ML_DESeq2_workshop

2023-02-07

DESeq2 workshop

The following pipeline is based on DESeq2 vignette by Love et al. found here: http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html#data-quality-assessment-by-sample-clustering-and-visualization

The data used in the vignette and that we will be using is from an experiment on Drosophila melanogaster cell cultures and investigation of the effect of RNAi knock-down of the splicing factor pasilla (Brooks et al. 2011).

Loading the packages

The first thing, is to load the required packages.

```
## Loading required package: DEXSeq
```

```
## Loading required package: BiocParallel
```

```
## Loading required package: Biobase
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
## Loading required package: SummarizedExperiment

## Loading required package: MatrixGenerics

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians

##
## Attaching package: 'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars

## The following object is masked from 'package:Biobase':
##
##     rowMedians

## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
## Loading required package: IRanges

## Loading required package: GenomeInfoDb

## Loading required package: DESeq2

## Loading required package: AnnotationDbi

## Loading required package: RColorBrewer

## Loading required package: ggrepel
```

Importing the data and generating the DESeq dataset

DESeq2 allows for a quick and easy import of test data

We look at the countmatrix (cts) and the column data i.e. data groups

```
head(cts,10)
```

```
##             untreated1 untreated2 untreated3 untreated4 treated1 treated2
## FBgn0000003          0          0          0          0        0        0
## FBgn0000008         92        161         76         70      140       88
## FBgn0000014          5          1          0          0        4        0
## FBgn0000015          0          2          1          2        1        0
## FBgn0000017       4664       8714       3564       3150     6205     3072
## FBgn0000018        583        761        245        310      722      299
## FBgn0000022          0          1          0          0        0        0
## FBgn0000024         10         11          3          3       10        7
## FBgn0000028          0          1          0          0        0        1
## FBgn0000032       1446       1713        615        672     1698      696
##             treated3
## FBgn0000003        1
## FBgn0000008       70
## FBgn0000014        0
## FBgn0000015        0
## FBgn0000017     3334
## FBgn0000018      308
## FBgn0000022        0
## FBgn0000024        5
## FBgn0000028        1
## FBgn0000032      757
```

```
coldata
```

```
##              condition        type
## treated1fb     treated single-read
## treated2fb     treated  paired-end
## treated3fb     treated  paired-end
## untreated1fb untreated single-read
## untreated2fb untreated single-read
## untreated3fb untreated  paired-end
## untreated4fb untreated  paired-end
```

Note that these are not in the same order with respect to samples!

It is absolutely critical that the columns of the count matrix and the rows of the column data are in the same order. D

As they are not in the correct order as given, we need to re-arrange one or the other so that they are consistent in terms of sample order (if we do not, later functions would produce an error). We additionally need to chop off the “fb” of the row names of coldata, so the naming is consistent.

```
rownames(coldata) <- sub("fb", "", rownames(coldata))
all(rownames(coldata) %in% colnames(cts))
```

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

```
all(rownames(coldata) == colnames(cts))
```

```
## [1] FALSE
```

```
cts <- cts[, rownames(coldata)]
all(rownames(coldata) == colnames(cts))
```

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

The countmatrix is now in order and we can generate the DESeq dataset

```
dds <- DESeqDataSetFromMatrix(countData = cts,
                              colData = coldata,
                              design = ~ condition)
dds
```

```
## class: DESeqDataSet
## dim: 14599 7
## metadata(1): version
## assays(1): counts
## rownames(14599): FBgn0000003 FBgn0000008 ... FBgn0261574 FBgn0261575
## rowData names(0):
## colnames(7): treated1 treated2 ... untreated3 untreated4
## colData names(2): condition type
```

Prefiltering the data

It is not required to prefilter the data, but it is a good habit. It reduces the memory and thus the computing time, but it can also help in visualisation by removing “noise”.

```
keep <- rowSums(counts(dds)) >= 10
dds <- dds[keep,]
```

Setting factor levels

R chooses the reference when performing differential gene expression analysis by alphabetic order. Therefore, one should set the factor level manually

```
dds$condition <- factor(dds$condition, levels = c("untreated","treated"))
```

Data transformation and visualisation

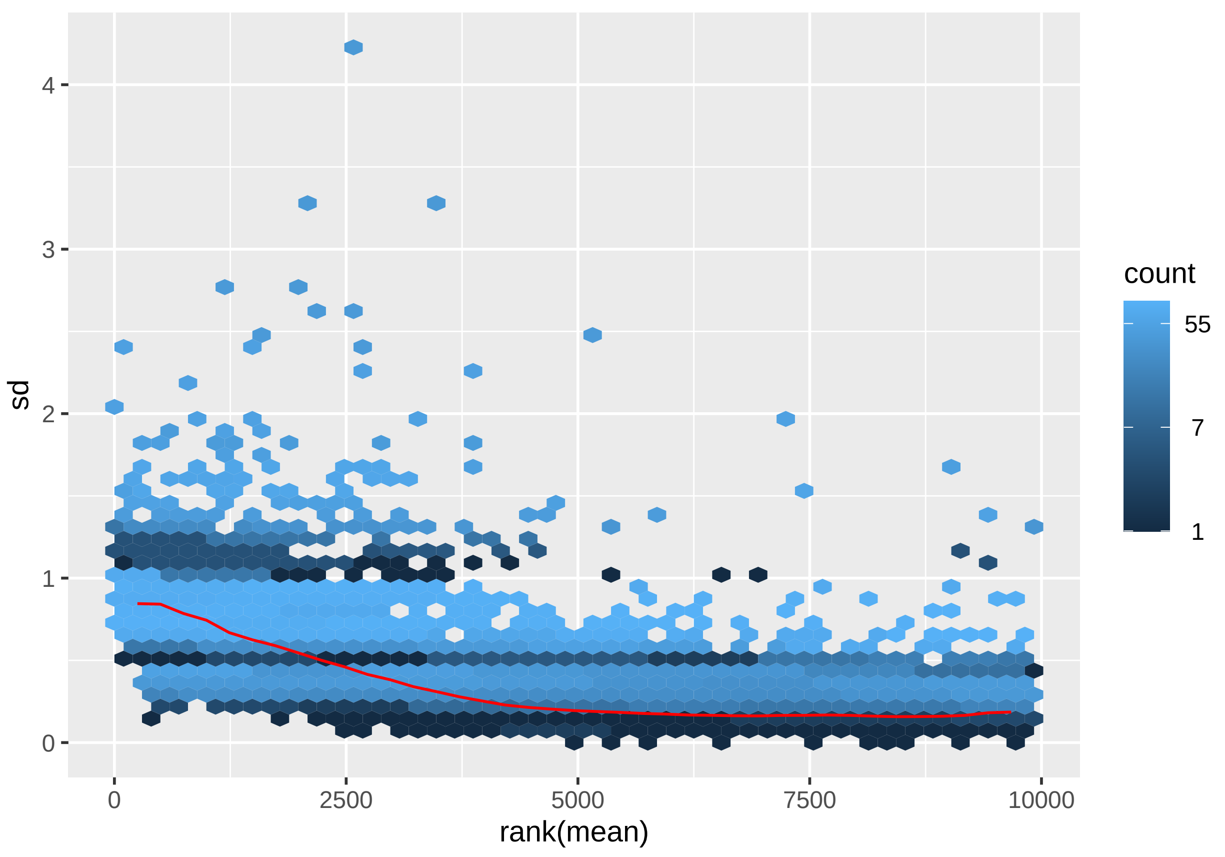
The differential gene expression analysis is done on raw counts. However, for some visualisation, transformed data has to be used. Transformed data implies scaled data. The most common method is logtransform, but other methods are available which are shown below.

The point of the two demonstrated transformations, the VST and the rlog, is to remove the dependence of the variance on the mean, particularly the high variance of the logarithm of count data when the mean is low. Both VST and rlog use the experiment-wide trend of variance over mean, in order to transform the data to remove the experiment-wide trend.

```
##              treated1  treated2  treated3 untreated1 untreated2 untreated3
## FBgn0000008  7.607917  7.834912  7.595052   7.567298   7.642174   7.844603
## FBgn0000014  6.318818  6.041221  6.041221   6.412782   6.173921   6.041221
## FBgn0000017 11.938311 12.024557 12.013565  12.045721  12.284647  12.455939
##             untreated4
## FBgn0000008   7.669147
## FBgn0000014   6.041221
## FBgn0000017  12.077404
```

Plotting the standard deviation of transformed data across all samples against the mean. Observe that the variance differ between the methods. In VST, the variance is stable across

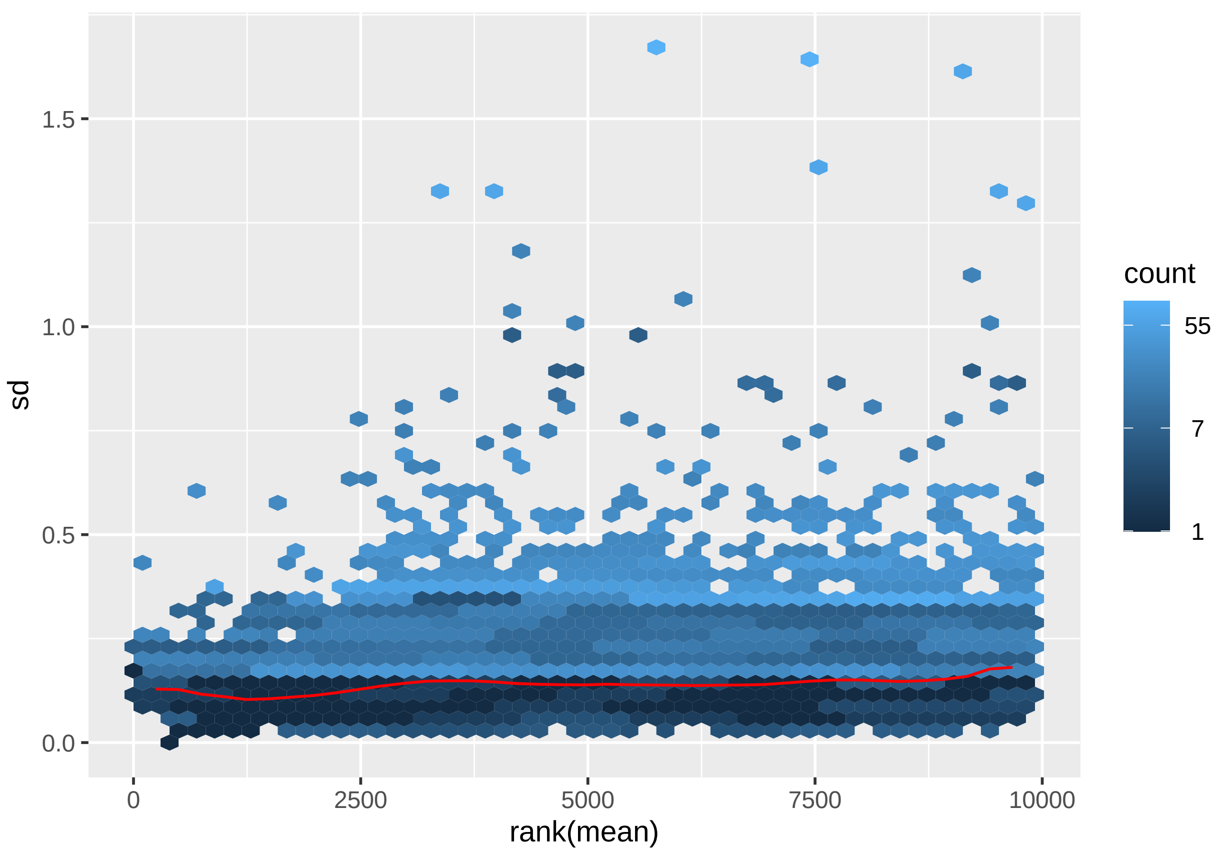
```
meanSdPlot(assay(ntd))
```



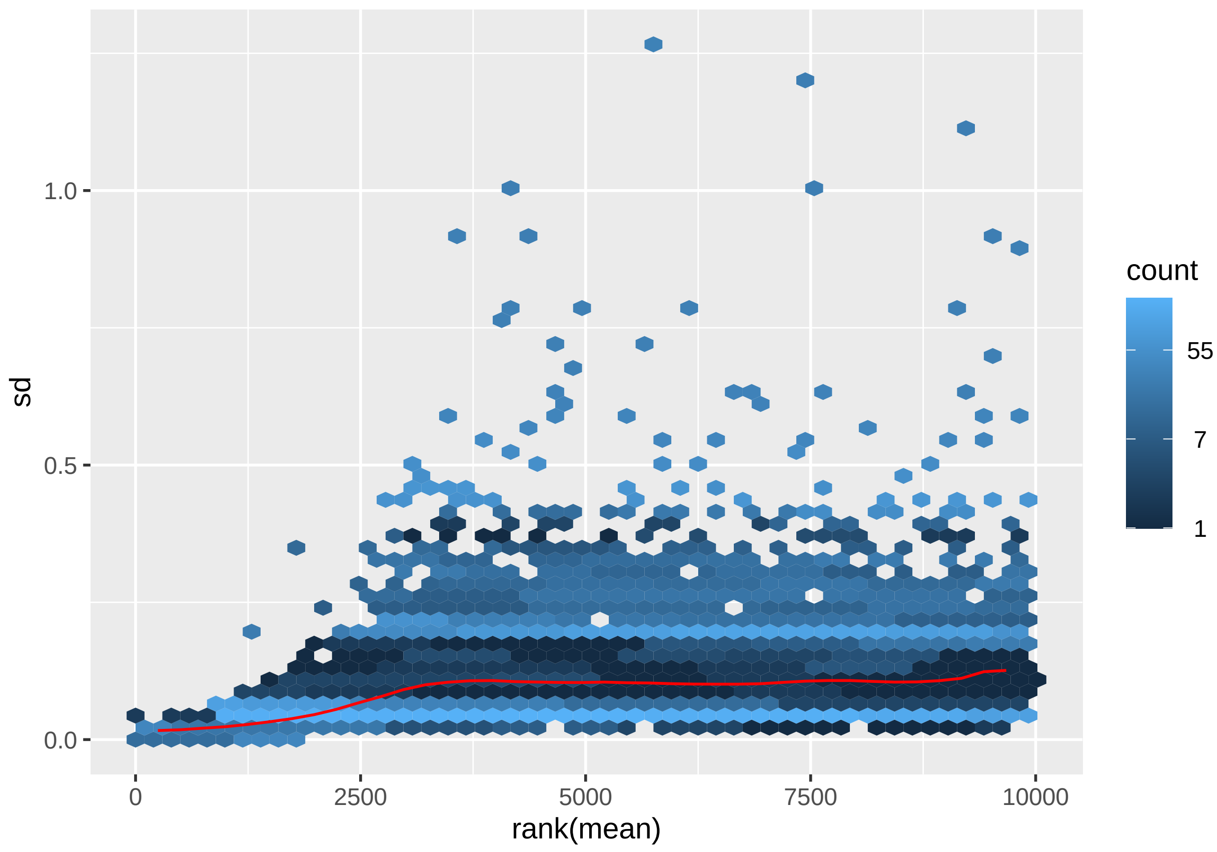
```
1
```

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

```
meanSdPlot(assay(vsd))
```



```
meanSdPlot(assay(rld))
```



Quality control by sample clustering and visaulization

Data quality assessment and quality control (i.e. the removal of insufficiently good data) are essential steps of any data analysis. These steps should typically be performed very early in the analysis of a new data set, preceding or in parallel to the differential expression testing.

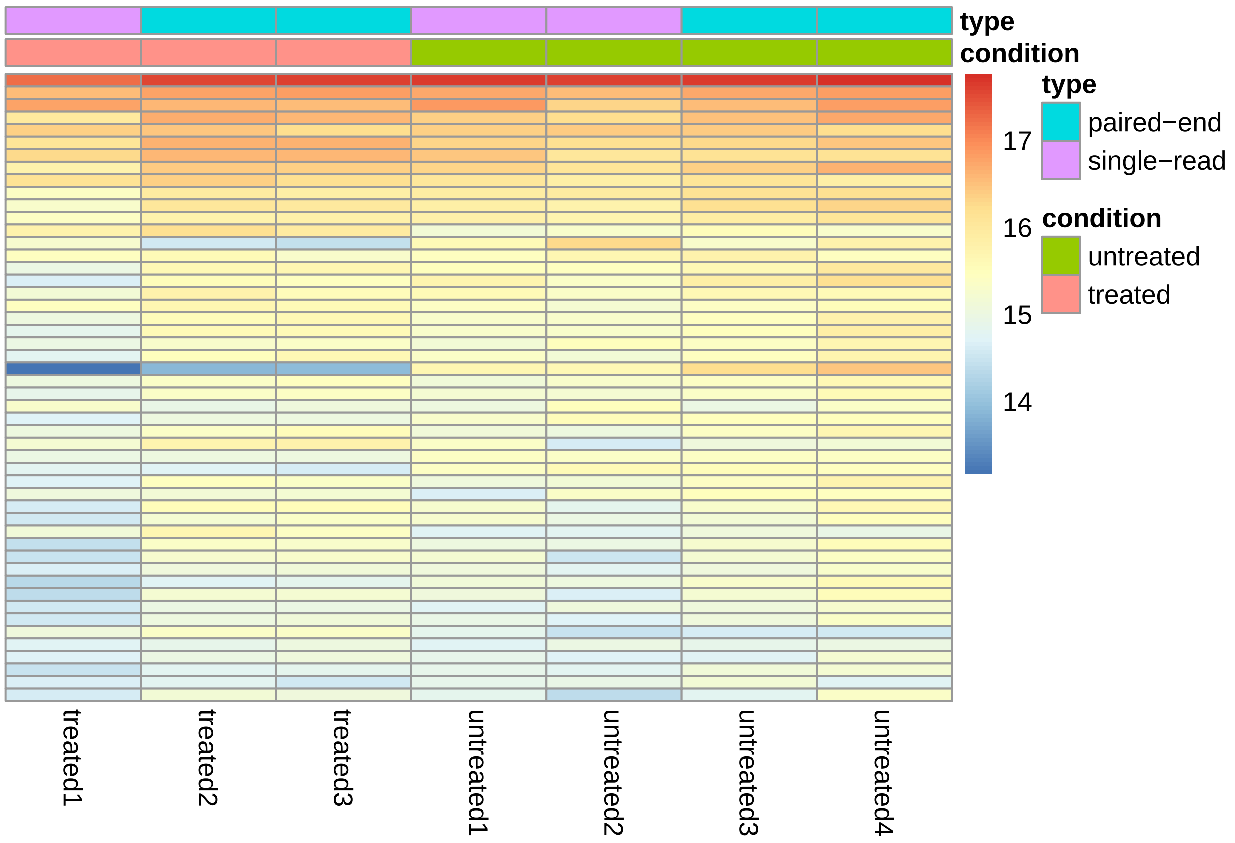
We define the term quality as fitness for purpose. Our purpose is the detection of differentially expressed genes, and we are looking in particular for samples whose experimental treatment suffered from an anormality that renders the data points obtained from these particular samples detrimental to our purpose.

We will generate three heatmaps. First, we select the top 50 expressed genes by the mean expression across all samples

```
select <- order(rowMeans(counts(dds,normalized=FALSE)),
                decreasing=TRUE)[1:50]
df <- as.data.frame(colData(dds)[,c("condition","type")])
```

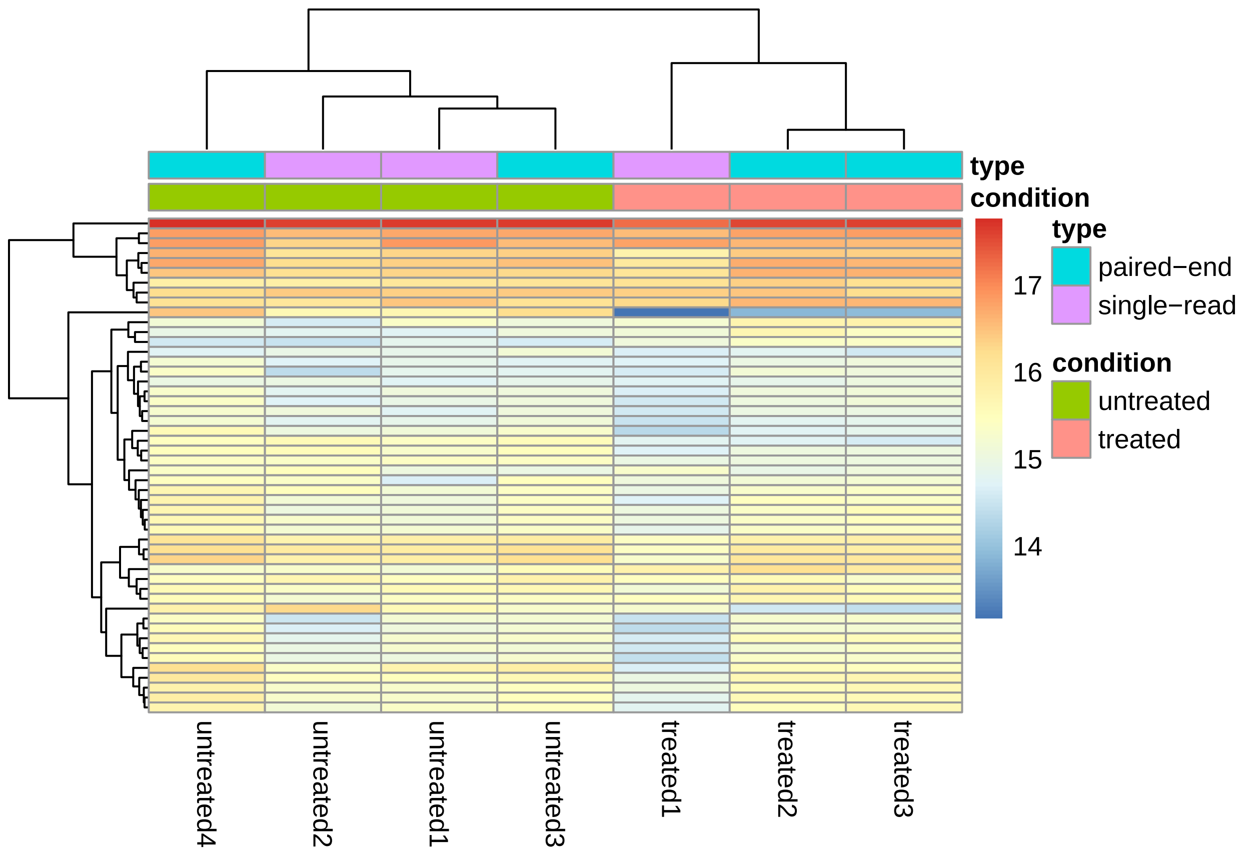
The first heatmap is without any clustering of the data

```
pheatmap(assay(vsd)[select,], cluster_rows=FALSE, show_rownames=FALSE,
         cluster_cols=FALSE, annotation_col=df)
```



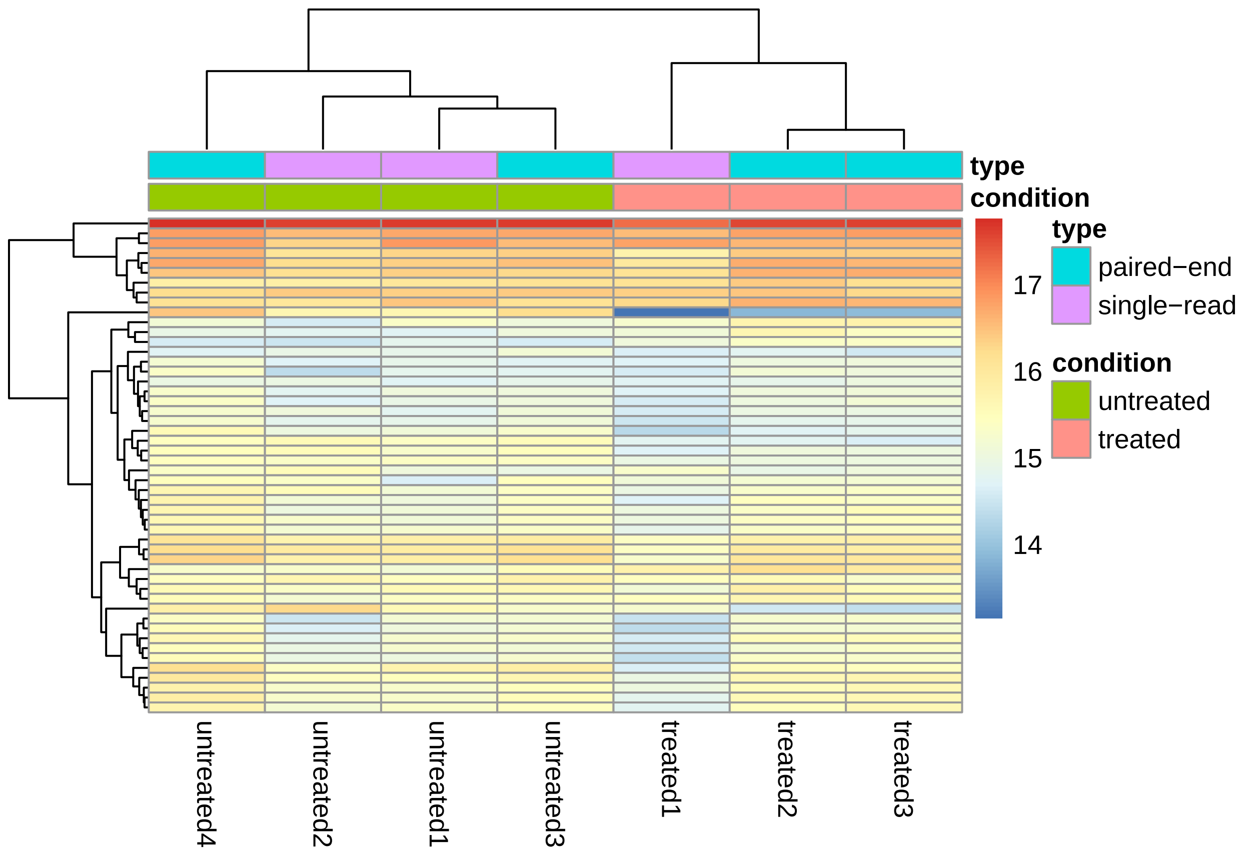
For the second, we test with hiearchical clustering

```
pheatmap(assay(vsd)[select,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df)
```



Lets compare with non-transformed data i.e. only log2 transformed

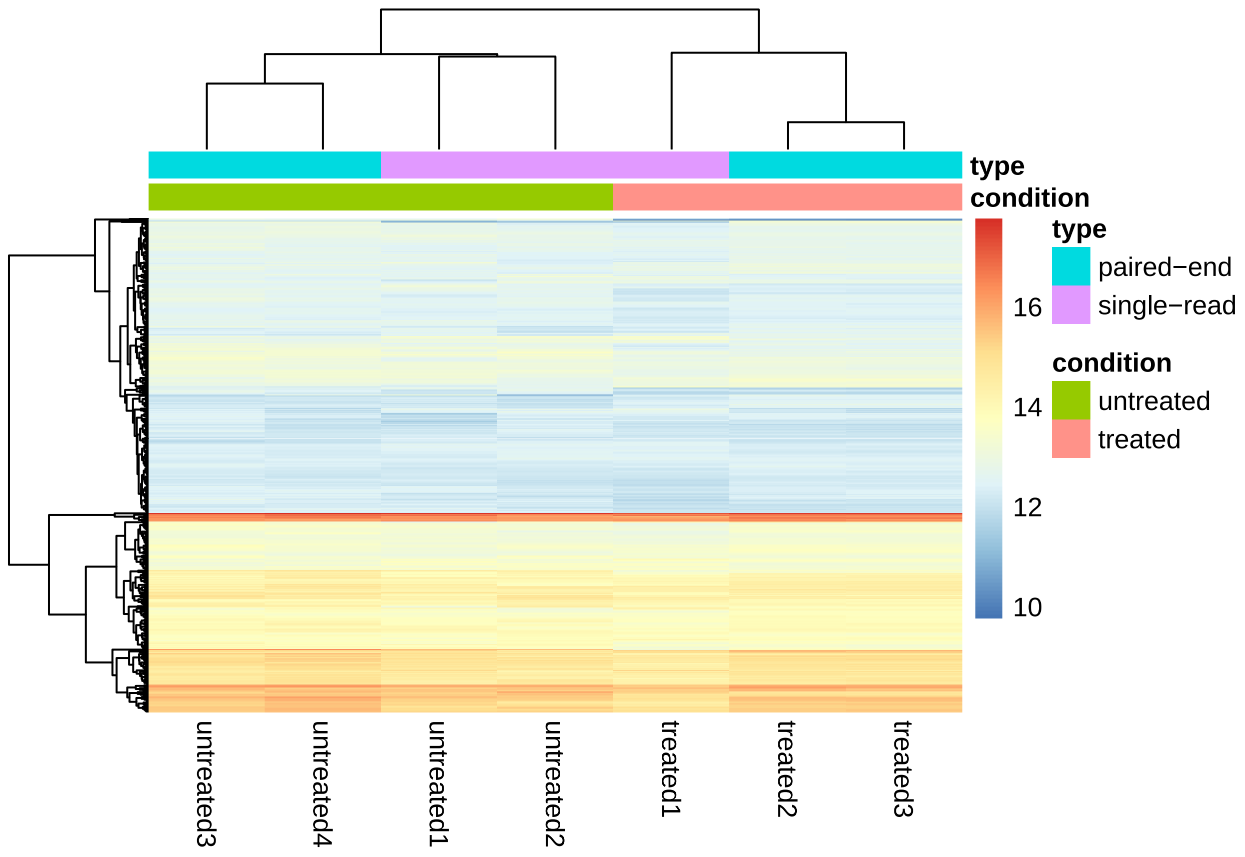
```
pheatmap(assay(ntd)[select,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df)
```



In this case, the data is not that different. . .

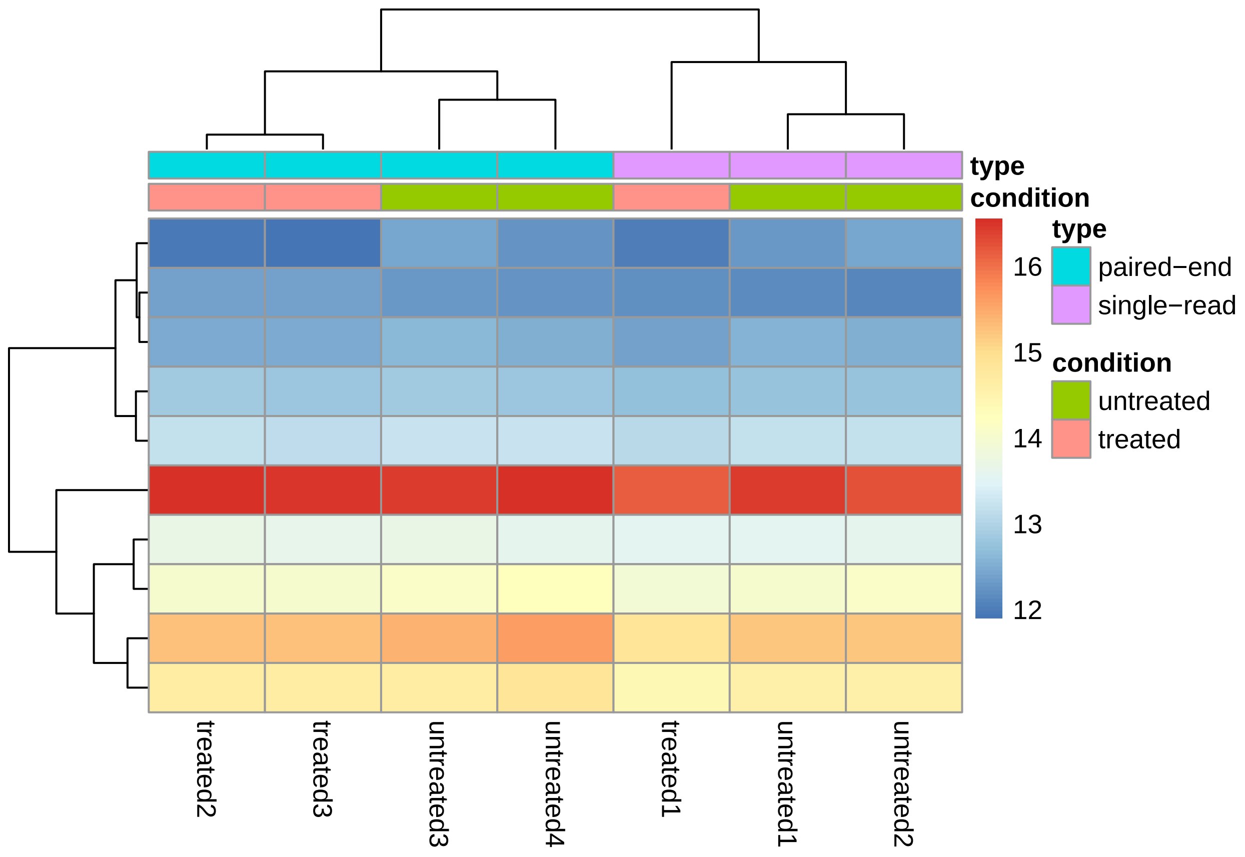
For the sake of fun, let us do one massive heatmap

```
select_500 <- order(rowMeans(counts(dds,normalized=FALSE)),
                  decreasing=TRUE)[1:500]
pheatmap(assay(vsd)[select_500,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df)
```



We can also play around with the kmeans to define the number of clusters to generate

```
select_500 <- order(rowMeans(counts(dds,normalized=FALSE)),
                    decreasing=TRUE)[1:500]
Set_K = 10
pheatmap(assay(vsd)[select_500,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df, kmeans_k = Set_K)
```



By setting the Kmeans, the rows are aggregated before generating the heatmap

Another way to determine whether the samples are similar or not, is to calculate the euclidean distances between the samples and store the values in a distance matrix which then can be visualised

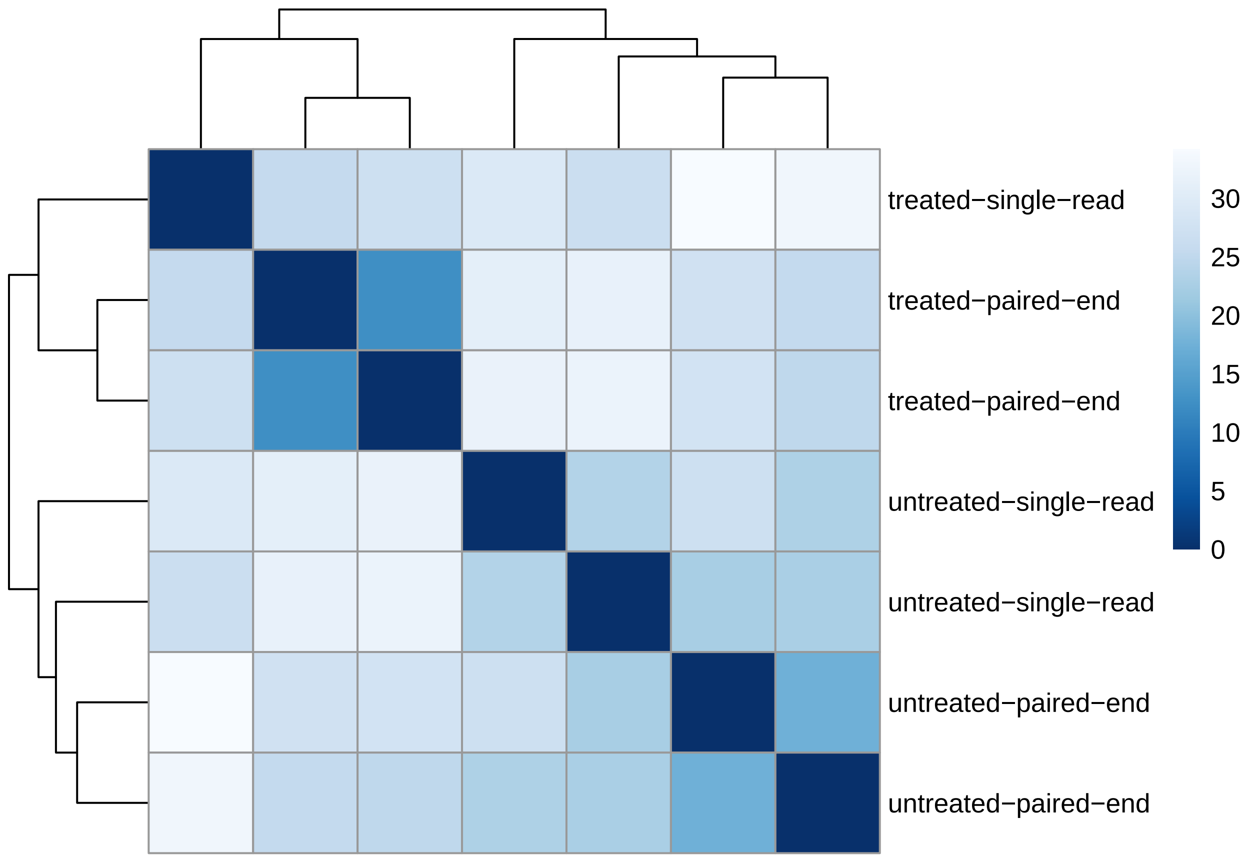
Calculating the distances:

```
sampleDists <- dist(t(assay(vsd)), method = "euclidean")
head(sampleDists)
```

```
## [1] 25.50386 26.95680 29.40565 26.55427 34.19351 33.10003
```

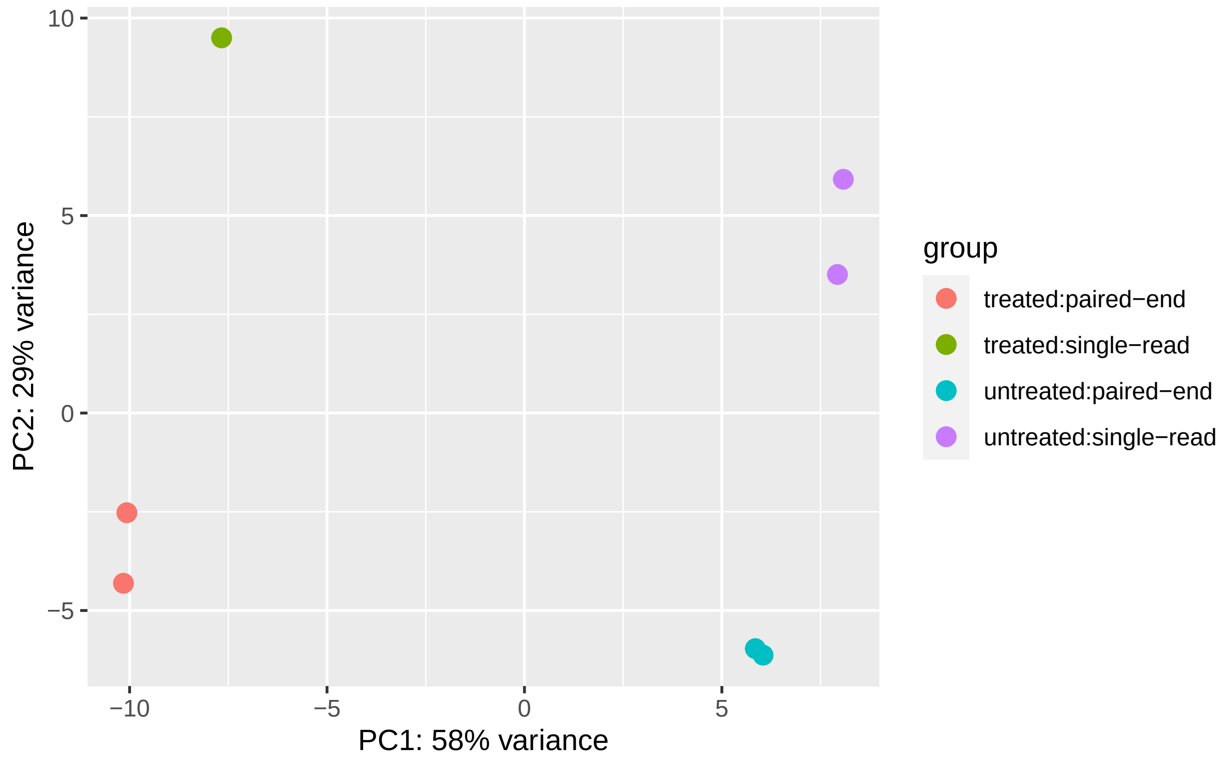
And plotting it:

```
sampleDistMatrix <- as.matrix(sampleDists)
rownames(sampleDistMatrix) <- paste(vsd$condition, vsd$type, sep="-")
colnames(sampleDistMatrix) <- NULL
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix,
         clustering_distance_rows=sampleDists,
         clustering_distance_cols=sampleDists,
         col=colors)
```



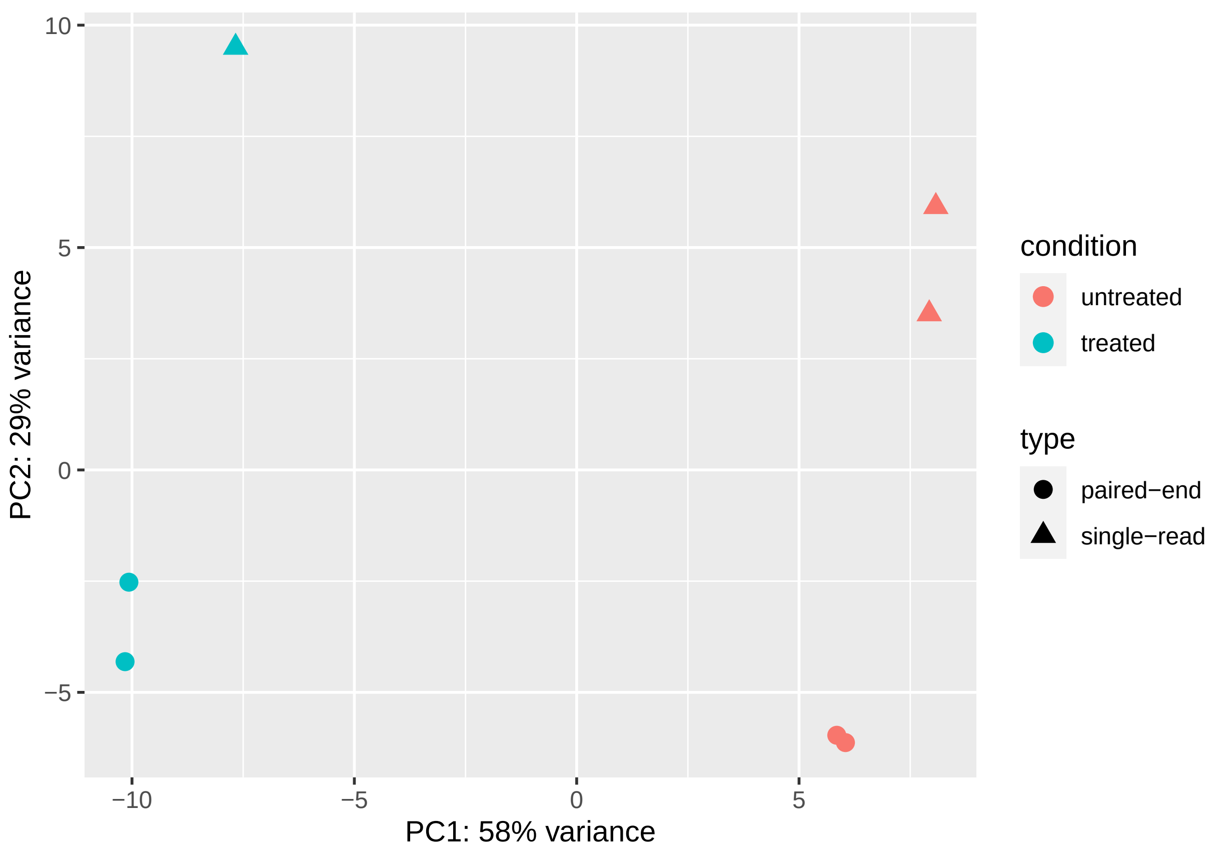
We leave the best to last, which is the PCA loved by most.

```
plotPCA(vsd, intgroup=c("condition", "type"))
```



The PCA plot can be further customised with ggplot

```
pcaData <- plotPCA(vsd, intgroup=c("condition", "type"), returnData=TRUE)
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(PC1, PC2, color=condition, shape=type)) +
  geom_point(size=3) +
  xlab(paste0("PC1: ",percentVar[1],"% variance")) +
  ylab(paste0("PC2: ",percentVar[2],"% variance")) +
  coord_fixed()
```



The data looks good. The majority of the variance is shown in PC1 in which we see that the data cluster based on the condition untreated or not. PC2 is explained by the technical difference, being sequence either paired-end or single-read.

Differential gene expression analysis

Differential expression analysis based on the Negative Binomial (a.k.a. Gamma-Poisson) distribution, further described in the DESeq2 publication (Love, Huber, and Anders 2014).

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

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

```
## fitting model and testing
```

```
res <- results(dds)
res
```

```
## log2 fold change (MLE): condition treated vs untreated
## Wald test p-value: condition treated vs untreated
## DataFrame with 9921 rows and 6 columns
##               baseMean log2FoldChange     lfcSE       stat    pvalue      padj
##              <numeric>      <numeric> <numeric>  <numeric> <numeric> <numeric>
## FBgn0000008   95.14429     0.00227644  0.223729   0.010175 0.9918817  0.997211
## FBgn0000014    1.05652    -0.49512039  2.143186  -0.231021 0.8172987        NA
## FBgn0000017 4352.55357    -0.23991894  0.126337  -1.899041 0.0575591  0.288002
## FBgn0000018  418.61048    -0.10467391  0.148489  -0.704927 0.4808558  0.826834
## FBgn0000024    6.40620     0.21084779  0.689588   0.305759 0.7597879  0.943501
## ...                ...            ...       ...        ...       ...       ...
## FBgn0261570 3208.38861      0.2955329  0.127350  2.3206264  0.020307  0.144240
## FBgn0261572    6.19719     -0.9588230  0.775315 -1.2366888  0.216203  0.607848
## FBgn0261573 2240.97951      0.0127194  0.113300  0.1122634  0.910615  0.982657
## FBgn0261574 4857.68037      0.0153924  0.192567  0.0799327  0.936291  0.988179
## FBgn0261575   10.68252      0.1635705  0.930911  0.1757102  0.860522  0.967928
```

Shrinkage of effect size (LFC estimates) is useful for visualization and ranking of genes. To shrink the LFC, we pass the dds object to the function lfcShrink. Below we specify to use the apeglm method for effect size shrinkage (Zhu, Ibrahim, and Love 2018), which improves on the previous estimator.

Apeglm is based on a generalised linear model. It provides Bayesian shrinkage estimators for effect sizes for a variety of GLM models, using approximation of the posterior for individual coefficients.

We provide the dds object and the name or number of the coefficient we want to shrink, where the number refers to the order of the coefficient as it appears in resultsNames(dds).

```
resLFC <- lfcShrink(dds, coef="condition_treated_vs_untreated", 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
```

```
resLFC
```

```
## log2 fold change (MAP): condition treated vs untreated
## Wald test p-value: condition treated vs untreated
## DataFrame with 9921 rows and 5 columns
##               baseMean log2FoldChange     lfcSE    pvalue      padj
##              <numeric>      <numeric> <numeric> <numeric> <numeric>
## FBgn0000008   95.14429     0.00119920  0.151897 0.9918817  0.997211
## FBgn0000014    1.05652    -0.00473412  0.205468 0.8172987        NA
## FBgn0000017 4352.55357    -0.18989990  0.120377 0.0575591  0.288002
## FBgn0000018  418.61048    -0.06995753  0.123901 0.4808558  0.826834
## FBgn0000024    6.40620     0.01752715  0.198633 0.7597879  0.943501
## ...                ...            ...       ...       ...       ...
## FBgn0261570 3208.38861     0.24110290 0.1244469  0.020307  0.144240
## FBgn0261572    6.19719    -0.06576173 0.2141351  0.216203  0.607848
## FBgn0261573 2240.97951     0.01000619 0.0993764  0.910615  0.982657
## FBgn0261574 4857.68037     0.00843552 0.1408267  0.936291  0.988179
## FBgn0261575   10.68252     0.00809101 0.2014704  0.860522  0.967928
```

We can extract the number of differentially expressed genes and summarise the results:

```
resOrdered <- res[order(res$padj),]
head(resOrdered, 10)
```

```
## log2 fold change (MLE): condition treated vs untreated
## Wald test p-value: condition treated vs untreated
## DataFrame with 10 rows and 6 columns
##              baseMean log2FoldChange     lfcSE      stat       pvalue
##             <numeric>      <numeric> <numeric> <numeric>    <numeric>
## FBgn0039155   730.568       -4.61874 0.1691240  -27.3098 3.24446e-164
## FBgn0025111  1501.448        2.89995 0.1273576   22.7701 9.07164e-115
## FBgn0029167  3706.024       -2.19691 0.0979154  -22.4368 1.72030e-111
## FBgn0003360  4342.832       -3.17954 0.1435677  -22.1466 1.12417e-108
## FBgn0035085   638.219       -2.56024 0.1378126  -18.5777  4.86844e-77
## FBgn0039827   261.911       -4.16243 0.2325942  -17.8957  1.27484e-71
## FBgn0034736   225.871       -3.51132 0.2147628  -16.3498  4.36736e-60
## FBgn0029896   489.877       -2.44494 0.1522149  -16.0625  4.67705e-58
## FBgn0000071   342.246        2.67973 0.1824175   14.6901  7.46383e-49
## FBgn0051092   153.069        2.32791 0.1772255   13.1353  2.06706e-39
##                     padj
##                <numeric>
## FBgn0039155 2.71918e-160
## FBgn0025111 3.80147e-111
## FBgn0029167 4.80594e-108
## FBgn0003360 2.35542e-105
## FBgn0035085  8.16049e-74
## FBgn0039827  1.78075e-68
## FBgn0034736  5.22898e-57
## FBgn0029896  4.89979e-55
## FBgn0000071  6.95048e-46
## FBgn0051092  1.73240e-36
```

```
summary(res)
```

```
##
## out of 9921 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 518, 5.2%
## LFC < 0 (down)     : 536, 5.4%
## outliers [1]       : 1, 0.01%
## low counts [2]     : 1539, 16%
## (mean count < 6)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
print("Number of DEGs:")
```

```
## [1] "Number of DEGs:"
```

```r
sum(res$padj < 0.1, na.rm=TRUE)
```

```
## [1] 1054
```

The parameters can also be adjusted to show adjusted p-value of 0.05

```r
res05 <- results(dds, alpha=0.05)
summary(res05)
```

```
##
## out of 9921 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0 (up)       : 407, 4.1%
## LFC < 0 (down)     : 431, 4.3%
## outliers [1]       : 1, 0.01%
## low counts [2]     : 1347, 14%
## (mean count < 5)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```r
print("Number of DEGs:")
```

```
## [1] "Number of DEGs:"
```

```r
sum(res05$padj < 0.05, na.rm=TRUE)
```

```
## [1] 838
```

Comparing the results to the LFC shrinken

```r
summary(resLFC)
```

```
##
## out of 9921 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 518, 5.2%
## LFC < 0 (down)     : 536, 5.4%
## outliers [1]       : 1, 0.01%
## low counts [2]     : 1539, 16%
## (mean count < 6)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```r
print("Number of DEGs:")
```

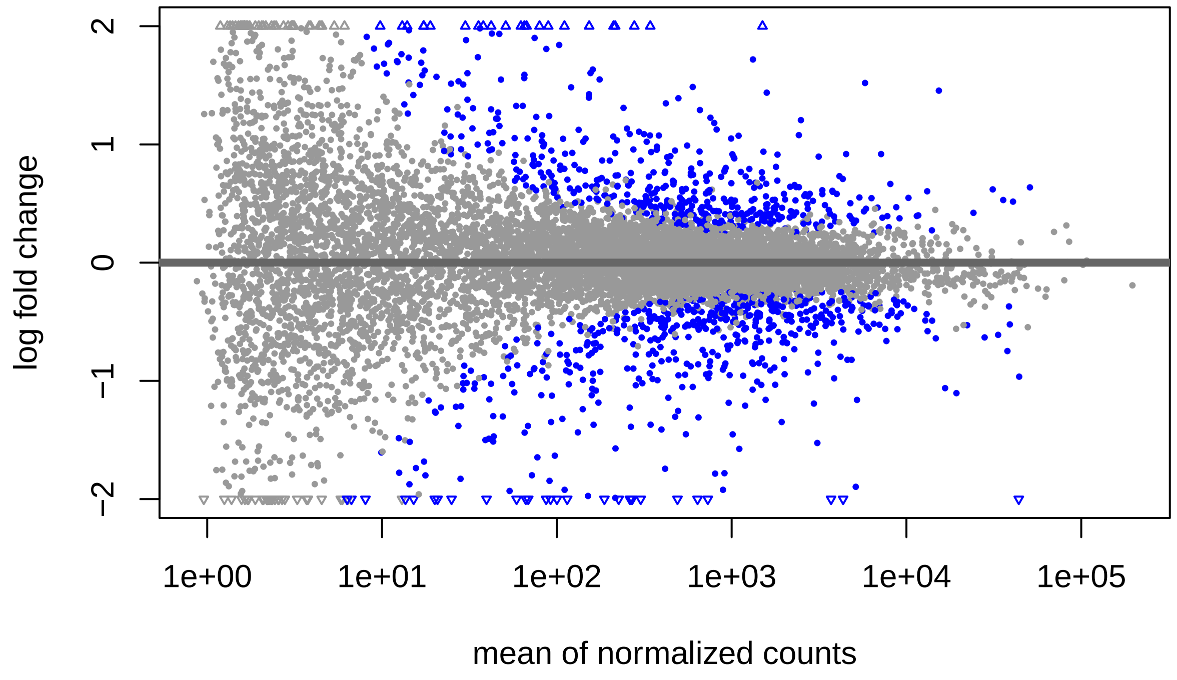
```
## [1] "Number of DEGs:"
```

```
sum(resLFC$padj < 0.05, na.rm=TRUE)
```

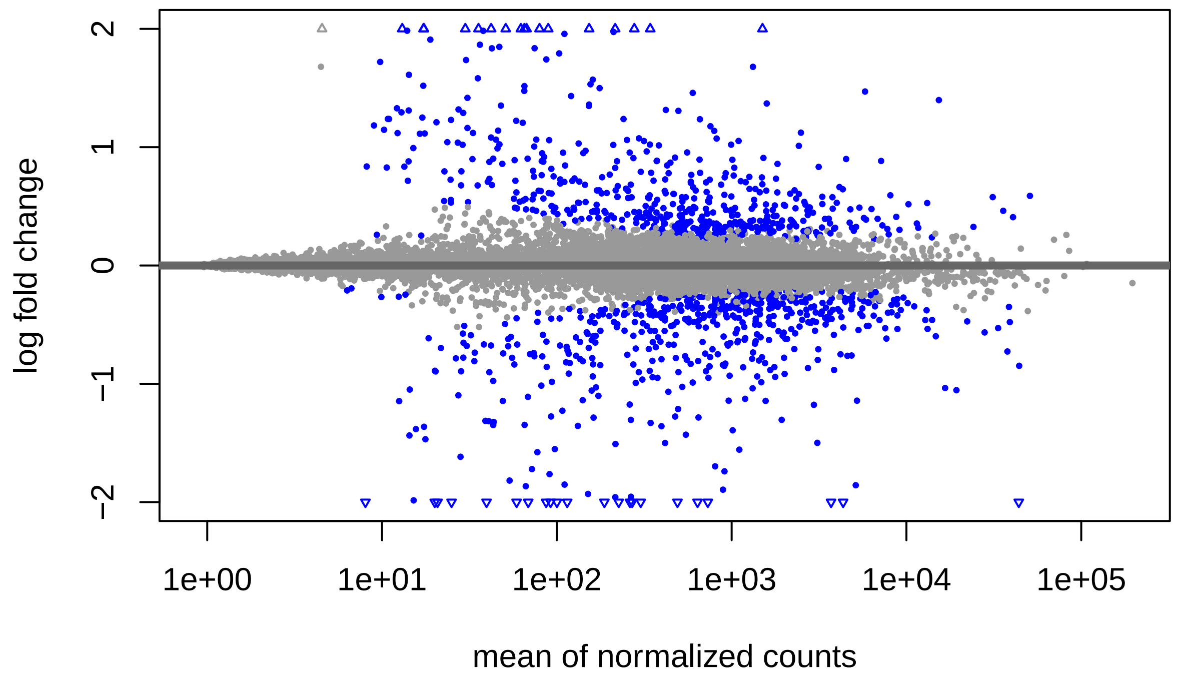
```
## [1] 841
```

At adjusted p-value 0.1, there are no differences in number of DEG's. But as we lower the cutoff, there is. The non shrunken data and shrunken can also be compared visually to further explain what shrinkage does:

```
plotMA(res, ylim=c(-2,2))
```



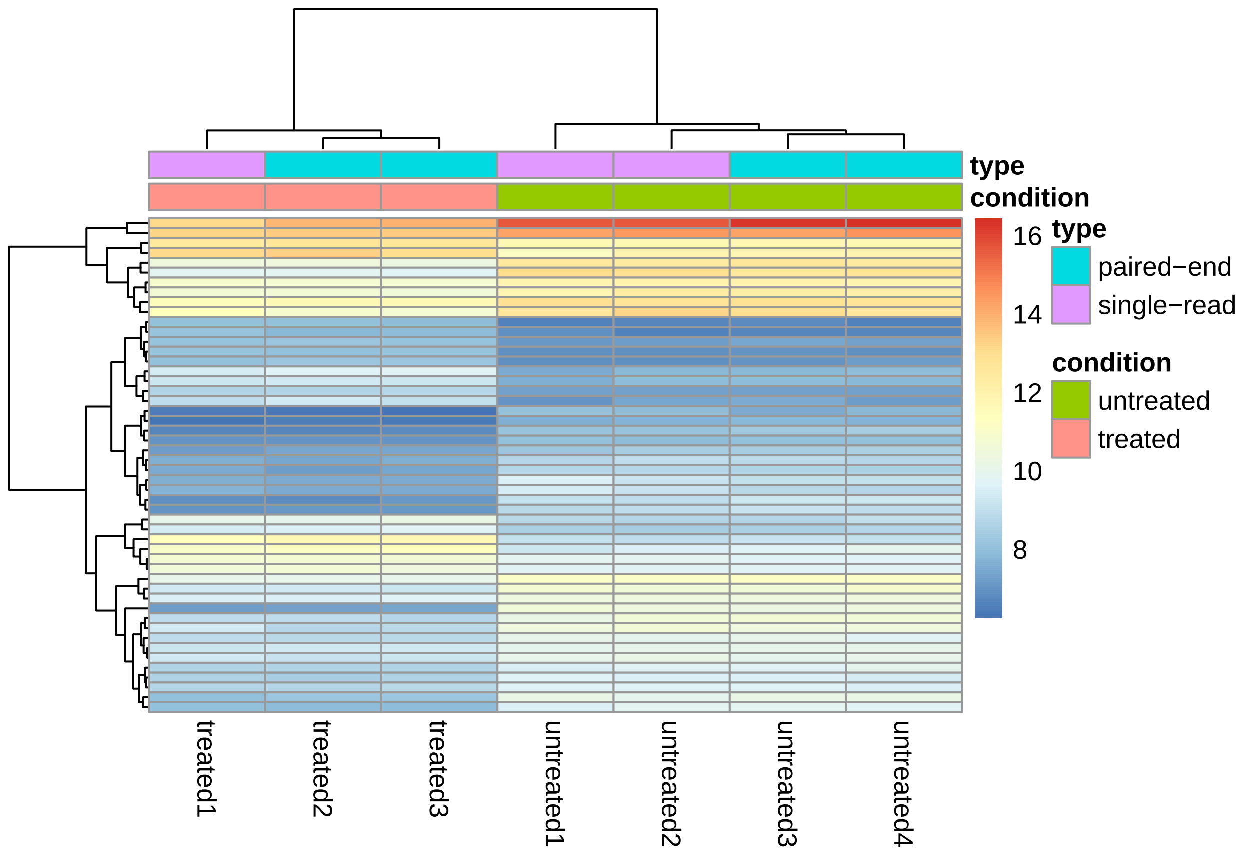
```
plotMA(resLFC, ylim=c(-2,2))
```



From the APEGLM paper: "When the read counts are low or highly variable, the maximum likelihood estimates for the LFCs has high variance, leading to large estimates not representative of true differences, and poor ranking of genes by effect size. One approach is to introduce filtering thresholds and pseudocounts to exclude or moderate estimated LFCs. Filtering may result in a loss of genes from the analysis with true differences in expression, while pseudocounts provide a limited solution that must be adapted per dataset.[. . .] The proposed method, Approximate Posterior Estimation for generalized linear model, apeglm, has lower bias than previously proposed shrinkage estimators, while still reducing variance for those genes with little information for statistical inference."

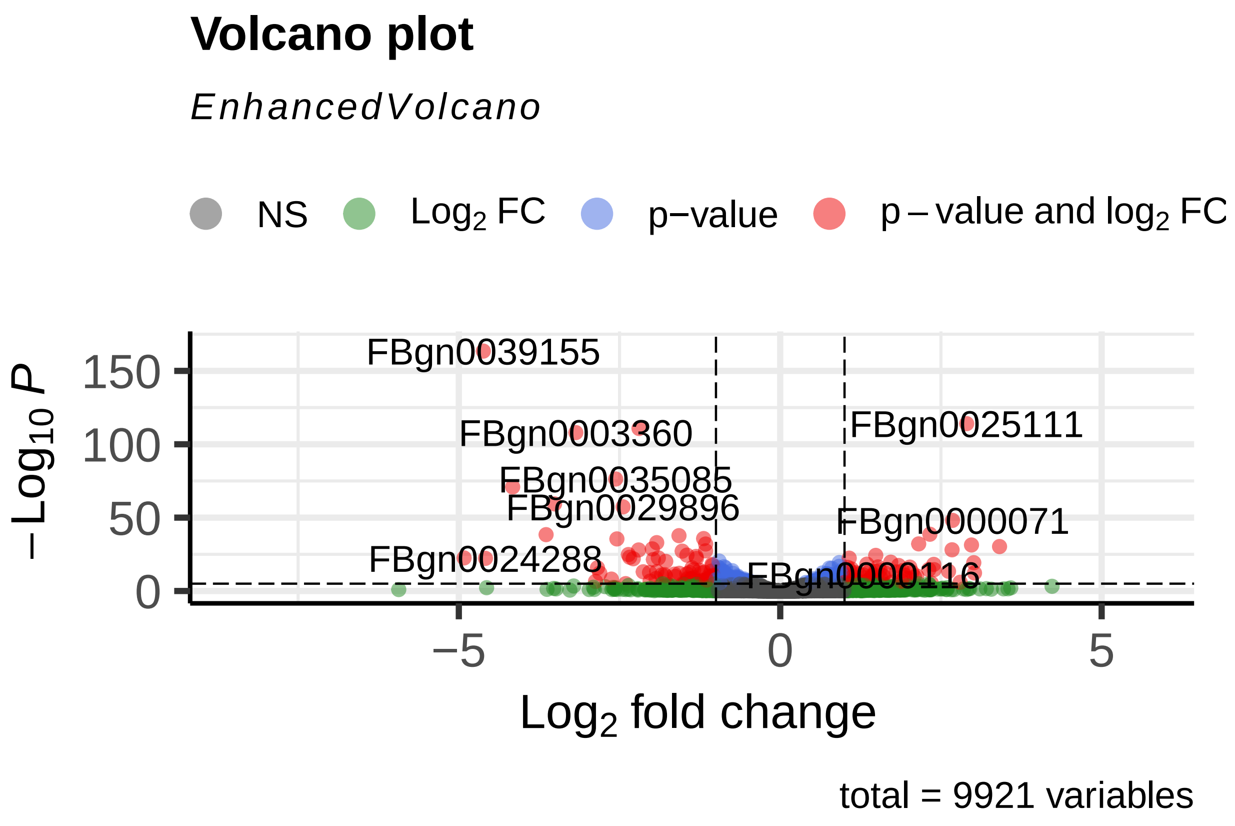
Lets do a heatmap of our top 50 differentially expressed genes based on padj, showing genes with a padj <= 0.05

```
#resLFC <- resLFC[!is.na(resLFC$padj),]
#DE.genes = rownames(resLFC[resLFC$padj < .05 & abs(resLFC$log2FoldChange) > 1,])

resOrdered.LFC <- resLFC[order(resLFC$padj, decreasing = FALSE),]
resOrdered.LFC.genes = rownames(resOrdered.LFC)[1:50]

pheatmap(assay(vsd)[resOrdered.LFC.genes,], cluster_rows=TRUE, show_rownames=FALSE,
         cluster_cols=TRUE, annotation_col=df)
```



We can also visualize the DEG's with a Volcano plot

```
EnhancedVolcano(res,
    lab = rownames(res),
    x = 'log2FoldChange',
    y = 'pvalue')
```


Volcano plot
EnhancedVolcano
NS
Log2 FC
p−value
p − value and log2 FC
FBgn0039155
FBgn0003360
FBgn0025111
FBgn0035085
FBgn0029896
FBgn0000071
FBgn0024288
FBgn0000116
150
100
50
0
−5
0
5
−Log10 P
Log2 fold change
total = 9921 variables