Complete bulk RNA-seq pipeline. Part 2: analysis of differential gene expression between two conditions

Author: Tomàs Montserrat Ayuso

September 17, 2023

Introduction

Once the bioinformatic preprocessing of the *fastq* files with the sequencing data has been carried out, we have a *txt* file with the counts of transcripts captured for each gene in each sample.

In this post we will process this matrix to prepare the data for statistical analysis and find out the differentially expressed genes between different conditions.

We should remember that the design of the experiment is as follows: HMLE-Twist cells (express markers of the epithelial-mesenchymal transition; EMT) and HMLE-pBp (do not express EMT markers) were sequenced. We have three biological replicates of each cell line in each experimental condition (with and without salinomycin). The authors of the original study considered the two experimental conditions for each cell line to represent paired samples, probably because these samples were generated on the same day. For more information you can consult this website.

The purpose of this entry is to carry out the statistical analysis of the data to answer the following questions:

1. Which genes respond to treatment in cells of the EMT lineage?
2. Which genes respond differently to treatment between EMT and non-EMT cells?

To do this, we will follow these steps:

- Import the data into R.
- Check the quality of the biological signal and merge the technical replicates.
- Create a `DGEList` object from the `edgeR` package in order to manipulate the expression data and metadata of each sample in a relevant way.
- Filter the underexpressed genes, as they are not of biological interest and will reduce the number of statistical tests to be applied.
- Normalize the data to ensure that the distribution of gene expression in each sample is similar and that the data are comparable.

- Do the statistical tests of the data to answer the two questions raised. Before that, we will transform the data with the *voom* method so that they are suitable for working with linear models.

Loading the data

Let's start the analysis by loading all the packages we will need to use:

```
library(DESeq2)
library(limma)
library(edgeR)
library(RNAseqQC)
library(ggplot2)
library(ggrepel)
library(dplyr)
library(pheatmap)
library(AnnotationDbi)
library(org.Hs.eg.db)
library(Glimma)
```

Now, we'll import the count matrix with the `read.table()` function and take a look at the object:

```
counts <- read.table(
  file = "../quantification/genes_count_matrix.txt",
  header = TRUE
)
> counts[1:5, 1:3]
           Geneid alignment.SRR8435265.bam alignment.SRR8435266.bam
1 ENSG00000160072                     120                     196
2 ENSG00000234396                       2                       1
3 ENSG00000225972                       0                       0
4 ENSG00000224315                       0                       0
5 ENSG00000198744                       2                       8
```

The counts object is a `data.frame` that contains a column for the identifier of each gene (*Geneid*) and for the number of reads detected in each gene in each technical replicate of each sample.

We'll do some work on formatting this `data.frame` correctly. First, we put the IDs of each gene as row names:

```
rownames(counts) <- counts$Geneid
```

Then we remove the *Geneid* column because we no longer need it and want each variable to represent a different sample:

```
counts$Geneid <- NULL
```

Now we only need to give the rest of the columns the name of the sample identifier:

```
# Empty vector to store the name of the samples
sample_names <- c()

# Recover sample names
for (i in 1:length(colnames(counts))) {
```

```
  sample_names[i] <- strsplit(colnames(counts)[i], split = "[.]")[[1]][2]
}

# Change sample names
colnames(counts) <- sample_names
```

In the previous code, we created an empty vector to store the names we want to give to each sample. In the following `for` loop we iterate each column of the matrix and save the identifier of each sample in the previous vector. Finally, in the last line we change the name of the `data.frame` variables.

The final appearance of the object is this:

```
> counts[1:5, 1:3]
                SRR8435265 SRR8435266 SRR8435267
ENSG00000160072        120        196        206
ENSG00000234396          2          1          0
ENSG00000225972          0          0          0
ENSG00000224315          0          0          0
ENSG00000198744          2          8          4
```

We now have the count matrix ready for analysis. Now we are missing the so-called metadata of the samples. This is a tabulated text file that we must have previously created with information for each sample. In our case, the most important thing about this file is to know which samples are technical replicates, which experimental group they belong to and which samples are paired. Let's import this file into R and take a look at its contents:

```
metadata <- read.table(
  file = "../quantification/metadata.txt",
  header = TRUE
)
> head(metadata)
         run experiment_accession cell_line treatment   group plate   sample
1 SRR8435265           SRX5242845       EMT      mock EMTmock  EMT1 EMTmock1
2 SRR8435266           SRX5242845       EMT      mock EMTmock  EMT1 EMTmock1
3 SRR8435267           SRX5242845       EMT      mock EMTmock  EMT1 EMTmock1
4 SRR8435268           SRX5242845       EMT      mock EMTmock  EMT1 EMTmock1
5 SRR8435269           SRX5242846       EMT      mock EMTmock  EMT2 EMTmock2
6 SRR8435270           SRX5242846       EMT      mock EMTmock  EMT2 EMTmock2
```

We see that we have all the information necessary to carry out the analysis. The first two columns refer to the replicate identifier (*run*) and the sample to which it belongs (*experiment_accession*). The rest of the columns refer to the experimental group of each sample, except for the *plate* column, which gives us the necessary information to match the samples.

Now, we will transform the variables, which are now encoded as character vectors, into factors. To do this, we'll first create a function to automate the process and then call it:

```
to_factor <- function(dataframe, variables_to_convert) {
  for (variable in variables_to_convert) {
    dataframe[, variable] <-  factor(dataframe[[variable]])
  }
  return(dataframe)
}
```

```
metadata <- to_factor(metadata, colnames(metadata))
```

Finally, we need the names of the columns in the `data.frame counts` to match the names of the rows in the `data.frame metadata`. We can achieve this by moving the *run* column to the row name:

```
rownames(metadata) <- metadata$run
```

Before continuing, we will check that the name of the columns in `counts` and the name of the rows in `metadata` match:

```
> all(colnames(counts) %in% rownames(metadata))
[1] TRUE
> all(colnames(counts) == rownames(metadata))
[1] TRUE
```

Merge of technical replicates

Now we can start the analysis. Technical replicates in bulk RNA-seq samples are considered excellent, so it is advisable to merge these replicates and stay only with biological replicates. Although the *fastq* or *bam* files of the technical replicates could have been concatenated, we will choose to add the counts. However, we want to be sure that there is no significant batch effect between the runs.

We will write a function that will not only return the count matrix and the metadata with the collapsed technical replicates, but will also generate some plots so that we can evaluate the presence or not of batch effects between the runs:

```
mergeReplicatesCounts <- function(counts, metadata, replicates.variable, variables.to.remove, group)
{

  # Create a DESeqDataSet object
  dds <- DESeqDataSetFromMatrix(
    countData = counts,
    colData = metadata,
    design = ~ 1
  )

  # Variance stabilizing transformation to moderate variance accross the mean
  vst <- vst(dds, blind = TRUE)

  # Plot to inspect the stabilization of the variance
  vst.plot <- mean_sd_plot(vst)
  vst_mat <- assay(vst)

  # Compute PCA from the VST matrix
  pca <- prcomp(t(vst_mat))

  # Extract loads
  loads <- round(pca$sdev^2/sum(pca$sdev^2) * 100, 1)

  # Prepare labels
  xlab <- c(paste("PC1", loads[1], "%"))
  ylab <- c(paste("PC2", loads[2], "%"))
```

```
  # Create data frame with metadata and PC1 and PC2 values for input to ggplot
  df <- cbind(metadata, pca$x)

  # Create PCA plot
  pca.plot <- ggplot(df, aes_string(
    x="PC1", y="PC2",
    color = replicates.variable,
    shape = group),
    size = 6) + geom_point() +
    geom_text_repel(aes_string(label = replicates.variable), max.overlaps = 200) +
    xlab(xlab) +
    ylab(ylab)

  # Hierarchical clustering
  # Compute pairwise correlation values
  vst_cor <- cor(vst_mat)
  clust.plot <- pheatmap(vst_cor)

  # Collapse technical replicates
  dds.collapsed <- collapseReplicates(
    object = dds,
    groupby = dds[[replicates.variable]],
    run = dds$run
  )

  # Extract new counts matrix
  new.counts <- counts(dds.collapsed, normalized = FALSE)

  # Extract new metadata
  new.metadata <- as.data.frame(colData(dds.collapsed))

  # Remove not needed columns
  new.metadata[, variables.to.remove] <- NULL

  return(list(
    "counts" = new.counts, "metadata" = new.metadata,
    "vst.plot" = vst.plot, "pca.plot" = pca.plot,
    "clust.plot" = clust.plot
    )
  )

}
```

This function creates a `DESeq2` object and uses the `collapseReplicates()` function of this package to aggregate the counts of the technical replicates. The function also generates a plot of the first two principal components as well as a heatmap of the correlations between the samples:

```
collapsed.data <- mergeReplicatesCounts(
  counts = counts,
  metadata = metadata,
  replicates.variable = "sample",
  variables.to.remove = c("run", "experiment_accession"),
  group = "group"
)
```

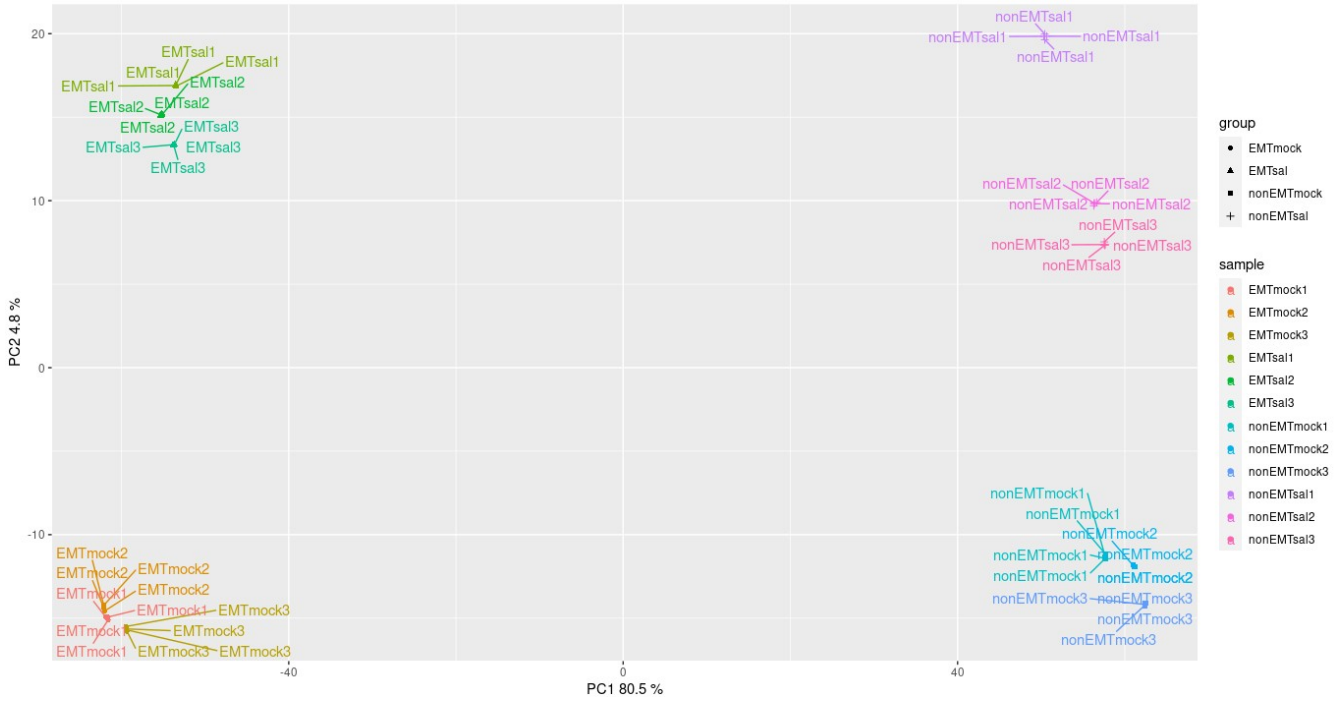
The returned object is a list containing a matrix with the collapsed counts, a `data.frame` with the metadata and three plots: one to assess the normalization made for the calculation of the principal components another of the first two principal components and a heatmap:

```
> names(collapsed.data)
[1] "counts"     "metadata"   "vst.plot"   "pca.plot"   "clust.plot"
```

Let's take a look at the counts and the metadata:

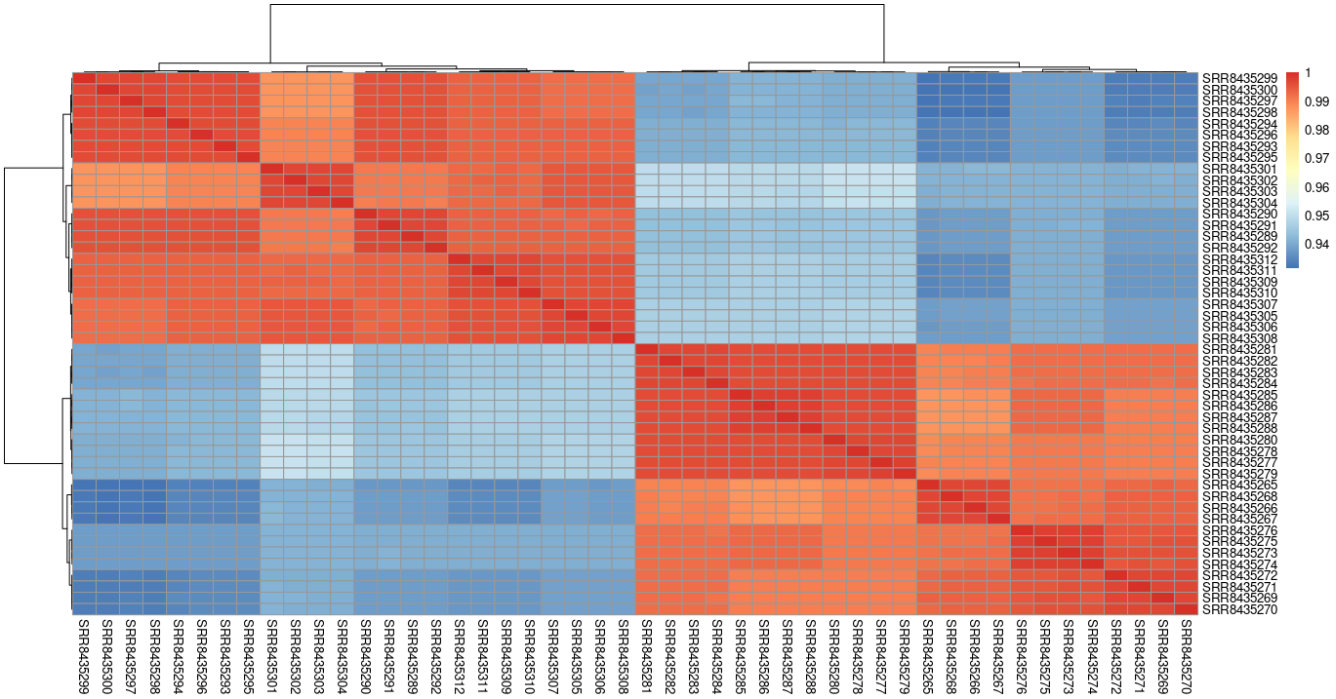
```
> collapsed.data$counts[1:3, 1:3]
                 EMTmock1 EMTmock2 EMTmock3
ENSG00000160072      697      770     1694
ENSG00000234396        3        0        0
ENSG00000225972        1        0        7
> head(collapsed.data$metadata)
         cell_line treatment   group plate   sample                                runsCollapsed
EMTmock1       EMT      mock EMTmock  EMT1 EMTmock1 SRR8435265,SRR8435266,SRR8435267,SRR8435268
EMTmock2       EMT      mock EMTmock  EMT2 EMTmock2 SRR8435269,SRR8435270,SRR8435271,SRR8435272
EMTmock3       EMT      mock EMTmock  EMT3 EMTmock3 SRR8435273,SRR8435274,SRR8435275,SRR8435276
EMTsal1        EMT       sal  EMTsal  EMT1  EMTsal1 SRR8435277,SRR8435278,SRR8435279,SRR8435280
EMTsal2        EMT       sal  EMTsal  EMT2  EMTsal2 SRR8435281,SRR8435282,SRR8435283,SRR8435284
EMTsal3        EMT       sal  EMTsal  EMT3  EMTsal3 SRR8435285,SRR8435286,SRR8435287,SRR8435288
```

We also look at the plots of the first two principal components and the heatmap of the correlations between all the runs. Let's focus first on the first two principal components:



We can clearly see how all the technical replicates of the same sample cluster very close to each other. This confirms that there is no significant batch effect between runs.

Let's see the heatmap now. Although we expect correlations close to 1 between all samples because there should only be significant expression differences in relatively few genes, technical and same-group replicates should be more similar to each other:



Indeed, the technical replicates cluster and the samples are clearly divided into two groups: samples of cells that perform EMT and those that do not.

Before continuing with the analysis, we will save the count matrix and the metadata in different objects and check again that the names of the columns of the count matrix match the names of the rows of the metadata:

```
counts.collapsed <- collapsed.data$counts
metadata.collapsed <- collapsed.data$metadata
> all(colnames(counts.collapsed) %in% rownames(metadata.collapsed))
[1] TRUE
> all(colnames(counts.collapsed) == rownames(metadata.collapsed))
[1] TRUE
```

Data preprocessing

We already have the data in a suitable format: a matrix with the identifier of each gene in the rows and the name of each sample in the columns; and a `data.frame` with the metadata of each sample to be able to assign it an experimental group. In addition, the technical replicates have been collapsed by adding the counts.

In this section we will carry out all the necessary transformations of these data for their statistical treatment under the framework of the linear models offered by the `limma` package.

Creation of a `DGEList` object

The first step is to create an object in R that is suitable for us to have all the data (counts and metadata) organized and synchronized correctly. This object will be a `DGEList` object from the `edgeR` package:

```
dge.list <- DGEList(
  counts = counts.collapsed,
  samples = metadata.collapsed,
  genes = rownames(counts.collapsed)
)
colnames(dge.list$genes) <- "ENSEMBL"
```

A `DGEList` object is like a list that contains information about counts, samples (metadata) and genes. That's why in the second line of code we changed the name of the *genes* column of the `data.frame` `genes` to *ENSEMBL*.

Next, we will add to this `data.frame` another column with the gene symbols. We will do this with the `mapIds()` function of the `AnnotationDbi` package that we loaded at the beginning:

```
dge.list$genes$SYMBOL <- mapIds(
  x = org.Hs.eg.db,
  keys = dge.list$genes$ENSEMBL,
  column = "SYMBOL",
  keytype = "ENSEMBL",
  multiVals = "first"
)
```

The first argument of this function is the database we want to use (in this case `org.Hs.eg.db`, which we loaded at the start). The argument `keys` are the identifiers of the genes we want to convert, `column` the type of identifier we want and `keytype` the type of identifier we entered in `keys`. Finally, `multiVals` specifies that it should return the first occurrence when the ENSEMBL/SYMBOL mapping is not unique.

Removal of underexpressed genes from the dataset

Examining the number of genes with 0 counts in all samples, we see that they are many:

```
> table(rowSums(counts.collapsed == 0) == 12)

FALSE  TRUE
33812 28048
```

Up to 28048 genes have no counts in any of the samples. Genes that are not expressed in any condition or that accumulate very few reads in any or all samples should be removed from the dataset. Biologically speaking, genes that are so poorly expressed in all conditions are not of interest and only add noise to the data. On the other hand, eliminating the genes that are not of interest, and that it is very likely there would be

no significant differences between the conditions, makes it possible to reduce the number of statistical tests to be performed.

To filter the genes that have few counts in all the samples we will use the `filterByExpr()` function of the `edgeR` package, which returns a logical vector for the rows of the count matrix (genes) that must be maintained. By default, the function keeps those genes with at least 10 counts in a sufficient number of samples:

```
keep.exprs <- filterByExpr(dge.list, group = "group")
dge.list <- dge.list[keep.exprs, keep.lib.size = FALSE]
```

Normalization of expression data

In order to proceed with the analysis, we must first normalize the data to avoid bias in the results due to differences in library size between the samples. We will apply `edgeR`'s own TMM normalization which estimates the effective library sizes, or in other words, normalizes the library sizes. This normalization allows the data to be normalized according to the library size of each sample, taking into account the effect of extreme differentially expressed genes.

The function that performs this normalization is `calcNormFactors()`, which returns the normalization factors that will be used downstream to calculate the effective library size, multiplying the original library size of each sample by its normalization factor:

```
dge.list <- calcNormFactors(dge.list, method = "TMM")
```

The estimated factors are automatically stored in the metadata:

```
> dge.list$samples[1:3, 1:5]
           group lib.size norm.factors cell_line treatment
EMTmock1 EMTmock 21014589     1.039857       EMT      mock
EMTmock2 EMTmock 20465413     1.054415       EMT      mock
EMTmock3 EMTmock 28344002     1.029786       EMT      mock
```

Data exploration

The next step before fitting the statistical models and finding the differentially expressed genes between the conditions of interest is to explore the data. The aim is to verify, in a multivariate analysis, that the samples are grouped together logically, that is to say, that the biological replicates cluster and that the experimental condition influences gene expression. In addition, at this point in the analysis we must also study if there are other variables that significantly affect the variability of the data to add them later to the statistical model.

First, we extract the normalized and log2-transformed count matrix:

```
lcpm <- cpm(dge.list, log=TRUE)
```

For the multivariate exploration of the data we will use two techniques: principal component analysis and hierarchical clustering, as we did to verify that the technical replicates were similar. With the following order we will perform the analysis of main components:

```
pca <- prcomp(t(lcpm))
```

Next, we will extract the loadings of each main component. The loadings allow us to know the percentage of data variability explained by each main component:

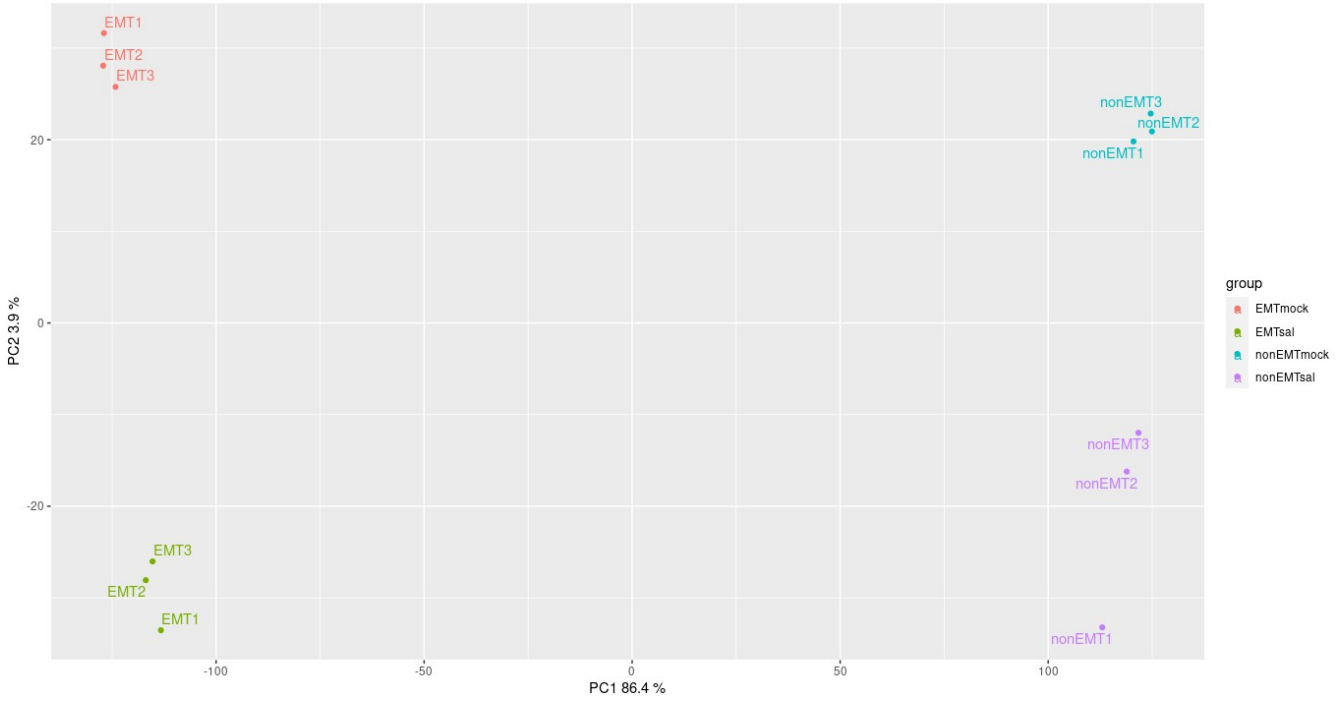
```
loads <- round(pca$sdev^2/sum(pca$sdev^2) * 100, 1)
```

We will create the visualizations with the `ggplot2` package. This package needs the data to be contained in a `data.frame`. Since we want to visualize pairs of principal components while exploring their relationship to metadata variables, we will include in a `data.frame` both the principal component values for each sample and the metadata:

```
df <- cbind(dge.list$samples, pca$x)
```

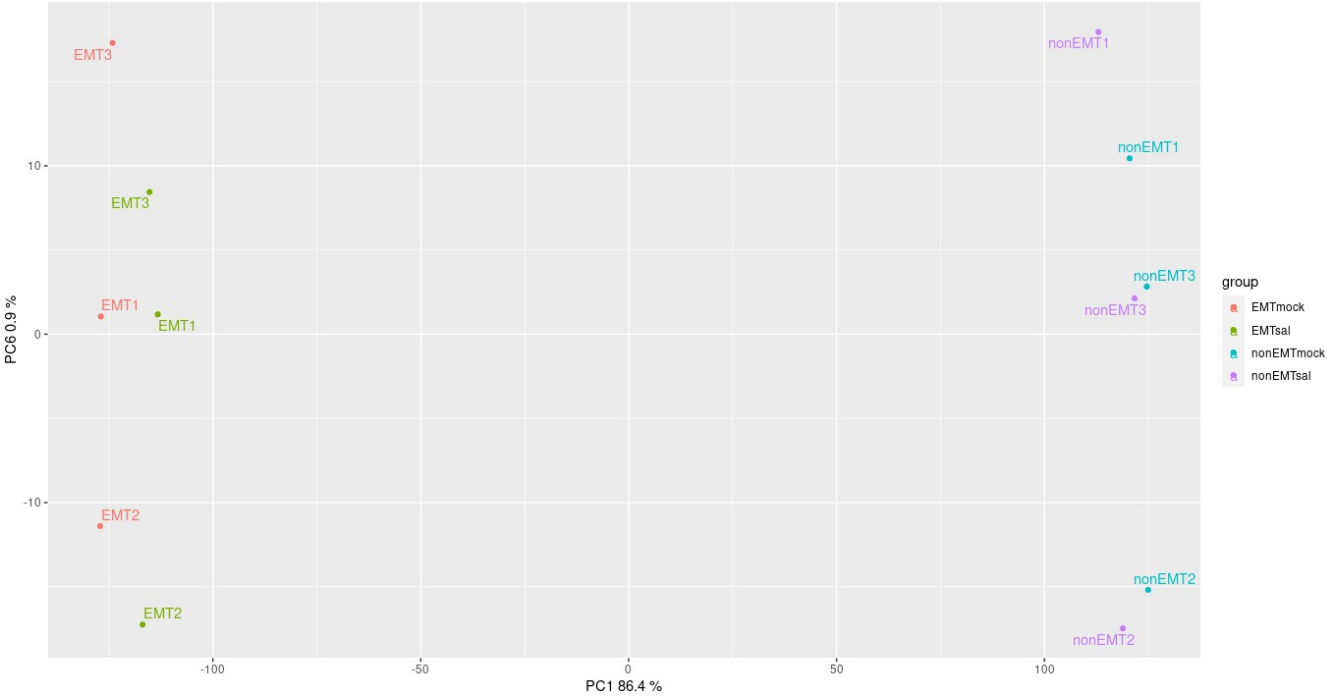
We begin the exploration with a scatter plot of the two principal components, painting the points according to their experimental group:

```
ggplot(
  data = df,
  mapping = aes(x=PC1, y=PC2, color = group),
  size = 6
) + geom_point() +
  geom_text_repel(aes_string(label = plate), max.overlaps = 200) +
  xlab(c(paste("PC1", loads[1], "%"))) +
  ylab(c(paste("PC2", loads[2], "%")))
```



From the above image we can draw two conclusions: that the type of cell line explains most of the variability and that salinomycin treatment has an effect on gene expression. We can explore whether the *plate* variable (which warrants analysis as paired data) has an effect on gene expression by studying the relationship of this variable with higher principal components. Here we will show the scatter plot of the first and sixth principal components:

```
ggplot(
  data = df,
  mapping = aes(x=PC1, y=PC6, color = group),
  size = 6
) + geom_point() +
  geom_text_repel(aes_string(label = plate), max.overlaps = 200) +
  xlab(c(paste("PC1", loads[1], "%"))) +
  ylab(c(paste("PC6", loads[6], "%")))
```



It appears that the sixth principal component correlates sufficiently with the *plate* variable. Although it may seem that the variability explained by this main component is very small (0.9%), we must take into account that 86% of the total variability is explained by the cell line, so even though the general effect of this variable seems small, it can be important enough within each treatment, so it seems a good idea to include the effect of this variable in the statistical model.

When we have more complex experimental designs than this project, it is convenient to explore the relationships of the principal components with different metadata variables, not only studying the relationship with the first two principal components, but also with higher principal components, especially if they explain some non-negligible percentage of the variability, as we did before. If we find variables that explain much of the variability in the data and are not of interest to the study (for example, if we see that the

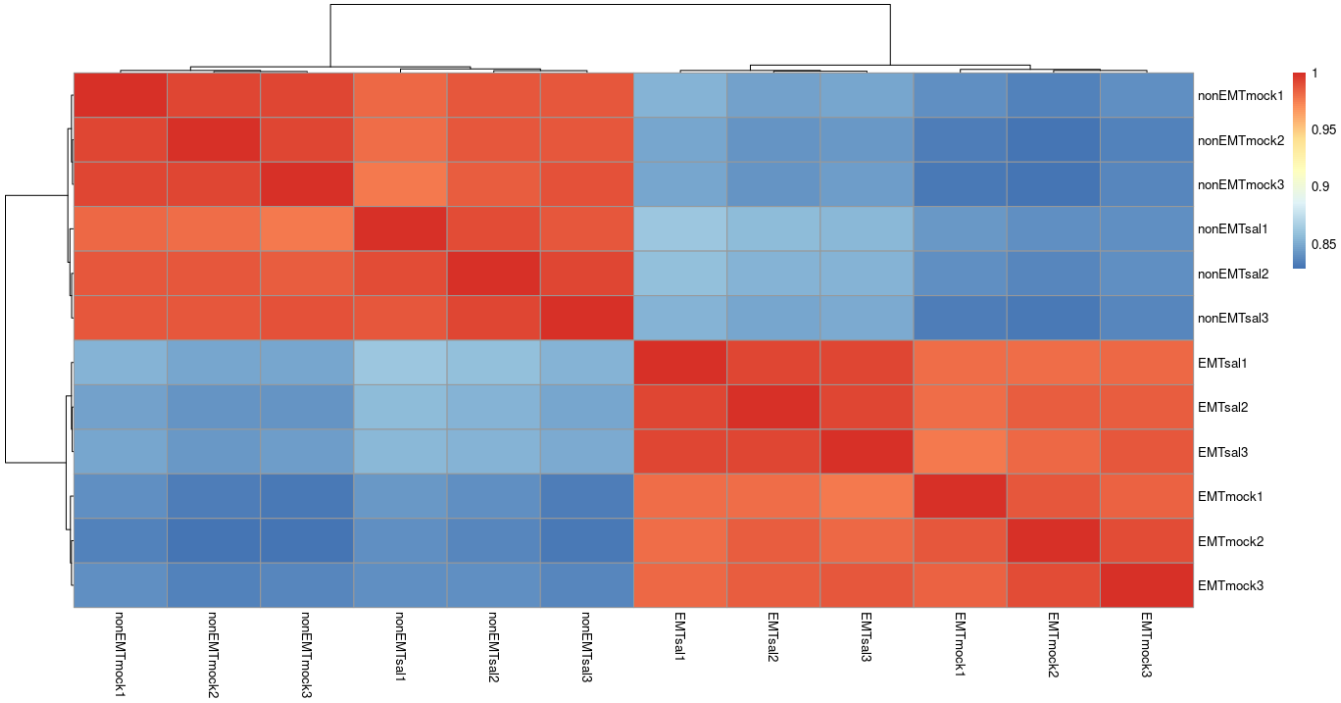
samples are grouped by sex or the day they were processed) then we will need to include these variables in the statistical model.

The hierarchical clustering should confirm the observations we made in the PCA. The prior step is to calculate the correlations between the samples:

```
lcpm_cor <- cor(lcpm)
```

Now, we use the `pheatmap()` function to generate the heatmap and the hierarchical clustering:

```
clust.plot <- pheatmap(lcpm_cor)
```



Indeed, the samples are grouped first by cell line and, second, by treatment, as we observed in the PCA.

Differential expression analysis

We have already done all the necessary preprocessing of the data, as well as explored the data to decide which variables to include in the linear models. So, we can perform the statistical tests to answer the two questions we posed at the beginning:

1. Which genes do cells of the EMT line respond to treatment?

2. Which genes respond differently to treatment between EMT and non-EMT cells?

The design of the experiment is 2x2 factorial, since each sample can be EMT or non-EMT and may or may not have received salinomycin treatment. In our metadata we have the group variable that correctly classifies the samples into the four groups of interest:

```
> dge.list$samples$group
 [1] EMTmock     EMTmock     EMTmock     EMTsal      EMTsal      EMTsal      nonEMTmock nonEMTmock
nonEMTmock
[10] nonEMTsal  nonEMTsal  nonEMTsal
Levels: EMTmock EMTsal nonEMTmock nonEMTsal
```

We will use the `limma` package to perform the statistical tests. This program builds a linear model for each gene to determine whether it is differentially expressed between the conditions of interest or not. In summary, the expression of each gene, according to a linear model, is as follows:

$$\hat{expression} = \hat{\beta}_1 EMTmock + \hat{\beta}_2 EMTsal + \hat{\beta}_3 nonEMTmock + \hat{\beta}_3 nonEMTsal$$

Where the β coefficients represent the average of the expression values for each group.

Design matrix and contrasts

The next step is to build the design and contrast matrices. First, we extract a vector of factor type with the experimental group of each sample:

```
group <- factor(dge.list$samples$group)
> group
 [1] EMTmock     EMTmock     EMTmock     EMTsal      EMTsal      EMTsal      nonEMTmock nonEMTmock
nonEMTmock
[10] nonEMTsal  nonEMTsal  nonEMTsal
Levels: EMTmock EMTsal nonEMTmock nonEMTsal
```

Next, we create the layout matrix with the `model.matrix()` function:

```
design <- model.matrix(~ 0 + group)
```

Specifying that we do not want intercept (`~ 0 + group`), the interpretation of the coefficients of the models will be the one we described in the previous section.

Finally, we give each column of the design matrix the same name as the experimental group it represents:

```
colnames(design) <- levels(group)
```

Let's take a look at the design matrix:

```
> design
   groupEMTmock groupEMTsal groupnonEMTmock groupnonEMTsal
1             1           0                0              0
2             1           0                0              0
3             1           0                0              0
4             0           1                0              0
5             0           1                0              0
6             0           1                0              0
7             0           0                1              0
```

```
8             0           0              1             0
9             0           0              1             0
10            0           0              0             1
11            0           0              0             1
12            0           0              0             1
```

The structure is very simple: there are as many rows as samples and as many columns as experimental groups. Each sample has a 1 in the experimental group to which it belongs.

Now we can specify the contrast matrix. In this matrix we specify the comparisons that interest us as a subtraction:

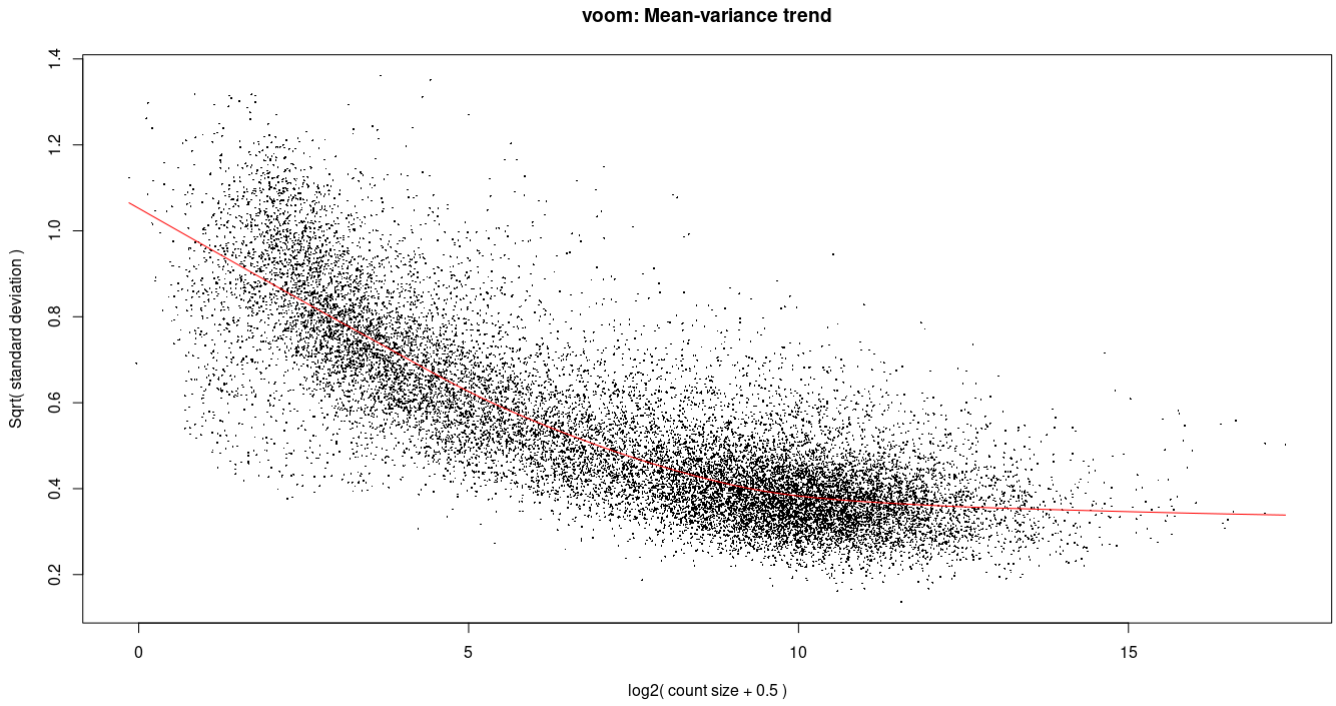
```
contr.matrix <- makeContrasts(
  EMTsal.vs.EMTmock = EMTsal - EMTmock,
  diff = (EMTsal-EMTmock) - (nonEMTsal - nonEMTmock),
  levels = design
)
```

Voom transformation and paired data control

In order for the data to be suitable for modeling the expression linearly, they must be transformed. The `voom()` function from `limma` allows us to perform this transformation:

```
voom.pre <- voom(counts = dge.list, design = design, plot = TRUE)
```

This function returns the log2-normalized counts (log2-CPM), estimates the mean-variance ratio for each gene and calculates its weights for each sample. In this way we obtain data suitable for linear modeling. The weights are used in the construction of the linear models to adjust the heteroscedasticity of the data. This heterocedasticity can be studied by looking at the graph returned by the function. Low mean values of log2-CPM show more variance than high values:



We only have to estimate the effect of the day of sample processing on the counts (*plate*). To do this, we will follow the recommendations of the authors of `limma`: perform the `voom()` transformation once without including the *plate* variable (which encodes the pairing of the samples) as a blocking variable in the model; estimate the correlation of the paired samples (intra-block correlation) with the function `duplicateCorrelation()`; perform the `voom()` transformation again, this time including the *plate* variable as a blocking variable and the intra-block correlation already calculated; and finally call the `duplicateCorrelation()` function again. By repeating the `voom()` transformation with the correlation we can obtain a better estimate of the mean-variance relationship and the weights. Since the `duplicateCorrelation()` function uses the weights, we also get an improvement in the intra-block correlation estimate:

```
voom.pre <- voom(counts = dge.list, design = design, plot = TRUE)
corfit <- duplicateCorrelation(
  object = voom.pre,
  design = design,
  block = voom.pre$targets$plate
)
> corfit$consensus.correlation
[1] 0.3906392
voom <- voom(
  counts = dge.list,
  design = design,
  block = dge.list$samples$plate,
  correlation = corfit$consensus.correlation,
  plot = TRUE
)
corfit <- duplicateCorrelation(voom, design, block = voom$targets$plate)
> corfit$consensus.correlation
```

```
[1] 0.391295
```

The intra-block correlation will allow us to include the *plate* variable as a random effect and adjust the model to the variability due to the day of generation of the samples.

Statistical tests

Now we can fit the linear models. The `lmFit()` function includes automatically the weights in the model:

```
vfit <- lmFit(
  object = voom,
  design = design,
  block = voom$targets$plate,
  correlation = corfit$consensus.correlation
)
```

The coefficients of the models can be seen in the following order:

```
> head(vfit$coefficients)
                  EMTmock     EMTsal nonEMTmock    nonEMTsal
ENSG00000160072  5.353297  5.8798082  7.0033104  6.550524144
ENSG00000225972 -3.655303 -2.1289380 -0.8469264 -0.521327137
ENSG00000198744  0.212315 -0.4707289  1.1880274 -0.001845786
ENSG00000279928 -2.508893 -2.3365608 -1.7191923 -2.737116899
ENSG00000225630  5.938579  6.9646690  7.3340459  7.947646102
ENSG00000157911  4.267858  4.1666555  4.4170283  4.145442225
```

Each coefficient represents the average log2-CPM expression of that gene in each group.

To carry out the hypothesis contrasts in which we were interested, we will call the function `contrasts.fit()`, in which we will include the matrix of contrasts in the second argument:

```
vfit <- contrasts.fit(fit = vfit, contrasts = contr.matrix)
```

We can see the coefficients of the contrasts by accessing `$coefficients`:

```
> head(vfit$coefficients)
                 Contrasts
                  EMTsal.vs.EMTmock      diff
  ENSG00000160072         0.5265117 0.9792979
  ENSG00000225972         1.5263652 1.2007659
  ENSG00000198744        -0.6830439 0.5068293
  ENSG00000279928         0.1723326 1.1902572
  ENSG00000225630         1.0260899 0.4124896
  ENSG00000157911        -0.1012025 0.1703836
```

The *EMTsal.vs.EMTmock* coefficient is easy to interpret. It is simply the subtraction between the model coefficient for *EMTsal* minus the coefficient for *EMTmock*, i.e. the log2FC (FC: fold change). Therefore, the first coefficient means that the average expression value for the first gene is $2^{0.53}=1.44$ (remember that these are values on a log2 scale) times higher in the *EMTsal* group than in the *EMTmock* group. Positive

values of log2FC indicate that there is a greater expression in the first group, while negative values indicate the opposite.

On the other hand, the *diff* coefficient is a bit more complicated. Remember that when we specified the matrix of contrasts we put that *diff = (EMTsal-EMTmock) - (nonEMTsal - nonEMTmock)*. The interpretation of the sign in this case is more difficult since there are many possible combinations. The absolute value of the coefficient, however, will help us find the genes that respond differently to the treatment depending on the cell line. The greater the absolute value of the coefficient of this contrast, the greater the difference of the effect of the treatment of that gene between the cell lines.

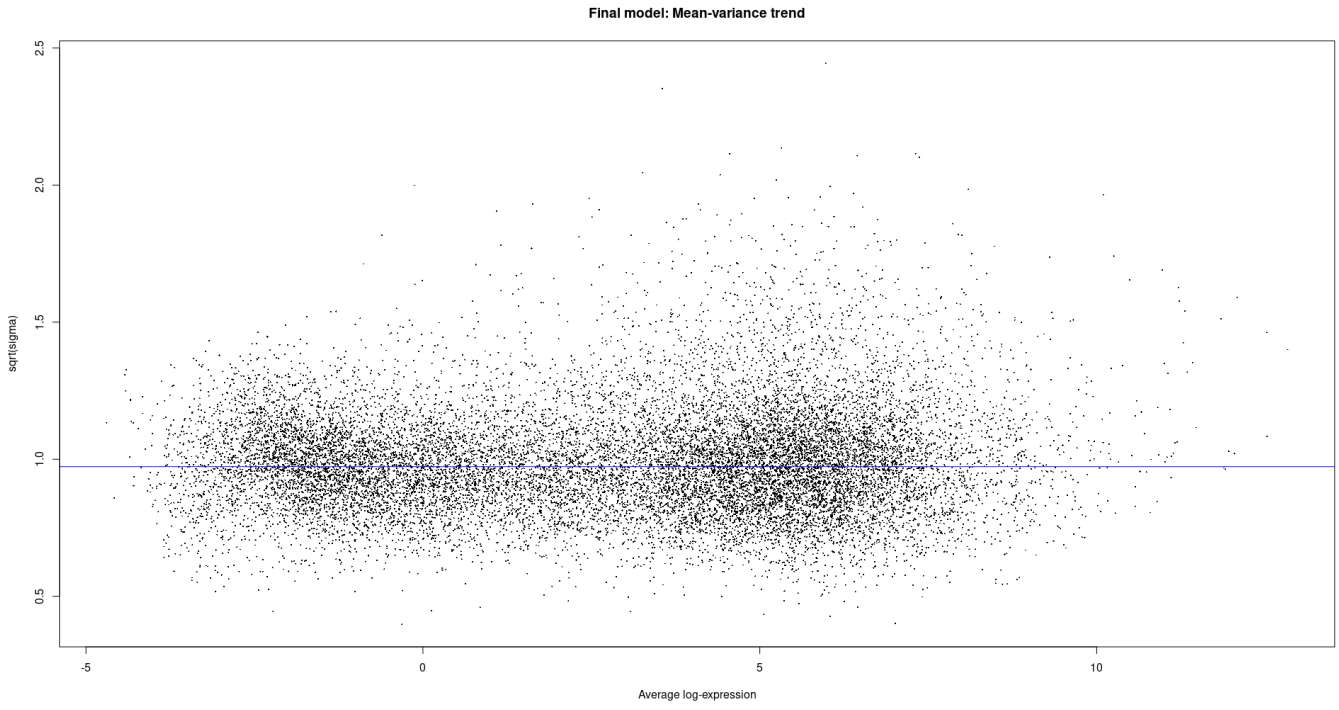
The null hypothesis of the statistical tests (t-test) that we will perform is to decide if the coefficients of the contrasts are significantly different from zero.

Although we already have the models fitted, before we can decide which genes are differentially expressed between conditions (or respond significantly differently to treatment based on cell line), we will use the `eBayes()` function to get an improved estimate of the variance of each gene and therefore of the t-test statistic (called moderated t-statistic). The function uses information from the expression of all genes to improve estimates for each individual gene:

```
efit <- eBayes(vfit)
```

In the following plot we can see that the heteroscedasticity of the data has disappeared when applying the weights calculated by `voom()` in the statistical models:

```
plotSA(efit, main="Final model: Mean-variance trend")
```



Once the tests are done, we can see the results using the `decideTests()` function:

```
> summary(decideTests(object = efit, adjust.method = "BH", p.value = 0.05))
       EMTsal.vs.EMTmock  diff
Down                3174  2069
NotSig             11985 14113
Up                  2815  1792
```

The `adjust.method` argument selects the method to adjust the p-value (Benjamini and Hochberg in our case) and `p.value` to the threshold value of the adjusted p-value from which to reject the null hypothesis.

The numbers in the *Down* and *Up* rows are the number of genes differentially expressed (or responding significantly differently to treatment depending on the cell line) in each contrast. Sometimes, however, it is convenient to look for those genes in which the coefficients of the contrasts are significantly different, but in addition the log2FC is above a minimum. In other words, it is often of interest that the observed differences are biologically significant.

A statistically correct way to find genes that are differentially expressed (or respond significantly differently to treatment depending on the cell line) and with a minimum log2FC is to do so with the `treat()` function. This function calculates p-values from the moderated t-statistic with a log2FC minimum. Instead of checking if log2FC is non-zero, `treat()` checks if it is greater (in absolute value) than a minimum:

```
tfit <- treat(vfit, lfc = 1)
> summary(decideTests(object = tfit, adjust.method = "BH", p.value = 0.05))
       EMTsal.vs.EMTmock  diff
Down                  18    26
NotSig             17894 17933
Up                    62    15
```

We asked `treat()` to check if log2FCs are greater than 1 with the `lfc` argument. Notice how the number of differentially expressed genes (or that behave differently to treatment depending on the cell line) is much lower.

Results

To get a `data.frame` with the results of the analysis we only need to call the `topTreat()` function (for the results of the `treat()` function). Let's do it for the EMTsal vs EMTmock contrast:

```
top.diff <- topTreat(fit = tfit, coef = "EMTsal.vs.EMTmock", n = Inf)
```

The `n` argument specifies the maximum number of genes to display. We have specified that it returns the maximum number. By default, the function returns the results by calculating the p-value adjusted by the Benjamini and Hochberg method and the genes ordered by p-value. In the following lines of code we will remove genes with a non-significant adjusted p-value and sort the table based on decreasing logFC:

```
top.tfit.EMTsal.vs.EMTmock <- top.tfit.EMTsal.vs.EMTmock %>%
  filter(adj.P.Val < 0.05) %>%
```

```
  arrange(desc(logFC))
```

Finally, we will rename the rows to numbers:

```
rownames(top.tfit.EMTsal.vs.EMTmock) <- c(1:nrow(top.tfit.EMTsal.vs.EMTmock))
```

Now we will do the same but for the contrast of the differences between cell lines. Unlike the previous results, we will order the genes based on the absolute logFC value:

```
top.tfit.diff <- topTreat(tfit, coef = "diff", n = Inf)
top.tfit.diff <- top.diff %>%
  filter(adj.P.Val < 0.05) %>%
  arrange(desc(abs(logFC)))
rownames(top.tfit.diff) <- c(1:nrow(top.tfit.diff))
```

The two `data.frames` contain the answers to the two questions we asked ourselves at the beginning:

- Which genes do cells of the EMT lineage respond to treatment? Let's show the first six lines of the `data.frame`:

```
> head(top.tfit.EMTsal.vs.EMTmock)
          ENSEMBL  SYMBOL    logFC    AveExpr         t      P.Value    adj.P.Val
1 ENSG00000109625     CPZ 4.330086 -2.9979253  6.330650 6.083185e-06 3.644639e-03
2 ENSG00000112379 ARFGEF3 4.057010  1.6977527  5.805998 1.584817e-05 6.624537e-03
3 ENSG00000169429   CXCL8 3.874474  1.7538305  5.115950 5.910896e-05 1.713588e-02
4 ENSG00000142552    RCN3 3.820515  0.6927719 18.569002 3.116616e-12 5.601805e-08
5 ENSG00000139269   INHBE 3.374912  2.3476290  8.715071 1.240304e-07 2.229322e-04
6 ENSG00000128342     LIF 3.138688  2.2993831  5.033519 6.932829e-05 1.947042e-02
```

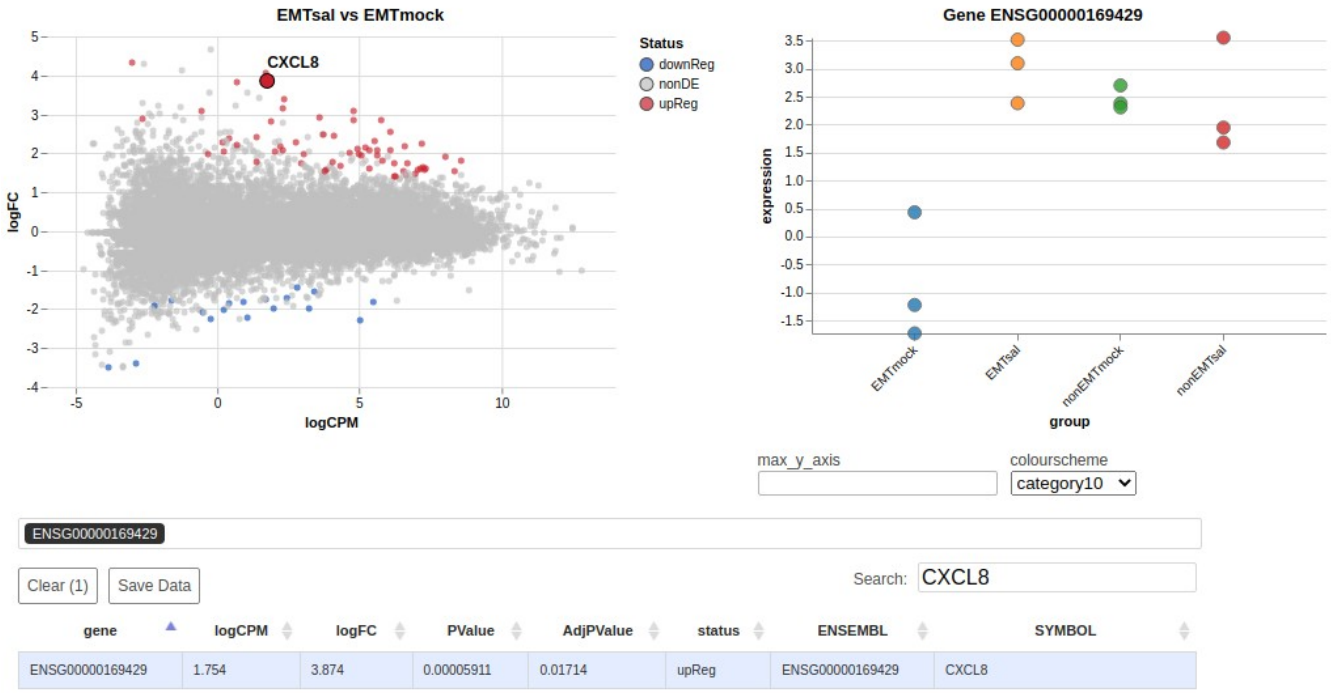
We can create an interactive visualization of these results with the `Glimma` package. First we extract the result of the tests for each gene with the function `decideTests()` and save the output in a variable:

```
dt <- decideTests(tfit)
```

Now we call the `glimmaMA()` function to generate the display:

```
glimmaMA(
  x = tfit,
  counts = voom$E,
  transform.counts = "none",
  groups = voom$targets$group,
  coef = "EMTsal.vs.EMTmock",
  main = "EMTsal vs EMTmock",
  status = dt[, "EMTsal.vs.EMTmock"]
)
```

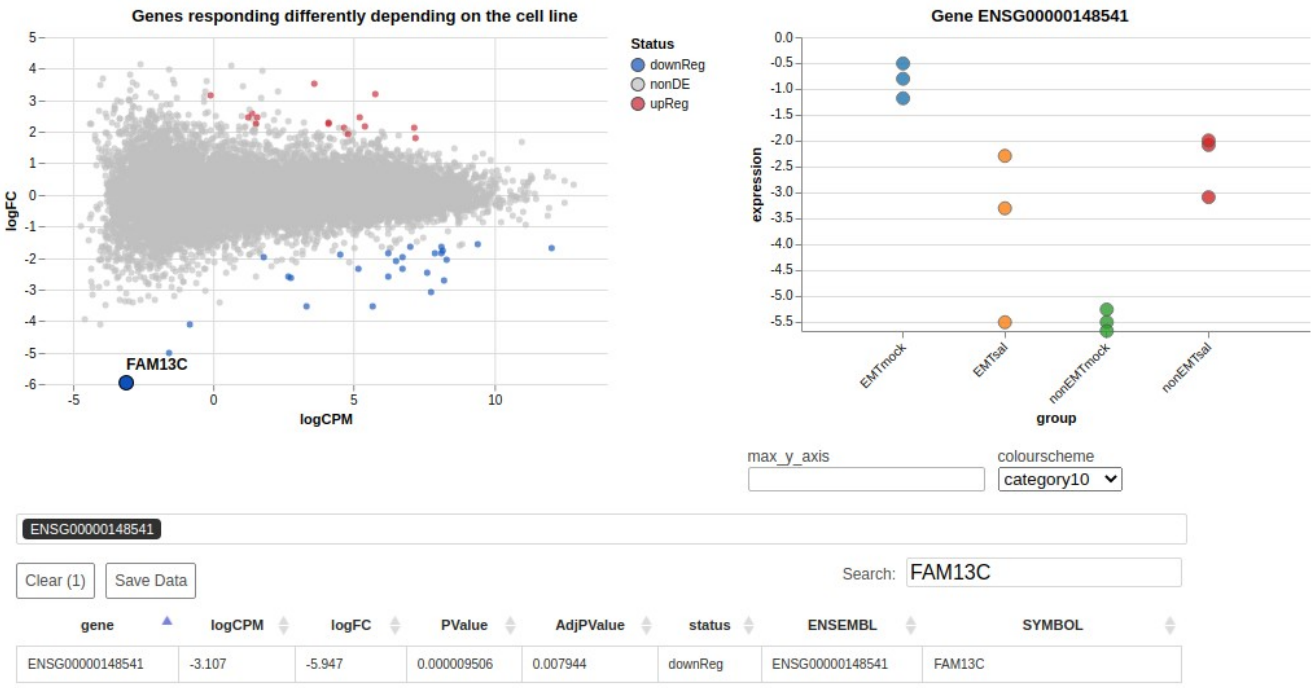
The execution of the previous command will generate an interactive plot where, on the left, we will have an MA plot (logFC between the contrasted groups against the average expression of all the samples of each gene) and, on the right, a dot plot with the counts of the gene we choose for each sample. The `counts` argument specifies the count matrix to use. Finally, `coef` and `status` determine the contrast to display. Here are the results for the CXCL8 gene:



- Which genes respond differently to treatment between EMT and non-EMT cells? The results for the first six genes:

```
> head(top.tfit.diff)
          ENSEMBL  SYMBOL     logFC    AveExpr          t      P.Value    adj.P.Val
1 ENSG00000148541  FAM13C -5.947224 -3.1069805  -6.091308 9.505585e-06 7.943649e-03
2 ENSG00000173930 SLCO4C1 -5.014336 -1.5529773  -4.832120 1.043780e-04 4.937079e-02
3 ENSG00000051523    CYBA -4.103989 -0.8202956  -5.466992 3.004055e-05 1.830706e-02
4 ENSG00000106772  PRUNE2 -3.556923  3.3412457  -5.632724 2.189806e-05 1.513829e-02
5 ENSG00000101187 SLCO4A1  3.540173  3.5889060   7.165047 1.429200e-06 2.261213e-03
6 ENSG00000131069   ACSS2 -3.520258  5.6690097 -14.961936 7.270477e-11 1.306796e-06
```

Let's view the results for the FAM13C gene:



gene	logCPM	logFC	PValue	AdjPValue	status	ENSEMBL	SYMBOL
ENSG00000148541	-3.107	-5.947	0.000009506	0.007944	downReg	ENSG00000148541	FAM13C

Conclusion

In this post we have seen how to carry out a basic analysis of the data from a bulk RNA-seq experiment that has allowed us to answer the two initial questions we had raised.

The previous exploratory analysis of the data is very important. First, to check that the different technical replicates are comparable. Second, to explore which variables contribute the most variability to the data and decide if it is necessary to include variables that are not of interest for the study in the statistical model. And third, to verify that the variables of interest (in this case the treatment with salinomycin and the cell line) have an effect on gene expression.

Although there are different frameworks for analyzing RNA-seq data (such as `edgeR`, `DESeq2` or `limma`) in this post we have decided on the linear models offered by `limma` because it allows us more flexibility when constructing the design matrix. For example, thanks to this flexibility we were able to incorporate the effect of the day of sample processing in the model thanks to the `duplicateCorrelation()` function.

Based on the gene lists obtained and the expression data, different fundamental functional analyzes can be performed, such as, for example, finding the GO terms (gene ontology) overrepresented in the different gene lists (GO enrichment analysis) or finding gene sets of interest that are differentially expressed in some of the experimental conditions (GSEA).

All the code used in this analysis is on the blog's GitHub.