Script2_Filtering out outliers (genes-samples)

Nickie Safarian

6/20/2022

R Markdown

This script explains steps to detect outliers and filter them out from the counts matrix.

Load Packages

Import the data

```
counts <-  readRDS("/external/rprshnas01/kcni/nsafarian/Toker_PD_Project/pd_complex-i_stratification/Da

metadataFinal <- read.csv("/external/rprshnas01/kcni/nsafarian/Toker_PD_Project/pd_complex-i_stratifica
```

Set rownames of the metadata

```
rownames(metadataFinal) <- metadataFinal$RNAseq_id_ParkOme2
```

Prepare the count matrix

```
# convert the count matrix into a dataframe and round the counts
cts <- counts %>% as.data.frame %>% round()

# only keep samples that have input in the metadataFinal
cts.PD <- cts[ , metadataFinal$RNAseq_id_ParkOme2]
```

Quality control of counts matrix

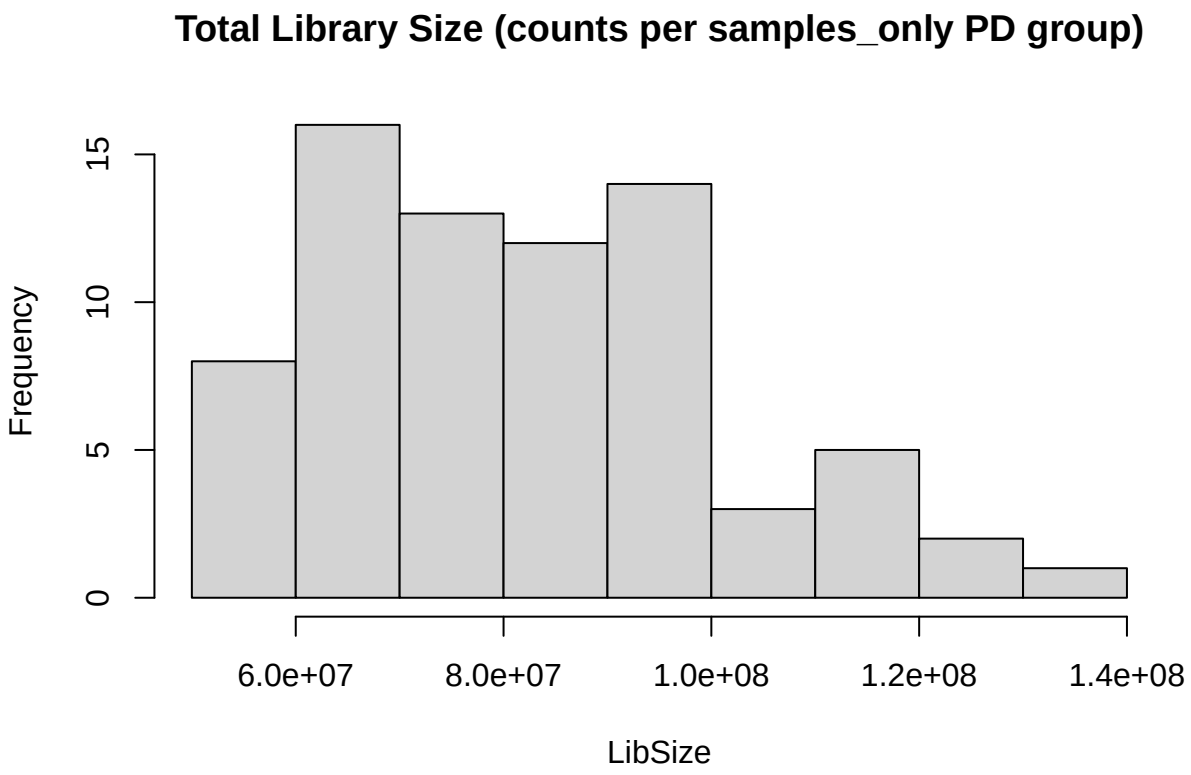
Step 1: calculate the library size

```
# Total gene counts per sample/ Library size
LibSize <- apply(cts.PD, 2, sum) # Total Library Size (counts per samples),
                                 # is from 50M to 140M
summary(LibSize)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##  50383965  64275945  80150594  81681350  95868654 133863330
```

Plot the LibSize

```
hist(LibSize, main="Total Library Size (counts per samples_only PD group)")
```



Note that few samples have total counts more than 100M and may be outlier.

You can add the library size column (info) to the metadata

```
SampleIDcol = "RNAseq_id_ParkOme2"
metadataFinal$LibSize <- LibSize[match(metadataFinal[[SampleIDcol]], names(LibSize))]
```

If removing outlier samples was necessary, follow the steps below

```
# 1) According to (https://doi.org/10.3389/fnmol.2022.903175) subjects can
# be deemed outliers and removed if they differed from the sample median of
# any of the first 5 latent components by more than 3 interquartile ranges .
# # Define the IR
# Interquartile.Range= Upper.Quartile - Lower.Quartile

# IR for LibSize => 94852986-64334628= 30518358
# 3IR => 3*30518358 = 91555074

# # How many samples meet the 100M threshold?
# outlier.samples= as.data.frame(LibSize[LibSize > 91555074]) #23 samples
```

```
# outlier.samples
#
# # Define a list of samples to be removed
# rm.samples <- colnames(cts.PD) %in% rownames(outlier.samples)
#
# # How many of the remaining samples are in each MT grouping
# metadata.filtered <- metadataFinal[!rm.samples, ]
#
# table(metadata.filtered$MT.Grouping, metadata.filtered$Cohort2)
# # NonMT_PD    MT_PD
# #       47       16
#
# # Filter count matrix for outlier samples
# cts.PD.filtered <- cts.PD[, !rm.samples]
```

Step 2: calculate the rowSums

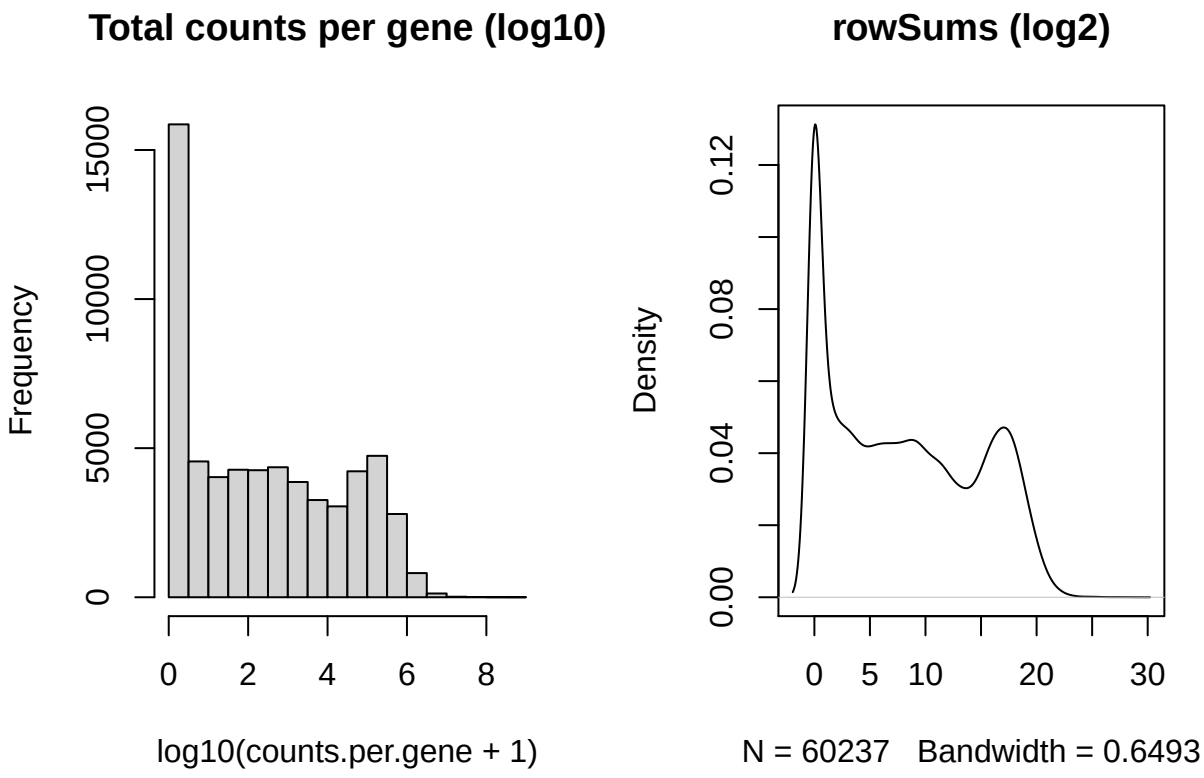
Count data is not normally distributed, so if we want to examine the distributions of the raw counts we need to log the counts.

```
counts.per.gene <- Matrix::rowSums(cts.PD)
summary(counts.per.gene)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##         0         2       146    100344     13172 322639685
```

Plot the rowSums

```
par(mfrow=c(1,2))
hist(log10(counts.per.gene+1), main="Total counts per gene (log10)")
plot(density(log2(counts.per.gene + 1)), main="rowSums (log2)")
```




```
# Note that so many genes appear to have 0 counts. Several genes also sow extreme high counts (log10 >=
```

Calculate the number of samples with counts for genes

```
count.dt = cts.PD

count.dt$count.num <- rowSums(count.dt !=c(0), na.rm=T)

table(count.dt$count.num)
```

```
##
##     0     1     2     3     4     5     6     7     8     9    10    11    12
## 11986  2794  1580  1164   921   766   644   610   503   442   430   367   308
##    13    14    15    16    17    18    19    20    21    22    23    24    25
##   329   260   278   253   247   239   244   224   215   217   200   188   215
##    26    27    28    29    30    31    32    33    34    35    36    37    38
##   186   203   199   169   178   184   181   151   174   140   171   166   157
##    39    40    41    42    43    44    45    46    47    48    49    50    51
##   152   152   177   190   139   149   170   165   162   177   166   184   171
##    52    53    54    55    56    57    58    59    60    61    62    63    64
##   147   140   169   191   188   168   177   193   199   206   208   238   244
##    65    66    67    68    69    70    71    72    73    74
##   251   268   273   332   394   410   503   719  1182 23400
```

```
# Note that 11986 genes have 0 counts across all 74 samples,
# and 21410 genes have less counts in less than 10 samples
```

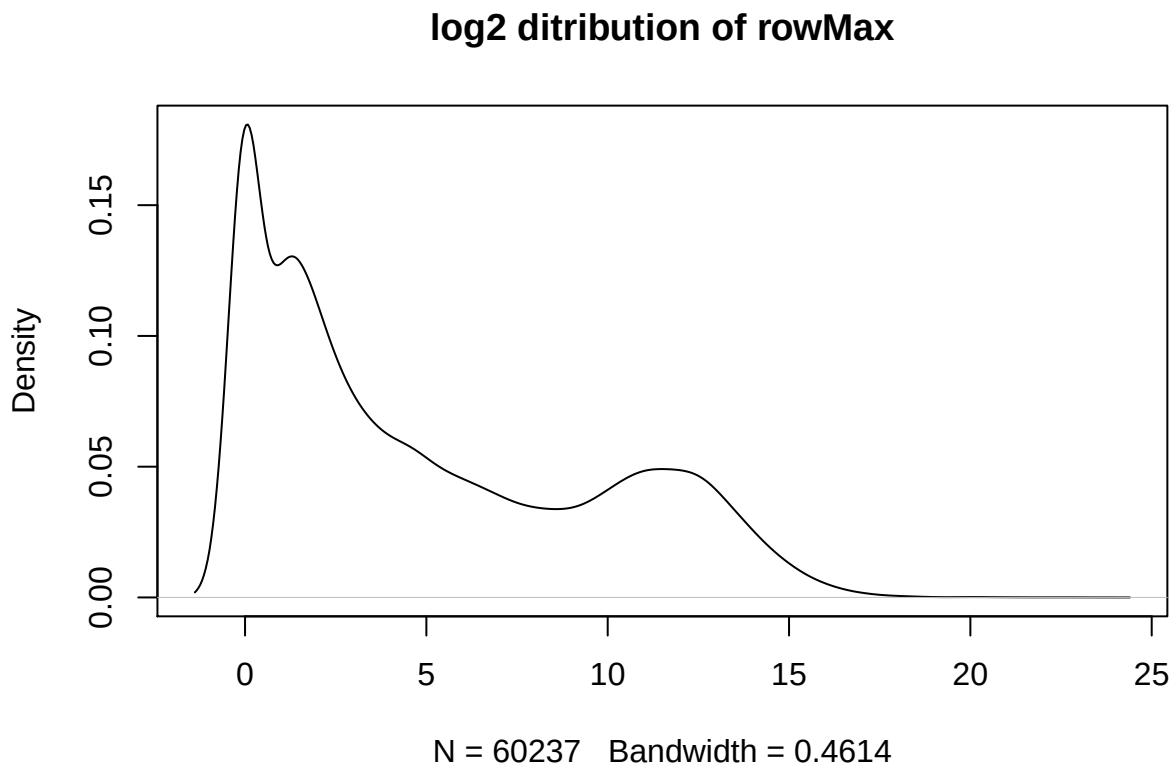
Step 3: check on the rowMax values

Using Max counts is another approch for filtering data. Let's take a look.

```
row.max <- apply (cts.PD, 1, max)
summary(row.max)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       1      10    2937     444 8481817
```

```
plot(density(log2(row.max + 1)), main="log2 ditribution of rowMax")
```



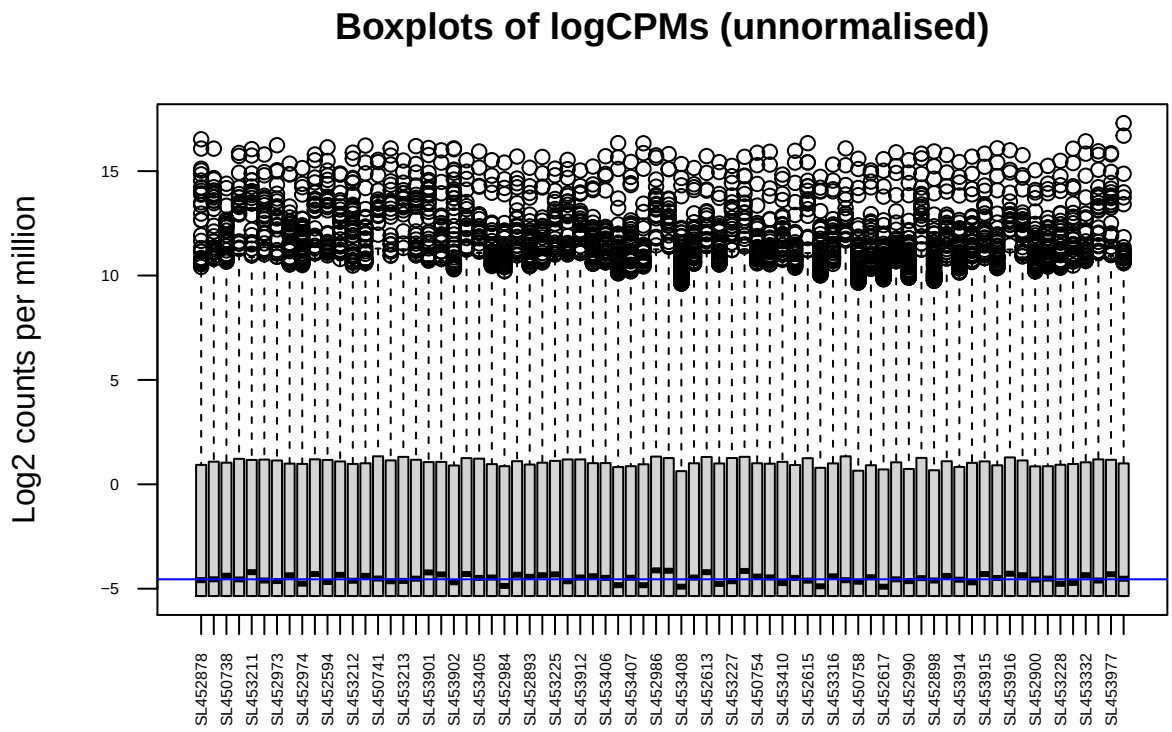

```
# what I understand from this plot is that rowMax=~ 15 (log2(15+1)=5) may serve as a cutoff threshold f
```

Step 4: Get the count.per.million values

We can use the cpm function to get log2 counts per million, which are corrected for the different library sizes. The cpm function also adds a small offset to avoid taking log of zero.

```
library(edgeR)
library(limma)
library(Rcpp)

# Get log2 counts per million
```

```
CPM <- cpm(cts.PD,log=TRUE)

# Check distributions of samples using boxplots
boxplot(CPM, xlab="", ylab="Log2 counts per million",las=2, cex.axis=0.5)
# Let's add a blue horizontal line that corresponds to the median logCPM
abline(h=median(CPM),col="blue")
title("Boxplots of logCPMs (unnormalised)")  # based on this plot samples
```




```
                                        # are not very dispersed.
```

Note: it's Strongly suggested that we do not filter reads in the data based on raw counts. Instead, we should just make the dds (DESeq2) object, then remove outliers on the normalized count matrix.

If using EdgeR package, you will need to filter data using CPM values. Usually a CPM of 0.5 is used as it corresponds to a count of 10-15 for the library sizes in this data set. If the count is any smaller, it is considered to be very low, indicating that the associated gene is not expressed in that sample. A requirement for expression in two or more libraries is used as each group contains two replicates. This ensures that a gene will be retained if it is only expressed in one group. Smaller CPM thresholds are usually appropriate for larger libraries. As a general rule, a good threshold can be chosen by identifying the CPM that corresponds to a count of 10, which in this case is about 0.5. You should filter with CPMs rather than filtering on the counts directly, as the latter does not account for differences in library sizes between samples.

Filter out data for genes with zero expression level in all samples

Here, before I proceed with making a dds object, I'll first remove all genes that have 0 counts across all samples, as they have no weights for DE analysis.

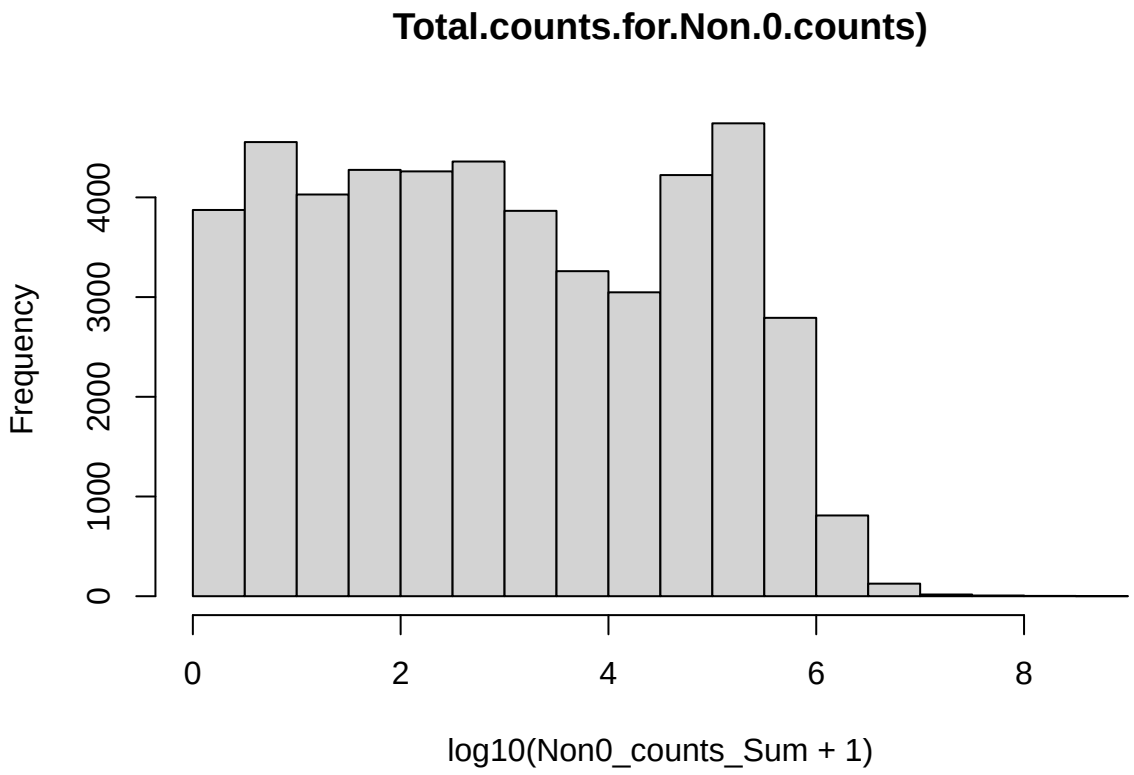
```
Non0_counts <- cts.PD %>% dplyr::filter(! rowSums(.) == 0) # 48251


# check if 0count reads are truly removed from the matrix
Non0_counts_Sum <- Matrix::rowSums(Non0_counts)
summary(Non0_counts_Sum)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##         1        27       710    125270     39184 322639685
```

Note that the counts Median (from 146 to 710) and mean (from 100344 to 125270) have increased after removing zero counts.

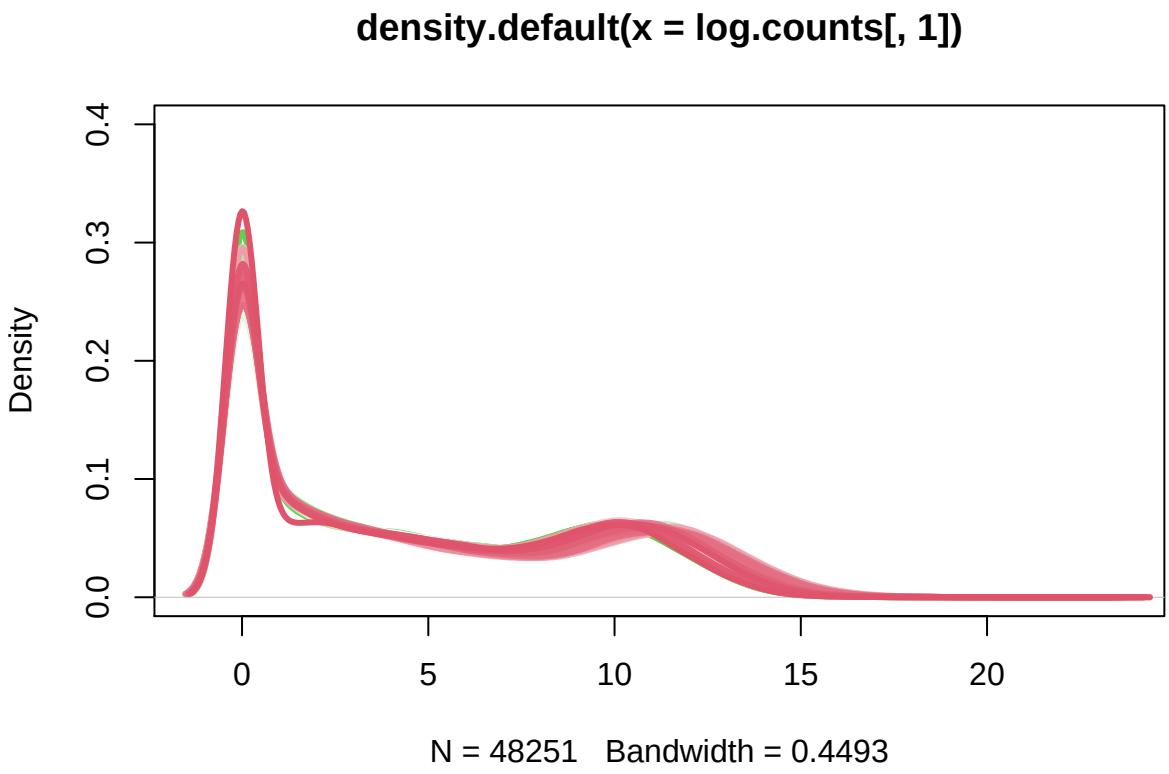
Plot distribution of counts in the Non0_matrix

```
hist(log10(Non0_counts_Sum+1), main="Total.counts.for.Non.0.counts)")
```



Show distributions for log2_Non0_counts for all samples

```
log.counts = log2(Non0_counts + 1)
colramp = colorRampPalette(c(3,"white",2))(74)
plot(density(log.counts[,1]),col=colramp[1],lwd=3,ylim=c(0,0.4))
for(i in 1:74){lines(density(log.counts[,i]),lwd=3,col=colramp[i])}
```



Make sure that the columns of the count data are in the same order as rows names of the metadata*

```
all(colnames(Non0_counts) == rownames(metadataFinal)) # same order check
```

```
## [1] TRUE
```

```
all(colnames(Non0_counts) %in% rownames(metadataFinal))
```

```
## [1] TRUE
```

```
                # same samples across two data
```

Create the DEseq2DataSet object

Step 1: Scale the numerical variables of the metadata

```
metadataFinal$Age = scale(metadataFinal$Age)
metadataFinal$DV200 = scale(metadataFinal$DV200)
metadataFinal$PropPos = scale(metadataFinal$PropPos)
metadataFinal$RIN = scale(metadataFinal$RIN)
metadataFinal$PMI = scale(metadataFinal$PMI)
metadataFinal$DV300 = scale(metadataFinal$DV300)
```

Step 2: Factor and level the categorical variables

```
metadataFinal$Cohort2 <- factor(metadataFinal$Cohort2,
                                levels= c("Barcelona", "Norway"))

metadataFinal$Sex <- factor(metadataFinal$Sex, levels=c("M", "F"))

metadataFinal$MT.Grouping <- factor(metadataFinal$MT.Grouping,
                                    levels=c("NonMT_PD", "MT_PD"))
```

Step 3: Make the dds object

```
dds <- DESeqDataSetFromMatrix(countData=Non0_counts,
                              colData= metadataFinal,
                              design= ~ Age+PMI+Sex+Cohort2+MT.Grouping+Sex:MT.Grouping)
```

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

```
saveRDS(dds, "dds.Non0counts.MultiFactorial.Rds")
```

After making the dds object we can apply a cutoff threshold and assess how removing those genes may affect the final DEG results.

Filter out genes that have less than 10 counts in at least 6 samples

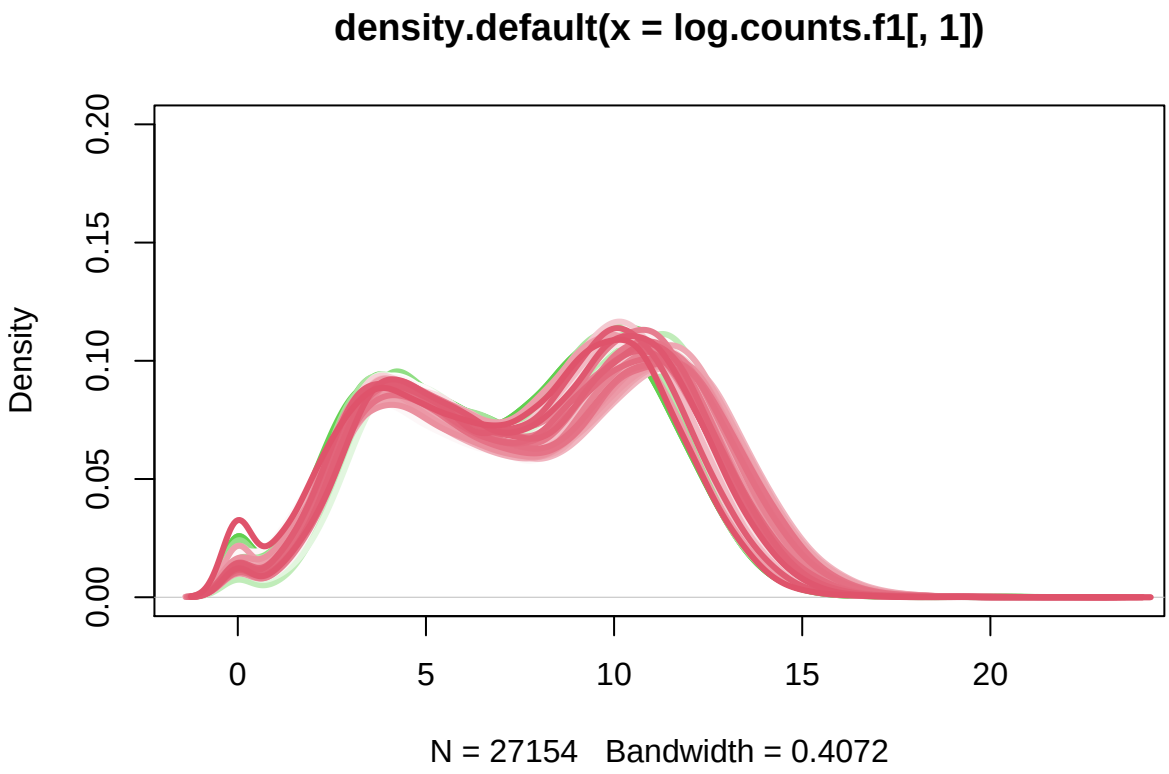
A count of 10 is the default, the minimum number of samples comes form the number of subjects in the smallest group. Somewhere between 40-70% of that number, which in the MT_PD group case with N=16 will be 6. Ass shown in lines 140-145, about 19211 genes have counts in <= 6 samples (11986+2794+1580 +1164 +921+766= 19211). So, using a cutoff of six samples will remove 19211 genes from the matrix.

Remove low count reads

```
dds <- estimateSizeFactors(dds)
idx <- rowSums(counts(dds, normalized=TRUE) >= 10 ) >= 6 # remove genes with
                                                         # counts less than 10
                                                         #in at least 6 samples
dds.f <- dds[idx,]   # save it as a new matrix, it has 27154 genes
```

Show distributions for log2 filtered counts

```
log.counts.f1 = log2(counts(dds.f) + 1)
colramp = colorRampPalette(c(3,"white",2))(74)
plot(density(log.counts.f1[,1]),col=colramp[1],lwd=3,ylim=c(0,.20))
for(i in 1:74){lines(density(log.counts.f1[,i]),lwd=3,col=colramp[i])}
```



Note: based on the previous analyses I learnd that outliers with extreme high counts also will trouble the downstream DE analysis. So, I'll remove all those genes as well.

```
# Remove genes with high counts
idy <- rowSums(counts(dds.f, normalized=TRUE) > 1000000)  # remove genes
                                                          # with more than 1M
                                                          # counts
dds.new <- dds.f[!idy,]
       # 27138 genes remain, meaning 16 more genes are removed
```

Note: in bulk tissue RNA-seq data usually Max.counts are around 10-20K sometimes to 100K. I used 100K and 1M as the cutoffs here.

Summary of the number of samples showing counts for the genes we removed by the High-counts filtering step:

With a cutoff of 1M, we remove 16 genes, from which only one has counts in all 74 samples.

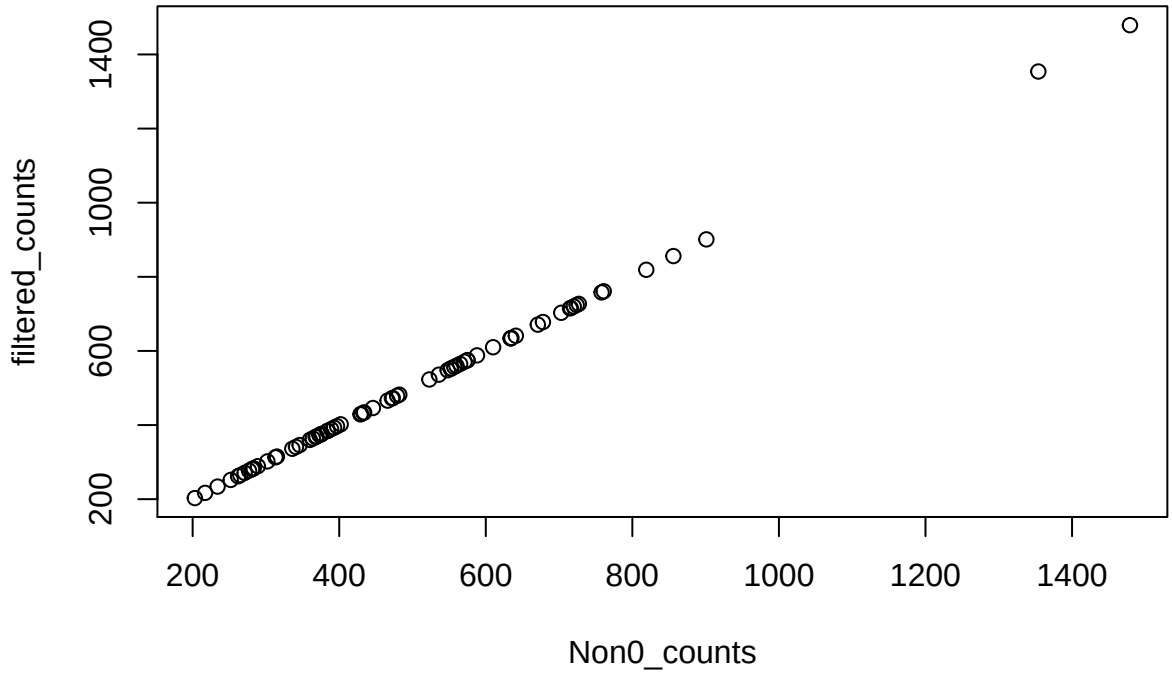
```
table(idy)
```

```
## idy
##     0     1     2     4     5     6     7     9    16    22    53    71    73
## 27138     1     1     1     2     1     2     1     2     1     1     1     1
##    74
##     1
```

```
# the top row show number of samples expressing that gene
# and the bottom row shows number of genes removed
```

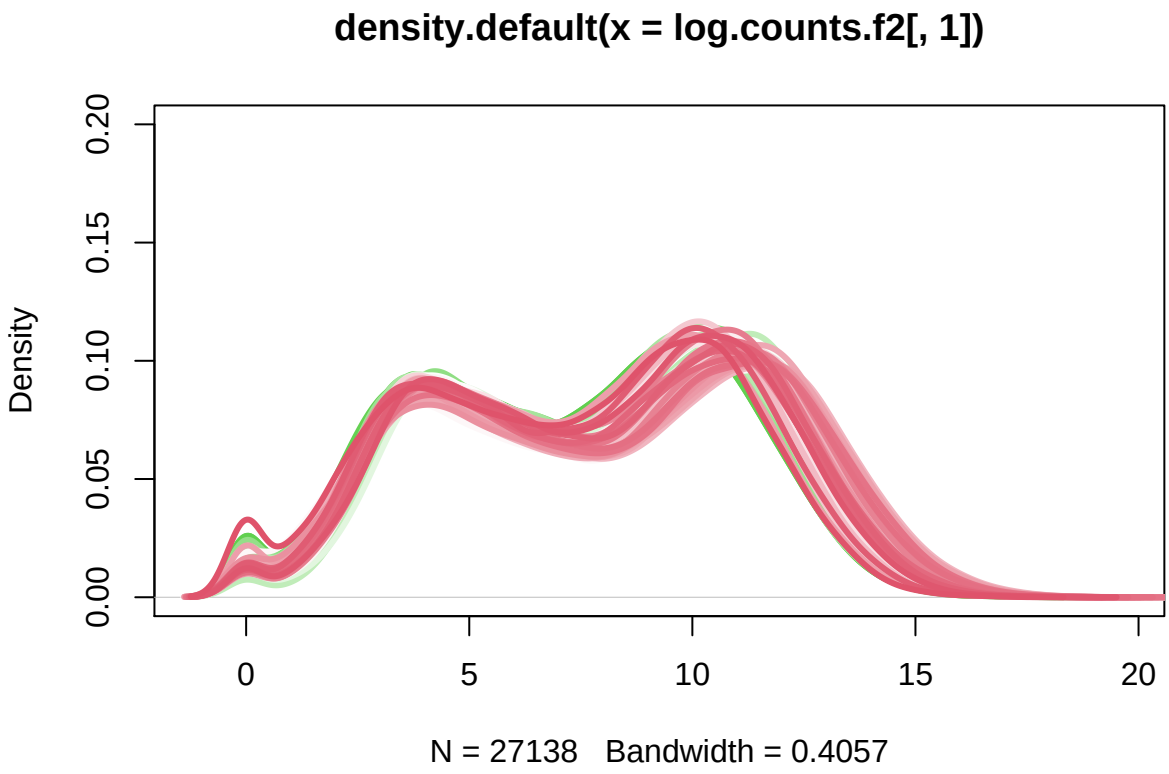
Let's have a look and see whether our thresholds of >10 and <1M do indeed correspond to a count of about 10-100K

```
# We will look at the first gene

X = counts(dds)[1, ]
Y = counts(dds.new)[1, ]

plot(X, Y, xlab="Non0_counts", ylab="filtered_counts")
```



Show distributions for log2 counts after second filteration for all samples

```
log.counts.f2 = log2(counts(dds.new) + 1)
colramp = colorRampPalette(c(3,"white",2))(74)
plot(density(log.counts.f2[,1]),col=colramp[1],lwd=3,ylim=c(0,.2))
for(i in 1:74){lines(density(log.counts.f2[,i]),lwd=3,col=colramp[i])}
```

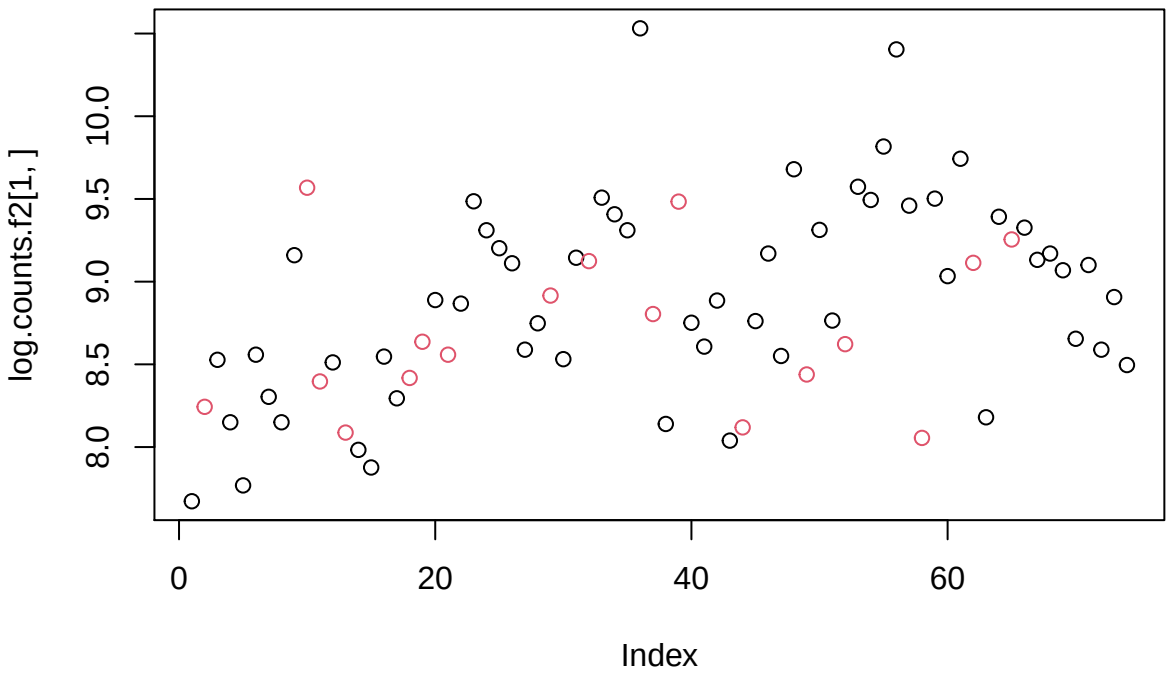


Note that the distributions aren't perfectly the same, but for the most part the distributions land right on top of each other.

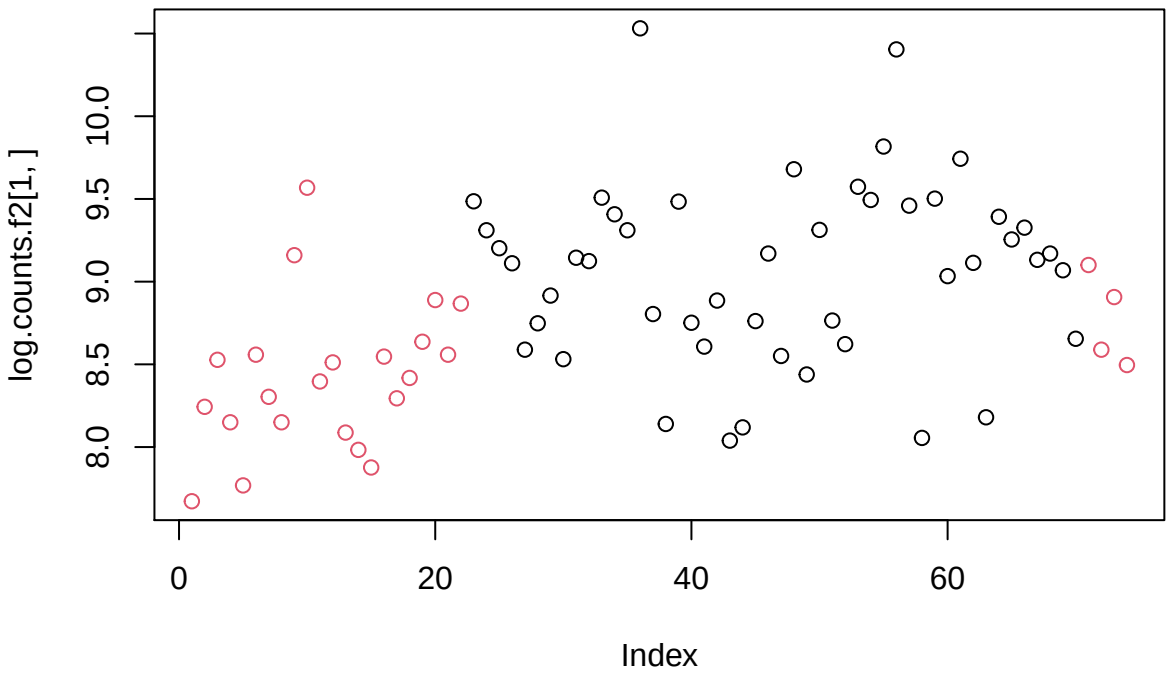
Matching distributions leaves variability

Normalization removes bulk differences due to technology. But there still may be differences you don't want after normalization. The only way to figure this out is to check.

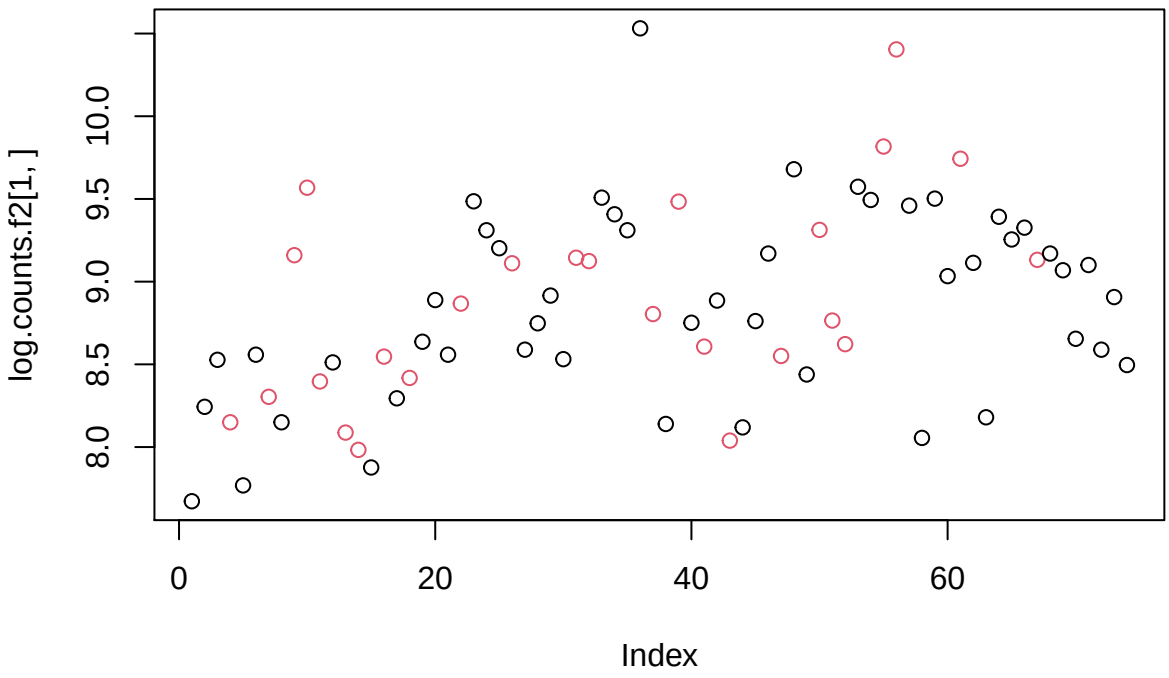
```
plot(log.counts.f2[1,], col=as.numeric(dds.new@colData$MT.Grouping))
```




```
plot(log.counts.f2[1,], col=as.numeric(dds.new@colData$Cohort2)) # major
```




```
                                                                  # effect
plot(log.counts.f2[1,], col=as.numeric(dds.new@colData$Sex))
```



Use Violin plot to show log counts:

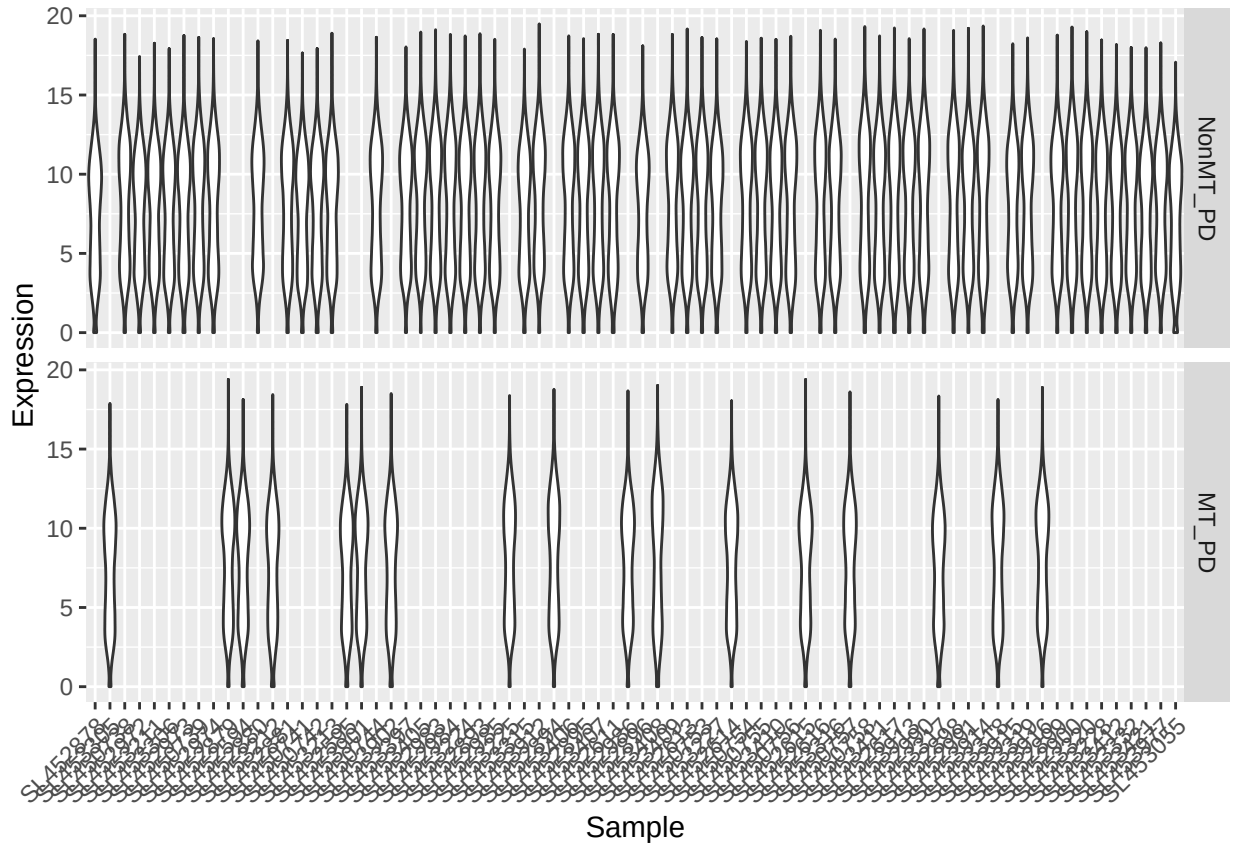
```
require(reshape2)
```

```
## Loading required package: reshape2
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
violinMatrix <- reshape2::melt(log.counts.f2)
colnames(violinMatrix) <- c("gene", "Sample","Expression")


Mit.type <- metadataFinal[, c("RNAseq_id_ParkOme2","MT.Grouping")]

new.violinMatrix <- merge(violinMatrix, Mit.type , by.x="Sample", by.y="RNAseq_id_ParkOme2" )

library(ggplot2)
ggplot(new.violinMatrix, aes(x=Sample, y=Expression)) +
  geom_violin() +
  theme(axis.text.x = element_text(angle=45, hjust=1))+
  facet_grid("MT.Grouping")
```



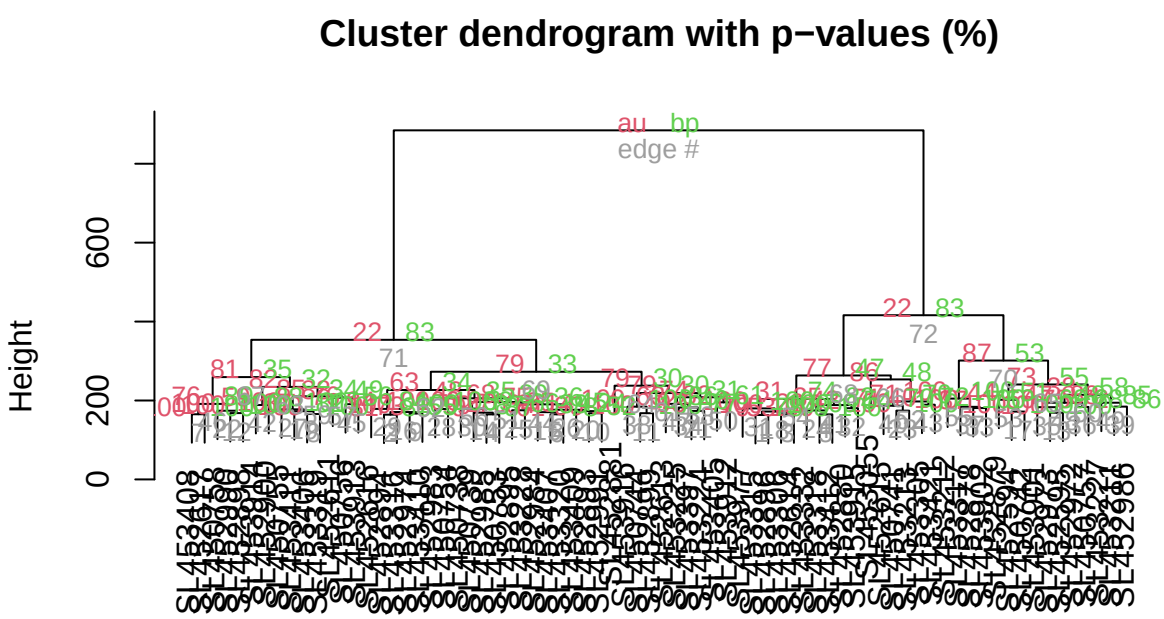
More for outlier detection (optional):

Bootstrapped hierarchical clustering (unsupervised - i.e. entire dataset) Using regularised log or variance stabilized counts:

```
library(pvclust)

pv <- pvclust(log.counts, method.dist="euclidean", method.hclust="ward.D2", nboot=10)
```

```
## Bootstrap (r = 0.5)... Done.
## Bootstrap (r = 0.6)... Done.
## Bootstrap (r = 0.7)... Done.
## Bootstrap (r = 0.8)... Done.
## Bootstrap (r = 0.9)... Done.
## Bootstrap (r = 1.0)... Done.
## Bootstrap (r = 1.1)... Done.
## Bootstrap (r = 1.2)... Done.
## Bootstrap (r = 1.3)... Done.
## Bootstrap (r = 1.4)... Done.
```

```
plot(pv)
```



Distance: euclidean
Cluster method: ward.D2

Save the dds files for future analysis

```
# saveRDS(dds.f, "dds.f.Above10counts.MultiFactorial.Rds")
# saveRDS(dds.new, "dds.new.10to1Mcounts.MultiFactorial.Rds")
```