Differential Expression of Bulk RNA-Seq

Tanisha

2024-04-02

Overview

Lecture 1

- loading from csv
- generating deseq set
- transforming the data
- library size

Lecture 2

- pca, screeplot

Lecture 3

- deseq2 contrast "Th0 vs Th2" memory cells

Exercise

- Th0 vs Th2 separately for naive cells / memory cell
- compare DEGs for both cell types to find overlap and difference

LECTURE 1: Loading and preprocessing.

Load the libraries

```
#if (!require("BiocManager", quietly = TRUE))
#  install.packages("BiocManager")

#BiocManager::install("DESeq2")
library(DESeq2)
```

```
## Loading required package: S4Vectors
```

```
## Loading required package: stats4
```

```
## 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

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:utils':
##
##     findMatches

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

## Loading required package: IRanges

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: SummarizedExperiment

## Loading required package: MatrixGenerics

## Loading required package: matrixStats

##
## 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

## Loading required package: Biobase

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

##
## Attaching package: 'Biobase'

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

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

```
#BiocManager::install("vsn")
library(vsn)
dir()
```

```
##  [1] "DATA_FSB_SET_1.csv"
##  [2] "DATA_FSB_SET_2.csv"
##  [3] "DATA_FSB_SET_3.csv"
##  [4] "DATA_FSB_SET_4A.csv"
##  [5] "DATA_FSB_SET_4B.csv"
##  [6] "Differential Expression of Bulk RNA Seq.Rmd"
##  [7] "Differential-Expression-of-Bulk-RNA-Seq.html"
##  [8] "Differential-Expression-of-Bulk-RNA-Seq.log"
##  [9] "Differential-Expression-of-Bulk-RNA-Seq.Rmd"
## [10] "Differential-Expression-of-Bulk-RNA-Seq.tex"
## [11] "genomic analysis.Rmd"
## [12] "genomic-analysis.html"
## [13] "genomic-analysis.log"
## [14] "genomic-analysis.tex"
## [15] "NCOMMS-19-7936188_bulk_RNAseq_metadata.txt.gz"
## [16] "NCOMMS-19-7936188_bulk_RNAseq_raw_counts.txt.gz"
## [17] "rsconnect"
## [18] "SECONDSESSIONOFR.Rproj"
## [19] "Untitled.R"
```

Loading the data

The first step of every analysis is parsing the data. Here we have compressed tab-separate file prepared by the authors of the publication. We can load the gene expression matrix with a single command. In general it is good practice to store data in a compressed format to save storage space.

```
ge_matrix <- read.table('NCOMMS-19-7936188_bulk_RNAseq_raw_counts.txt.gz',
                header = TRUE, sep = '\t')
dim(ge_matrix)
```

```
## [1] 58051    94
```

```
ge_matrix[1:4, 1:4]
```

```
##                 I0712 I0713 I0717 I0721
## ENSG00000223972     0     0     0     0
## ENSG00000227232    55    93   198   131
## ENSG00000278267     6     6    28     5
## ENSG00000243485     0     0     0     3
```

Loading the meta-data

In addition to the expression data, we also need the meta data i.e. which samples corresponds to which phenotype, cell type.

```
pheno_matrix <- read.table('NCOMMS-19-7936188_bulk_RNAseq_metadata.txt.gz',
                header = TRUE, sep = '\t', stringsAsFactors = TRUE)
pheno_matrix[1:4, 1:4]
```

```
##    sample_id  cell_type cytokine_condition stimulation_time
## 1      I0712  CD4_Naive               IFNB              16h
## 2      I0713  CD4_Naive               Th17              16h
## 6      I0717 CD4_Memory            Resting               5d
## 10     I0721  CD4_Naive                Th2               5d
```

Organize the data

We can assign the sample names to the expression matrix rows, which makes it easier to keep track of data after e.g. subsetting or shuffling.

```
rownames(pheno_matrix) <- pheno_matrix$sample_id

dim(pheno_matrix)
```

```
## [1] 94 10
```

```
head(pheno_matrix)
```

```
##       sample_id  cell_type cytokine_condition stimulation_time donor_id  sex
## I0712     I0712  CD4_Naive               IFNB              16h      257 Male
## I0713     I0713  CD4_Naive               Th17              16h      254 Male
## I0717     I0717 CD4_Memory            Resting               5d      265 Male
## I0721     I0721  CD4_Naive                Th2               5d      257 Male
## I0726     I0726 CD4_Memory               Th17               5d      264 Male
## I0731     I0731  CD4_Naive                Th0              16h      257 Male
##       age sequencing_batch cell_culture_batch rna_integrity_number
## I0712  38                1                  3                  9.5
## I0713  58                1                  3                  9.3
## I0717  59                1                  4                  9.7
## I0721  38                1                  3                 10.0
## I0726  27                1                  4                  9.9
## I0731  38                1                  3                  9.4
```

We can also check that both matrices are properly aligned.

```
all(rownames(pheno_matrix) == colnames(ge_matrix))
```

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

Now we need to select samples corresponding to the cell type and treatment that we want to focus on, in this case the CD4+ Memory cells after 5 days of treatment vs. control.

```
stimTime    <- '5d'
conditions  <- c('Th2', 'Th0')
celltype    <- 'CD4_Memory'
```

We can make an index as practiced before and apply index for subsetting

```
toSelect <- pheno_matrix$stimulation_time == stimTime &
    pheno_matrix$cytokine_condition %in% conditions &
    pheno_matrix$cell_type == celltype

pheno_matrix.subset <- pheno_matrix[toSelect, ]
ge_matrix.subset <- ge_matrix[ , toSelect]
```

Create a DESeq2 Object

```
dds <- DESeqDataSetFromMatrix(countData = ge_matrix.subset,
                              colData = , pheno_matrix.subset,
                              design = ~ cytokine_condition)
```

```
## factor levels were dropped which had no samples
```

A commonly performed step is the filtering of genes, which have too few counts, e.g. less than 10 reads over all samples.

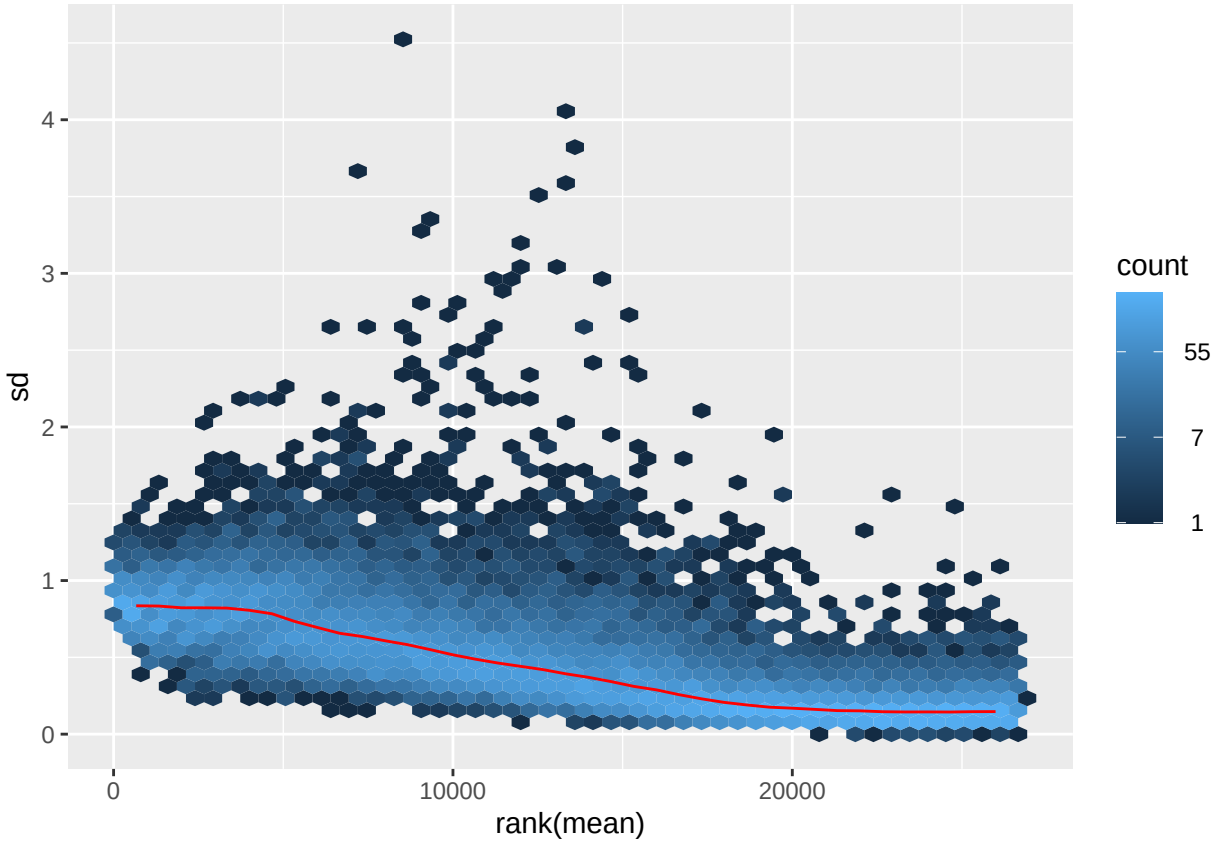
```
keep <- rowSums(counts(dds)) >= 10
dds <- dds[keep,]
dds
```

```
## class: DESeqDataSet
## dim: 26656 9
## metadata(1): version
## assays(1): counts
## rownames(26656): ENSG00000227232 ENSG00000278267 ... ENSG00000271254
##   ENSG00000275405
## rowData names(0):
## colnames(9): I0736 I0749 ... I0867 I0874
## colData names(10): sample_id cell_type ... cell_culture_batch
##   rna_integrity_number
```

Investigate the data

While DESeq2 operates on raw counts, visualization and downstream analyses often depend on data following a normal distribution. It is possible to apply a log-transformation on the raw data to achieve that. Unfortunately, the logarithm of count data tends to exhibit higher variance when the mean expression value is low. Let's take a look at our data. normTransform applies a normalization transformation, which adjusts for differences in library size and composition

```
# Apply a pseudocount of 1 and apply log2
normtfd <- normTransform(dds)
# Compare mean to sd
meanSdPlot(assay(normtfd))
```

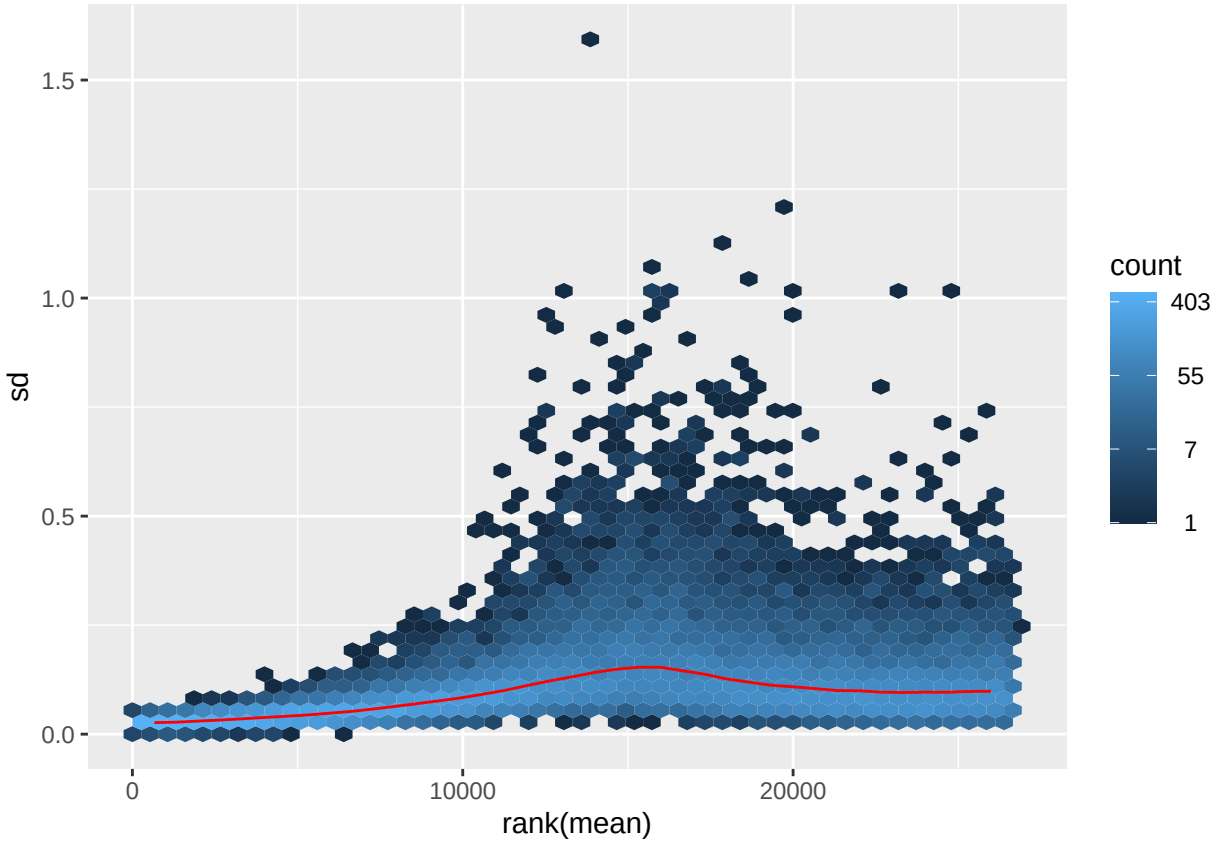


expression versus sd

"Preprocess"

Two methods are commonly used remove the dependence of the variance on the mean, variance stabilizing transformations (VST) (Tibshirani 1988; Huber et al. 2003; Anders and Huber 2010), and regularized logarithm (rlog), which places a prior on the sample differences (Love, Huber, and Anders 2014). rlog applies a variance-stabilizing transformation, which stabilizes the variance across expression values. since the standard deviation is not homogeneous for different levels of gene expression, we want to work with variance stabilizing transformations. for the larger expressed values have higher sd as a result of log transformation. generation of higher standard deviation.

```
# Let's calculate rlog values and take another look.
rltfd <- rlog(dds, blind=FALSE)
meanSdPlot(assay(rltfd))
```



Normalization

Before we can compare gene counts between samples and perform DE analysis, differences in sequencing depth per sample and RNA composition across samples need to be compensated. DESeq2 uses the median of ratios method where the counts are divided by sample-specific size factors determined by median ratio of gene counts relative to geometric mean per gene. non normalized data is very sensitive to outliers. one or two genes can be massively expressed and this will affect the entire data. so we need to identify those genes that are stable for some generations and are not too much expressed or too low and we use them as our reference plus the sequence to make the data a bit more comparable.

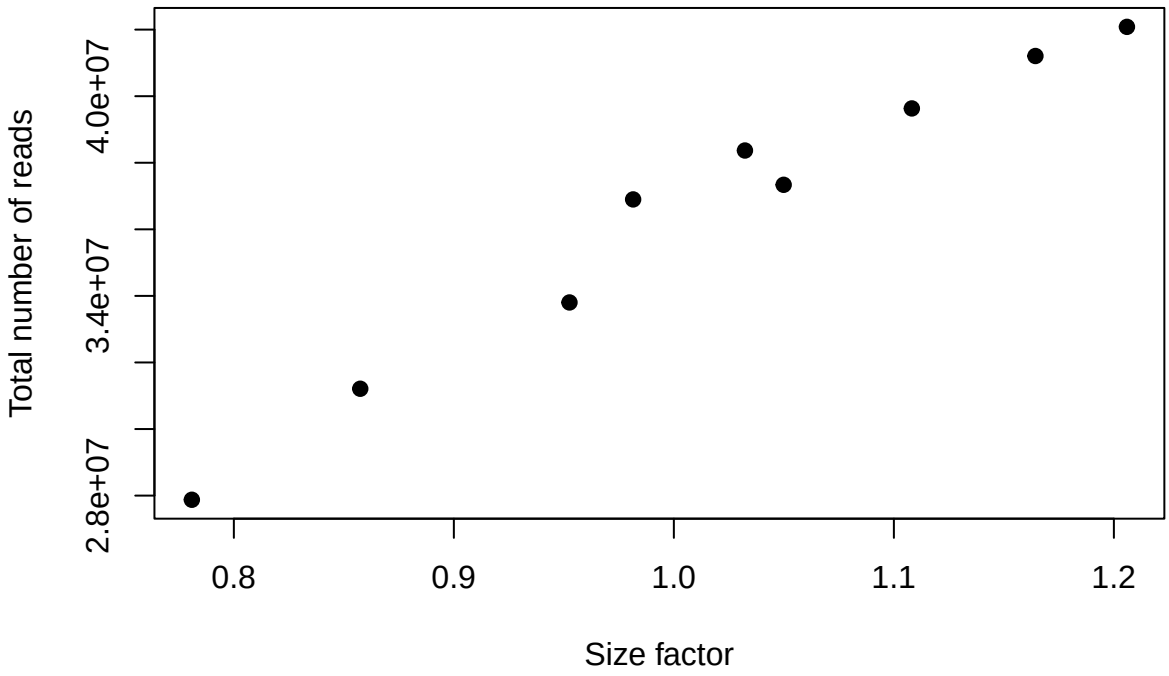
```
dds <- estimateSizeFactors(dds)
sizeFactors(dds)
```

```
##     I0736     I0749     I0760     I0801     I0841     I0862     I0865     I0867
## 1.1643462 1.0323102 1.2059462 0.9525591 0.9814876 0.7809045 1.0499075 0.8574515
##     I0874
## 1.1081705
```

We can compare these size factors to the total number of reads in each sample. Samples with more reads have larger size factors so that dividing the counts by the size factors accounts for the differences in sequencing depth between samples.

```
plot(sizeFactors(dds),
    colSums(counts(dds, normalized=F)),
```

```
    xlab = 'Size factor',
    ylab = 'Total number of reads',
    pch = 19)
```



LECTURE 2: Explorative analysis.

Load the libraries

```
library(DESeq2)
```

Computing the PCA

A principal component analysis is a good way to inspect similarities among the data, to e.g. spot strong confounding factors.

PCA requires normal-distributed data. We can use the rlog function to transform our data by library size and apply log2 transformation. One way to perform the PCA is then using the function prcomp.

```
rltfd.pca <- prcomp(t(assay(rltfd)), scale = TRUE)
```

PCA analysis

The first step is often to evaluate the complexity of the data, i.e. how much of the variance is explained by the first component, the second, ... We can use a scree plot to visualize that. the first graph explains the variability in the data

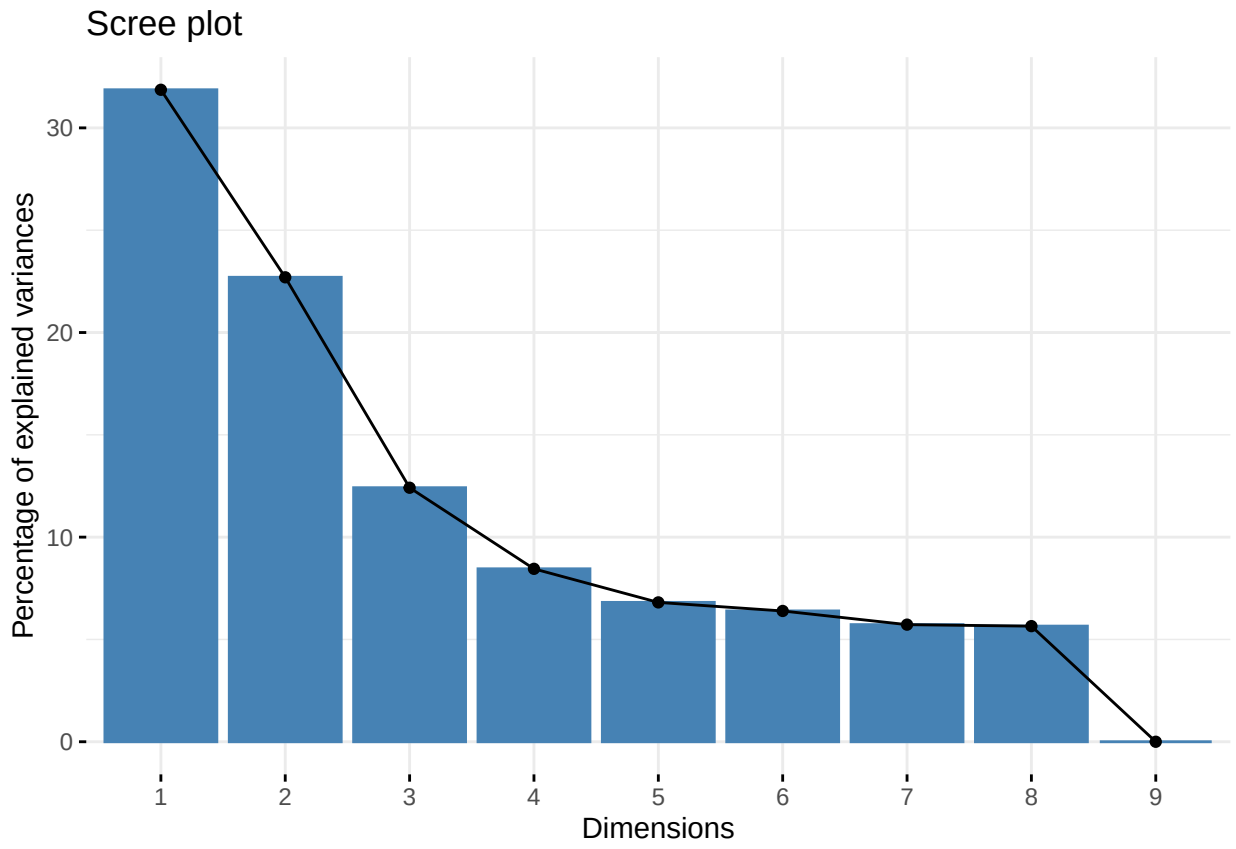
```
require(factoextra)
```

```
## Loading required package: factoextra
```

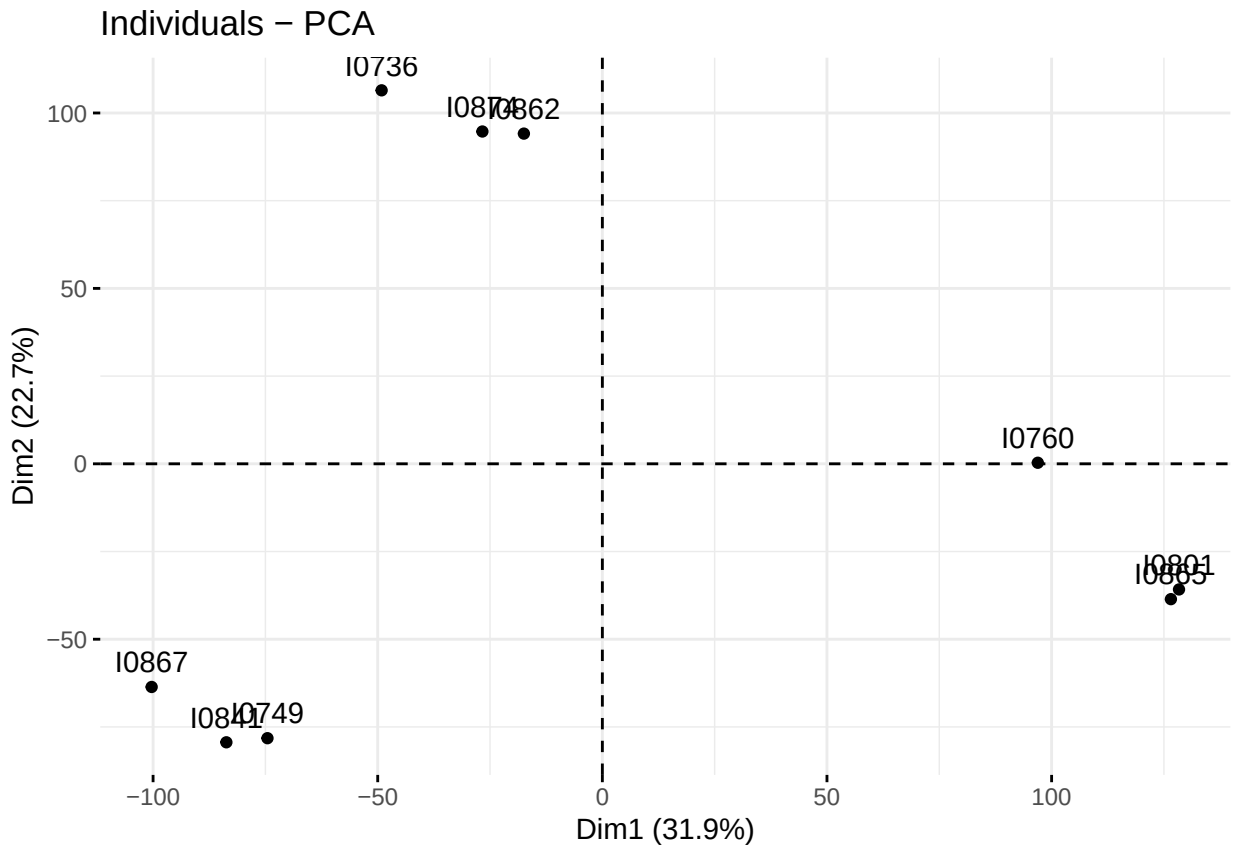
```
## Loading required package: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
fviz_eig(rltfd.pca)
```



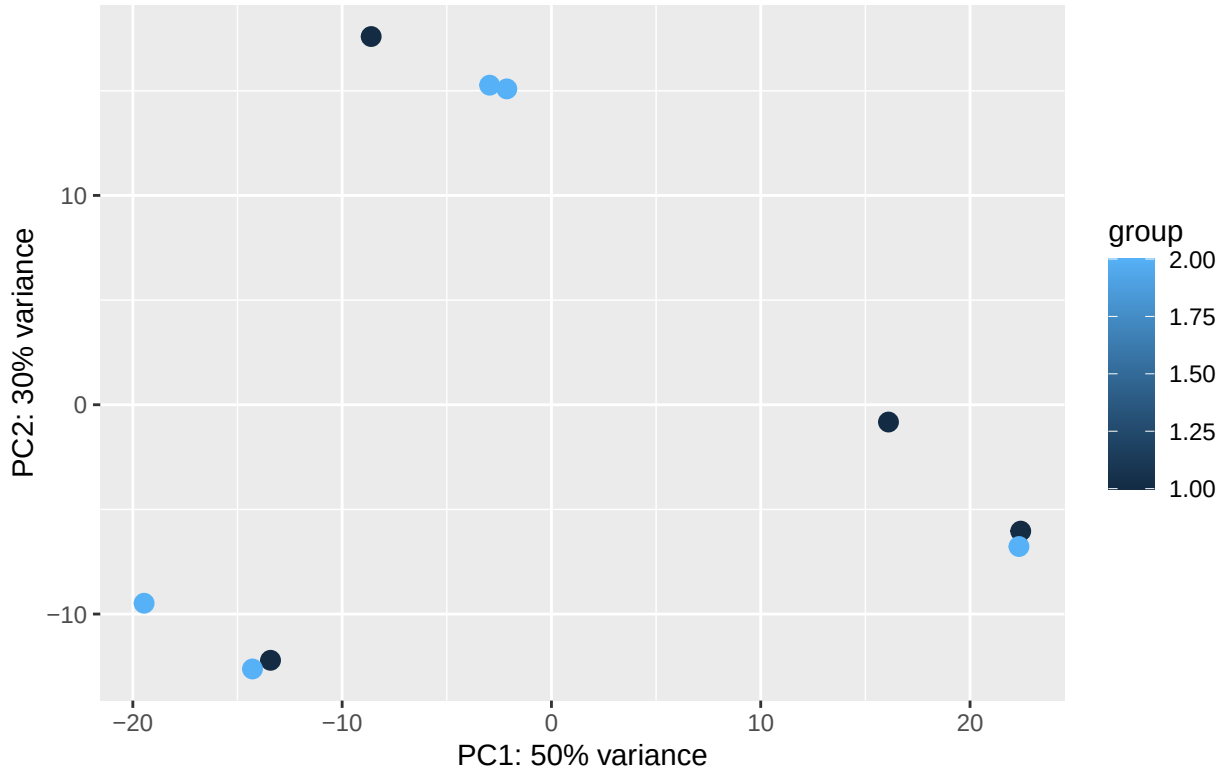

```
fviz_pca_ind(rltfd.pca)
```



Next, lets see if our samples group by sequencing batch, which would mean we have a technical confounding factor. Fortunately, it does not look like that is the case. # change of scale

```
plotPCA(rltfd, intgroup = 'sequencing_batch',ntop=26656)
```

```
## using ntop=26656 top features by variance
```




```
?plotPCA
```

```
## Help on topic 'plotPCA' was found in the following packages:
##
##   Package               Library
##   DESeq2                /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library
##   BiocGenerics          /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library
##
##
## Using the first match ...
```

```
getMethod("plotPCA","DESeqTransform")
```

```
## Method Definition:
##
## function (object, ...)
## {
##     .local <- function (object, intgroup = "condition", ntop = 500,
##         returnData = FALSE, pcsToUse = 1:2)
##     {
##         message(paste0("using ntop=", ntop, " top features by variance"))
##         rv <- rowVars(assay(object))
##         select <- order(rv, decreasing = TRUE)[seq_len(min(ntop,
##             length(rv)))]
##         pca <- prcomp(t(assay(object)[select, ]))
```

```
##         percentVar <- pca$sdev^2/sum(pca$sdev^2)
##         if (!all(intgroup %in% names(colData(object)))) {
##             stop("the argument 'intgroup' should specify columns of colData(dds)")
##         }
##         intgroup.df <- as.data.frame(colData(object)[, intgroup,
##             drop = FALSE])
##         group <- if (length(intgroup) > 1) {
##             factor(apply(intgroup.df, 1, paste, collapse = ":"))
##         }
##         else {
##             colData(object)[[intgroup]]
##         }
##         pcs <- paste0("PC", pcsToUse)
##         d <- data.frame(V1 = pca$x[, pcsToUse[1]], V2 = pca$x[,
##             pcsToUse[2]], group = group, intgroup.df, name = colnames(object))
##         colnames(d)[1:2] <- pcs
##         if (returnData) {
##             attr(d, "percentVar") <- percentVar[pcsToUse]
##             return(d)
##         }
##         ggplot(data = d, aes_string(x = pcs[1], y = pcs[2], color = "group")) +
##             geom_point(size = 3) + xlab(paste0(pcs[1], ": ",
##             round(percentVar[pcsToUse[1]] * 100), "% variance")) +
##             ylab(paste0(pcs[2], ": ", round(percentVar[pcsToUse[2]] *
##                 100), "% variance")) + coord_fixed()
##     }
##     .local(object, ...)
## }
## <bytecode: 0x162e91d30>
## <environment: namespace:DESeq2>
##
## Signatures:
##         object
## target  "DESeqTransform"
## defined "DESeqTransform"
```

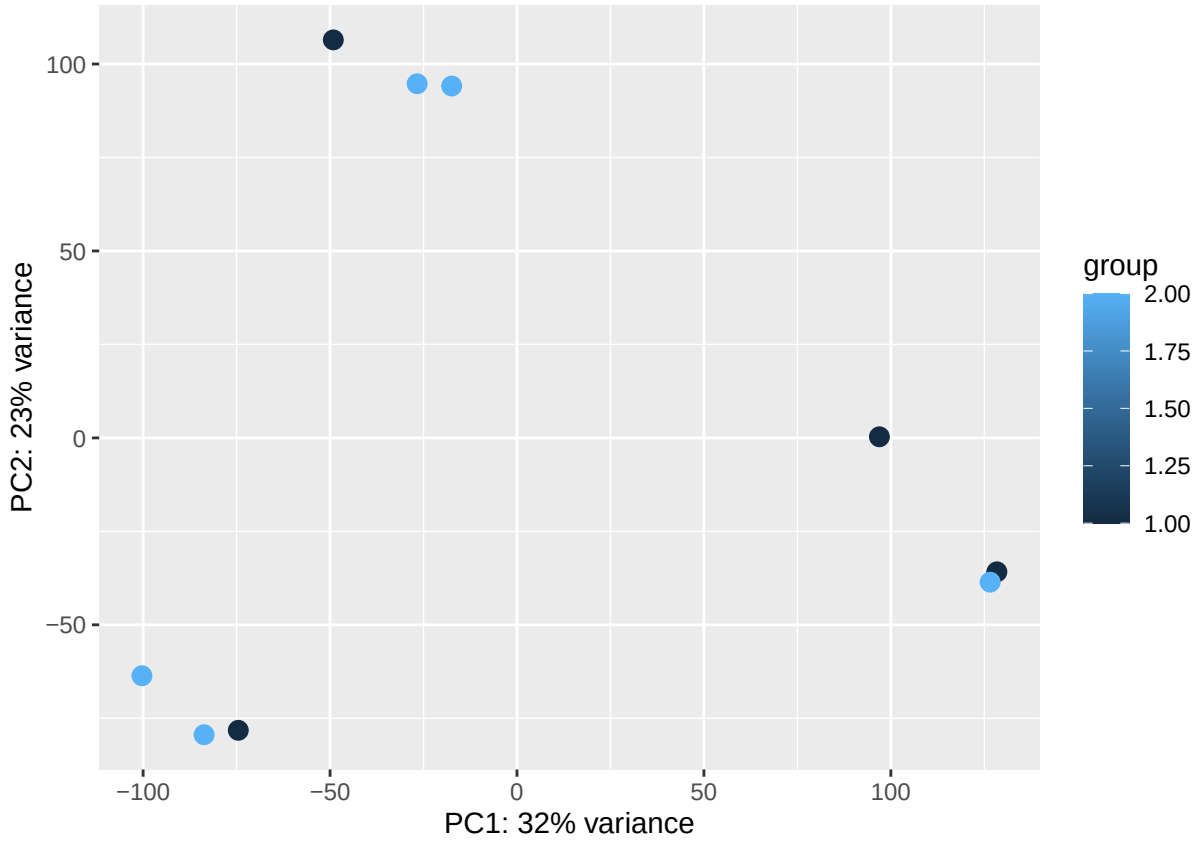
```
object=rltfd
intgroup = 'sequencing_batch'
ntop=26656
returnData = FALSE

      rv <- rowVars(assay(object))
      select <- order(rv, decreasing = TRUE)[seq_len(min(ntop,
            length(rv)))]
      #pca <- prcomp(t(assay(object)[, ]))
      pca <- prcomp(t(assay(object)[, ]),scale=TRUE)
      percentVar <- pca$sdev^2/sum(pca$sdev^2)

      if (!all(intgroup %in% names(colData(object)))) {
            stop("the argument 'intgroup' should specify columns of colData(dds)")
      }
        intgroup.df <- as.data.frame(colData(object)[, intgroup,
            drop = FALSE])
        group <- if (length(intgroup) > 1) {
```

```
        factor(apply(intgroup.df, 1, paste, collapse = ":"))
    } else {
        colData(object)[[intgroup]]
    }
    d <- data.frame(PC1 = pca$x[, 1], PC2 = pca$x[, 2], group = group,
        intgroup.df, name = colnames(object))
    if (returnData) {
        attr(d, "percentVar") <- percentVar[1:2]
        return(d)
    }
    ggplot(data = d, aes_string(x = "PC1", y = "PC2", color = "group")) +
        geom_point(size = 3) + xlab(paste0("PC1: ", round(percentVar[1] *
        100), "% variance")) + ylab(paste0("PC2: ", round(percentVar[2] *
        100), "% variance")) + coord_fixed()
```

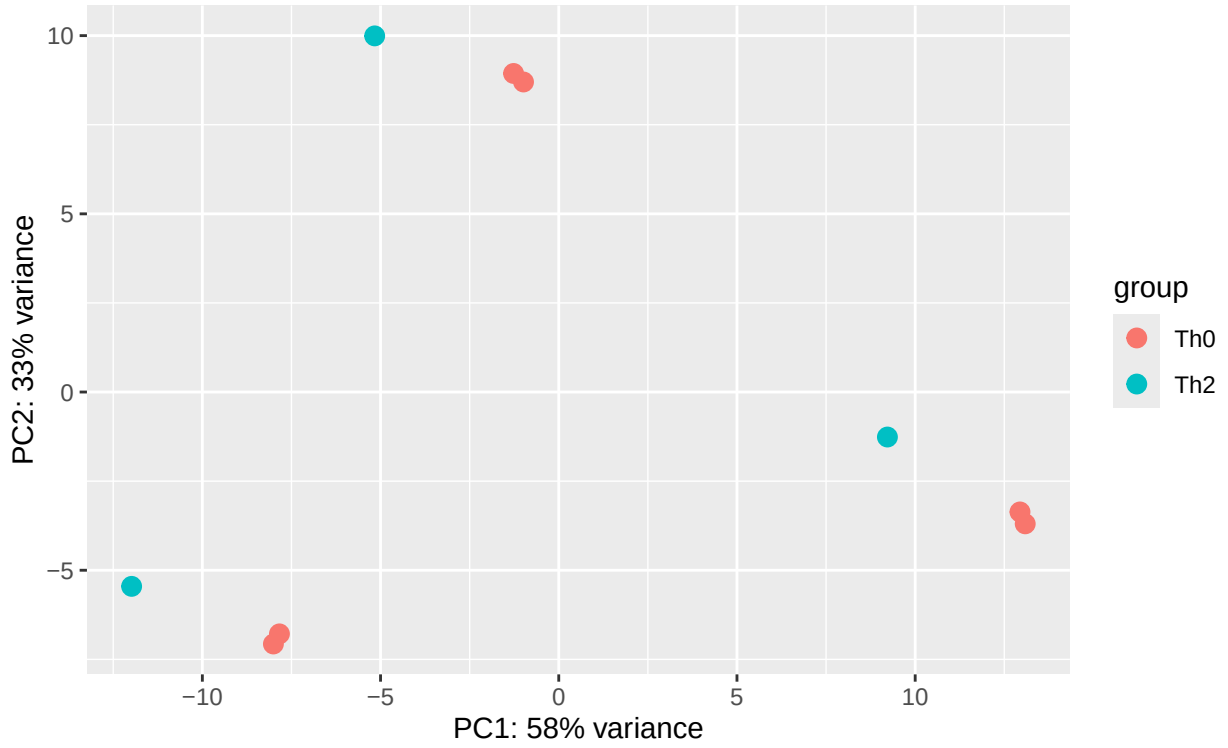
```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Secondly, we can check if our samples group by treatment/cytokine condition, which would mean we have a strong biological signal in our data that we can look forward to analyse. Interestingly, this is not the case. So we have to look deeper into the data.

```
plotPCA(rltfd, intgroup = 'cytokine_condition')
```

```
## using ntop=500 top features by variance
```



LECTURE 3: Differential Expression.

Load the libraries

```
library(DESeq2)
#BiocManager::install("EnhancedVolcano")
library(EnhancedVolcano)
```

```
## Loading required package: ggrepel
```

```
library(pheatmap)
```

DESeq object

We can now use the DESeqDataSetFromMatrix that we have created in lecture 1 to create the DESeq object and run the analysis.

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

```
## using pre-existing size factors
```

```
## estimating dispersions
```

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

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

```
## final dispersion estimates
```

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

Differential expression

This is the long-awaited table holding the estimated base mean expression, log-fold change and p-value for the differential expression for each of our genes.

```
res <- results(dds)

dim(res)
```

```
## [1] 26656     6
```

```
res
```

```
## log2 fold change (MLE): cytokine condition Th2 vs Th0
## Wald test p-value: cytokine condition Th2 vs Th0
## DataFrame with 26656 rows and 6 columns
##                   baseMean log2FoldChange     lfcSE       stat    pvalue
##                  <numeric>      <numeric> <numeric>  <numeric> <numeric>
## ENSG00000227232  88.95983      0.0223454  0.274133  0.0815127  0.935034
## ENSG00000278267   5.97875     -1.3153424  0.867645 -1.5159912  0.129522
## ENSG00000233750   1.53816      1.0398040  1.087345  0.9562782  0.338932
## ENSG00000268903  66.27922      0.1048259  0.377488  0.2776933  0.781248
## ENSG00000269981  50.99808     -0.1707947  0.355286 -0.4807244  0.630712
## ...                   ...            ...       ...        ...       ...
## ENSG00000276345  60.59217       0.227696  2.330880  0.0976868  0.922181
## ENSG00000277856   2.46383      -0.961505  0.975319 -0.9858360  0.324214
## ENSG00000275063   9.71461       0.523047  0.497266  1.0518458  0.292870
## ENSG00000271254  30.86838       0.123534  0.477830  0.2585305  0.795998
## ENSG00000275405   1.80653      -0.479047  1.294148 -0.3701642  0.711260
##                      padj
##                 <numeric>
## ENSG00000227232         1
## ENSG00000278267         1
## ENSG00000233750        NA
## ENSG00000268903         1
## ENSG00000269981         1
```

```
## ...                   ...
## ENSG00000276345         1
## ENSG00000277856         1
## ENSG00000275063         1
## ENSG00000271254         1
## ENSG00000275405        NA
```

Analysis of the outcome

How many significantly differentially expressed genes do we find for the current contrast Th2 vs Th0 in CD4+ memory cells?

```
sum(res$padj <= 0.01 &
      abs(res$log2FoldChange) > 1, na.rm = TRUE)
```

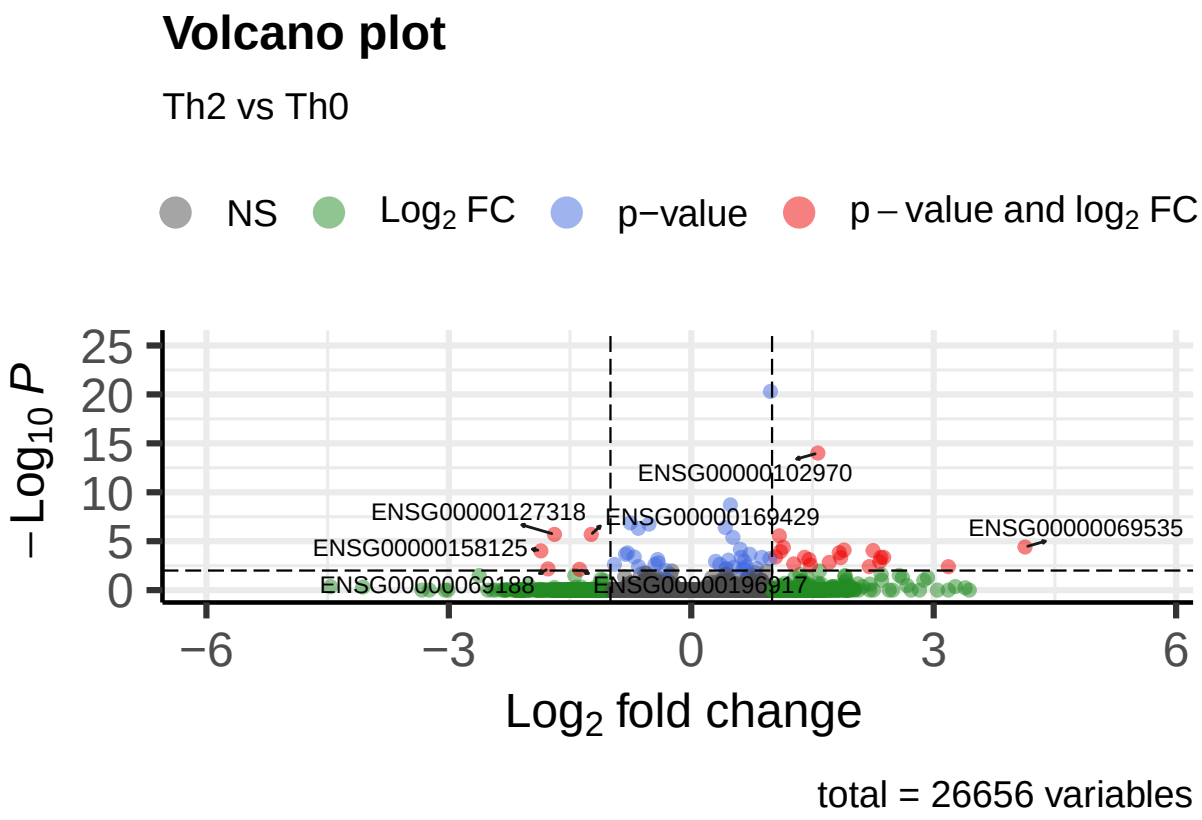
```
## [1] 25
```

Visualization 1: Volcano Plot

Volcano plots are a helpful tool to visualize the log-fold changes and corresponding differential expression p-values

```
EnhancedVolcano(res, lab = rownames(res),
                x = 'log2FoldChange', y = 'padj',
                subtitle = 'Th2 vs Th0', labSize = 3,
                pCutoff = 0.01,
                FCcutoff = 1,
                drawConnectors = TRUE)
```

```
## Warning: ggrepel: 18 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Visualization 2: Heatmap

Lastly, it can be helpful to visualize the individual expression values for a set of genes of interest over the different samples. This is commonly done using heatmaps.

First, we select our genes of interest, here the differentially expressed genes.

```
DEG.idx <- which(res$padj <= 0.01 &
      abs(res$log2FoldChange) > 1)
res[DEG.idx,]
```

```
## log2 fold change (MLE): cytokine condition Th2 vs Th0
## Wald test p-value: cytokine condition Th2 vs Th0
## DataFrame with 25 rows and 6 columns
##                   baseMean log2FoldChange     lfcSE      stat      pvalue
##                  <numeric>      <numeric> <numeric> <numeric>   <numeric>
## ENSG00000158125   55.3416       -1.86081  0.345086  -5.39230 6.95609e-08
## ENSG00000115896  259.0013        1.04492  0.206308   5.06484 4.08737e-07
## ENSG00000091181   38.7084        1.83120  0.345994   5.29257 1.20607e-07
## ENSG00000168386  138.2252        1.08978  0.179103   6.08468 1.16723e-09
## ENSG00000169429 1005.5434       -1.23768  0.201104  -6.15443 7.53456e-10
## ...                   ...            ...       ...       ...         ...
## ENSG00000092068  214.5830        1.47597  0.323572   4.56147 5.07972e-06
## ENSG00000102970  202.8555        1.56695  0.176911   8.85726 8.20053e-19
## ENSG00000069188   98.4945       -1.77297  0.409486  -4.32975 1.49276e-05
```

```
## ENSG00000101695 4525.4228        1.26926  0.273535   4.64023 3.48026e-06
## ENSG00000124212  277.0815        1.40168  0.279146   5.02130 5.13230e-07
##                        padj
##                   <numeric>
## ENSG00000158125 9.84982e-05
## ENSG00000115896 4.09964e-04
## ENSG00000091181 1.52803e-04
## ENSG00000168386 2.80976e-06
## ENSG00000169429 2.01524e-06
## ...                     ...
## ENSG00000092068 2.77907e-03
## ENSG00000102970 9.87016e-15
## ENSG00000069188 6.77994e-03
## ENSG00000101695 2.09442e-03
## ENSG00000124212 4.49525e-04
```

```
df <- as.data.frame(colData(dds)[,c("cytokine_condition","donor_id", "sequencing_batch")])
```

Secondly, we use the pheatmap function. Importantly, this function can perform a clustering of rows to group genes with similar expression patterns, as well as clustering of columns to group samples with similar patterns. Here, we see that samples cluster nicely by treatment (cytokine_condition). Also note, that the expression values are scaled by row, i.e.gene to compensate for differences in based expression and focus on expression changes between samples.

```
pheatmap(assay(rltfd)[DEG.idx,], annotation_col=df,
    treeheight_row = 0, treeheight_col = 0, scale = "row")
```