGE2-treated_1	89.01	103.2	0

	IFNg-treated_1
Untreated_1	93.91
IL4-treated_1	96.5
PGE2-treated_1	99.87

Table 2.2: Normalized distance data between samples from .tsv file

In table 2.2 is a matrix shown for distances between all samples for one gene.

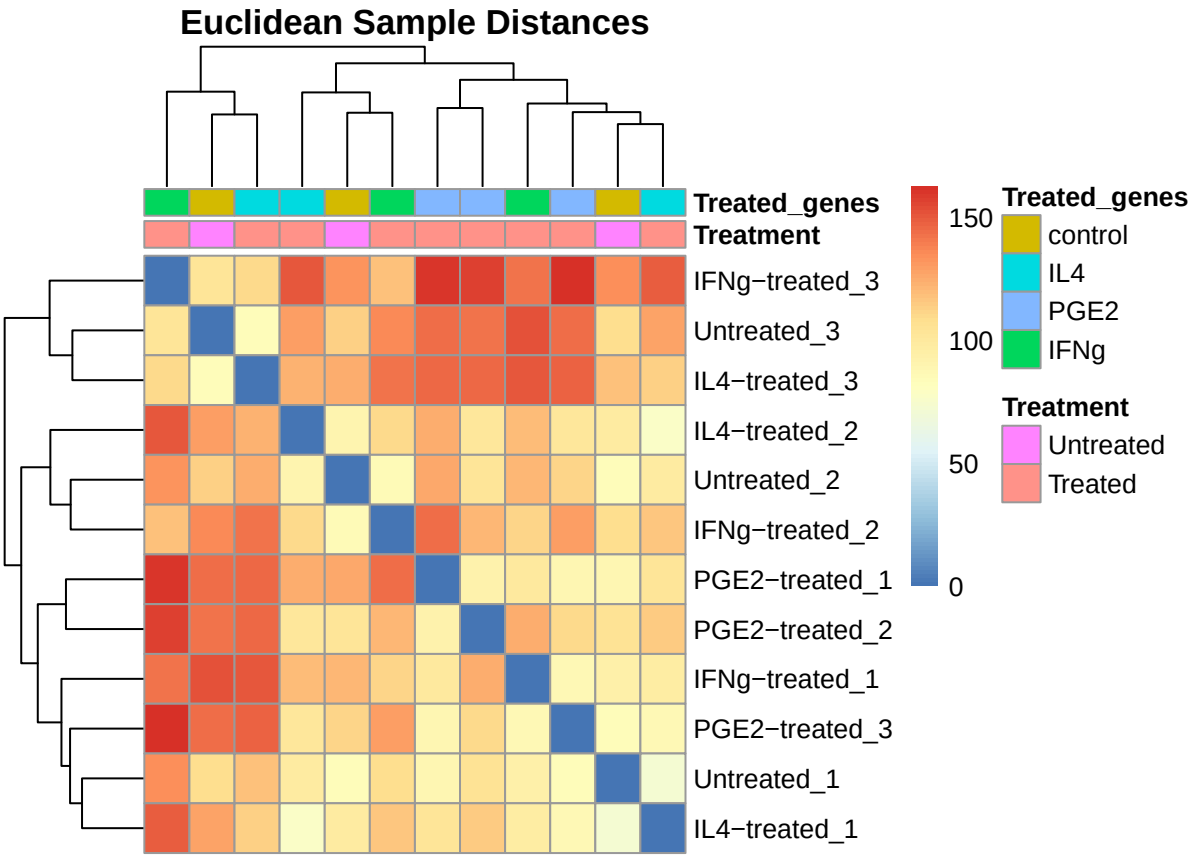
3.3 Sample distances using a Heatmap

```
# Get colnames for untreated an treated
Untreated <- c(1,5,9)
Treated <- c(2:4, 6:8, 10:12)
Treatment <- c(1:12)

Treatment[Untreated] <- 0
Treatment[Treated] <- 1
Treatment <- factor(Treatment, labels = c("Untreated", "Treated"))
Treated_genes <- factor(rep(c(0,1,2,3), 3),
                        labels = c("control", "IL4", "PGE2", "IFNg"))

annotation <- data.frame(Treatment, Treated_genes)
annotation
```

```
##    Treatment Treated_genes
## 1  Untreated       control
## 2    Treated           IL4
## 3    Treated          PGE2
```

```
## 4    Treated          IFNg
## 5  Untreated       control
## 6    Treated           IL4
## 7    Treated          PGE2
## 8    Treated          IFNg
## 9  Untreated       control
## 10   Treated           IL4
## 11   Treated          PGE2
## 12   Treated          IFNg
```

```
rownames(annotation) <- names(count_data)

pheatmap(DistMatrix, show_colnames = FALSE,
         annotation_col = annotation,
         clustering_distance_rows = distance_data,
         clustering_distance_cols = distance_data,
         main = "Euclidean Sample Distances")
```



IFNg-treated 3, IL4-treated 1

3.4 Multi-Dimensional Scaling

In the code block below several steps are performed for normalizing the raw count data and calculating preferable distances between the samples called the Poisson Distance.

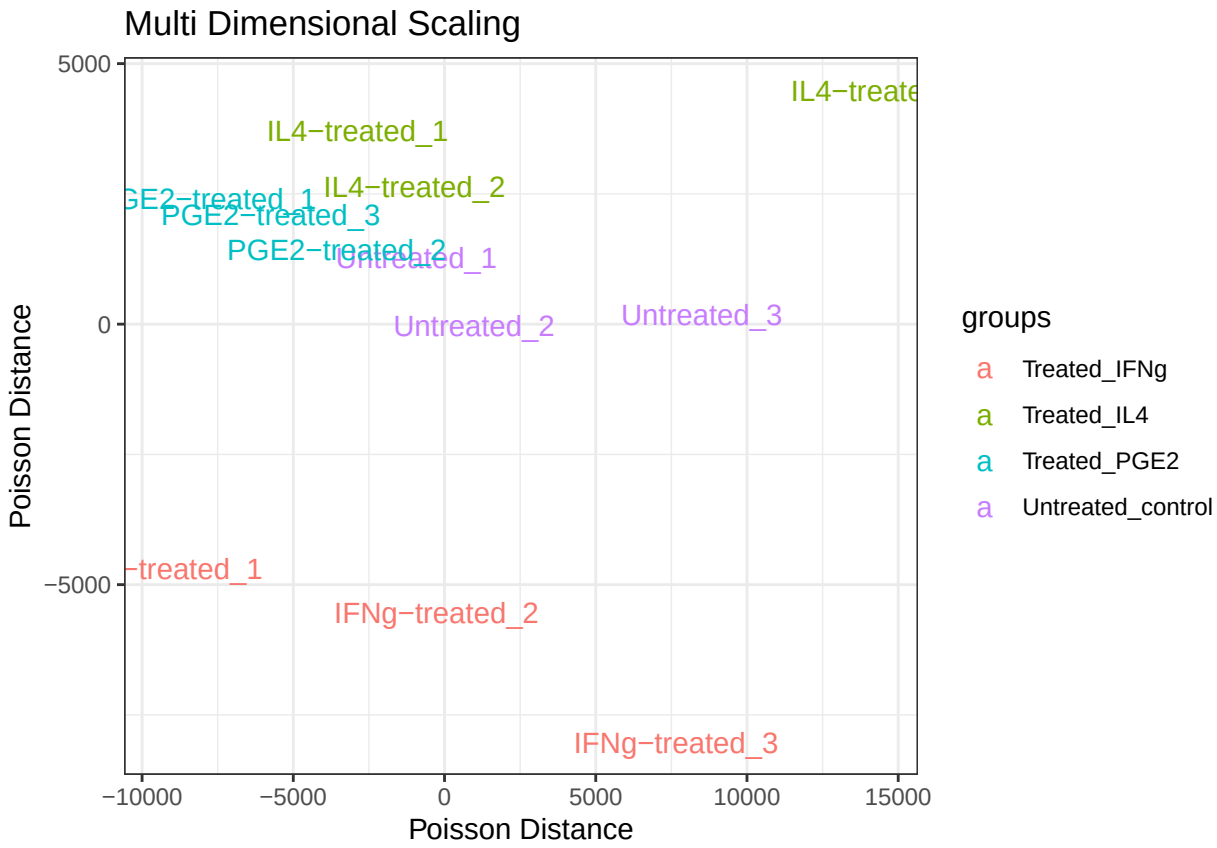
```
dds <- assay(ddsMat)
poisd <- PoissonDistance( t(dds), type = "deseq")
normalized.samplePoisDistMatrix <- as.matrix(poisd$dd)
mdsPoisData <- data.frame( cmdscale(normalized.samplePoisDistMatrix) )
names(mdsPoisData) <- c('x_coord', 'y_coord')
pander(mdsPoisData)
```

x_coord	y_coord
-924.3	1272
-2870	3710
-7857	2411
-9394	-4689
977	-16.49
-993.7	2650
-3556	1439
-260.4	-5532
8507	180.9
14457	4494
-5742	2102
7656	-8020

In the figure below a graph is drawn showing the Poisson Distance between the samples. First a factor is made *factor()* for the three replication groups assigning unique names to show in the legend. Furthermore the names of the samples *names()* are assigned to coldata, as the names for the samples will show up instead of dots.

The plotting is done with *ggplot()* . The function offers a prettier and clearer graph and is constructed using more parameters. Such as the colnames from the raw data, the colors chosen arbitrarily based on the length of groups and the graph theme is adjusted by *theme_bw* .

```
groups <- factor(paste(annotation$Treatment, annotation$Treated_genes, sep = "_"))

coldata <- names(count_data)


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()
```


Multi Dimensional Scaling
5000
0
−5000
Poisson Distance
IL4−treate
IL4−treated_1
IL4−treated_2
GE2−treated_1
PGE2−treated_3
PGE2−treated_2
Untreated_1
Untreated_2
Untreated_3
−treated_1
IFNg−treated_2
IFNg−treated_3
−10000
−5000
0
5000
10000
15000
Poisson Distance
groups
a Treated_IFNg
a Treated_IL4
a Treated_PGE2
a Untreated_control

4.1 Pre-processing

A different way of normalizing the data using the fragments per million mapped fragments (FPM) formula. As shown below the count data of the samples is divided by the sum of the count data divided by one million. The log2 is taken from the results plus one, as to remove zero values while normalization.

```
counts.fpm <- log2( (count_data / (colSums(count_data) / 1e6)) + 1 )
pander(counts.fpm[1:3, 1:5])
```

Table 10: Table continues below

	Untreated_1	IL4-treated_1	PGE2-treated_1
ENSG00000223972.5	1.399	1.171	1.798
ENSG00000227232.5	3.178	5.066	4.744
ENSG00000243485.5	0.3806	0.783	1.318

	IFNg-treated_1	Untreated_2
ENSG00000223972.5	1.805	0.03403
ENSG00000227232.5	6.128	1.011
ENSG00000243485.5	0.9783	0.1893

4.1 Assignment

In the code block below genes are kept if they hold a count higher than five for at least six samples. If this requirement is not met, then the gene (or row) will be filtered from the count data. In this case half of the samples (6) must meet the requirements.

```
# function that receives three parameters, the count_data object,
# a minimum count for a sample as threshold
# and n_samples for amount of samples that satisfy the threshold
normalize_data <- function(count_data, min_count, n_samples){
  keep <- rowSums(count_data >= min_count) >= n_samples
  kept_counts <- nrow(dds[keep,])
  return(kept_counts)
}

filtered_data <- normalize_data(count_data, 5, 6)

original_gene_count <- nrow(count_data)
difference_filtered_count <- filtered_data - original_gene_count
cat("After the filtering step with minimum count 5 for at least 6 samples the new
filtered count is substracted from the original count:", original_gene_count, "-",
filtered_data, "=", difference_filtered_count, "genes")
```

```
## After the filtering step with minimum count 5 for at least 6 samples the new
## filtered count is substracted from the original count: 24365 - 21527 = -2838 genes
```

4.2 Fold Change

In the code block below the means are calculated for two groups (experiment and control) per gene using the *rowMeans()* function. Then the log fold change is calculated by substracting the control averagescompanying

article showing differences in performance compared to other methods and packages. from the experiments averages.

Finally a histogram is made with breaks of forty for clearer visualisation of the deviation. Also an two vertical lines are added at -1 and 1 indicating low fold-change values in between the borders and vice versa.

```
# step 1
counts.fpm$experiment_1_avg <- rowMeans(counts.fpm[case2])
counts.fpm$control_avg <- rowMeans(counts.fpm[control])
LFC_column <- counts.fpm$experiment_1_avg - counts.fpm$control_avg
hist(LFC_column, breaks = 40, main = "Log fold change of group PGE2-treated")
abline(v = c(-1 ,1), lt = 2,  lwd = 2, col = "red")
```

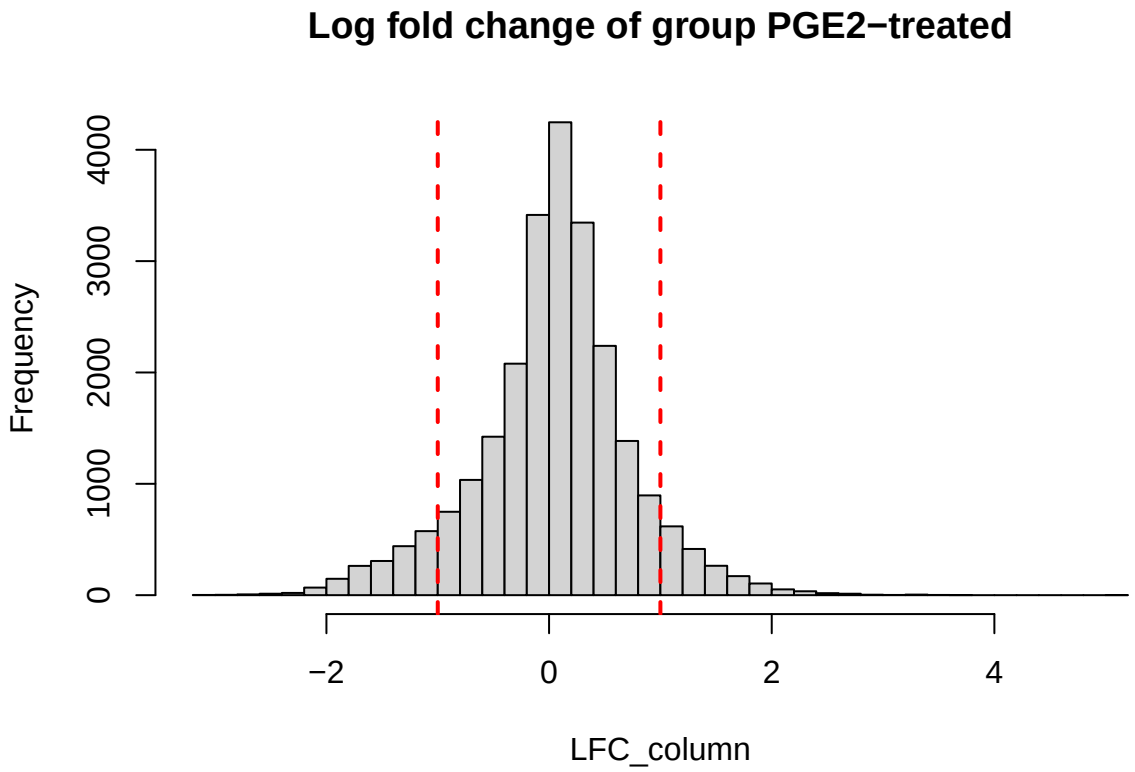


Figure 4: Log fold change of group PGE2-treated

In the figure 1.8 above a histogram is made of the log fold change values between two groups. A clear normal deviation is visible.

4.3 Using Bioconductor Packages

In the research's article it's mentioned that the t-test and chi-test are used for calculating the p-values.

4.3.1 The Design (matrix)

Here a matrix model is formed that can be used as the design in the ddsmatrix.

```
Treated_genes
```

```
##  [1] control IL4     PGE2    IFNg    control IL4     PGE2    IFNg    control
## [10] IL4     PGE2    IFNg
```

```
## Levels: control IL4 PGE2 IFNg
```

```
design.matrix <- model.matrix(~ Treated_genes, data=annotation)
design.matrix
```

```
##                (Intercept) Treated_genesIL4 Treated_genesPGE2 Treated_genesIFNg
## Untreated_1              1                0                 0                 0
## IL4-treated_1            1                1                 0                 0
## PGE2-treated_1           1                0                 1                 0
## IFNg-treated_1           1                0                 0                 1
## Untreated_2              1                0                 0                 0
## IL4-treated_2            1                1                 0                 0
## PGE2-treated_2           1                0                 1                 0
## IFNg-treated_2           1                0                 0                 1
## Untreated_3              1                0                 0                 0
## IL4-treated_3            1                1                 0                 0
## PGE2-treated_3           1                0                 1                 0
## IFNg-treated_3           1                0                 0                 1
## attr(,"assign")
## [1] 0 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$Treated_genes
## [1] "contr.treatment"
```

4.3.2 DESeq2

Below a new ddsMat is set up with as main difference passing an factor with annotated data to 'colData' and a model.matrix is passed as design.

After setting up this matrix passed as an DESeqDataSet object and the names of the individual groups are extracted using the 'resultsNames()' function.

Finally these names can be passed to the 'results()' and the summary of each result object shows the amount of up or down regulated genes for that group, also an adjusted p-value, outliers, low counts and a mean count.

```
(ddsMat <- DESeqDataSetFromMatrix(countData = count_data,
                                  colData = annotation,
                                  design = design.matrix))
```

```
## class: DESeqDataSet
## dim: 24365 12
## metadata(1): version
## assays(1): counts
## rownames(24365): ENSG00000223972.5 ENSG00000227232.5 ...
##   ENSG00000278817.1 ENSG00000277196.4
## rowData names(0):
## colnames(12): Untreated_1 IL4-treated_1 ... PGE2-treated_3
##   IFNg-treated_3
## colData names(2): Treatment Treated_genes
```

```
dds <- DESeq(ddsMat)
```

```
## using supplied model matrix
```

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

```
resultsNames(dds) # lists the names of individual effects
```

```
## [1] "Intercept"         "Treated_genesIL4"  "Treated_genesPGE2"
## [4] "Treated_genesIFNg"
```

```
untreated_lfc <- summary(results(dds, name="Intercept"))
```

```
##
## out of 24365 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 21570, 89%
## LFC < 0 (down)     : 0, 0%
## outliers [1]       : 114, 0.47%
## low counts [2]     : 0, 0%
## (mean count < 1)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
Treated_genesIL4_lfc <- summary(results(dds, name="Treated_genesIL4"))
```

```
##
## out of 24365 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 210, 0.86%
## LFC < 0 (down)     : 218, 0.89%
## outliers [1]       : 114, 0.47%
## low counts [2]     : 5621, 23%
## (mean count < 11)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
Treated_genesPGE2_lfc <- summary(results(dds, name="Treated_genesPGE2"))
```

```
##
## out of 24365 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 634, 2.6%
## LFC < 0 (down)     : 1013, 4.2%
## outliers [1]       : 114, 0.47%
## low counts [2]     : 9368, 38%
## (mean count < 28)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
Treated_genesIFNg_lfc <- summary(results(dds, name="Treated_genesIFNg"))
```

```
##
## out of 24365 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 571, 2.3%
## LFC < 0 (down)     : 317, 1.3%
```

```
## outliers [1]       : 114, 0.47%
## low counts [2]     : 6090, 25%
## (mean count < 12)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

A way for removing noise of log2 fold change data of low count genes is by getting the results from the 'lfcShrink()' instead of the 'results()' function.

It's beneficial to do this since when plotting so that the log fold changes stand out.

```
Treated_genesIL4_shrink <- lfcShrink(dds, coef="Treated_genesIL4", type="apeglm")
```

```
## using 'apeglm' for LFC shrinkage. If used in published research, please cite:
##     Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distributions for
##     sequence count data: removing the noise and preserving large differences.
##     Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895
```

```
Treated_genesPGE2_shrink <- lfcShrink(dds, coef="Treated_genesPGE2", type="apeglm")
```

```
## using 'apeglm' for LFC shrinkage. If used in published research, please cite:
##     Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distributions for
##     sequence count data: removing the noise and preserving large differences.
##     Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895
```

```
Treated_genesIFNg_shrink <- lfcShrink(dds, coef="Treated_genesIFNg", type="apeglm")
```

```
## using 'apeglm' for LFC shrinkage. If used in published research, please cite:
##     Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distributions for
##     sequence count data: removing the noise and preserving large differences.
##     Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895
```

```
par(mfrow = c(1,3))
plotMA(Treated_genesIL4_shrink, main = "Log2 fold change of IL4 with shrinkage")
plotMA(Treated_genesPGE2_shrink, main = "Log2 fold change of PGE2 with shrinkage")
plotMA(Treated_genesIFNg_shrink, main = "Log2 fold change of IFNg with shrinkage")
```

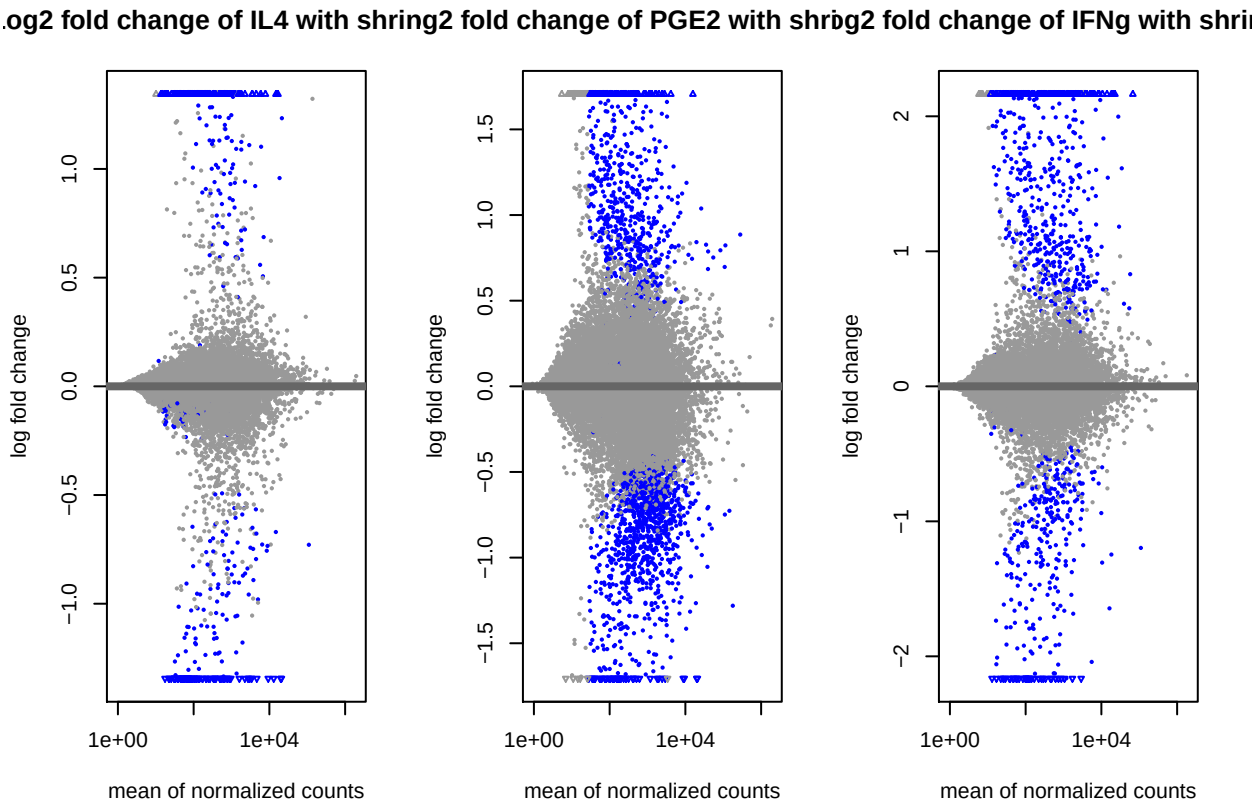


Figure 5: Log fold change of group PGE2-treated

5. Data analysis and visualisation

5.1 Volcano Plot

Bibliography

1. GEO Browser. Geraadpleegd op 14-02-2023. De link: https://www.ncbi.nlm.nih.gov/geo/browse/.
2. Geo Browser. Blood milieu in acute myocardial infarction reprograms human macrophages for trauma repair. Geraadpleegd op 14-02-2023. De link: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE215801