Group 5 RNASeq Mini Project: Differential Expression Analysis

EANBiT RT Group 5

8/19/2020

Introduction

In this section we perform Differential Expression following our previous analysis on alignment. Differential Expression is basically focused on finding out which genes are either being upregulated or down-regulated between the cases compared to controls.

In our analyis we had 6 samples (samples 37-42), for which samples 37-39 were normal samples and samples 40-42 were diseased samples.

Counts to be used have been generated from two approaches of alignment, ie;

- The classical alignment using **`HISAT2`**, and
- Pseudo-alignment using **`Kallisto`**

Loading Libraries

```
library(DESeq2)
library(GenomicFeatures)  #For creating the tx2gene file for kallisto counts import
library(tximport)  #Importing kallisto counts
library(pheatmap)  #For generating heatmaps
library(vsn)  #for plotting dispersion plot
library("RColorBrewer")

require(genefilter)
require(calibrate)

#BiocManager::install("org.Hs.eg.db")
library(org.Hs.eg.db)  #For including gene symbols

#BiocManager::install("reactome.db")
library(reactome.db)  #For gene function annotations
library(AnnotationDbi)
```

Importing data

Reading metadata

```
#Reading the metadata file
meta <- read.csv("practice.dataset.metadata.tsv", sep="\t", row.names = 1, header = T, stringsAsFactors
meta
```

```
##          Condition
## sample37    normal
## sample38    normal
## sample39    normal
## sample40   disease
## sample41   disease
## sample42   disease
```

```
levels(meta$Condition)
```

```
## [1] "disease" "normal"
```

Looking at our metadata file, the levels for the condition are set as `disease` followed by `normal` because by default R follows alphabetical order to arrange the levels of a factor. However, when using DESeq the first level is treated as the control (in our case its normal), and the second is taken as the case group (in our case the disease).
So we will have to rearrange our levels so that normal is picked as the reference/Control group and disease as the case group.

To achieve this will use the **`relevel`** function and specify normal as our reference.

```
#Releveling the metadata Conditions
meta$Condition <- relevel(meta$Condition, ref = "normal")
#Check our levels again
levels(meta$Condition)
```

```
## [1] "normal"  "disease"
```

We now proceed to import the counts from the 2 programs we used during our alignment in 2 different ways as specified by the DESeq2 manual.

Importing Counts matrices

DESeq2 has basically 4 ways of importing counts to create a `DESeqDataSet` object that can be used for downstream analysis. These inlcude;

- Importing Count matrix generated by featureCounts from either a hisat2 or STAR alignment.
- Importing pseudo-alignments output eg from kallisto or Salmon using `tximport`.
- Reading counts from `htseq-count` files.
- Reading counts from a `SummarizedExperiment` object.

a. Importing HISAT2's featurecounts matrix

Featurecounts produces a single matrix file containing the counts for all the samples used. To import feature counts matrix, we use the **DESeqDataSetFromMatrix** function from DESeq2 to create an object that can be converted into a DESeq object.

```
# Importing HISAT2's featurecounts

countdata <- read.csv("Hisat_featurecounts/hisat_counts.txt", sep="\t", header=T, row.names=1, comment.
#countdata <- read.csv("hisat2/hisat2_counts.txt", sep="\t", header=T, row.names=1, comment.char = "#")
head(countdata)
```

```
##                                  Chr
## ENSG00000223972     1;1;1;1;1;1;1;1;1
## ENSG00000227232 1;1;1;1;1;1;1;1;1;1;1
## ENSG00000278267                    1
## ENSG00000243485            1;1;1;1;1
## ENSG00000284332                    1
## ENSG00000237613            1;1;1;1;1
##                                                                        Start
## ENSG00000223972            11869;12010;12179;12613;12613;12975;13221;13221;13453
## ENSG00000227232 14404;15005;15796;16607;16858;17233;17606;17915;18268;24738;29534
## ENSG00000278267                                                        17369
## ENSG00000243485                                    29554;30267;30564;30976;30976
## ENSG00000284332                                                        30366
## ENSG00000237613                                    34554;35245;35277;35721;35721
##                                                                          End
## ENSG00000223972            12227;12057;12227;12721;12697;13052;13374;14409;13670
## ENSG00000227232 14501;15038;15947;16765;17055;17368;17742;18061;18366;24891;29570
## ENSG00000278267                                                        17436
## ENSG00000243485                                    30039;30667;30667;31109;31097
## ENSG00000284332                                                        30503
## ENSG00000237613                                    35174;35481;35481;36073;36081
##                                 Strand Length sample37_hisat_sorted.bam
## ENSG00000223972     +;+;+;+;+;+;+;+;+   1735                         0
## ENSG00000227232 -;-;-;-;-;-;-;-;-;-;-   1351                        52
## ENSG00000278267                     -     68                         7
## ENSG00000243485             +;+;+;+;+   1021                         0
## ENSG00000284332                     +    138                         0
## ENSG00000237613             -;-;-;-;-   1219                         0
##                 sample38_hisat_sorted.bam sample39_hisat_sorted.bam
## ENSG00000223972                         0                         0
## ENSG00000227232                        48                       187
## ENSG00000278267                        11                        36
## ENSG00000243485                         0                         0
## ENSG00000284332                         0                         0
## ENSG00000237613                         0                         0
##                 sample40_hisat_sorted.bam sample41_hisat_sorted.bam
## ENSG00000223972                         0                         0
## ENSG00000227232                        56                        59
## ENSG00000278267                         2                         3
## ENSG00000243485                         1                         0
## ENSG00000284332                         0                         0
## ENSG00000237613                         1                         1
```

```
##                 sample42_hisat_sorted.bam
## ENSG00000223972                         0
## ENSG00000227232                        69
## ENSG00000278267                         4
## ENSG00000243485                         0
## ENSG00000284332                         0
## ENSG00000237613                         0
```

```
#Check the column names
colnames(countdata)
```

```
##  [1] "Chr"                         "Start"
##  [3] "End"                         "Strand"
##  [5] "Length"                      "sample37_hisat_sorted.bam"
##  [7] "sample38_hisat_sorted.bam" "sample39_hisat_sorted.bam"
##  [9] "sample40_hisat_sorted.bam" "sample41_hisat_sorted.bam"
## [11] "sample42_hisat_sorted.bam"
```

```
#Remove the unwanted columns
countdata[c("Chr", "Start", "End", "Strand", "Length")] <- NULL

#Renaming the Colnames (to remove the suffix "_hisat_sorted.bam")

#colnames(countdata) <- gsub("hisat2", "", colnames(countdata))
#colnames(countdata) <- gsub("_hisat2_sorted.bam", "", colnames(countdata))
colnames(countdata) <- gsub("_hisat_sorted.bam", "", colnames(countdata))
head(countdata)
```

```
##                 sample37 sample38 sample39 sample40 sample41 sample42
## ENSG00000223972        0        0        0        0        0        0
## ENSG00000227232       52       48      187       56       59       69
## ENSG00000278267        7       11       36        2        3        4
## ENSG00000243485        0        0        0        1        0        0
## ENSG00000284332        0        0        0        0        0        0
## ENSG00000237613        0        0        0        1        1        0
```

```
#Check if samples in the countdata matrix have a corresponding entry in the metadata file
all(rownames(meta) %in% colnames(countdata))
```

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

```
#Check if the order of the samples in the matrix is similar to that in the metadata file.
all(colnames(countdata) == rownames(meta))
```

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

```
#In case the Order is not the same, we shall rearrange our count matrix using rownames of meta using;
#countdata <- countdata[, rownames(meta)]

#Importing the data into DESeqDataSet Object
dds <- DESeqDataSetFromMatrix(countData = countdata,
                              colData = meta,
                              design = ~ Condition)
dds
```

```
## class: DESeqDataSet
## dim: 60683 6
## metadata(1): version
## assays(1): counts
## rownames(60683): ENSG00000223972 ENSG00000227232 ... ENSG00000277475
##   ENSG00000268674
## rowData names(0):
## colnames(6): sample37 sample38 ... sample41 sample42
## colData names(1): Condition
```

b. Importing output from Kallisto

Kallisto unlike featurecounts, it outputs each sample's count matrix in a separate file. We then use tximport to help us gather all these sample files from the provided directory so that DESeq2 can concatenate them into a single DESeqdataset object used for downstream analysis.

One extra file required by tximport is a file that maps the transcript IDs in our alignment to the gene IDs, which we shall call **tx2gene**. This file is created using the `GenomicFeatures` Bioconductor package, which requires the annotation of the tracscriptome used during the pseudo alignment.

```
# Kallisto

#Download the annotation file (Run this in case the file is not yet downloaded.)
url="ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_34/gencode.v34.annotation.gff3.gz"
#download.file(url = url, "gencode.v34.annotation.gff3.gz")

#Create a table that links transcripts to genes for this dataset.
#We use the GenomicFeatures for this to create the tx2gene

# We create the TxDb obeject from the transcript annotation in the gff3 format
txdb <- makeTxDbFromGFF("gencode.v34.annotation.gff3.gz")
```

```
## Import genomic features from the file as a GRanges object ... OK
## Prepare the 'metadata' data frame ... OK
## Make the TxDb object ...

## Warning in .get_cds_IDX(mcols0$type, mcols0$phase): The "phase" metadata column contains non-NA valu
##   stop_codon. This information was ignored.

## OK
```

```
txdb
```

```
## TxDb object:
## # Db type: TxDb
## # Supporting package: GenomicFeatures
## # Data source: gencode.v34.annotation.gff3.gz
## # Organism: NA
## # Taxonomy ID: NA
## # miRBase build ID: NA
## # Genome: NA
## # Nb of transcripts: 228048
```

```
## # Db created by: GenomicFeatures package from Bioconductor
## # Creation time: 2020-08-26 21:03:26 +0300 (Wed, 26 Aug 2020)
## # GenomicFeatures version at creation time: 1.41.2
## # RSQLite version at creation time: 2.2.0
## # DBSCHEMAVERSION: 1.2
```

```
gene_id <- keys(txdb, keytype = "GENEID")  #Extracct the GeneIDs
#Matching the transcripts with the geneIDs
tx2gene <- AnnotationDbi::select(txdb, keys = gene_id, keytype = "GENEID", columns = "TXNAME")
```

```
## 'select()' returned 1:many mapping between keys and columns
```

Now that we have out **tx2gene** object ready, we go ahead to read the counts using tximport. We use the **DESeqDataSetFromTximport** function from DESeq2 to achive this.

```
#Dircetory for Kallisto counts
kal_dir <- "Kallisto"
files <- file.path(kal_dir, rownames(meta),"abundance.h5" )
names(files) <- rownames(meta)
files
```

```
##                          sample37                          sample38
## "Kallisto/sample37/abundance.h5" "Kallisto/sample38/abundance.h5"
##                          sample39                          sample40
## "Kallisto/sample39/abundance.h5" "Kallisto/sample40/abundance.h5"
##                          sample41                          sample42
## "Kallisto/sample41/abundance.h5" "Kallisto/sample42/abundance.h5"
```

```
#Reading the files for the different samples
txi <- tximport(files, type="kallisto",tx2gene = tx2gene, txOut = T)
```

```
## 1 2 3 4 5 6
```

```
names(txi)
```

```
## [1] "abundance"           "counts"              "infReps"
## [4] "length"              "countsFromAbundance"
```

```
#Construct the  DESeqDataSet object
kallisto_dds <- DESeqDataSetFromTximport(txi,
                                         colData = meta,
                                         design = ~ Condition)
```

```
## using counts and average transcript lengths from tximport
```

```
kallisto_dds
```

```
## class: DESeqDataSet
## dim: 228048 6
## metadata(1): version
```

```
## assays(2): counts avgTxLength
## rownames(228048):
##   ENST00000456328.2|ENSG00000223972.5|OTTHUMG00000000961.2|OTTHUMT00000362751.1|DDX11L1-202|DDX11L1|
##   ENST00000450305.2|ENSG00000223972.5|OTTHUMG00000000961.2|OTTHUMT00000002844.2|DDX11L1-201|DDX11L1|
##   ...
##   ENST00000387460.2|ENSG00000210195.2|-|-|MT-TT-201|MT-TT|66|Mt_tRNA|
##   ENST00000387461.2|ENSG00000210196.2|-|-|MT-TP-201|MT-TP|68|Mt_tRNA|
## rowData names(0):
## colnames(6): sample37 sample38 ... sample41 sample42
## colData names(1): Condition
```

In our analysis, we shall focus more on the counts produced by HISAT2 and featurecounts as our primary results for the differential expression analysis. If one is interested in usisng the output from kallisto as the primary results, then simply rename the names of the kallisto object to that being used here (ie that generated by the hisat2)

Differential Expression

We shall pass our DESeqDataset object through the **DESeq()** function which will first normalize our counts, then performs an estimation of size factors (which control for differences in the library size of the sequencing experiments) and finally the estimation of dispersion for each gene by fitting the object to a generalized linear model,glm and producing the corresponding pvalues at a 90% confidence.
Normalization of counts is such an import step because of 2 main reasons;

- Some genes are considerably larger than others hence we expect more reads to align onto these genes compared to smaller genes, something that is no where associated to the gene's expression profile.
- Some samples may be deeply sequenced as compared to others, hence we expect such samples to have more reads mapping to genes than others.

We then use the **`result()`** function to extract out the results from the DESeq() output.

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

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

```
#Extracting the results from the formed dds object
res <- results(dds)
head(res, 10)
```

```
## log2 fold change (MLE): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 10 rows and 6 columns
##                  baseMean log2FoldChange     lfcSE       stat    pvalue
##                 <numeric>      <numeric> <numeric>  <numeric> <numeric>
## ENSG00000223972  0.000000             NA        NA         NA        NA
## ENSG00000227232 71.877192      -0.617388  0.547883 -1.1268609 0.2598013
## ENSG00000278267  9.328531      -2.539808  1.067946 -2.3782174 0.0173966
## ENSG00000243485  0.152211       0.780782  4.080473  0.1913459 0.8482546
## ENSG00000284332  0.000000             NA        NA         NA        NA
## ENSG00000237613  0.333464       1.771850  4.030219  0.4396412 0.6601970
## ENSG00000268020  0.000000             NA        NA         NA        NA
## ENSG00000240361  0.000000             NA        NA         NA        NA
## ENSG00000186092  0.000000             NA        NA         NA        NA
## ENSG00000238009  1.382485      -0.177357  3.283507 -0.0540145 0.9569236
##                      padj
##                 <numeric>
## ENSG00000223972        NA
## ENSG00000227232  0.574020
## ENSG00000278267  0.137254
## ENSG00000243485        NA
## ENSG00000284332        NA
## ENSG00000237613        NA
## ENSG00000268020        NA
## ENSG00000240361        NA
## ENSG00000186092        NA
## ENSG00000238009        NA
```

Understanding the results output in the res object.

The res object generated has 6 columns with the rownames being the individual genes.

- **baseMean**; This is the avarage of the counts for a particular gene from the different samples.
- **log2FoldChange**; this is basically the log-ratio of a gene's or transcript's expression values in 2 different conditions.
- **lfcSE**; this is just the standard error in determining the DE signifince level of a particular gene.
- **stat**; is the Wald Statistic
- **pvalue**; this is the raw p-value indicating how significant a gene is differentially expressed.
- **padj**; this is the adjusted pvalue after applying the Benjamini-Hochberg (BH) adjustment for multiple comparisons.

And for values assigned as NAs are based on the criteria below;

- If baseMean for all samples is zero, the p values and lfc will be assigned NA
- If a row contains any sample with extreme count outlier, the p-value and padj are set to NA
- If a row is filetered due to low counts then only the padj is set to NA

```
#Generate a summary of res object
summary(res)
```

```
##
## out of 38260 with nonzero total read count
```

```
## adjusted p-value < 0.1
## LFC > 0 (up)       : 1497, 3.9%
## LFC < 0 (down)     : 1004, 2.6%
## outliers [1]       : 287, 0.75%
## low counts [2]     : 12931, 34%
## (mean count < 3)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

The summary of res above gives us an idea of the possible number of differentially expressed genes (both upregulated and downregulated), as well possible counts of outliers in the our data at 0.1 significance level by default. From this we can see that we have **1497 upregulated genes** (constituting about 3.9%) and **1004 downregulated genes** (constituting about 2.6%) whereas 287 (0.75) and 12931 (34%) of the counts are considered as outliers and low counts respectively.

Data Quaity assessment with plots

A quality assessment of our the results is one of the most crucial steps reccommended to be done before any further analysis is performed. This typically informs us how good or bad our data is and how best it is suited for the analysis to be performed and whether any results need to filtered out.

However with DESeq2, this is not so much of a major concern since the quality control is already implemented in the package function DESeq() that caters for any low counts and outliers.

Here we shall simply use plots to give us an idea of the data we are planning to use for our downstream analysis.
The 2 common plots to achieve this include the heatmap and PCA.

For generating heatmaps and more quality plots it is reccommended to first transform our data.

Data Transformation

Testing for DE operates on raw counts, however for visualizations and clusterings, its better to work with transformed count data. There are 2 common waays of transformation; which produce transformed data on the log2 scale which has been normalized with respect to library size or other normalization factors.

- Variance Stabilizing Transformations (VST) and
- regularized logarithm, rlog

The goal of transformation is to *eliminate any variance arising when the mean is low.*

One common argument used during transformation is **blind** which tells whether transformation should be blind to sample information specified in the the design (ie in an unbiased manner), hence using only the intercept. However this is not the best choice especially if the counts difference are attributed to the design and if one is to use the transformed data for downstream analysis. If `blind = FALSE` is set, we take into account already estimated dispersion due to design as we are transforming the data, and make the process much faster.

In Comparison: VST runs faster than rlog. If the library size of the samples and therefore their size factors vary widely, the rlog transformation is a better option than VST. Both options produce log2 scale data which has been normalized by the DESeq2 method with respect to library size.

```
#Count data Transformations
vsd <- vst(dds, blind=FALSE)  #computing for Variance Stabilizing Transformation
rld <- rlog(dds, blind=FALSE) #Computing for regularised logarithm

#A look at the generated matrices in contrast to the original
#Original count matrix
head(countdata)
```

```
##                 sample37 sample38 sample39 sample40 sample41 sample42
## ENSG00000223972        0        0        0        0        0        0
## ENSG00000227232       52       48      187       56       59       69
## ENSG00000278267        7       11       36        2        3        4
## ENSG00000243485        0        0        0        1        0        0
## ENSG00000284332        0        0        0        0        0        0
## ENSG00000237613        0        0        0        1        1        0
```

```
#Count matrix for vsd
head(assay(vsd)) #Assay extracts a matrix of normalized counts
```

```
##                 sample37 sample38 sample39 sample40 sample41 sample42
## ENSG00000223972 5.281749 5.281749 5.281749 5.281749 5.281749 5.281749
## ENSG00000227232 7.177809 6.902050 7.635498 6.856577 7.026329 6.915044
## ENSG00000278267 6.020474 6.088279 6.404361 5.593753 5.698105 5.694927
## ENSG00000243485 5.281749 5.281749 5.281749 5.502584 5.281749 5.281749
## ENSG00000284332 5.281749 5.281749 5.281749 5.281749 5.281749 5.281749
## ENSG00000237613 5.281749 5.281749 5.281749 5.502584 5.522688 5.281749
```

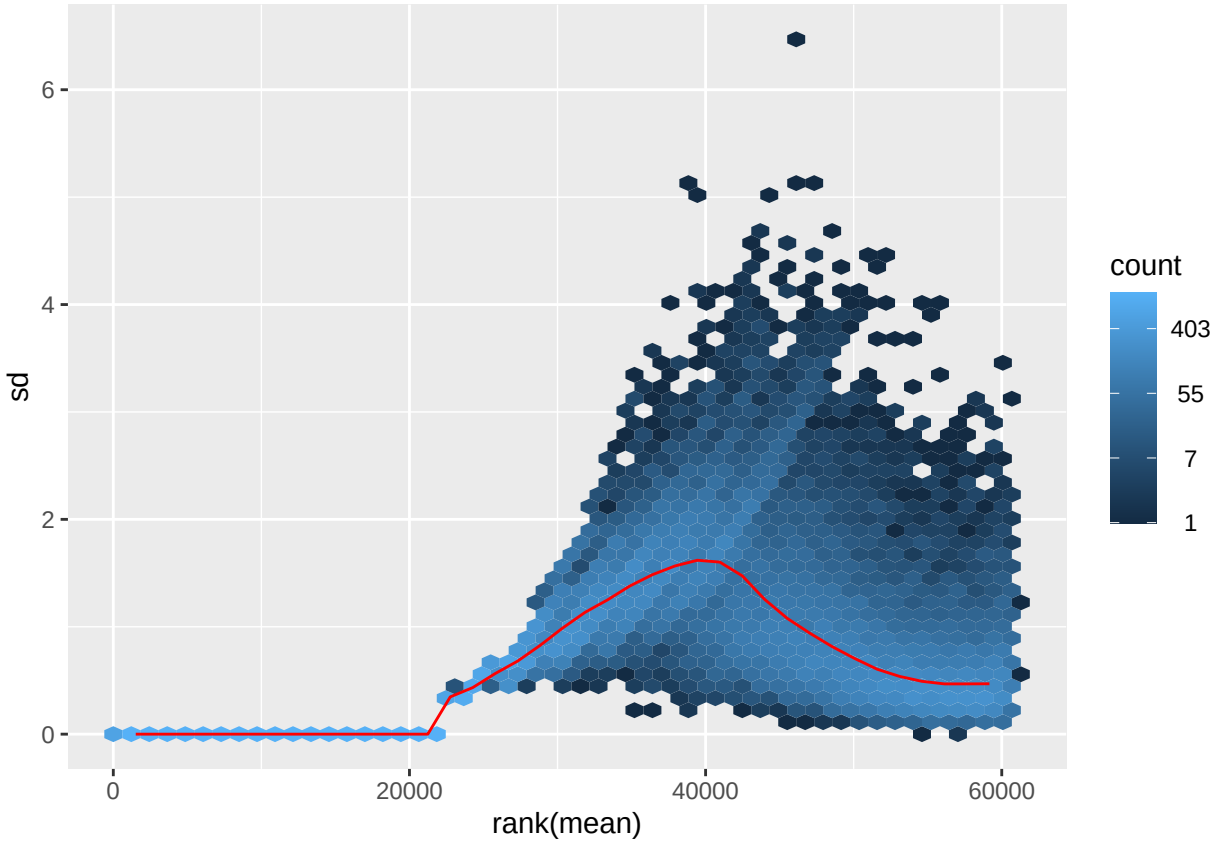
```
#Count matrix produced by rlog
head(assay(rld))
```

```
##                  sample37  sample38  sample39  sample40  sample41  sample42
## ENSG00000223972  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## ENSG00000227232  6.210644  5.891107  6.701883  5.835198  6.038031  5.905750
## ENSG00000278267  3.095538  3.190424  3.618631  2.546669  2.670309  2.660245
## ENSG00000243485 -1.979000 -1.982334 -1.988459 -1.960492 -1.982850 -1.986563
## ENSG00000284332  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
## ENSG00000237613 -1.503514 -1.510533 -1.523192 -1.464545 -1.460042 -1.519438
```

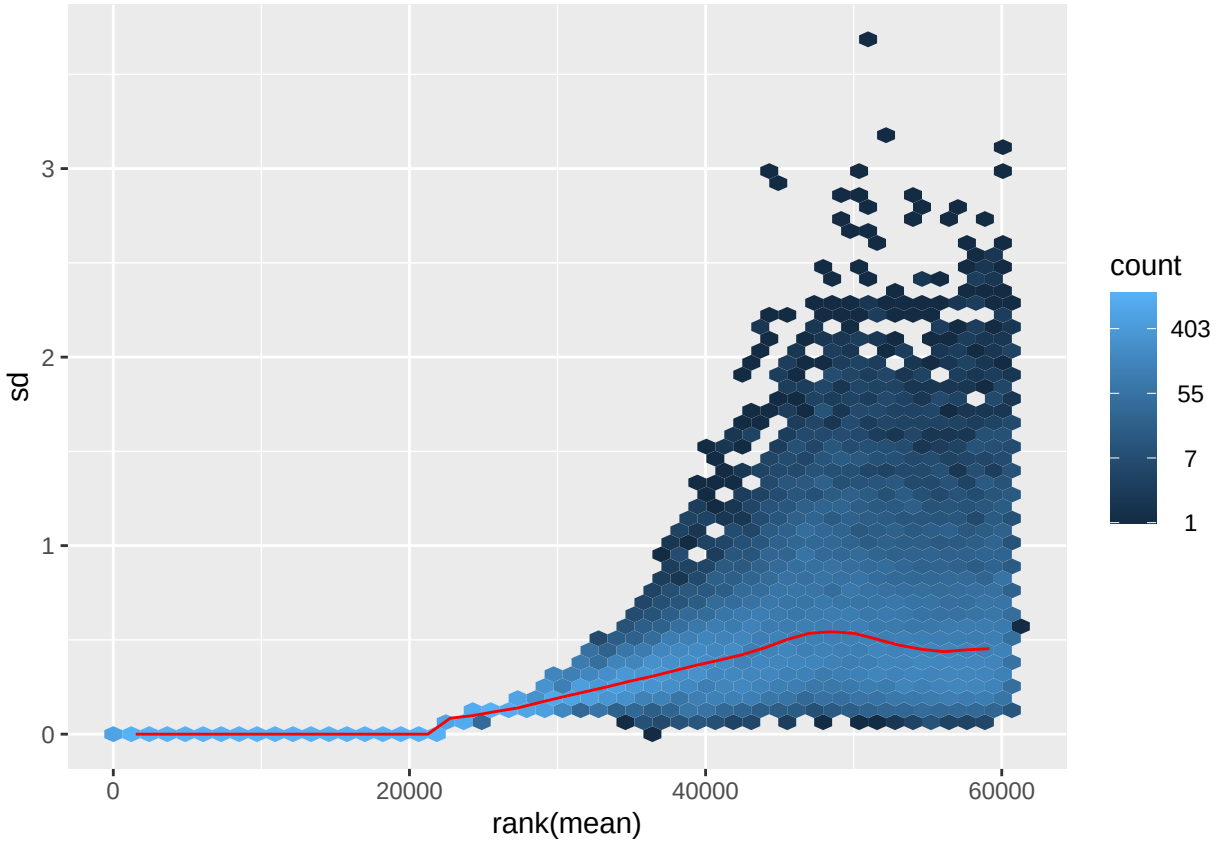
Effects of transformation on the variance We will use the dispersion plot to explore the effects of transformation on the variance in our data. This plots the standard deviation of the transformed data across samples, against the mean, using the shifted logarithm transformation, the regularized log transformation and the variance stabilizing transformation.

What we expect: The shifted logarithm has elevated standard deviation in the lower count range, and the regularized log to a lesser extent, while for the variance stabilized data the standard deviation is roughly constant along the whole dynamic range.

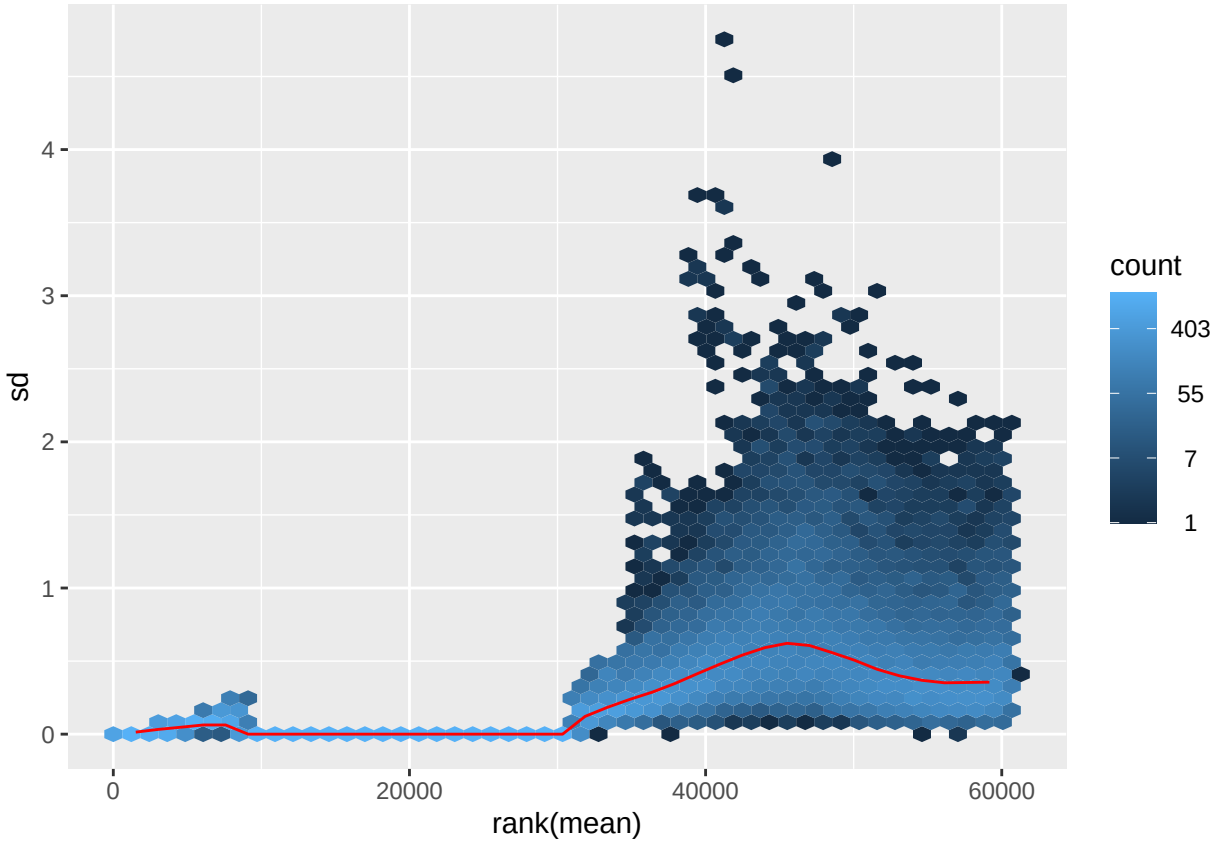
```
library(vsn)
#Dispersion Plots

#for the untransformed dds
ntd <- normTransform(dds)
meanSdPlot(assay(ntd))
```




```
#for vst
meanSdPlot(assay(vsd))
```




```
#For rlog
meanSdPlot(assay(rld))
```



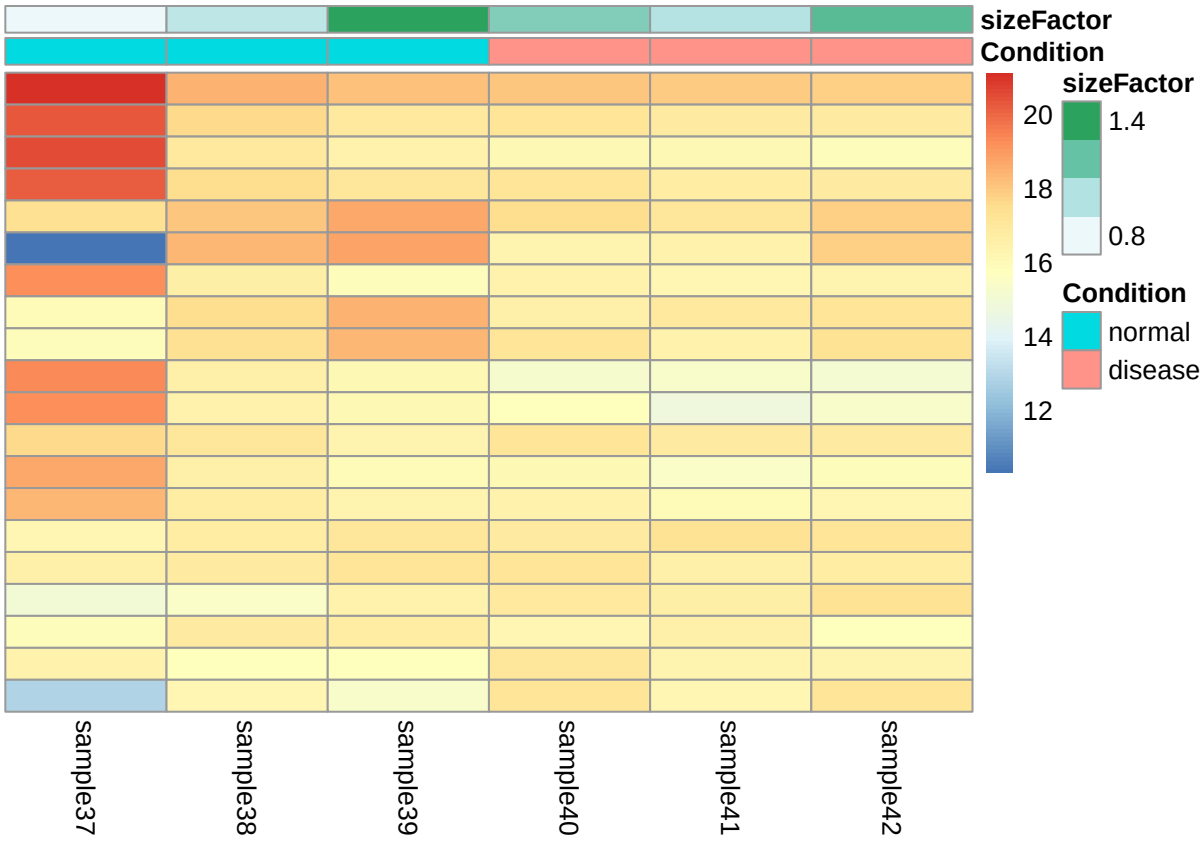
We generally do not see much change between rlog and vst in this case, however there is a noticiable improvement in the the variance as compared to the untransformed data, shown by a fairly lower fitted line.

We will use the vst transformed data obtained from above for the plots, since its the most reccomended approach.

1. Heatmap of Count matrix.

We will first use the heatmap to explore the quality for the count matrix.

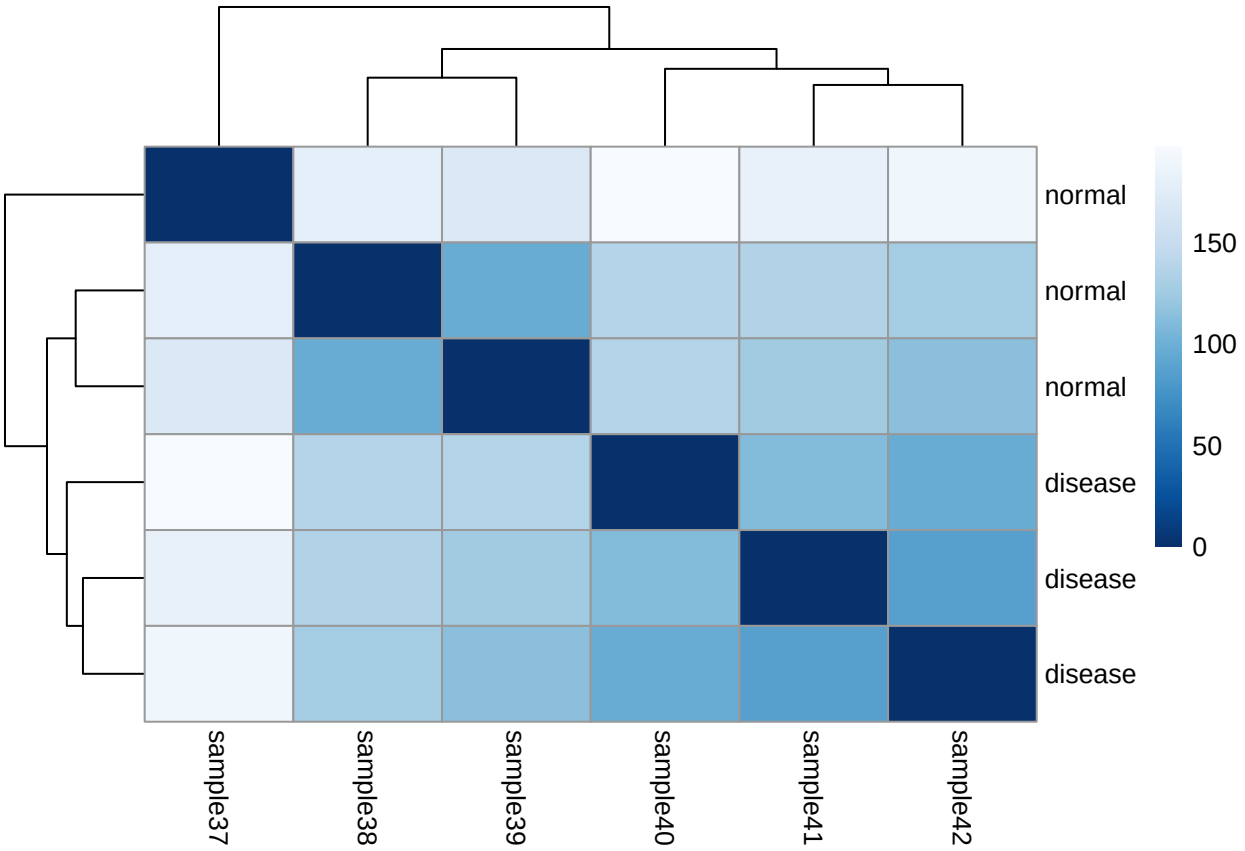
```
select <- order(rowMeans(counts(dds,normalized=TRUE)), decreasing=TRUE)[1:20]
df <- as.data.frame(colData(dds))
pheatmap(assay(vsd)[select,], cluster_rows=FALSE, show_rownames=FALSE,
cluster_cols=FALSE, annotation_col=df)
```



2. Heatmap of Sample-Sample distances

This heatmap gives us an overview of the similarities and dissimilarities between the samples.

```
#Calculate sample-2-sample distances
sampleDists <- dist(t(assay(vsd)))

sampleDistMatrix <- as.matrix(sampleDists) #Converting the dist object to matrix
rownames(sampleDistMatrix) <- vsd$Condition
colnames(sampleDistMatrix) <- rownames(meta)

#Defining the colors
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)

pheatmap(sampleDistMatrix,
clustering_distance_rows=sampleDists,
clustering_distance_cols=sampleDists,
col=colors)
```



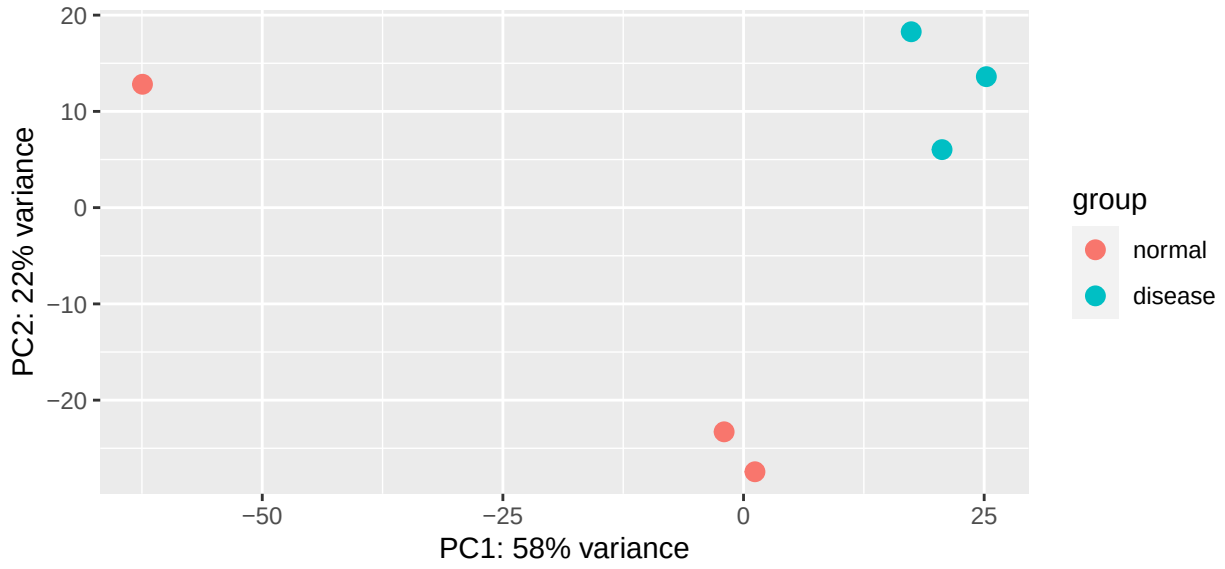
From this matrix we can generally observe a larger simarity between intra-sample groups eg more similarity between samples 40-41 and samples 37-39, and we can also note major dissimilarities between disease group compared to normal samples shown by much lighter colors.

However we do note one unexpected difference here. Sample37 which belongs to the normal samples seems to be very different from the other samples both in the same category and different category as well, which is one thing we can't have a conclusive answer for yet as to wether the changes were generated after the experiment or they are phsyiological changes.

3. Principal component plot of the samples

This plot is similar to the distane matrix above which will basically cluster the samples basing on their properties.
This type of plot is useful for visualizing the overall effect of experimental covariates, relatedness and batch effects.

```
plotPCA(vsd, intgroup="Condition")
```

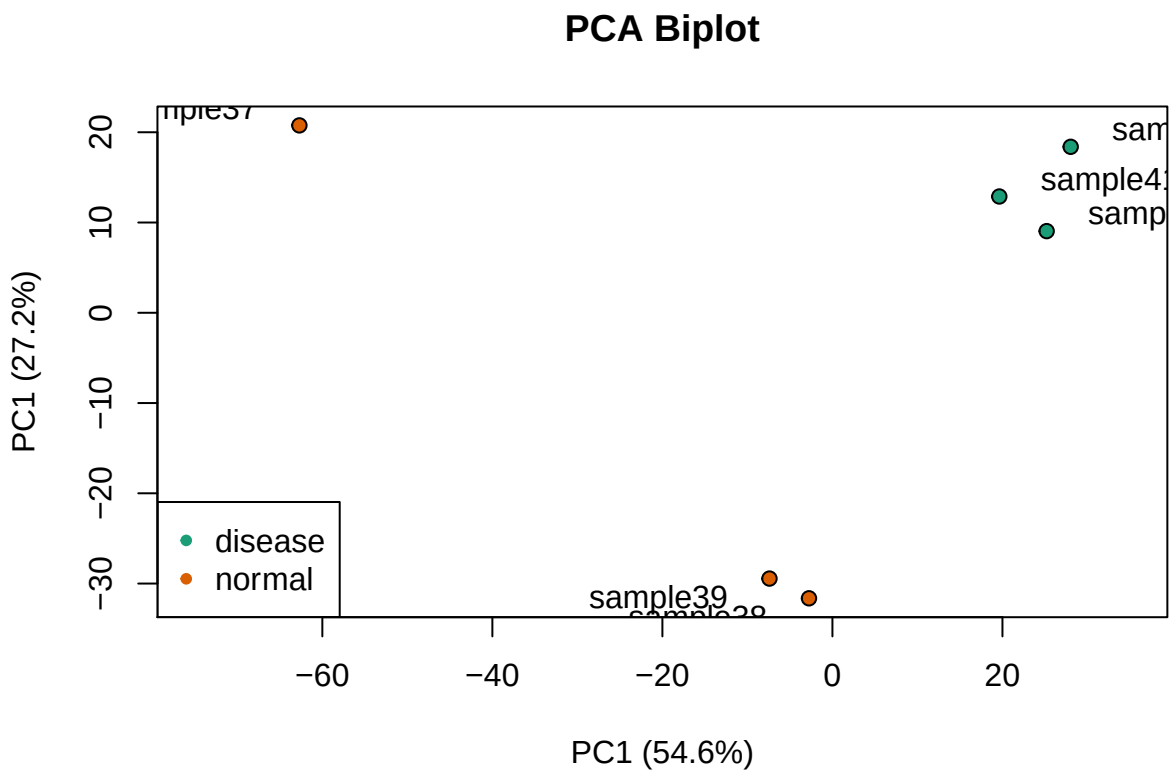


From this plot we generally see disease samples clustering together, and 2 of the normal samples clustering together apart from one.

We will redraw the plots to include their sample labels to be able to tell which sample is generally different among the normal.

```
#Including samples in the pca plot
#PCA Analysis
mycols <- brewer.pal(8, "Dark2")[1:length(unique(meta$Condition))]

rld_pca <- function (rld, intgroup = "Condition", ntop = 500, colors=NULL, legendpos="bottomleft", main=

require(RColorBrewer)
rv = rowVars(assay(rld))
select = order(rv, decreasing = TRUE)[seq_len(min(ntop, length(rv)))]
pca = prcomp(t(assay(rld)[select, ]))
fac = factor(apply(as.data.frame(colData(rld)[, intgroup, drop = FALSE]), 1, paste, collapse = " : "))
if (is.null(colors)) {
if (nlevels(fac) >= 3) {
colors = brewer.pal(nlevels(fac), "Paired")
} else {
colors = c("black", "red")
}
}

pc1var <- round(summary(pca)$importance[2,1]*100, digits=1)
pc2var <- round(summary(pca)$importance[2,2]*100, digits=1)
```

```
pc1lab <- paste0("PC1 (",as.character(pc1var),"%)")
pc2lab <- paste0("PC1 (",as.character(pc2var),"%)")
plot(PC2~PC1, data=as.data.frame(pca$x), bg=colors[fac], pch=21, xlab=pc1lab, ylab=pc2lab, main=main, .
with(as.data.frame(pca$x), textxy(PC1, PC2, labs=rownames(as.data.frame(pca$x)), cex=textcx))
legend(legendpos, legend=levels(fac), col=colors, pch=20)
}
rld_pca(rld, colors=mycols, intgroup="Condition", xlim=c(-75, 35))
```



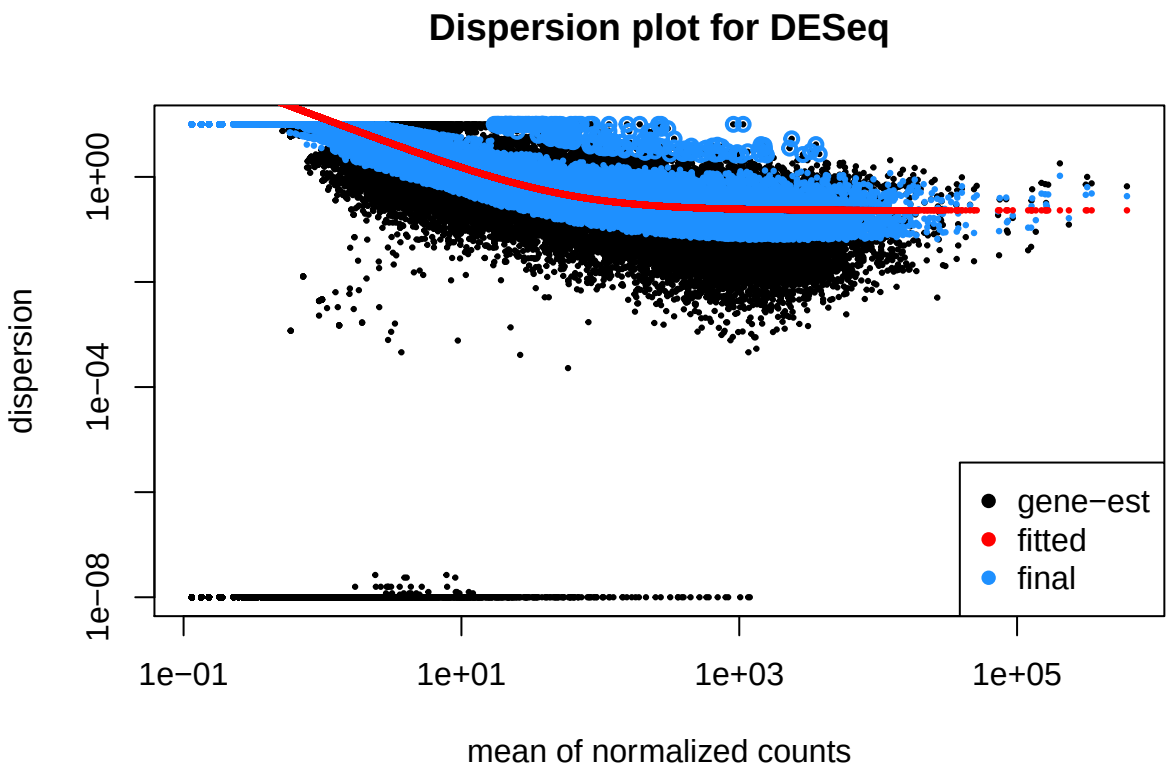
After adding labels, we can clearly tell that indeed it was sample37 that was clustering differently compared to the rest.

4. Dispersion plot

This plot generally show how gene expression data differs across samples in the same treatment group.

- The black dots are the dispersion estimates for each gene separately
- The red line is the fitted trend which shows the dispersion's dependence on the mean.
- The blue dots are genes that have been shrunk/fitted towards the red line.
- The outer blue circles with black dots inside are considered outliers and therefore not shrunk towards the fitted line

```
#Dispersion plot
plotDispEsts(dds, main="Dispersion plot for DESeq")
```



Exploring the results with plots

MA plots

This plot is a quick visual for genes that are upregulated as well those downregulated, possible outliers, and those condidered not significant. Its a better practice to plot the shrinked values for this this plot as compared to the initial results.

Log fold change shrinkage

To generate plots and ranking of genes, its better we first perform a Shrinkage of effect size (ie the LFC estimates) that helps in shrinking any low counts/ low effect size that are close to zero towards the zero line. This helps in cleaning out the plots (very useful for the MA plot) for easier interpretation *without even the need of filtering out low counts in the results.*

> There are generally 3 methods for performing effect size shrinkage ie using the **`apeglm`** which is usally the default method, **`ashr`** and **`normal`** which was the initial method implemented by DESeq2. Among this the best recommended is the apeglm and most discouraged for use is normal.

For exploration purposes we will test all of the above methods on our results to better understand the difference it makes (if any).

One thing we need to provide to the `lfcshrink()` function on top of the method to use is the coefficient obtained from the `resultsNames()` which basically tells the shrink function what Condition is to be compared against what.

```
#Obtain the coefficient to use for shrinkage
resultsNames(dds)
```

```
## [1] "Intercept"                   "Condition_disease_vs_normal"
```

In our case its `Condition_disease_vs_normal` or the second object in the output which basically tells us that we are comparing diseased samples against normal samples.

```
#Computing for size shrikage using apeglm
resLFC <- lfcShrink(dds, coef="Condition_disease_vs_normal", type="apeglm")
```

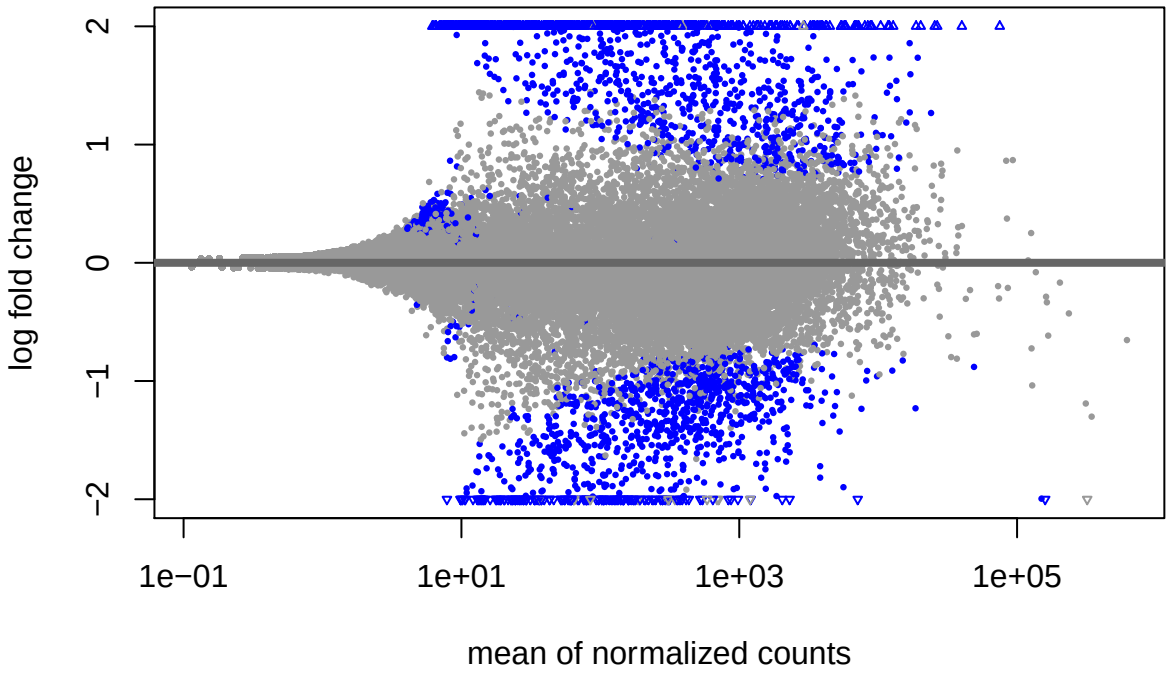
```
## using 'apeglm' for LFC shrinkage. If used in published research, please cite:
##     Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distributions for
##     sequence count data: removing the noise and preserving large differences.
##     Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895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

```
head(resLFC)
```

```
## log2 fold change (MAP): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 6 rows and 5 columns
##                   baseMean log2FoldChange     lfcSE    pvalue      padj
##                  <numeric>      <numeric> <numeric> <numeric> <numeric>
## ENSG00000223972  0.000000             NA        NA        NA        NA
## ENSG00000227232 71.877192     -0.3271584  0.433315 0.2598013  0.574020
## ENSG00000278267  9.328531     -1.1171275  1.567943 0.0173966  0.137254
## ENSG00000243485  0.152211      0.0341144  0.535381 0.8482546        NA
## ENSG00000284332  0.000000             NA        NA        NA        NA
## ENSG00000237613  0.333464      0.0441123  0.536771 0.6601970        NA
```

Lets now generate the MA plot for the shrinked value.

```
#MA plot for shrinked values
plotMA(resLFC, ylim=c(-2,2))
```

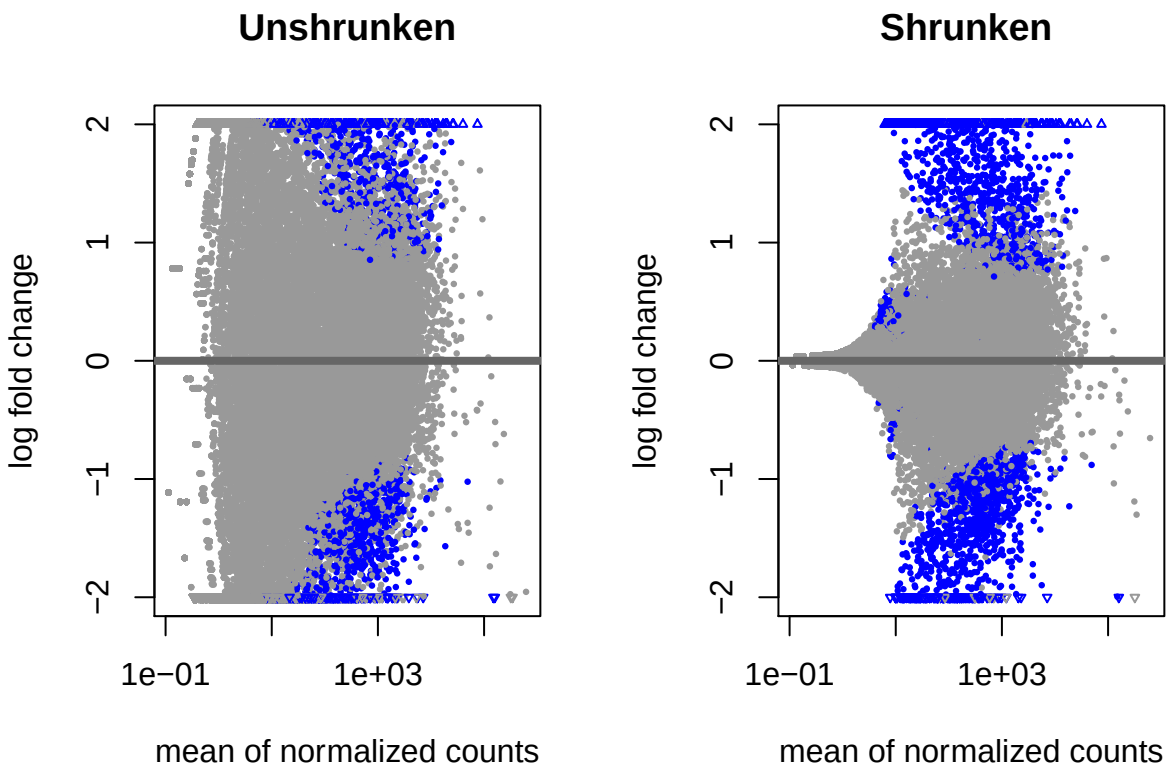


This plot basically has 3 features of our interest explained below. Each dot represented here is a gene.

1. The `grey` dots represent the genes whose LFC is considered not signicant to our condition.
2. The `blue` dots represent genes whose LFC is considered significant. Dots above the line represent upregulated genes, and dots below the line represent downregulated genes
3. The genes represented with triangles either coloured or not are considered as outliers basing on the limit we have set (in this case its from -2 to 2).

We shall now visualize the MA plot with first the initial res object where we hadn't performed the size shrinkage and compare it to when we have performed shrinkage using apeglm.

```
# comparison of MA plot for unshrinked values and shrinked valued
par(mfrow=c(1,2))
plotMA(res, ylim=c(-2,2), main="Unshrunken")  #Unshrinked values
plotMA(resLFC, ylim=c(-2,2), main="Shrunken ")  #Shrinked values
```



One quick observation we can give from the plots above is that we have alot of unsignificants genes being represented in the unshrunken plot as compared to the shrunken. This is the exact reason why we perform size shrinkage, ie clear the plots and make it more easily interpretable.

Compare the shrinkage methods

We will now compute the size shrinkage using the other 2 methods and determine the difference it would make.

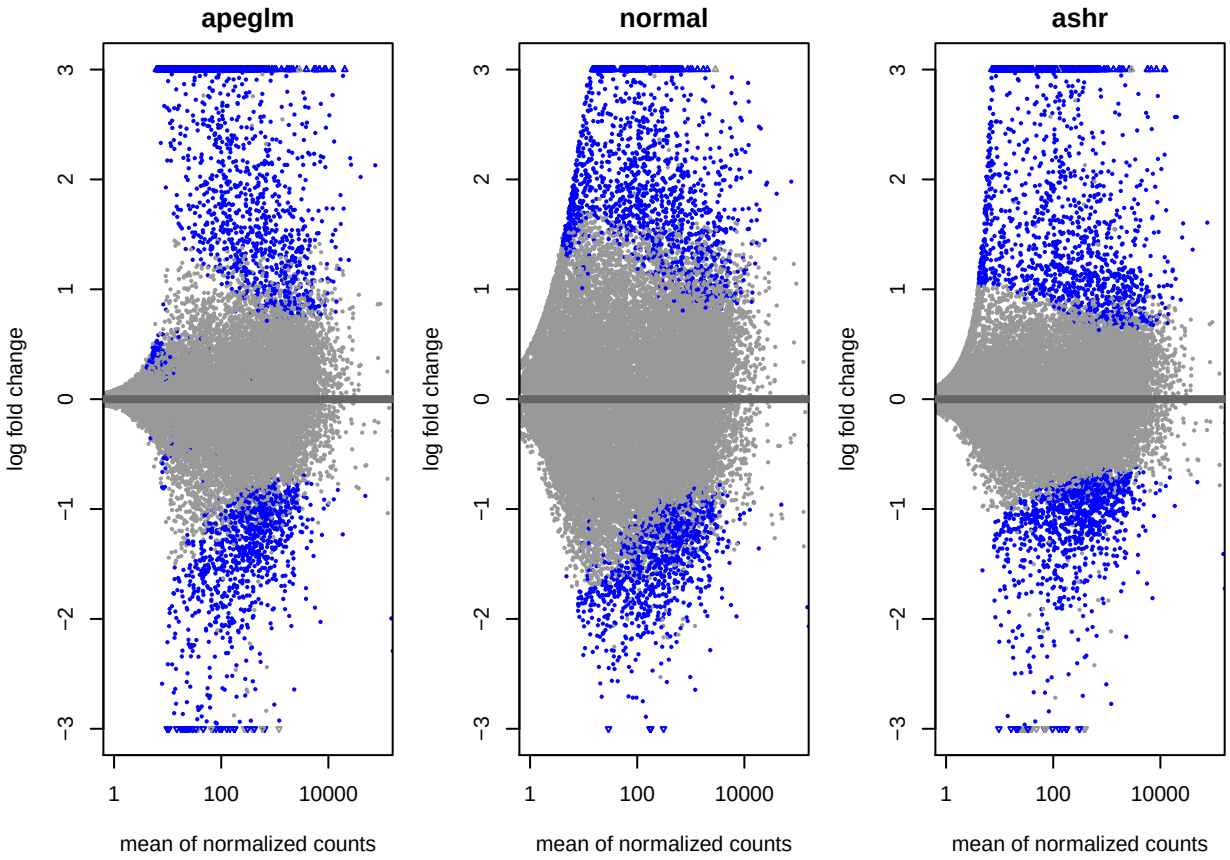
```
#Alternative shrinkage estimators

#using the ashr method
resAsh <- lfcShrink(dds, coef=2, type="ashr")
```

```
## using 'ashr' for LFC shrinkage. If used in published research, please cite:
##     Stephens, M. (2016) False discovery rates: a new deal. Biostatistics, 18:2.
##     https://doi.org/10.1093/biostatistics/kxw041
```

```
#Using the normal method
resNorm <- lfcShrink(dds, coef=2, type="normal")
```

```
## using 'normal' for LFC shrinkage, the Normal prior from Love et al (2014).
##
## Note that type='apeglm' and type='ashr' have shown to have less bias than type='normal'.
## See ?lfcShrink for more details on shrinkage type, and the DESeq2 vignette.
## Reference: https://doi.org/10.1093/bioinformatics/bty895
```

```
#Generating MA plots comparing the 3 shrinkage estimators
#mar sets the margin sizes in the following order: bottom, left, top, right
par(mfrow=c(1,3), mar=c(4,4,2,1))
xlim <- c(1,1e5); ylim <- c(-3,3)
plotMA(resLFC, xlim=xlim, ylim=ylim, main="apeglm")
plotMA(resNorm, xlim=xlim, ylim=ylim, main="normal")
plotMA(resAsh, xlim=xlim, ylim=ylim, main="ashr")
```

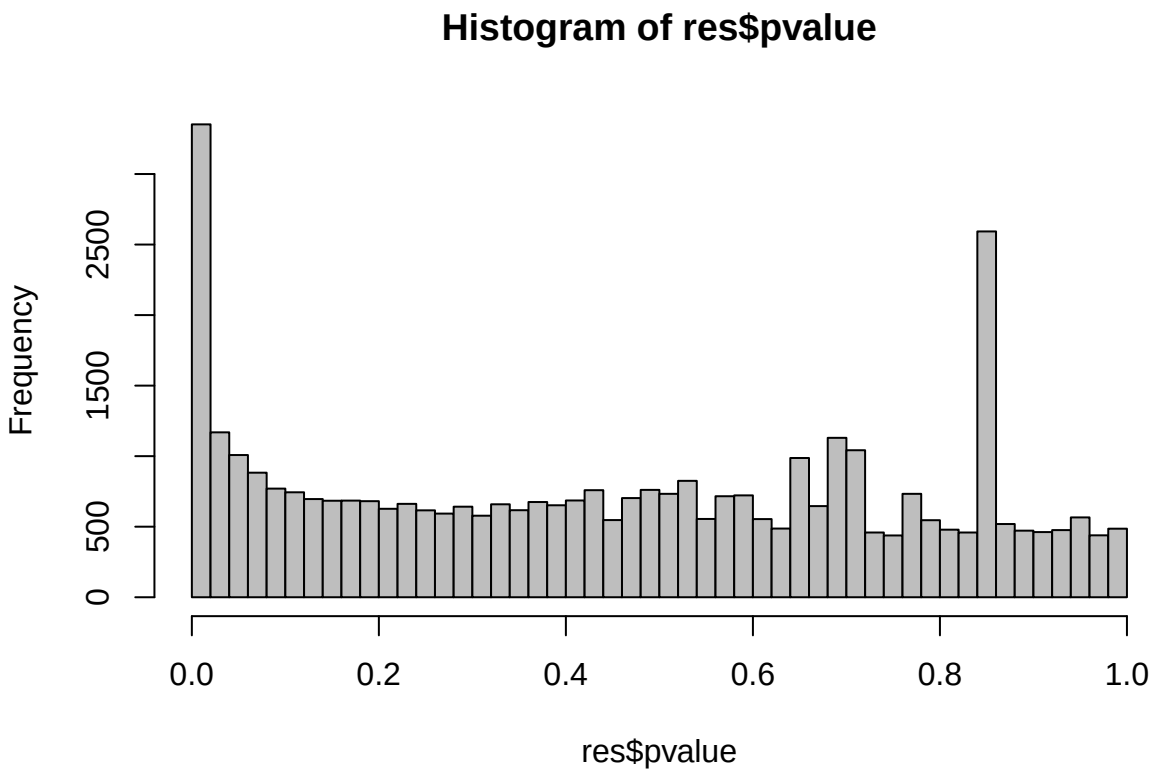


Among these plots, the plot for shrinakge using the normal estimate contains more unsignificant genes followed by ashr which would probably explain why apeglm is a better prefered estimator.

Exploring Significant Results

Lets first make a histogram plot of p-value frequency.

```
#Exploring p-value frequency
hist(res$pvalue, breaks=50, col="grey")
```



We can see from this that we have some good number of genes whose pvalue is significant (ie fall below 0.1), however most of the pvalues for the genes is not considered significant.

From our res object we will go head to only subset out the significant results. We will consider a gene's expression significant if its pvalue is below 0.05, ie 95% confidence.

```
#Reorder res based on pvalues
resOrdered <- res[order(res$pvalue),]
head(resOrdered)
```

```
## log2 fold change (MLE): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 6 rows and 6 columns
##                   baseMean log2FoldChange     lfcSE      stat      pvalue
##                  <numeric>      <numeric> <numeric> <numeric>   <numeric>
## ENSG00000039537   431.823        6.92551  0.731458   9.46809 2.85007e-21
## ENSG00000124237   241.613        7.98886  0.855855   9.33435 1.01608e-20
## ENSG00000160401   507.498        6.03790  0.658602   9.16775 4.82988e-20
## ENSG00000007908  1332.119        6.21421  0.693271   8.96362 3.14198e-19
## ENSG00000188817   409.116        5.82887  0.667047   8.73832 2.36594e-18
## ENSG00000168658   330.530       11.73927  1.400477   8.38234 5.18865e-17
##                       padj
##                  <numeric>
## ENSG00000039537 7.13715e-17
## ENSG00000124237 1.27223e-16
## ENSG00000160401 4.03166e-16
## ENSG00000007908 1.96703e-15
```

```
## ENSG00000188817 1.18496e-14
## ENSG00000168658 1.78339e-13
```

```
#Only subsettting genes with Significant padj value
resSig <- subset(resOrdered, padj < 0.05)

#Sum of significant genes
nrow(resSig)
```

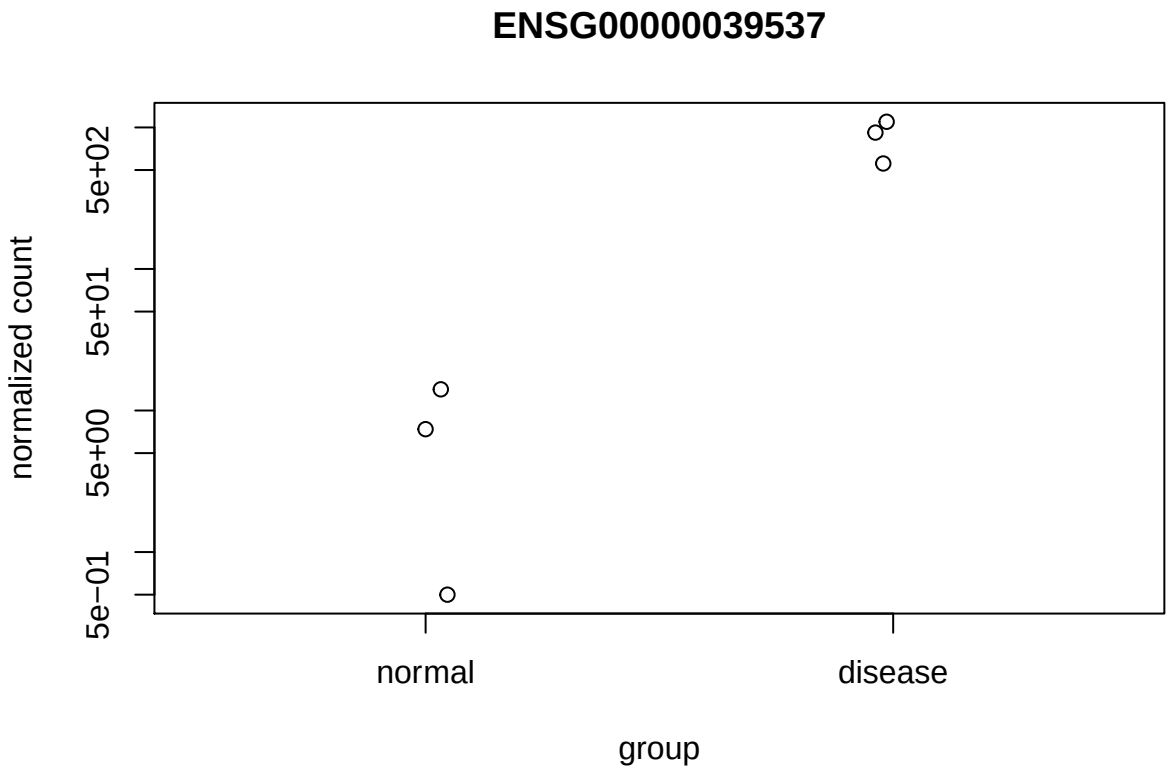
```
## [1] 1671
```

Plot Counts

This plots the normalized counts for each sample in the particular gene

Lets start by plotting the counts of the most significant gene ie the one with the lowest pvalue.

```
#For the gene with the minimum padj
plotCounts(dds, gene=which.min(res$padj), intgroup="Condition")
```



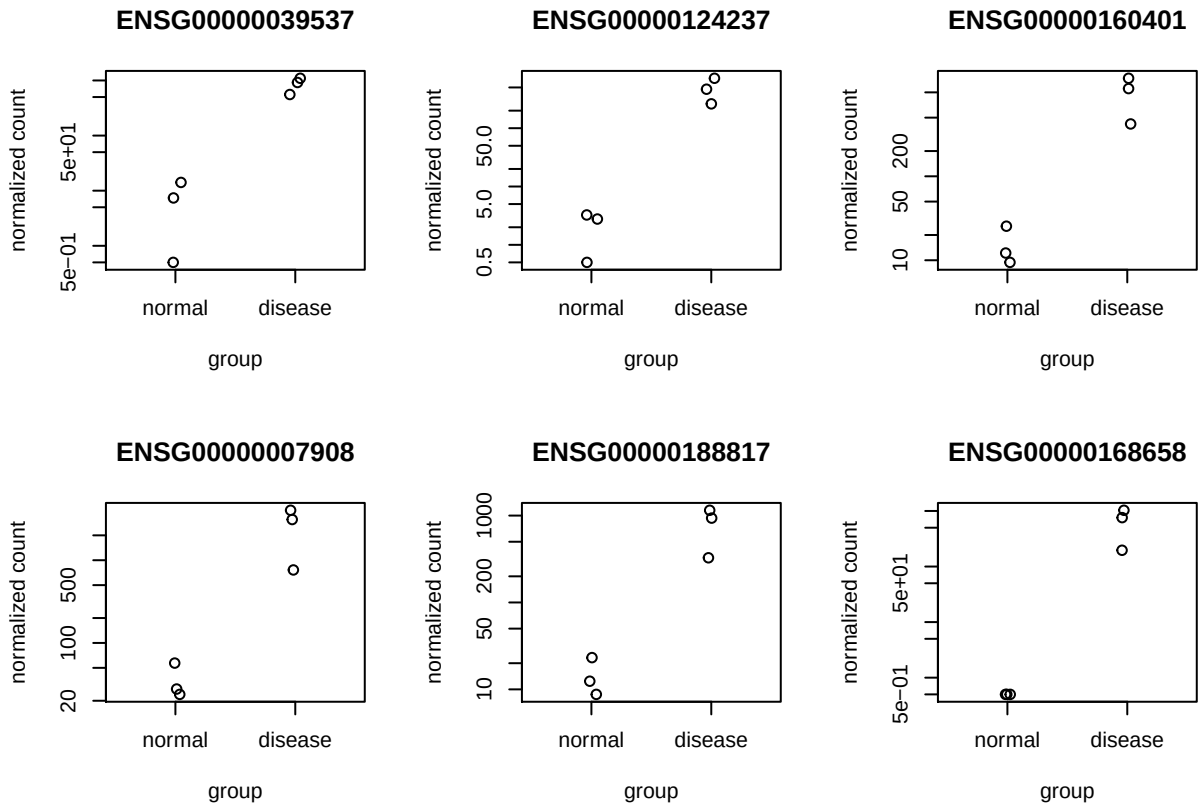
From this we can see that normal samples are having a fairly equal count for the gene as well as the disease samples, however there is a clear cut difference between the counts for this gene in the 2 categories.

Our interests will now be generating a count for the top 6 significant genes, the lowest 6 significant genes, as well as the unsignificant genes for better contrast.

```r
#Top 6 significant genes
head(resSig)
```

```
## log2 fold change (MLE): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 6 rows and 6 columns
##                  baseMean log2FoldChange     lfcSE      stat      pvalue
##                 <numeric>      <numeric> <numeric> <numeric>   <numeric>
## ENSG00000039537   431.823        6.92551  0.731458   9.46809 2.85007e-21
## ENSG00000124237   241.613        7.98886  0.855855   9.33435 1.01608e-20
## ENSG00000160401   507.498        6.03790  0.658602   9.16775 4.82988e-20
## ENSG00000007908  1332.119        6.21421  0.693271   8.96362 3.14198e-19
## ENSG00000188817   409.116        5.82887  0.667047   8.73832 2.36594e-18
## ENSG00000168658   330.530       11.73927  1.400477   8.38234 5.18865e-17
##                       padj
##                  <numeric>
## ENSG00000039537 7.13715e-17
## ENSG00000124237 1.27223e-16
## ENSG00000160401 4.03166e-16
## ENSG00000007908 1.96703e-15
## ENSG00000188817 1.18496e-14
## ENSG00000168658 1.78339e-13
```

```r
#Plot counts for the top 6
par(mfrow=c(2,3))
plotCounts(dds, gene="ENSG00000039537", intgroup="Condition")
plotCounts(dds, gene="ENSG00000124237", intgroup="Condition")
plotCounts(dds, gene="ENSG00000160401", intgroup="Condition")
plotCounts(dds, gene="ENSG00000007908", intgroup="Condition")
plotCounts(dds, gene="ENSG00000188817", intgroup="Condition")
plotCounts(dds, gene="ENSG00000168658", intgroup="Condition")
```

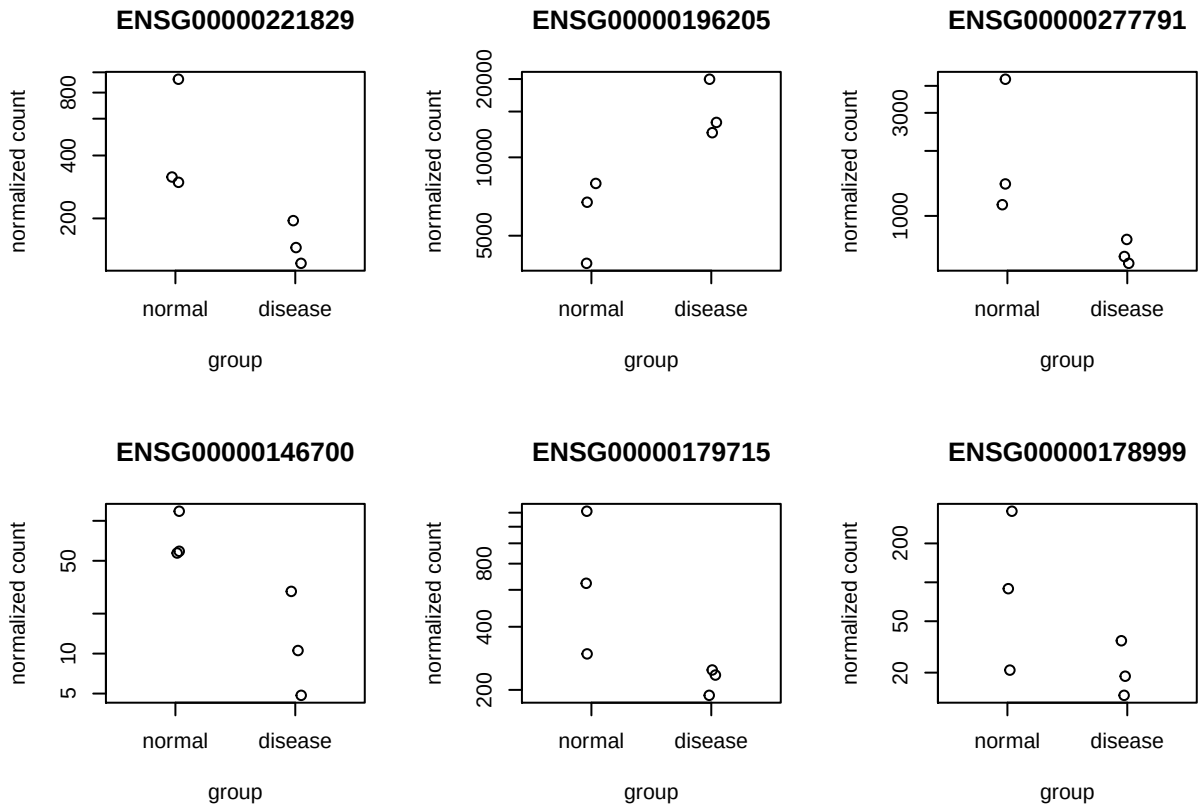



```
#Lowest 6 Significant genes
tail(resSig)
```

```
## log2 fold change (MLE): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 6 rows and 6 columns
##                   baseMean log2FoldChange     lfcSE      stat     pvalue
##                  <numeric>      <numeric> <numeric> <numeric>  <numeric>
## ENSG00000221829   333.2119       -1.73864  0.591901  -2.93738 0.00331003
## ENSG00000196205 10767.8096        1.30891  0.445730   2.93655 0.00331885
## ENSG00000277791  1475.5847       -1.75008  0.596127  -2.93575 0.00332746
## ENSG00000146700    46.0253       -2.40770  0.820261  -2.93529 0.00333235
## ENSG00000179715   505.7102       -1.81465  0.618222  -2.93527 0.00333258
## ENSG00000178999    88.2522       -2.81577  0.959294  -2.93526 0.00333273
##                       padj
##                  <numeric>
## ENSG00000221829 0.0497538
## ENSG00000196205 0.0498565
## ENSG00000277791 0.0499451
## ENSG00000146700 0.0499451
## ENSG00000179715 0.0499451
## ENSG00000178999 0.0499451
```

```
par(mfrow=c(2,3))
plotCounts(dds, gene="ENSG00000221829", intgroup="Condition")
```

```
plotCounts(dds, gene="ENSG00000196205", intgroup="Condition")
plotCounts(dds, gene="ENSG00000277791", intgroup="Condition")
plotCounts(dds, gene="ENSG00000146700", intgroup="Condition")
plotCounts(dds, gene="ENSG00000179715", intgroup="Condition")
plotCounts(dds, gene="ENSG00000178999", intgroup="Condition")
```

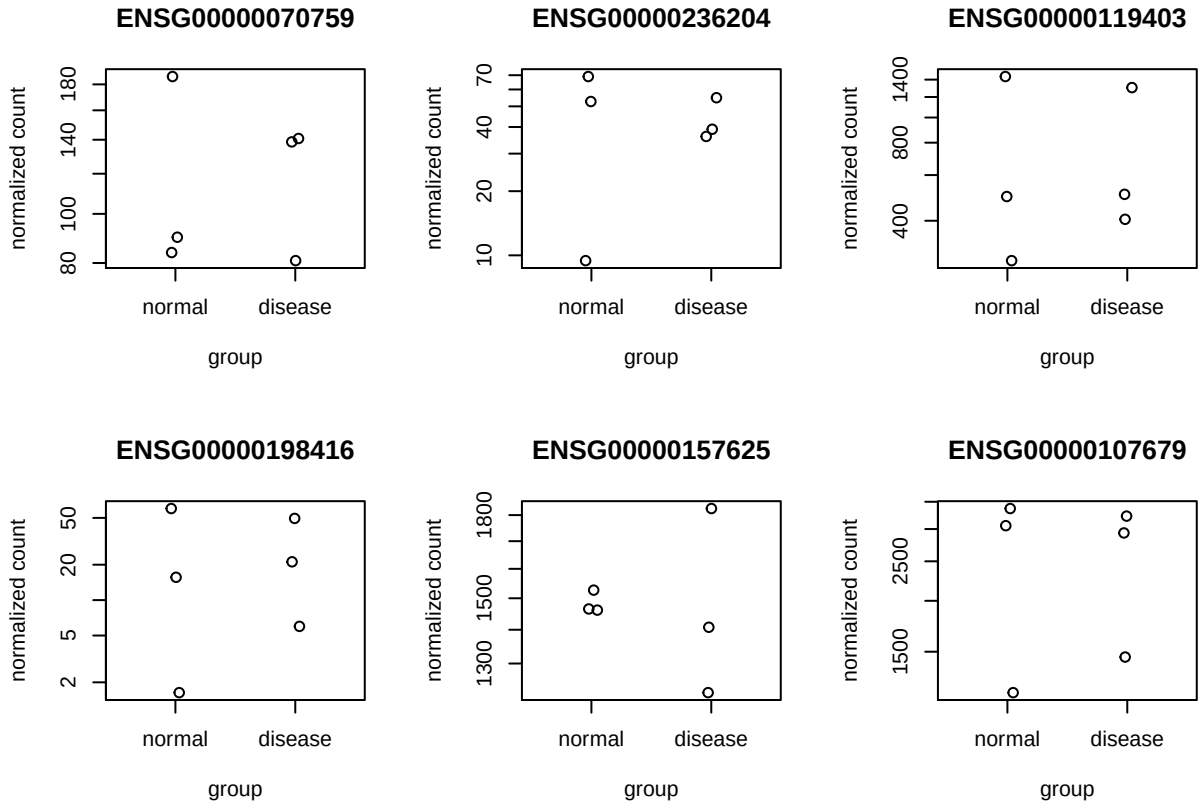



```
#Confirming the plots with that of unsignificant genes
tail(na.omit(resOrdered))
```

```
## log2 fold change (MLE): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 6 rows and 6 columns
##                  baseMean log2FoldChange     lfcSE         stat    pvalue
##                 <numeric>      <numeric> <numeric>    <numeric> <numeric>
## ENSG00000070759  119.6107   -3.45224e-04  0.552954 -0.000624328  0.999502
## ENSG00000236204   43.0285   -3.28937e-04  0.819029 -0.000401618  0.999680
## ENSG00000119403  737.9646    2.70902e-04  0.733961  0.000369096  0.999706
## ENSG00000198416   25.1621   -2.94144e-04  1.266748 -0.000232204  0.999815
## ENSG00000157625 1484.0039    4.87119e-05  0.346662  0.000140517  0.999888
## ENSG00000107679 2538.6277    3.49144e-05  0.559391  0.000062415  0.999950
##                      padj
##                 <numeric>
## ENSG00000070759  0.999701
## ENSG00000236204  0.999825
## ENSG00000119403  0.999825
```

```
## ENSG00000198416  0.999895
## ENSG00000157625  0.999928
## ENSG00000107679  0.999950
```

```
par(mfrow=c(2,3))
plotCounts(dds, gene="ENSG00000070759", intgroup="Condition")
plotCounts(dds, gene="ENSG00000236204", intgroup="Condition")
plotCounts(dds, gene="ENSG00000119403", intgroup="Condition")
plotCounts(dds, gene="ENSG00000198416", intgroup="Condition")
plotCounts(dds, gene="ENSG00000157625", intgroup="Condition")
plotCounts(dds, gene="ENSG00000107679", intgroup="Condition")
```



Gene Function Annotation

After us discovering which genes are differentially expressed in the 2 connditions, our next goal will now be to determine the function they play and determine whether the biological process played by the gene has an attachment to the condition of study.

To achieve this we will first need to attach gene IDs to the pre-existing Ensembl gene IDs. These gene IDs will later be used to access the corresponding entry information for a particular gene including its function.

We will make use of the functions obtained from the tutorial here.

Adding gene names/ IDs

We shall use the **org.Hs.eg.db** package that has information stored in as AnnotationDbi database. This contains information linking to a number of databases however our primary interest is with the Ensembl database.

```
#list of all available key types
columns(org.Hs.eg.db)
```

```
##  [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
##  [6] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
## [11] "GO"           "GOALL"        "IPI"          "MAP"          "OMIM"
## [16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
## [21] "PROSITE"      "REFSEQ"       "SYMBOL"       "UCSCKG"       "UNIGENE"
## [26] "UNIPROT"
```

```
convertIDs <- function( ids, fromKey, toKey, db, ifMultiple=c( "putNA", "useFirst" ) ) {
stopifnot( inherits( db, "AnnotationDb" ) )
ifMultiple <- match.arg( ifMultiple )
suppressWarnings( selRes <- AnnotationDbi::select(
db, keys=ids, keytype=fromKey, columns=c(fromKey,toKey) ) )
if( ifMultiple == "putNA" ) {
duplicatedIds <- selRes[ duplicated( selRes[,1] ), 1 ]
selRes <- selRes[ ! selRes[,1] %in% duplicatedIds, ] }
return( selRes[ match( ids, selRes[,1] ), 2 ] )
}
res1 <- res #making a backup
res$hgnc_symbol <- convertIDs( row.names(res), "ENSEMBL", "SYMBOL", org.Hs.eg.db )
res$entrezid <- convertIDs( row.names(res), "ENSEMBL", "ENTREZID", org.Hs.eg.db )

head(res, 10)
```

```
## log2 fold change (MLE): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 10 rows and 8 columns
##                   baseMean log2FoldChange     lfcSE       stat    pvalue
##                  <numeric>      <numeric> <numeric>  <numeric> <numeric>
## ENSG00000223972  0.000000             NA        NA         NA        NA
## ENSG00000227232 71.877192      -0.617388  0.547883 -1.1268609 0.2598013
## ENSG00000278267  9.328531      -2.539808  1.067946 -2.3782174 0.0173966
## ENSG00000243485  0.152211       0.780782  4.080473  0.1913459 0.8482546
## ENSG00000284332  0.000000             NA        NA         NA        NA
## ENSG00000237613  0.333464       1.771850  4.030219  0.4396412 0.6601970
## ENSG00000268020  0.000000             NA        NA         NA        NA
## ENSG00000240361  0.000000             NA        NA         NA        NA
## ENSG00000186092  0.000000             NA        NA         NA        NA
## ENSG00000238009  1.382485      -0.177357  3.283507 -0.0540145 0.9569236
##                      padj  hgnc_symbol    entrezid
##                 <numeric>  <character> <character>
## ENSG00000223972        NA      DDX11L1   100287102
## ENSG00000227232  0.574020           NA          NA
## ENSG00000278267  0.137254    MIR6859-1   102466751
## ENSG00000243485        NA           NA          NA
```

```
## ENSG00000284332        NA    MIR1302-2   100302278
## ENSG00000237613        NA      FAM138A      645520
## ENSG00000268020        NA           NA          NA
## ENSG00000240361        NA           NA          NA
## ENSG00000186092        NA        OR4F5       79501
## ENSG00000238009        NA LOC100996442   100996442
```

We now have the `hgnc_symbol` added to our genes. We will use this to search for the corresponding gene function in the reactome database.

Gene-set enrichment analysis

The goal of this is to examine whether the observed genes with a strong/significant up- or down-regulation have either something in common or have something to do with the biological process causing the condition.

```
#Extracting only genes that have corresponding data in reactome db and padj value is not an NA
res2 <- res[ res$entrezid %in% keys( reactome.db, "ENTREZID" ) & !is.na( res$padj ) , ]
head(res2)
```

```
## log2 fold change (MLE): Condition disease vs normal
## Wald test p-value: Condition disease vs normal
## DataFrame with 6 rows and 8 columns
##                   baseMean log2FoldChange     lfcSE      stat     pvalue
##                  <numeric>      <numeric> <numeric> <numeric>  <numeric>
## ENSG00000188976 1426.2300      -0.472664  0.395878 -1.193964 0.23249214
## ENSG00000187642   10.3347      -1.514005  1.689232 -0.896268 0.37010940
## ENSG00000187608  458.4554      -1.542325  0.662844 -2.326828 0.01997440
## ENSG00000188157 5994.5542      -0.601310  0.341325 -1.761694 0.07812093
## ENSG00000162571   77.5936       2.363003  0.796479  2.966812 0.00300905
## ENSG00000186891  119.8685      -2.817709  0.929541 -3.031289 0.00243512
##                       padj hgnc_symbol    entrezid
##                  <numeric> <character> <character>
## ENSG00000188976 0.5427853       NOC2L       26155
## ENSG00000187642 0.6758754       PERM1       84808
## ENSG00000187608 0.1492830       ISG15        9636
## ENSG00000188157 0.3184607        AGRN      375790
## ENSG00000162571 0.0469611      TTLL10      254173
## ENSG00000186891 0.0411751    TNFRSF18        8784
```

```
#Next we use select for AnnotationDbi to map entrez_id to Reactome_IDs
reactomeTable <- AnnotationDbi::select( reactome.db,
                                        keys=as.character(res2$entrezid), keytype="ENTREZID",
                                        columns=c("ENTREZID","REACTOMEID") )

head(reactomeTable, 10)
```

```
##    ENTREZID    REACTOMEID
## 1     26155  R-HSA-212436
## 2     26155 R-HSA-3700989
## 3     26155 R-HSA-5633007
## 4     26155 R-HSA-6804756
## 5     26155   R-HSA-73857
```

```
## 6     26155   R-HSA-74160
## 7     84808 R-HSA-1592230
## 8     84808 R-HSA-1852241
## 9     84808 R-HSA-2151201
## 10     9636  R-HSA-110313
```

```
#We then tell which genes are members of which Reactome Paths.
incm <- do.call( rbind, with(reactomeTable, tapply(
   ENTREZID, factor(REACTOMEID), function(x) res2$entrezid %in% x ) ))
colnames(incm) <- res2$entrez
str(incm)
```

```
##  logi [1:2230, 1:9089] FALSE FALSE FALSE FALSE FALSE FALSE ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2230] "R-HSA-1059683" "R-HSA-109581" "R-HSA-109582" "R-HSA-109606" ...
##   ..$ : NULL
```

```
#We remove all rows corresponding to Reactome Paths with less than 20 or more than 80 assigned genes.
within <- function(x, lower, upper) (x>=lower & x<=upper)
incm <- incm[ within(rowSums(incm), lower=20, upper=80), ]

#We test whether the genes in a Reactome Path behave in a special way in our experiment,
testCategory <- function(reactomeID) {
   isMember <- incm[ reactomeID, ]
   data.frame(
      reactomeID = reactomeID,
      numGenes = sum( isMember ),
      avgLFC = mean( res2$log2FoldChange[isMember] ),
      sdLFC = sd( res2$log2FoldChange[isMember] ),
      zValue = mean( res2$log2FoldChange[isMember] ) /sd( res2$log2FoldChange[isMember] ),
      strength = sum( res2$log2FoldChange[isMember] ) / sqrt(sum(isMember)),
      pvalue = t.test( res2$log2FoldChange[ isMember ] )$p.value,
      reactomeName = reactomePATHID2NAME[[reactomeID]],
      stringsAsFactors = FALSE ) }

#We call the function for all Paths in our incidence matrix and collect the results in a data frame:
reactomeResult <- do.call( rbind, lapply( rownames(incm), testCategory ) )
head(reactomeResult)
```

```
##     reactomeID numGenes     avgLFC     sdLFC     zValue   strength      pvalue
## 1 R-HSA-109606       53  0.1293626 0.7695864  0.1680937  0.9417739 0.226566384
## 2 R-HSA-109704       33  0.5389266 0.9878471  0.5455567  3.0958978 0.003678512
## 3 R-HSA-110313       38 -0.2547910 0.7408372 -0.3439231 -1.5706373 0.040767947
## 4 R-HSA-110314       29 -0.2207728 0.6433232 -0.3431755 -1.1888977 0.075182923
## 5 R-HSA-110328       34 -0.5681174 1.4837575 -0.3828910 -3.3126653 0.032475183
## 6 R-HSA-110329       34 -0.5681174 1.4837575 -0.3828910 -3.3126653 0.032475183
##                                                                                reactor
## 1                                                  Homo sapiens: Intrinsic Pathway for Apo
## 2                                                                Homo sapiens: PI3K Ca
## 3        Homo sapiens: Translesion synthesis by Y family DNA polymerases bypasses lesions on DNA te
## 4                          Homo sapiens: Recognition of DNA damage by PCNA-containing replication co
## 5 Homo sapiens: Recognition and association of DNA glycosylase with site containing an affected pyri
## 6                                                Homo sapiens: Cleavage of the damaged pyrim
```

```r
#Now we perform Multiple Testing Adjustment again since we had to compare multiple elememts
reactomeResult$padjust <- p.adjust( reactomeResult$pvalue, "BH" )

#We obtain the reactomeID with significant p.adj values in our comparison of normal vs disease
reactomeResultSignif <- reactomeResult[ reactomeResult$padjust < 0.05, ]

head( reactomeResultSignif[ order(-reactomeResultSignif$strength), ] ,20)[c("reactomeID", "numGenes" ,"
```

```
##        reactomeID numGenes
## 200 R-HSA-2129379       37
## 239 R-HSA-3000178       70
## 64  R-HSA-1566948       44
## 369 R-HSA-5083635       35
## 203  R-HSA-216083       80
## 371 R-HSA-5173214       36
## 44  R-HSA-1296071       79
## 47   R-HSA-140877       28
## 488 R-HSA-6803157       33
## 238 R-HSA-3000171       58
## 91   R-HSA-166658       45
## 51  R-HSA-1442490       57
## 72  R-HSA-1592389       26
## 237 R-HSA-3000170       27
## 266  R-HSA-375280       28
## 45  R-HSA-1296072       33
## 294 R-HSA-3906995       62
## 665  R-HSA-977606       37
## 84  R-HSA-1650814       63
## 599 R-HSA-8948216       41
##                                                          reactomeName
## 200             Homo sapiens: Molecules associated with elastic fibres
## 239                                  Homo sapiens: ECM proteoglycans
## 64                                Homo sapiens: Elastic fibre formation
## 369  Homo sapiens: Defective B3GALTL causes Peters-plus syndrome (PpS)
## 203                   Homo sapiens: Integrin cell surface interactions
## 371    Homo sapiens: O-glycosylation of TSR domain-containing proteins
## 44                                    Homo sapiens: Potassium Channels
## 47           Homo sapiens: Formation of Fibrin Clot (Clotting Cascade)
## 488                                Homo sapiens: Antimicrobial peptides
## 238               Homo sapiens: Non-integrin membrane-ECM interactions
## 91                                    Homo sapiens: Complement cascade
## 51                                  Homo sapiens: Collagen degradation
## 72               Homo sapiens: Activation of Matrix Metalloproteinases
## 237                                 Homo sapiens: Syndecan interactions
## 266                       Homo sapiens: Amine ligand-binding receptors
## 45                       Homo sapiens: Voltage gated Potassium channels
## 294 Homo sapiens: Diseases associated with O-glycosylation of proteins
## 665                       Homo sapiens: Regulation of Complement cascade
## 84           Homo sapiens: Collagen biosynthesis and modifying enzymes
## 599                           Homo sapiens: Collagen chain trimerization
```

Since the samples that were used for this analysis to develop the script had no accompayning metedata details about the nature of the disease under investigation, we can not really do much going forward. However in

case one knew the condition under study as well an idea of the biological processes probably affecetd by the condition, they would go ahead and subset out the genes that have a function associated with that.

Exporting Results to a file

As a Conluding step, we will export our significant results both before the gene annotation and after annotation to include the gene functions to a local file.

We will use the `write.table()` function to save a tab delimited file of a dataframe version of the results.

```
#Exporting the resSig genes
write.table(as.data.frame(resSig),
            file="Condition_significant_results.csv")

#Exporting the genes containing their annotation
write.table(as.data.frame(reactomeResultSignif),
            file="Gene_function_annotated_significant_results.csv")
```

References

- http://folk.uio.no/jonbra/MBV-INF4410_2017/exercises/2017-12-07_R_DESeq2_exercises_without_results.html

- http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html
- http://www.sthda.com/english/wiki/rna-seq-differential-expression-work-flow-using-deseq2#running-the-deseq2-pipeline