RNA-seq analysis in R

Contents

Disclaimer

The content of this tutorial is largely taken and adapted from the COMBINE RNA-seq Workshop, and from two tutorials using the DESeq2 package (tutorial 1 and tutorial 2). We encourage you to visit these webpages to get extra material, including lecture slides and introductory R material if needed.

Setup

The code used to generate this document along with the data that you will need to reproduce the results presented here are available from the github repository `VoisinneG/RNA_seq_analysis_in_R`

Clone this repository from the terminal with the command:

```
git clone "https://github.com/VoisinneG/RNA_seq_analysis_in_R"
```

First things first : create a R project to store data, code, output tables and figures in separate sub-folders. Just click on "File > New Project. . . "

Create a new project. Add directories `./data`, `./R`, `./output` and `./figures`. Create a new R script file `script.R` in the `./R` directory. We will use it to copy paste useful command lines below. We can re-run all commands sequentially by clicking on the `source` button in the top right corner of the R studio editor.

Report

You will have to report your analysis in the same format as this document. This document has been built from a **Rmarkdown** file using the `knitr` package. You can download the original `.Rmd` file here which could

provide a useful starting point. You can find more information about **Rmarkdown** on the official website and in this cheatsheet .

Data

The data for this tutorial comes from a Nature Cell Biology paper, *EGF-mediated induction of Mcl-1 at the switch to lactation is essential for alveolar cell survival* (Fu et al. 2015). Both the raw data (sequence reads) and processed data (counts) can be downloaded from Gene Expression Omnibus database (GEO) under accession number GSE60450.

This study examines the expression profiles of basal stem-cell enriched cells (B) and committed luminal cells (L) in the mammary gland of virgin, pregnant and lactating mice. Six groups are present, with one for each combination of cell type and mouse status. Each group contains two biological replicates. We will first use the counts file as a starting point for our analysis.

Data files are available from: https://figshare.com/s/1d788fd384d33e913a2a

Download the `SampleInfo_Corrected.txt` and `GSE60450_Lactation-GenewiseCounts.txt` files and place them in a `./data` directory within your project. Alternatively, these files are available from the github repository you have just cloned in the `demo_data` folder. You can copy these file in your `./data` folder.

Load data

We can now start by loading the data into the RStudio environment.

```
seqdata <- read.delim("./demo_data/GSE60450_Lactation-GenewiseCounts.txt",
                      stringsAsFactors = FALSE)
sampleinfo <- read.delim("./demo_data/SampleInfo_Corrected.txt")
```

You now have two objects in your environment. You can have a look at them using the `head()` or `View()` commands or by double clicking on an object within the environment. The `seqdata` object contains information about genes (one gene per row), the first column has the Entrez gene id, the second has the gene length and the remaining columns contain information about the number of reads aligning to the gene in each experimental sample.

```
names(seqdata)
```

```
##  [1] "EntrezGeneID"                      "Length"
##  [3] "MCL1.DG_BC2CTUACXX_ACTTGA_L002_R1" "MCL1.DH_BC2CTUACXX_CAGATC_L002_R1"
##  [5] "MCL1.DI_BC2CTUACXX_ACAGTG_L002_R1" "MCL1.DJ_BC2CTUACXX_CGATGT_L002_R1"
##  [7] "MCL1.DK_BC2CTUACXX_TTAGGC_L002_R1" "MCL1.DL_BC2CTUACXX_ATCACG_L002_R1"
##  [9] "MCL1.LA_BC2CTUACXX_GATCAG_L001_R1" "MCL1.LB_BC2CTUACXX_TGACCA_L001_R1"
## [11] "MCL1.LC_BC2CTUACXX_GCCAAT_L001_R1" "MCL1.LD_BC2CTUACXX_GGCTAC_L001_R1"
## [13] "MCL1.LE_BC2CTUACXX_TAGCTT_L001_R1" "MCL1.LF_BC2CTUACXX_CTTGTA_L001_R1"
```

The sampleinfo file contains basic information about the samples that we will need for the analysis today.

```
sampleinfo
```

```
##                            FileName SampleName CellType   Status
## 1  MCL1.DG_BC2CTUACXX_ACTTGA_L002_R1    MCL1.DG    basal   virgin
## 2  MCL1.DH_BC2CTUACXX_CAGATC_L002_R1    MCL1.DH    basal   virgin
## 3  MCL1.DI_BC2CTUACXX_ACAGTG_L002_R1    MCL1.DI    basal pregnant
## 4  MCL1.DJ_BC2CTUACXX_CGATGT_L002_R1    MCL1.DJ    basal pregnant
## 5  MCL1.DK_BC2CTUACXX_TTAGGC_L002_R1    MCL1.DK    basal  lactate
## 6  MCL1.DL_BC2CTUACXX_ATCACG_L002_R1    MCL1.DL    basal  lactate
```

```
## 7  MCL1.LA_BC2CTUACXX_GATCAG_L001_R1    MCL1.LA  luminal   virgin
## 8  MCL1.LB_BC2CTUACXX_TGACCA_L001_R1    MCL1.LB  luminal   virgin
## 9  MCL1.LC_BC2CTUACXX_GCCAAT_L001_R1    MCL1.LC  luminal pregnant
## 10 MCL1.LD_BC2CTUACXX_GGCTAC_L001_R1    MCL1.LD  luminal pregnant
## 11 MCL1.LE_BC2CTUACXX_TAGCTT_L001_R1    MCL1.LE  luminal  lactate
## 12 MCL1.LF_BC2CTUACXX_CTTGTA_L001_R1    MCL1.LF  luminal  lactate
```

Formatting

We will be manipulating and reformatting the counts matrix into a suitable format for downstream analysis. The first two columns in the `seqdata` dataframe contain annotation information. We need to make a new matrix `countdata` containing only the counts, but we can store the gene identifiers (the `EntrezGeneID` column) as rownames.

```
# Remove first two columns from seqdata
countdata <- seqdata[,-(1:2)]
# Store EntrezGeneID as rownames
rownames(countdata) <- seqdata[,1]
```

The column names of `countdata` are the sample names which are pretty long so we'll shorten these to contain only the relevant information about each sample. We will use the `substr` command to extract the first 7 characters and use these as the colnames.

```
colnames(countdata) <- substr(colnames(countdata),start=1,stop=7)
```

Note that the column names are now the same as `SampleName` in the `sampleinfo` file. This is good because it means our sample information in `sampleinfo` is in the same order as the columns in `countdata`. Let's also simplify `sampleinfo` by putting `SampleName` as row names.

```
rownames(sampleinfo) <- sampleinfo[,2]
sampleinfo <- sampleinfo[, -(1:2)]
table(colnames(countdata)==rownames(sampleinfo))
```

```
##
## TRUE
##   12
```

Filtering

Genes with very low counts across all libraries provide little evidence for differential expression and they interfere with some of the statistical approximations that are used later in the pipeline. They also add to the multiple testing burden when estimating false discovery rates, reducing power to detect differentially expressed genes. These genes should be filtered out prior to further analysis. Here we perform a minimal pre-filtering to keep only genes that have at least 1 read total.

```
keep <- rowMeans(countdata) >= 1
countdata <- countdata[keep, ]
```

If you want to normalize data and perform differential expression analysis, you can jump to section Creating a `DESeqDataSet`

Annotations

The only information we have about genes is their Entrez Gene ID, which is not very informative. We would like to add some annotation information. There are a number of ways to do this. We will demonstrate how to do this using the org.Mm.eg.db package.

First we need to decide what information we want. In order to see what we can extract we can run the columns function on the annotation database.

```
library(org.Mm.eg.db)
columns(org.Mm.eg.db)
```

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

For now, let's get gene symbols and build up our annotation information in a separate data frame using the select function

```
annot <- select(org.Mm.eg.db, keys=rownames(countdata), columns=c("ENTREZID","SYMBOL"))
```

Let's double check that the ENTREZID column matches exactly to our countdata rownames.

```
table(annot$ENTREZID==rownames(countdata))
```

```
##
##  TRUE
## 18178
```

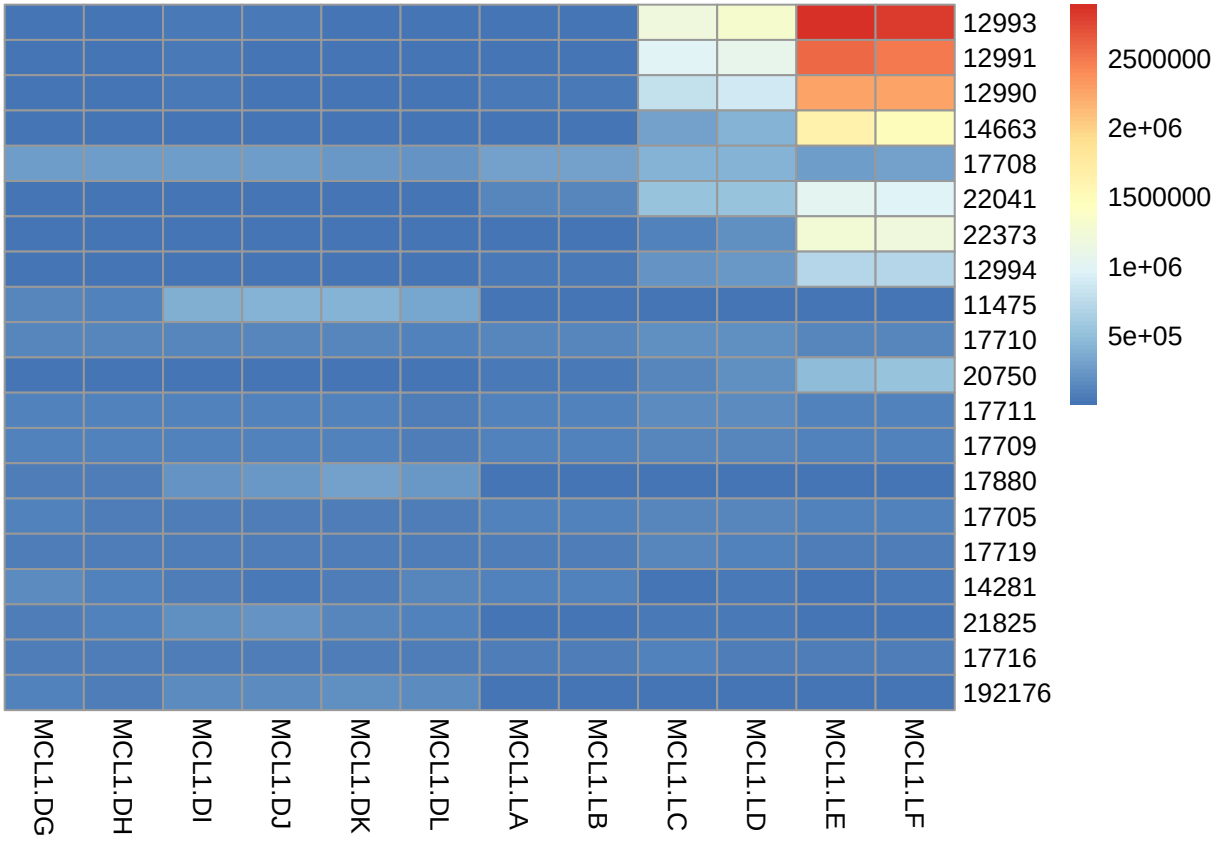
Visualization

We are working we raw count data in this section. Data normalization and transformation will come later.

Heatmaps

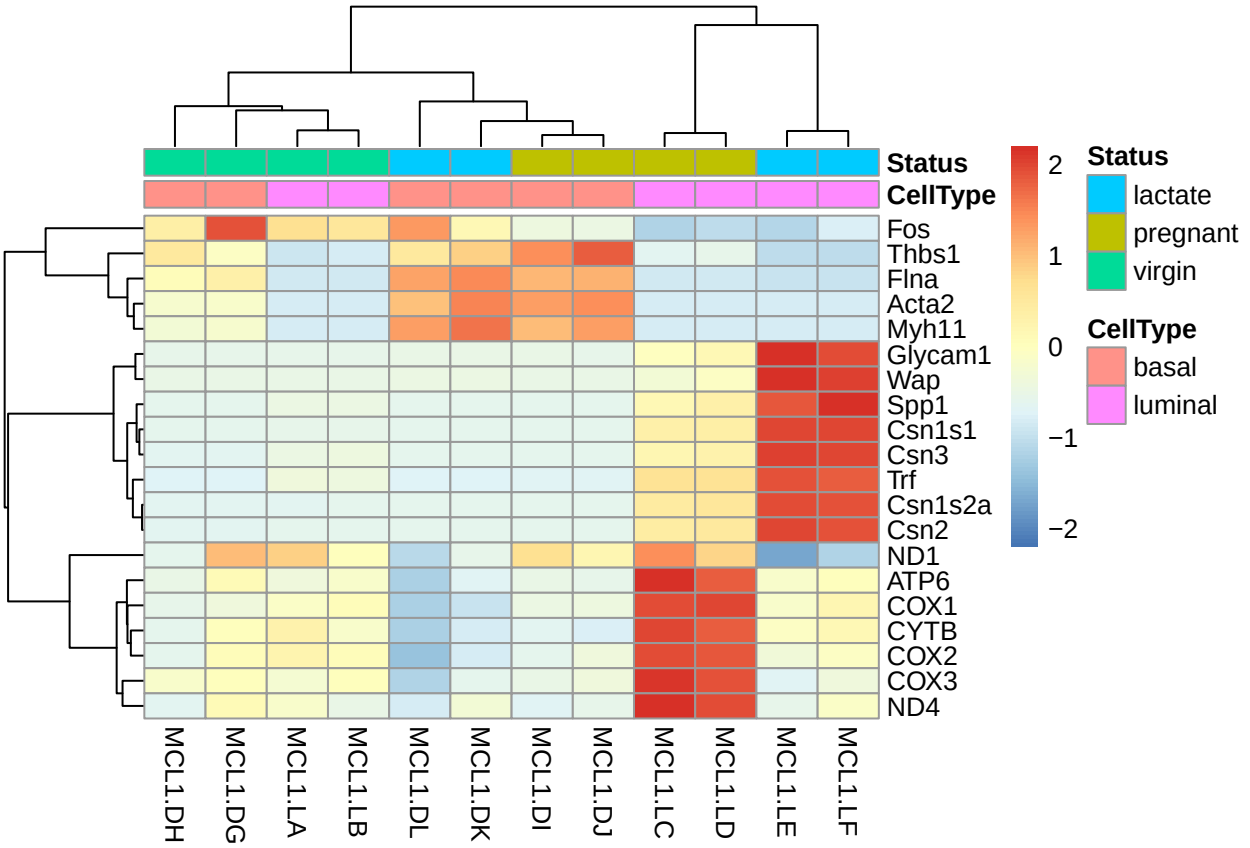
To explore a count matrix, it is often instructive to look at it as a heatmap. Below we show how to produce such a heatmap. We focus on the 20 genes with the highest average counts.

```
library("pheatmap")

select <- order(rowMeans(countdata), decreasing=TRUE)[1:20]

pheatmap(countdata[select,],
         cluster_rows=FALSE,
         cluster_cols=FALSE)
```



Look up the documentation for `pheatmap` in Rstudio. The parameter `annotation_col` allow us to add annotation columns. We'll also cluster rows and columns , scale values by rows and change row labels using gene symbol.

```
pheatmap(countdata[select,],
         show_rownames=TRUE,
         cluster_cols=TRUE,
         annotation_col=sampleinfo[ , c("CellType","Status")] ,
         scale = "row",
         labels_row = annot$SYMBOL[select]
         )
```



If you like interactive objects, you could try the `heatmaply()` function to generate the heatmap.

```
library(heatmaply)

heatmaply(countdata[select,], scale = "row")
```

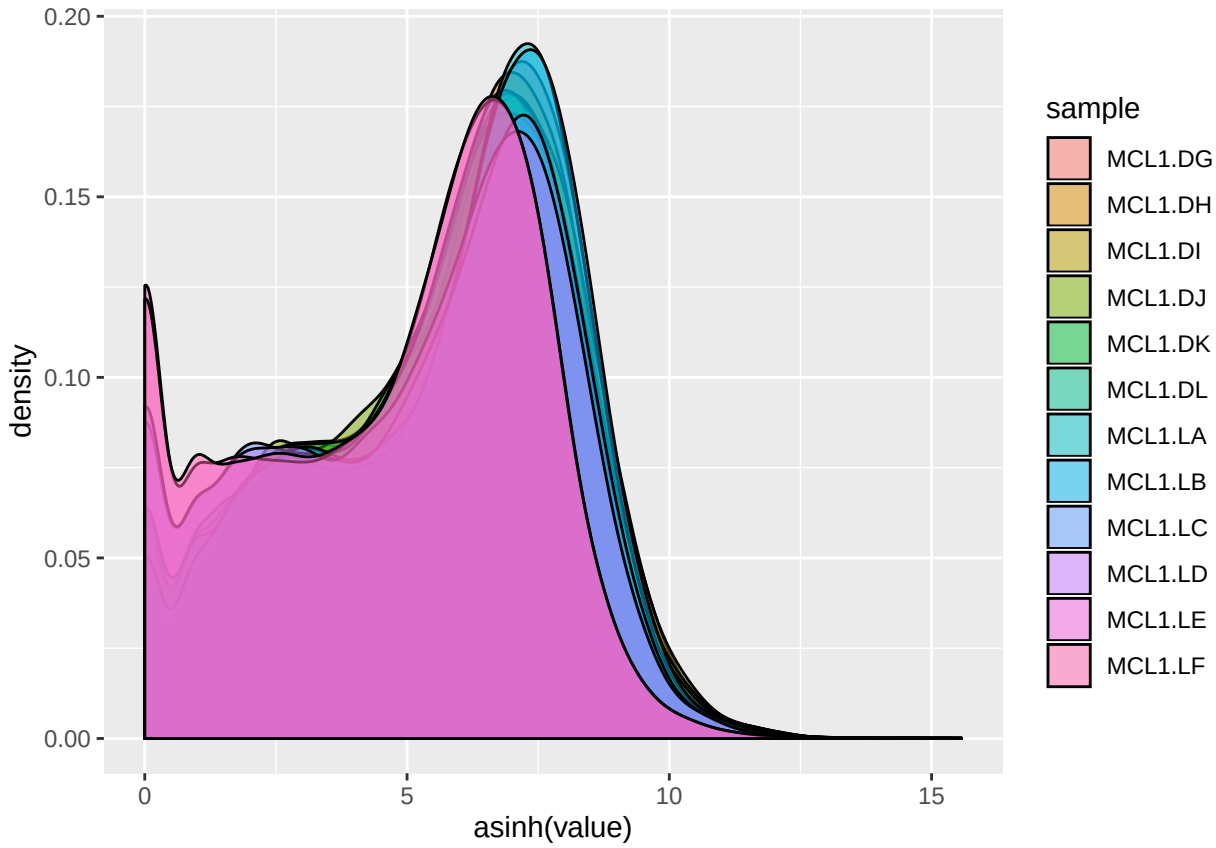
Data visualization with ggplot2

Data visualization with `ggplot2` works with data frames. Here, we convert our data into a data frame where all count values are stored in the same column named `value` (long format). This will allow us to fully exploit `ggplot2` features.

```
library(reshape2)
library(dplyr)
df <- countdata
df$GeneID <- rownames(countdata)
df_melt <- melt(df, id.vars = "GeneID")
df_melt <- rename(df_melt,  sample = variable)
```

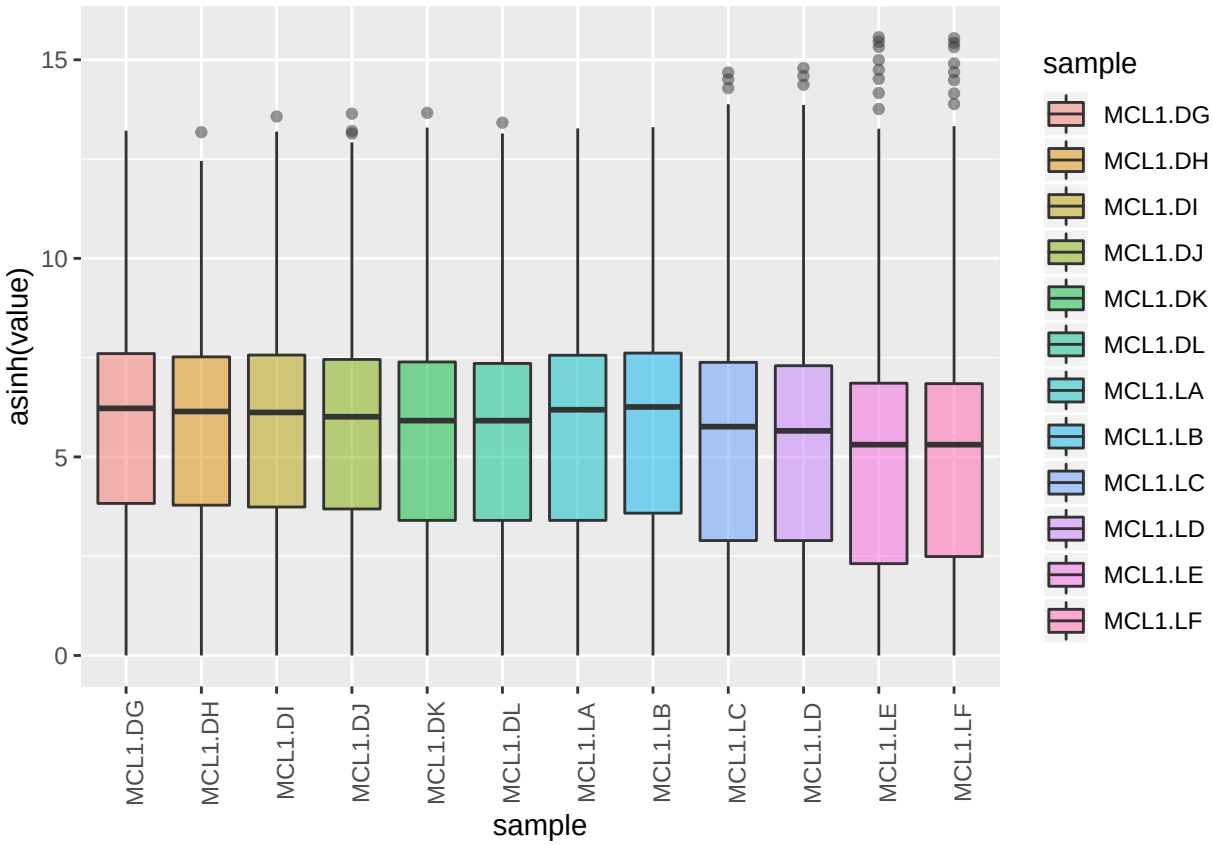
As an example, we can now plot the distribution of asinh transformed count values for each sample (You can offset histograms for better readability using the `ggridges` package).

```
plot <- ggplot(df_melt, aes(x=asinh(value), fill = sample)) +
  geom_density(alpha = 0.5)
plot
```



Box plots allow to see a summary of these distributions.

```
plot <- ggplot(df_melt, aes(x=sample, y = asinh(value), fill = sample)) +
  theme(axis.text.x = element_text(angle = 90)) +
  geom_boxplot(alpha = 0.5)
plot
```



The last two samples seem to differ quite neatly from the other samples. Some normalization between samples will be needed before going further into the analysis.

Here we use the `dplyr` package to group rows and return group summary (see the corresponding chapter in R for data science). It makes it easy to compute the median count value per sample.

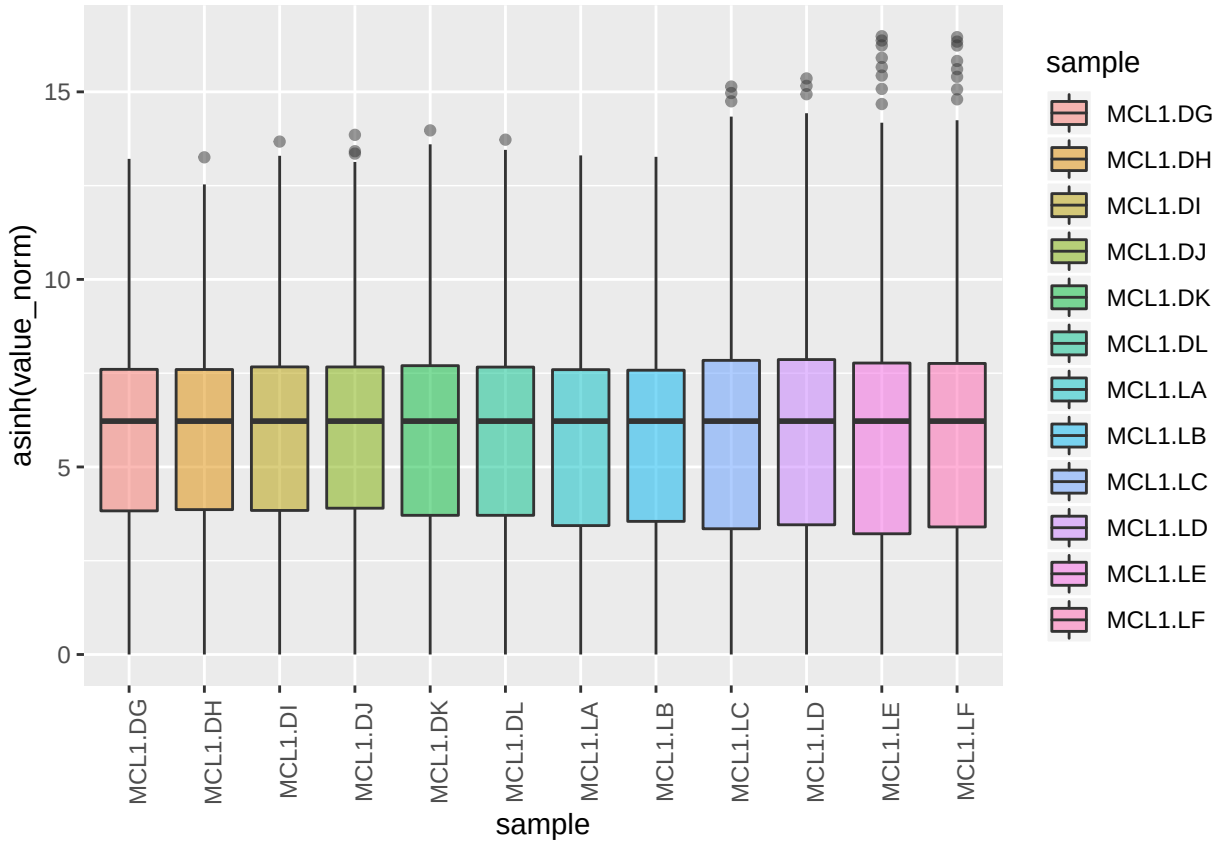
```
df_median <-
  df_melt %>%
  group_by(sample) %>%
  summarise(median = median(value, na.rm = TRUE))

df_median
```

```
## # A tibble: 12 x 2
##    sample  median
##    <fct>    <dbl>
##  1 MCL1.DG   252
##  2 MCL1.DH   233
##  3 MCL1.DI   228.
##  4 MCL1.DJ   204
##  5 MCL1.DK   185
##  6 MCL1.DL   185
##  7 MCL1.LA   244.
##  8 MCL1.LB   261
##  9 MCL1.LC   159
## 10 MCL1.LD   143
## 11 MCL1.LE   101
## 12 MCL1.LF   101
```

Build a data frame with the median and mean count per gene across samples.

Normalization will be carried out properly later. Here we show how we can use the `dplyr` package to normalize the data using the median:

```
median_first <- df_median$median[1]

df_melt_norm <-
  df_melt %>%
  group_by(sample) %>%
  mutate(value_norm = value/median(value, na.rm = TRUE)*median_first)

ggplot(df_melt_norm, aes(x=sample, y = asinh(value_norm), fill = sample)) +
  theme(axis.text.x = element_text(angle = 90)) +
  geom_boxplot(alpha = 0.5)
```



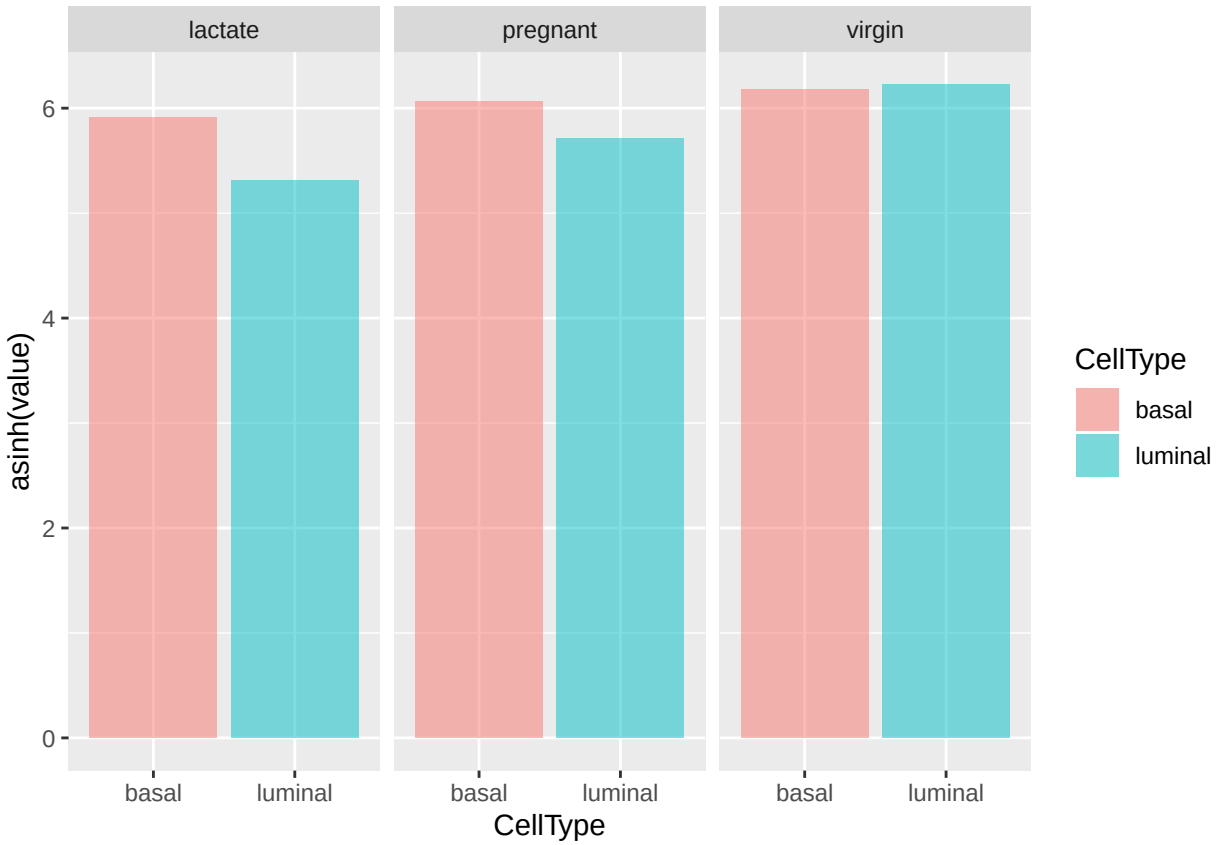
Using metadata

Things get more interesting when we include metadata. Let's map sample information to `df_melt`

```
idx_match <- match(df_melt$sample, rownames(sampleinfo))
df_melt <- cbind(df_melt, sampleinfo[idx_match, ])
```

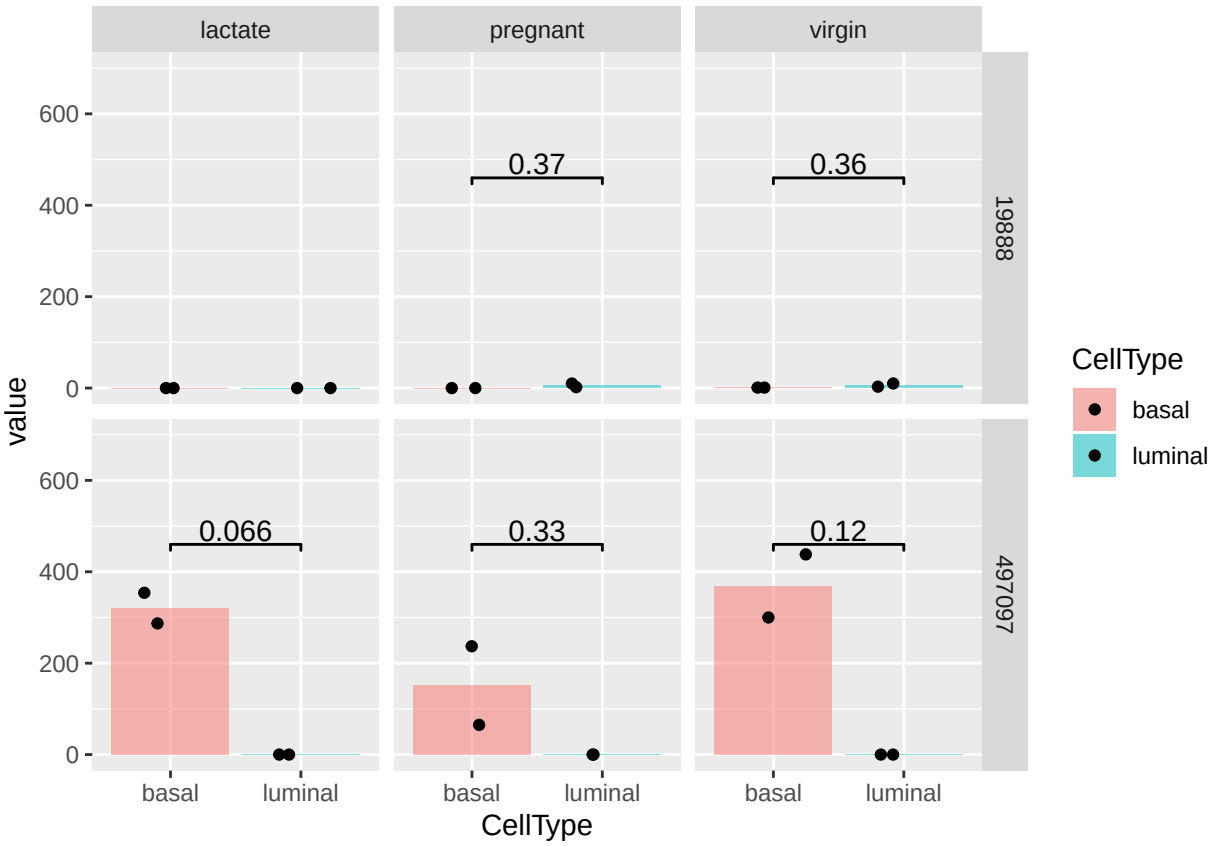
Now we have access to two more variables, `CellType` and `Status`, that we could use to form groups.

```
ggplot(df_melt, aes(x = CellType, y = asinh(value), fill = CellType)) +
  geom_bar(alpha = 0.5, stat = "summary", fun.y = "median", position = "dodge") +
  facet_wrap(~Status)
```



We can also focus on a particular set of GeneIDs and split plots using `facet_grid`. Let's add an unpaired t-test on top of that.

```
library(ggsignif)
idx_select <- df_melt$GeneID %in% df_melt$GeneID[1:2]
ggplot(df_melt[idx_select, ], aes(x = CellType, y = value, fill = CellType)) +
  geom_bar(alpha = 0.5, stat = "summary", fun.y = "median", position = "dodge") +
  geom_point(position = position_jitter(width = 0.25, height = 0))+
  geom_signif(comparisons = list(1:2),
              na.rm = TRUE, test = "t.test",
              test.args = list("paired" = FALSE),
              position = "identity") +
  ylim(c(0,700))+
  facet_grid(GeneID~Status)
```



PCA

Principal component analyisis (PCA) is a great tool to see the overall "shape" of the data. It allows to identify which samples are similar to one another and which are very different. This can enable us to identify groups of samples that are similar and work out which variables make one group different from another. We use the `prcomp` function to run the PCA. Usually, features (here genes) are columns while observation (here samples) are rows so we'll transpose `countdata` to before running the PCA.

```
pca_res <- prcomp( t(countdata), center = TRUE, scale. = TRUE)
summary(pca_res)
```

```
## Importance of components:
##                             PC1     PC2      PC3      PC4     PC5      PC6
## Standard deviation     83.8370 72.0165 40.90397 38.68490 31.0671 22.45107
## Proportion of Variance  0.3867  0.2853  0.09204  0.08233  0.0531  0.02773
## Cumulative Proportion   0.3867  0.6720  0.76401  0.84633  0.8994  0.92716
##                             PC7      PC8      PC9     PC10     PC11      PC12
## Standard deviation     18.77350 17.49166 16.17556 15.66165 12.60086 1.652e-13
## Proportion of Variance  0.01939  0.01683  0.01439  0.01349  0.00873 0.000e+00
## Cumulative Proportion   0.94655  0.96338  0.97777  0.99127  1.00000 1.000e+00
```

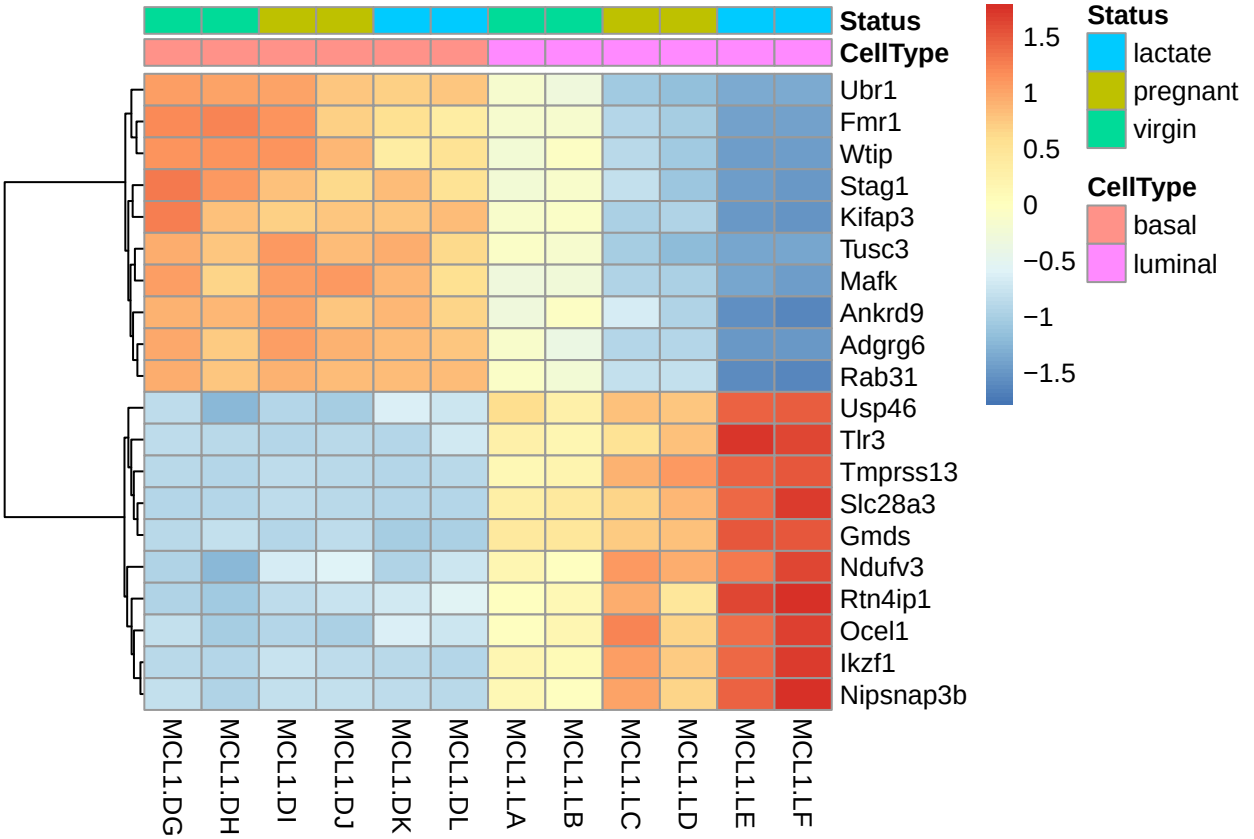
You obtain 12 principal components, each one explaining a percentage of the total variation in the dataset (PC1 explains 39%, PC2 29% and so on). The relationship (correlation or anticorrelation, a.k.a loadings) between the initial variables and the principal components is in `$rotation`. Let's see which genes have the greatest loadings onto PC1:

```
idx_up <- order(pca_res$rotation[ , 1], decreasing = TRUE)
idx_down <- order(pca_res$rotation[ , 1], decreasing = FALSE)
head(pca_res$rotation[idx_up, 1:2])
```

```
##                PC1           PC2
## 214531 0.01182908  0.0006886089
## 22778  0.01176338  0.0001653767
## 114304 0.01174112  0.0010381224
## 142980 0.01173840 -0.0003178144
## 66536  0.01171238 -0.0003867467
## 69727  0.01170085  0.0012227270
```

Let's look at the count data heatmap for genes with greatest and lowest loadings onto PC1. How does it look for PC2?

```
idx_gene <- c(idx_up[1:10], idx_down[1:10])
pheatmap(countdata[idx_gene, ],
         show_rownames=TRUE,
         cluster_cols=FALSE,
         annotation_col=sampleinfo[ , c("CellType","Status")],
         scale = "row",
         labels_row = annot$SYMBOL[idx_gene]
         )
```



The values of each sample in terms of the principal components is in `$x`. Check taht the row names `pca_res$x` are the same as that of `sampleinfo`:

```
pca_res$x[ , 1:2]
```

```
##                PC1        PC2
## MCL1.DG -91.91003  21.174002
## MCL1.DH -79.97473  -2.987728
## MCL1.DI -82.15576 -12.811208
## MCL1.DJ -67.73988 -40.945893
## MCL1.DK -61.12426 -67.292207
## MCL1.DL -55.00054 -67.320751
## MCL1.LA  17.84360 117.724564
## MCL1.LB  12.05523 137.439876
## MCL1.LC  75.36071  44.429079
## MCL1.LD  80.94125  16.790115
## MCL1.LE 125.65812 -73.007205
## MCL1.LF 126.04628 -73.192642
```

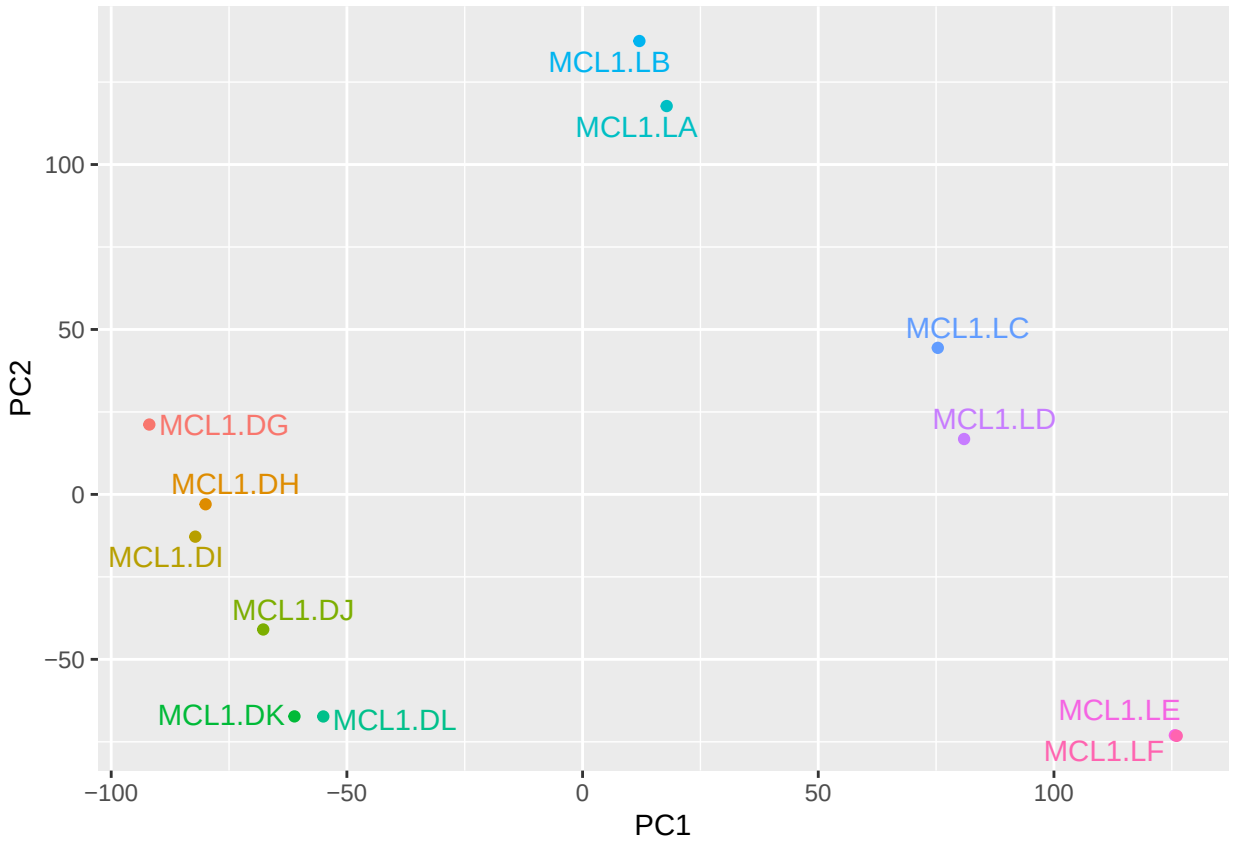
We'll use the `ggplot2` package to plot the results of the PCA. See the R for data science book for an introduction to ggplot2. We first need to create a data frame from the pca results.

```
df_pca <- as.data.frame(pca_res$x)
df_pca$sample <- rownames(df_pca)
names(df_pca)
```

```
##  [1] "PC1"    "PC2"    "PC3"    "PC4"    "PC5"    "PC6"    "PC7"    "PC8"
##  [9] "PC9"    "PC10"   "PC11"   "PC12"   "sample"
```

We can now use the `ggplot()` function and choose to draw one point per sample.

```
library(ggplot2)
library(ggrepel)

pca_plot <- ggplot(df_pca, aes(x=PC1, y=PC2, color = sample, label=sample)) +
  geom_point(show.legend = FALSE) +
  geom_text_repel(show.legend = FALSE)
pca_plot
```



As an exercise, check that the row names `pca_res$x` are the same as that of `sampleinfo` (check it) and map metadata directly to the `df_pca` dataframe. Color points according to the sample `Status` or `CellType`.

–>

Do the same using t-SNE

```
library(Rtsne)
tsne_res <- Rtsne(t(countdata), perplexity = 3)
df_tsne <- as.data.frame(tsne_res$Y)
colnames(df_tsne) <- c("tSNE1", "tSNE2")
df_tsne$sample <- colnames(countdata)

ggplot(df_tsne, aes(x=tSNE1, y=tSNE2, color = sample, label=sample)) +
  geom_point(alpha = 0.25, show.legend = FALSE) +
  geom_text_repel(show.legend = FALSE)
```

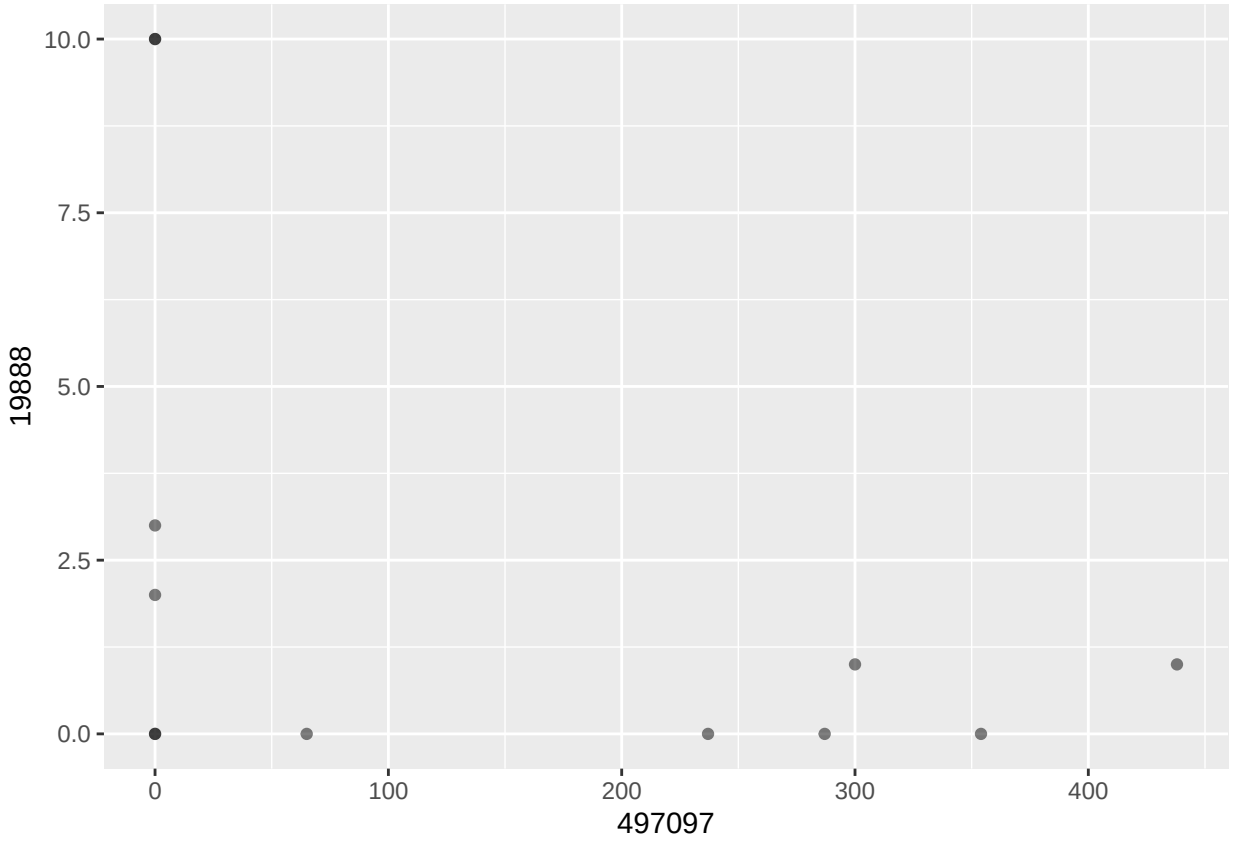
Correlations

At some point in the analysis, we might want to perform pairwise comparisons between genes. Let's first create a data frame (note that we use the transpose of the countdata matrix so available variables are in `names(df)`).

```
df <- as.data.frame(t(countdata))
```

Plotting the data for two different genes is a good way to identify functionnal relationships between them. Let's pick two genes,

```
library(ggplot2)
xvar <- as.name(names(df)[1])
yvar <- as.name(names(df)[2])
p <- ggplot(df, aes_string(x=xvar, y=yvar)) + geom_point(alpha = 0.5)
p
```



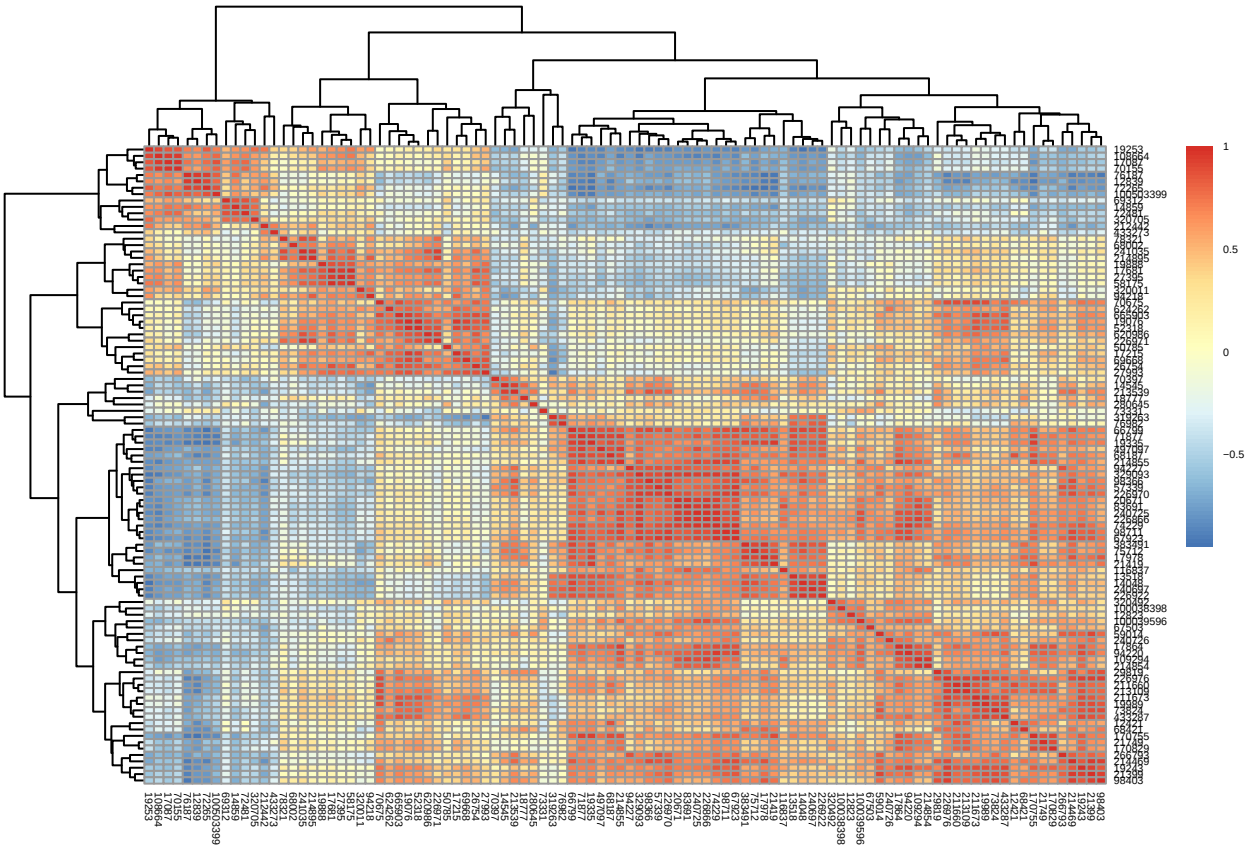
and fit a linear model to the data and compute Pearson's correlation coefficient.

```
lm_res <- lm(formula = `19888` ~ `497097`, data = df)
sm<-summary.lm(lm_res)
sqrt(sm$r.squared)
```

```
## [1] 0.4371161
```

Now we want to repeat this for all pairs of genes (we use only the 100 first genes here) and build a correlation matrix

```
library(Hmisc)
corr <- Hmisc::rcorr(t(countdata[1:100, ]))
pheatmap(corr$r, fontsize = 4)
```



We can explore correlations by constructing a data frame with the correlation corefficient, its associated p-value and the number of points used to compute the correlation

```
library(reshape2)
df_R_coeff <- melt(corr$r)
df_R_coeff$variable <- "R"
df_pval <- melt(corr$P)
df_pval$variable <- "P"
df_nval <- melt(corr$n)
df_nval$variable <- "n"

df_corr <- rbind(df_R_coeff, df_pval, df_nval)
df_corr <- dcast(df_corr, Var1 + Var2 ~ variable, mean)
df_corr$gene_name_1 <- annot$SYMBOL[match(df_corr$Var1, annot$ENTREZID)]
df_corr$gene_name_2 <- annot$SYMBOL[match(df_corr$Var2, annot$ENTREZID)]
df_corr <- df_corr[df_corr$Var1 != df_corr$Var2, ]
```

We can look at the whole correlation data frame using the interactive function `datatable`. It is convenient if you wish to search for certain variables or order results.

```
library(DT)
DT::datatable(df_corr)
```

Now we can filter and keep only the most correlated pairs

```
df_corr_filtered <- df_corr[!is.na(df_corr$P) & df_corr$n>10 & df_corr$P<0.05 & df_corr$R>0.9, ]
```

We use that restricted set to build a network.

```
library(igraph)

net <- igraph::graph.data.frame(df_corr_filtered[ , c("gene_name_1", "gene_name_2")], directed=FALSE)
net <- igraph::simplify(net)
clusters <- igraph::cluster_fast_greedy(as.undirected(net))
group = clusters$membership
names(group) <- clusters$names
```

And we render it in an interactive environment

```
library(networkD3)
net_d3 <- networkD3::igraph_to_networkD3(net, group = group)
p <- forceNetwork(Links = net_d3$links, Nodes = net_d3$nodes,
                  Source = 'source', Target = 'target',
                  fontFamily = "arial",
                  NodeID = 'name', Group = 'group',
                  colourScale = JS("d3.scaleOrdinal(d3.schemeCategory20);"),
                  charge = -10, opacity = 1,
                  linkColour = rgb(0.75, 0.75, 0.75),
                  fontSize = 12, bounded = TRUE, zoom=TRUE, opacityNoHover = 1)
p
```

Alternatively, we can save it as a text file and use another software such as *cytoscape* to manipulate this network.

```
dir.create("./output/")
write.table(df_corr_filtered, file = "./output/df_corr_filtered.txt", sep = "\t", row.names = FALSE, qu
write.table(data.frame(clusters$membership, clusters$names), file = "./output/group.txt", sep = "\t", r
```

Exercise: add an extra column `df$score <- 1:length(rownames(df))` that contains a score associated to each sample to our data frame. Which genes most strongly correlate with this score?

Differential expression analysis

Creating a DESeqDataSet

We will use the `DESeq2` package to conduct the core steps of the analysis. It should be already installed so we just need to load it in the RStudio session.

```
library(DESeq2)
```

We can use the `help("DESeq2-package")` command to browse the package documentation in Rstudio. We will use the `DESeqDataSetFromMatrix` function to build a `DESeqDataSet` object.

```
dds <- DESeqDataSetFromMatrix(countData = countdata,
                              colData = sampleinfo,
                              design = ~ CellType + Status)
```

The design indicates how to model the samples, i.e how the counts for each gene depend on the variables in `colData = sampleinfo`. Here we want to measure the effect of the cell type and of the pregnancy status of the mice. Note that the two factor variables `CellType` and `Status` should be columns of `sampleinfo`. Now the `dds` object contains count data along with the metadata and the experiment design. The count data is obtained using `counts(dds)` and the metadata is obtained using `colData(dds)`.

Now that we have a `DESeqDataSet` object, we can analyse the data using the many tools available in the `DESeq2` package.

Differential expression analyis

The standard differential expression analysis steps are wrapped into a single function, `DESeq`. It is then straightforward to perform differential expression analysis.

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

We can see the comparisons carried out by DESeq using `resultsNames()`.

```
resultsNames(dds)
```

```
## [1] "Intercept"                  "CellType_luminal_vs_basal"
## [3] "Status_pregnant_vs_lactate" "Status_virgin_vs_lactate"
```

Results tables are generated using the function `results`, which extracts a results table with log2 fold changes, p values and adjusted p values. We specify which coomparison we want to extract using the `name` parameter. Here we choose to compare basal and luminal cell types. We select significant results with adjusted p-values (correction for multiple tests computed with the Benjamini–Hochberg procedure) below 0.01 using `alpha`.

```
res <- results(dds, alpha = 0.001, pAdjustMethod="BH", name="CellType_luminal_vs_basal")
head(res)
```

```
## log2 fold change (MLE): CellType luminal vs basal
## Wald test p-value: CellType luminal vs basal
## DataFrame with 6 rows and 6 columns
##                baseMean     log2FoldChange              lfcSE              stat
##               <numeric>          <numeric>          <numeric>         <numeric>
## 497097 126.481585433975  -10.1820437902142  0.930694407250738 -10.9402653662569
## 19888  1.95697709143844   3.06715205617399   1.23908077239746  2.47534472691353
## 20671  50.5979096011351  -2.71203870035567  0.337521822904564 -8.03515066675411
## 27395  358.719720532762  0.799404687906072  0.127971895259294   6.2467207060295
## 18777  744.356910502922  0.106722300368205  0.134155732952959 0.795510545983351
## 21399  1097.91010559132 -0.361177585114154 0.0647349105685968 -5.57933241803787
##                      pvalue                 padj
##                   <numeric>            <numeric>
## 497097 7.39830424216018e-28  9.6398635364105e-27
## 19888    0.0133107627219475   0.0207548596323924
## 20671  9.34639816135234e-16  5.9580304660091e-15
## 27395  4.19159587750374e-10 1.67328660652554e-09
## 18777     0.426316578845322    0.482508269994292
## 21399  2.41443432722022e-08 8.05872225396731e-08
```

We can get a summary of these results using `summary()`

```
summary(res)
```

```
##
## out of 18178 with nonzero total read count
## adjusted p-value < 0.001
## LFC > 0 (up)       : 4185, 23%
## LFC < 0 (down)     : 4741, 26%
## outliers [1]       : 0, 0%
## low counts [2]     : 705, 3.9%
## (mean count < 2)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

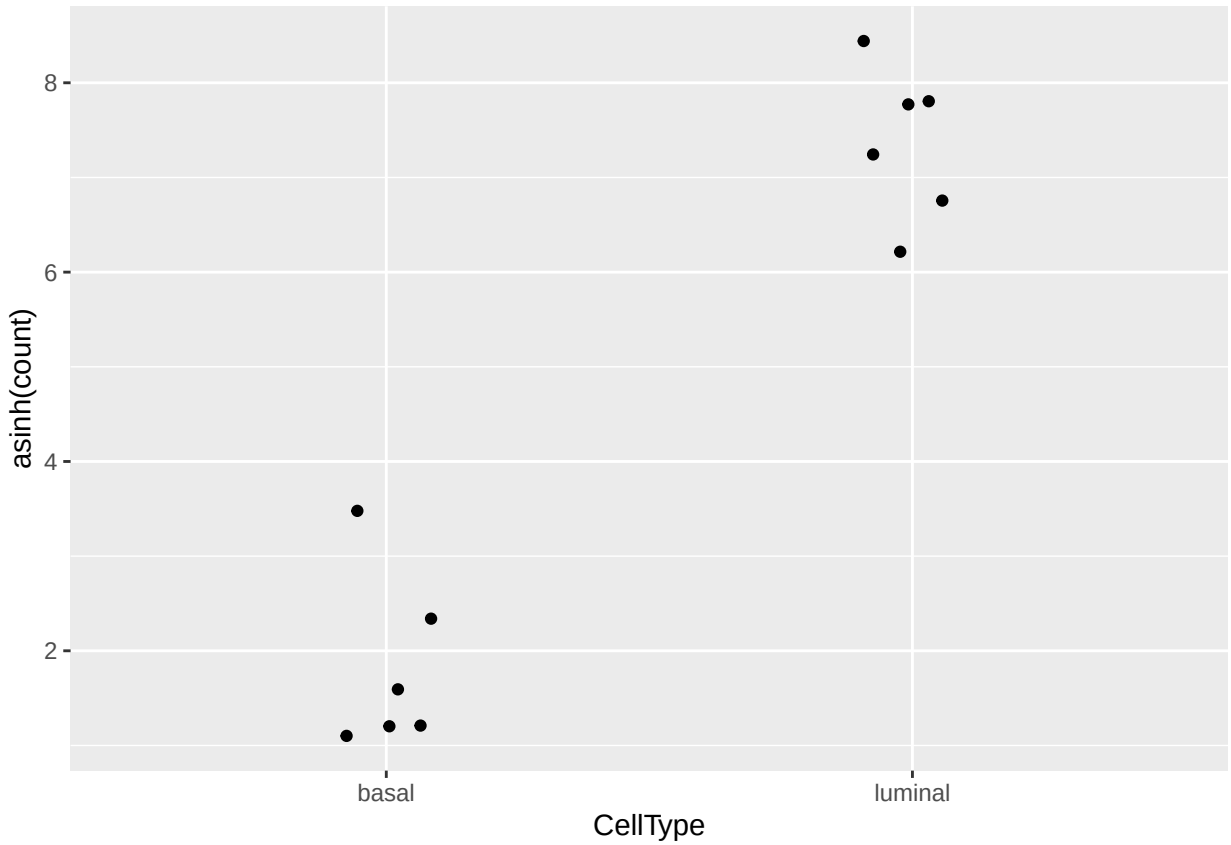
Let's select genes have an a adjusted p-value lower than `0.001` and a fold-change greater than `2^5`:

```
enrich <- data.frame(ID=rownames(res), res@listData)

enrich  <- enrich  %>%
  filter(padj < 0.001, log2FoldChange > 5) %>%
  arrange(desc(log2FoldChange))

enrich %>% head(10)
```

```
##           ID  baseMean log2FoldChange     lfcSE      stat       pvalue
## 1     192200 510.96091       7.836400 0.6605401 11.863625 1.828812e-32
## 2     791407  72.08411       7.785638 0.7407422 10.510590 7.720906e-26
## 3      21990 977.00600       7.686764 0.5023793 15.300719 7.561686e-53
## 4     217316  58.82642       7.482389 0.7202352 10.388813 2.788000e-25
## 5     382036  16.46267       7.472099 0.8865860  8.427946 3.517839e-17
## 6      12310 450.32143       7.388458 0.6800650 10.864340 1.704520e-27
## 7      20309 366.10363       7.296678 0.4709648 15.493044 3.865462e-54
## 8     246133  16.26991       7.047306 1.1689753  6.028618 1.653672e-09
## 9      71601  27.35394       7.010957 0.8244734  8.503558 1.838672e-17
## 10 100503562  99.14758       6.949689 0.4788576 14.513059 1.001526e-47
##             padj
## 1  3.011766e-31
## 2  9.183620e-25
## 3  2.735514e-51
## 4  3.202809e-24
## 5  2.457705e-16
## 6  2.185112e-26
## 7  1.471486e-52
## 8  6.262377e-09
## 9  1.312919e-16
## 10 3.108290e-46
```

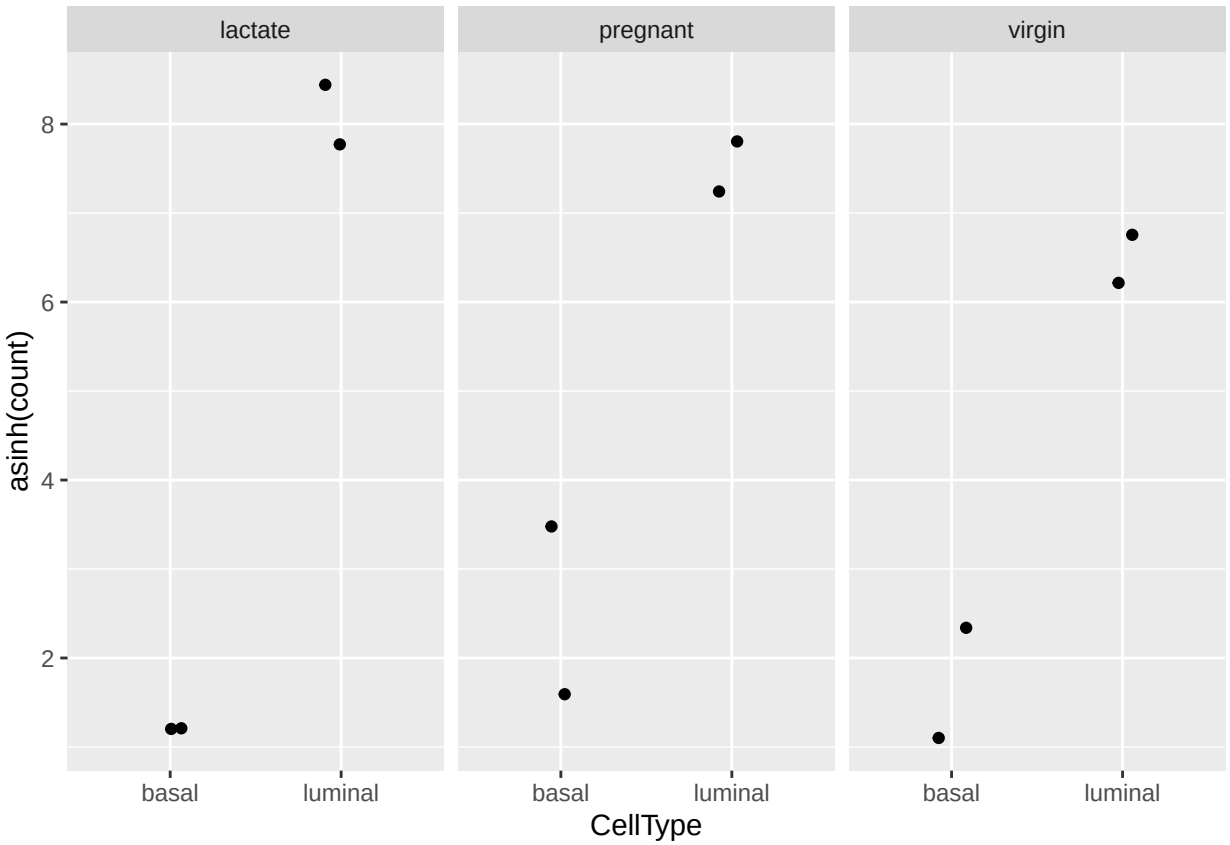
Have a look at the most enriched gene

```
d <- plotCounts(dds, gene=as.character(enrich$ID[1]), intgroup=c("CellType","Status"),
                returnData=TRUE)
library("ggplot2")
p <- ggplot(d, aes(x=CellType, y=asinh(count))) +
  geom_point(position=position_jitter(w=0.1,h=0))
p
```



We can also facet the plot according to `Status`. It could be interesting to see how the `CellType` effect depends on `Status`. See section Interactions to address such questions.

```
p + facet_wrap(~Status)
```



We have used default parameters here. Data normalization and model fitting have been performed in the background by DESeq. If you want to perform these steps separately, have a look at the Additional Material section.

Interactions

Interaction terms allow to test, for example, if the log2 fold change attributable to a given condition is different based on another factor, for example if the `CellType` effect differs across `Status`.

Interaction terms can be added to the design formula but a simpler approach to take into account interactions consists in: -combining the factors of interest into a single factor with all combinations of the original factors -changing the design to include just this factor, e.g. ~ group

```
dds.int <- dds
dds.int$group <- factor(paste(dds.int$Status, dds.int$CellType, sep="."))
design(dds.int) <- ~ group
dds.int <- DESeq(dds.int)
```

Now we can compare basal and luminal cell types within the lactate status.

```
res.int <- results(dds.int, contrast = c("group", "lactate.basal", "lactate.luminal"))
res.int
```

```
## log2 fold change (MLE): group lactate.basal vs lactate.luminal
## Wald test p-value: group lactate.basal vs lactate.luminal
## DataFrame with 18178 rows and 6 columns
##                   baseMean     log2FoldChange             lfcSE
##                  <numeric>          <numeric>         <numeric>
## 497097    126.481585433975   9.99193033684966  1.70778319837909
```

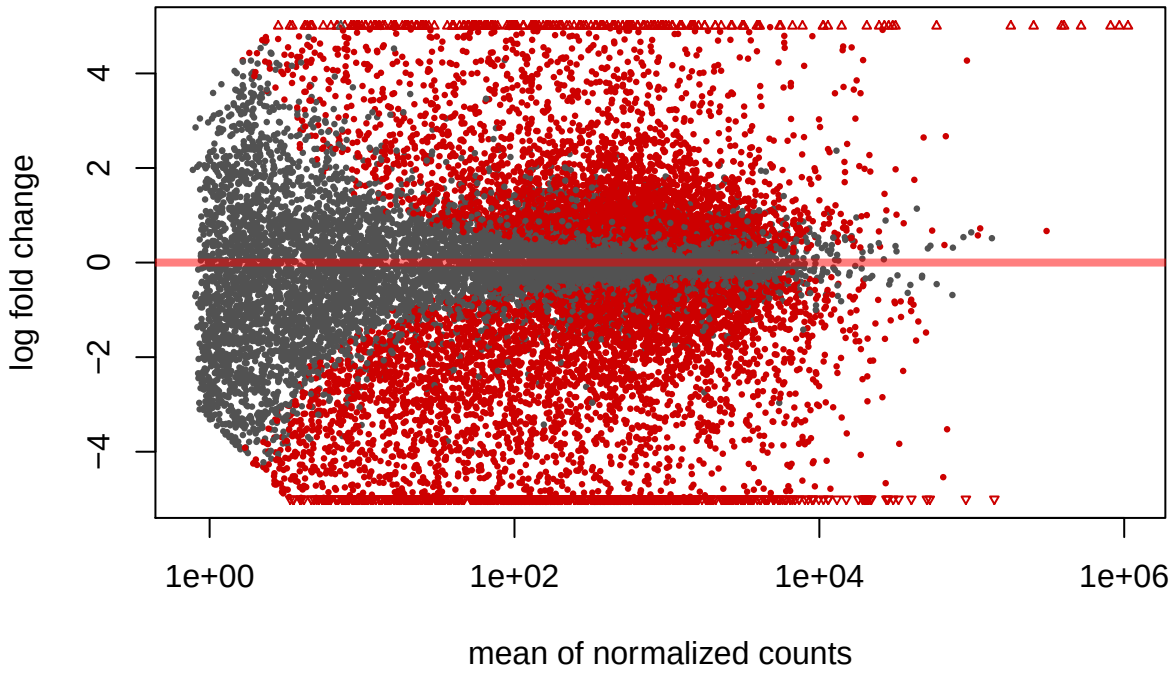
```
## 19888     1.95697709143844                  0  2.74448002307268
## 20671     50.5979096011351   2.49566717862667 0.627259620850239
## 27395     358.719720532762 -0.986661733396426 0.201363861154175
## 18777     744.356910502922 -0.162747242371505 0.141156023502934
## ...                    ...                ...                ...
## 100862004 3409.10829251673 -0.114306553961538 0.192976178931272
## 107626    2.09893260798678 -0.188503357421951  1.84799787880061
## 100861837 60.0354369270688   1.76258722180287 0.413367940836069
## 100861924 70.3085529839158 -0.757018298034853 0.470176666408391
## 170942    407.572590388916 -0.321253809935335 0.267531015052703
##                         stat               pvalue                 padj
##                    <numeric>            <numeric>            <numeric>
## 497097      5.85081897183043 4.89158399055341e-09 1.52128680547955e-08
## 19888                      0                    1                    1
## 20671       3.97868298176733 6.92980676689751e-05 0.000153603252540743
## 27395      -4.89989478618998 9.58879910203484e-07  2.5232366832193e-06
## 18777      -1.15295995404774     0.24892682182977    0.319245926853504
## ...                      ...                  ...                  ...
## 100862004 -0.592335046711895     0.553626251835777    0.626716777050115
## 107626    -0.102004098372826     0.918753422023646    0.940801019915832
## 100861837   4.26396691102339 2.00829197398357e-05 4.70326352783732e-05
## 100861924  -1.61007202636746     0.107382133165361    0.151258614233238
## 170942     -1.20080959537364     0.22982506889867    0.297801946953625
```

Visualizing results

MA-plot

In DESeq2, the function plotMA shows the log2 fold changes attributable to a given variable over the mean of normalized counts for all the samples in the DESeqDataSet.

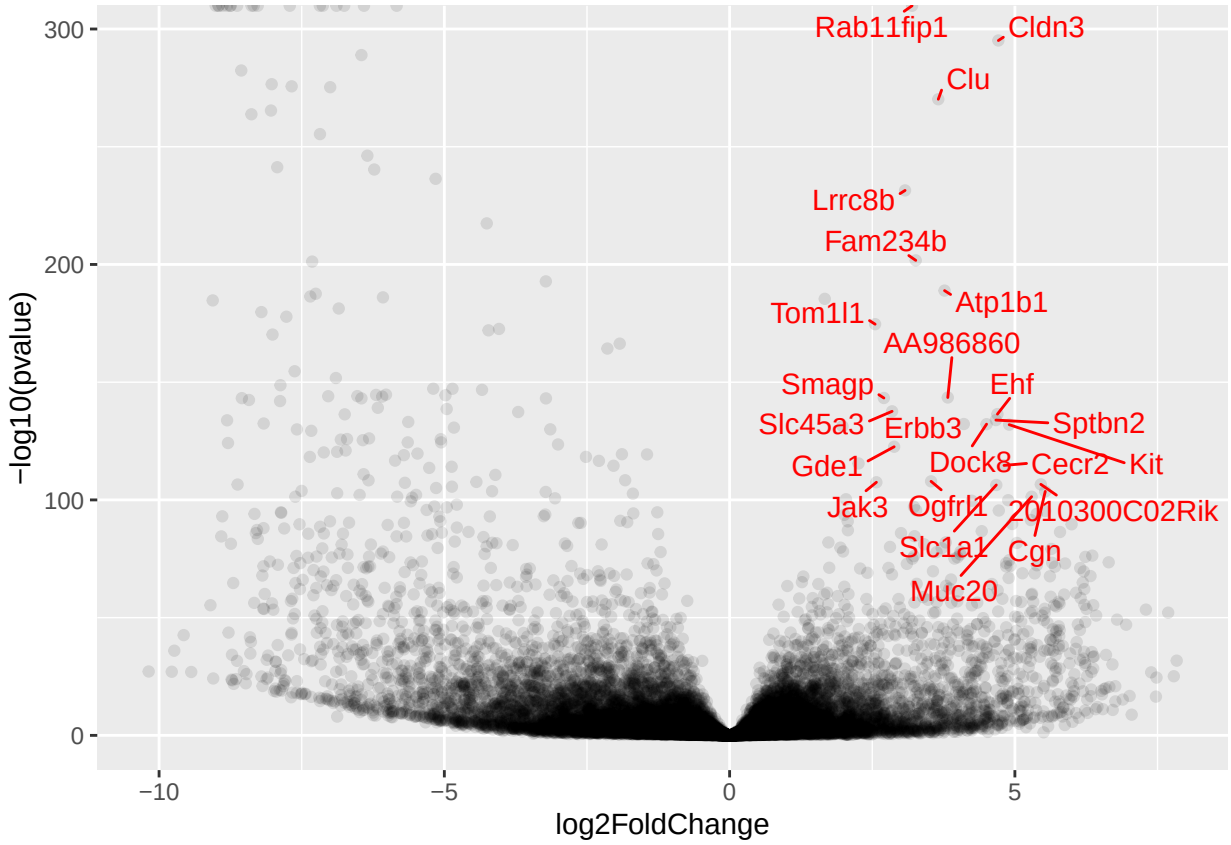
```
plotMA(res, ylim=c(-5, 5))
```



Volcano plot

Volcano plots allow to visualize p-values and fold-change. We add gene names for a set of selected genes

```
df<-as.data.frame(res@listData)
df$ID <- res@rownames
df$names <- annot$SYMBOL[match(df$ID, annot$ENTREZID)]
select <- df$log2FoldChange > 2.5 & log10(df$pvalue) < -100
ggplot(df, aes(x=log2FoldChange, y=-log10(pvalue), label = names)) +
  geom_point(alpha = 0.1) +
  geom_text_repel(data = df[select, ], color = "red", min.segment.length = 0)
```



Annotation enrichment analysis

Now that we have identified a set of genes that are differentially expressed between basal and luminal conditions, we can use available gene annotations to figure out whether this set of genes is enriched in terms corresponding to specific biological processes, molecular functions or pathways. We'll use the `clusterProfiler` package to perform this analysis on GO annotation terms corresponding to molecular functions (GO MF ontology).

```
library(clusterProfiler)
enrichgo_result_over_MF = enrichGO( gene = df$ID[select], keyType = "ENTREZID",
                            universe = df$ID,
                            OrgDb = org.Mm.eg.db,
                            ont = "MF",
                            pvalueCutoff = 0.05,
                            qvalueCutoff = 0.2,
                            readable = TRUE)
```

Results are stored in the `result` slot

```
datatable( enrichgo_result_over_MF@result )
```

Check the `clusterProfiler` documentation if you wish to look at other ontologies.

GSEA

Complementary to the previous approach, GSEA (Gene Set Enrichment Analysis) consists in analyzing how sets of genes of interest (also called "genesets" or "signatures") are distributed along some ranking (i.e. of ratios) between two conditions (i.e. luminal versus basal). It calculates an enrichment score and some statistics (p-value and FDR) for each geneset. Download the gene set file from here. Several sets can be dowloaded. We choose the Hallmark gene sets with "gene symbols" and save it in `./data`. You should now have a file `h.all.v7.1.symbols.gmt` in the `./data` folder. Let's read it:

```
#read the geneset file
pathwaysH <- read.csv("./demo_data/h.all.v7.1.symbols.gmt", sep="\t", header=F, stringsAsFactor=FALSE)
```

The gene set file needs to be in a specific format so we need to do a little formatting before using the `fgsea()` function:

```
#read the geneset file
# the format is different than when we do: load("human_H_v5p2.rdata")
# this causes a problem.
genesets <- lapply( 1:nrow(pathwaysH), function(x){
  row <- pathwaysH[x, ]
  current_string =  paste( unlist( row[ 3:length( row)], use.names = FALSE), collapse="\t")
  current_string <- gsub("[\t]+$","", current_string)
  current_string <- strsplit(current_string, split="\t")[[1]]
  return(current_string)}
)
names(genesets) <- pathwaysH[[1]]
```

We also need to rank genes. We choose to rank them according to the log2 fold-change stored in our `res` object. But before that, we need to remove duplicate genes in order to keep only distinct gene symbols. We decide to keep the most regulated ones.

```
ranks <- data.frame(ID = rownames(res), log2FC = res$log2FoldChange)
idx_match <- match(rownames(res), annot$ENTREZID)
ranks$symbol <- toupper(annot$SYMBOL[idx_match])
ranks <- ranks[order(abs(ranks$log2FC), decreasing = TRUE), ]
ranks <- distinct(ranks, symbol, .keep_all = TRUE)
ranks <- ranks[order(ranks$log2FC, decreasing = TRUE), ]
stats <- ranks$log2FC
names(stats) <- ranks$symbol
```

Run GSEA

```
library(fgsea)
fgseaRes <- fgsea(pathways = genesets, stats = stats, minSize=15, maxSize = 500, nperm=1000)
```

Lets look at the top 10 most enriched gene sets:

```
topUp <- fgseaRes %>%
  filter(NES > 0) %>%
  arrange(padj, desc(NES)) %>%
  head(10)
```

```
topUp %>% dplyr::select(pathway, padj, NES)
```

```
##                                pathway       padj      NES
##  1: HALLMARK_OXIDATIVE_PHOSPHORYLATION 0.07607777 1.937665
##  2:               HALLMARK_E2F_TARGETS 0.07607777 1.843079
##  3:            HALLMARK_G2M_CHECKPOINT 0.07607777 1.544091
##  4:   HALLMARK_ESTROGEN_RESPONSE_EARLY 0.07607777 1.436568
##  5:            HALLMARK_MYC_TARGETS_V1 0.07607777 1.379836
##  6:    HALLMARK_ESTROGEN_RESPONSE_LATE 0.10775862 1.228705
##  7:     HALLMARK_FATTY_ACID_METABOLISM 0.11422045 1.281048
##  8:              HALLMARK_ADIPOGENESIS 0.11574074 1.236591
##  9:         HALLMARK_PROTEIN_SECRETION 0.13670540 1.243309
## 10:   HALLMARK_CHOLESTEROL_HOMEOSTASIS 0.20304569 1.223285
```

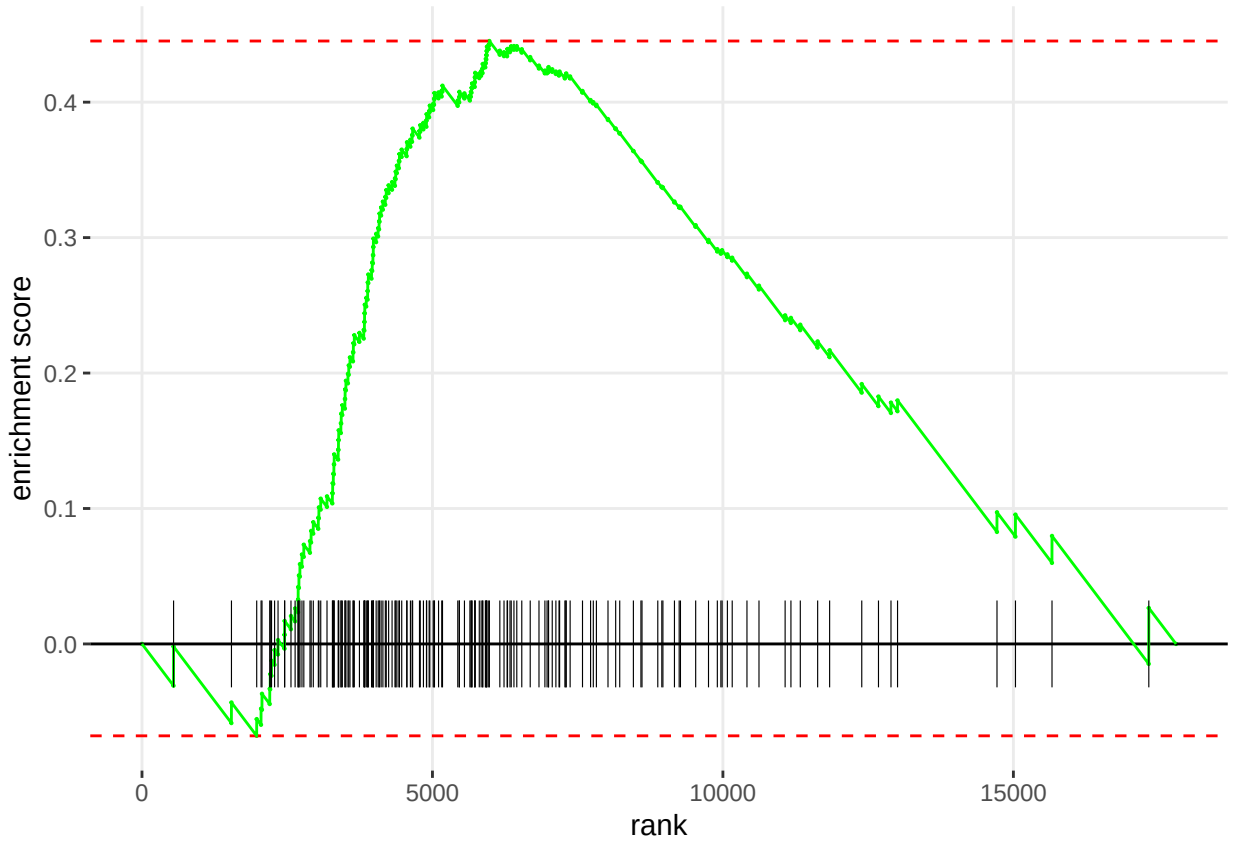
and at the top 10 most depleted gene sets :

```
topDown <- fgseaRes %>%
  filter(NES < 0) %>%
  arrange(padj, desc(NES)) %>%
  head(10)

topDown %>% dplyr::select(pathway, padj, NES)
```

```
##                                      pathway       padj       NES
##  1:                          HALLMARK_HYPOXIA 0.01104972 -1.504631
##  2:                    HALLMARK_UV_RESPONSE_DN 0.01104972 -1.705186
##  3:                   HALLMARK_APICAL_JUNCTION 0.01104972 -1.736047
##  4:                       HALLMARK_MYOGENESIS 0.01104972 -1.887309
##  5: HALLMARK_EPITHELIAL_MESENCHYMAL_TRANSITION 0.01104972 -2.373933
##  6:        HALLMARK_WNT_BETA_CATENIN_SIGNALING 0.04397537 -1.658711
##  7:                HALLMARK_HEDGEHOG_SIGNALING 0.04705439 -1.662703
##  8:                 HALLMARK_KRAS_SIGNALING_DN 0.07607777 -1.374792
##  9:                      HALLMARK_COAGULATION 0.07728604 -1.421915
## 10:                     HALLMARK_ANGIOGENESIS 0.08410801 -1.530256
```

Let'see how the genes within a pathway are distributed along the ranked genes:

```
plotEnrichment(genesets[[topUp$pathway[1]]], stats)
```



We can also look at several pathways at once:

```
topPathways <- bind_rows(topUp, topDown) %>%
  arrange(-ES)

plotGseaTable(genesets[topPathways$pathway],
              stats,
              fgseaRes,
              gseaParam = 0.5)
```

Pathway	Gene ranks	NES	pval	padj
K_OXIDATIVE_PHOSPHORYLATION		1.94	1.6e−02	7.6e−02
HALLMARK_E2F_TARGETS		1.84	1.9e−02	7.6e−02
HALLMARK_G2M_CHECKPOINT		1.54	1.9e−02	7.6e−02
K_ESTROGEN_RESPONSE_EARLY		1.44	1.7e−02	7.6e−02
K_CHOLESTEROL_HOMEOSTASIS		1.22	8.1e−02	2.0e−01
HALLMARK_MYC_TARGETS_V1		1.38	1.9e−02	7.6e−02
HALLMARK_PROTEIN_SECRETION		1.24	5.2e−02	1.4e−01
MARK_FATTY_ACID_METABOLISM		1.28	3.9e−02	1.1e−01
HALLMARK_ADIPOGENESIS		1.24	4.2e−02	1.2e−01
RK_ESTROGEN_RESPONSE_LATE		1.23	3.4e−02	1.1e−01
HALLMARK_KRAS_SIGNALING_DN		−1.37	2.0e−02	7.6e−02
HALLMARK_COAGULATION		−1.42	2.2e−02	7.7e−02
HALLMARK_HYPOXIA		−1.50	1.1e−03	1.1e−02
HALLMARK_APICAL_JUNCTION		−1.74	1.1e−03	1.1e−02
HALLMARK_UV_RESPONSE_DN		−1.71	1.1e−03	1.1e−02
HALLMARK_ANGIOGENESIS		−1.53	2.5e−02	8.4e−02
HALLMARK_MYOGENESIS		−1.89	1.1e−03	1.1e−02
_WNT_BETA_CATENIN_SIGNALING		−1.66	5.3e−03	4.4e−02
LLMARK_HEDGEHOG_SIGNALING		−1.66	6.6e−03	4.7e−02
IAL_MESENCHYMAL_TRANSITION		−2.37	1.1e−03	1.1e−02

Re-run the analysis using the GO MF gene set. What differentiate GSEA from our previous annotation enrichment analysis?

Additional Material

Normalization

Normalization is performed using `estimateSizeFactors`. This function first normalize rows by the geometric mean and returns the median value for each column.

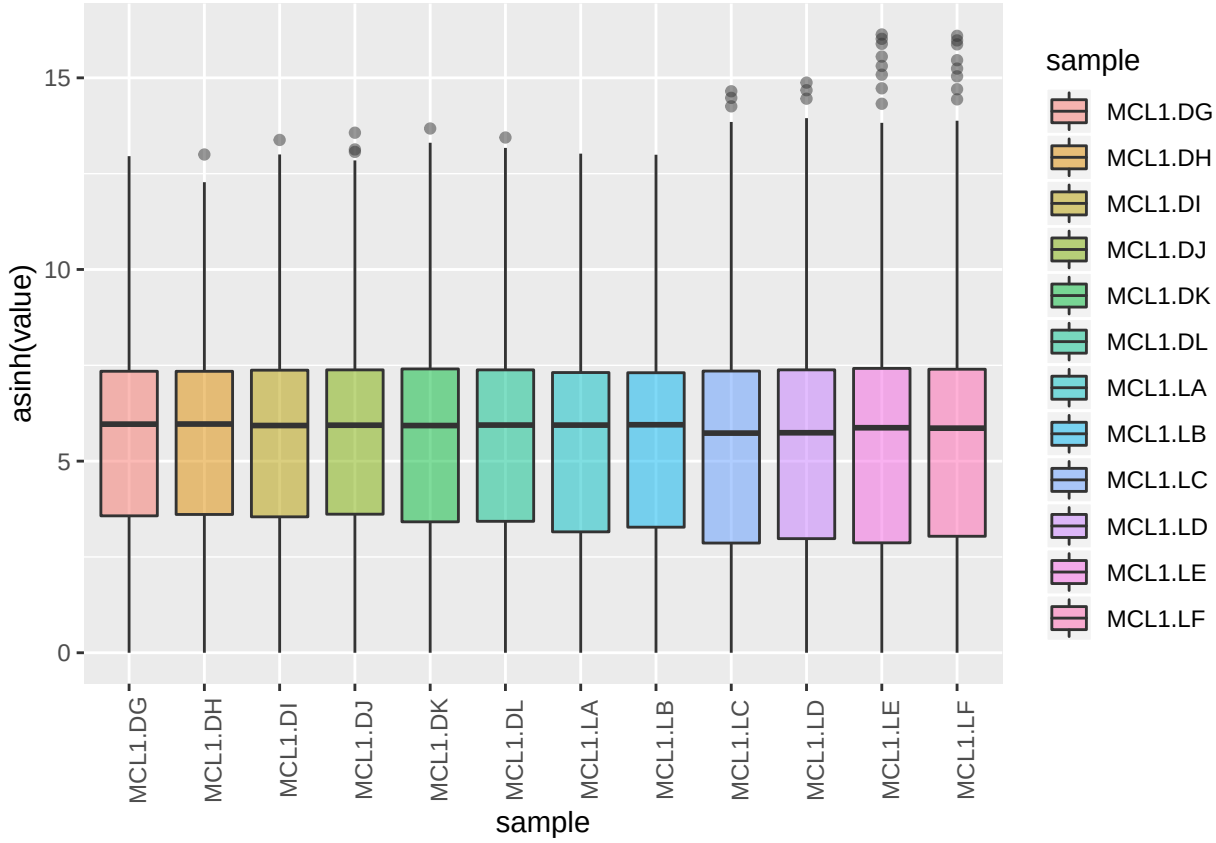
```
dds.norm <-  estimateSizeFactors(dds)
sizeFactors(dds)
```

```
##   MCL1.DG   MCL1.DH   MCL1.DI   MCL1.DJ   MCL1.DK   MCL1.DL   MCL1.LA   MCL1.LB
## 1.2934903 1.1933709 1.2113142 1.0768623 0.9859083 0.9729893 1.2822154 1.3615404
##   MCL1.LC   MCL1.LD   MCL1.LE   MCL1.LF
## 1.0315620 0.9190180 0.5697346 0.5758505
```

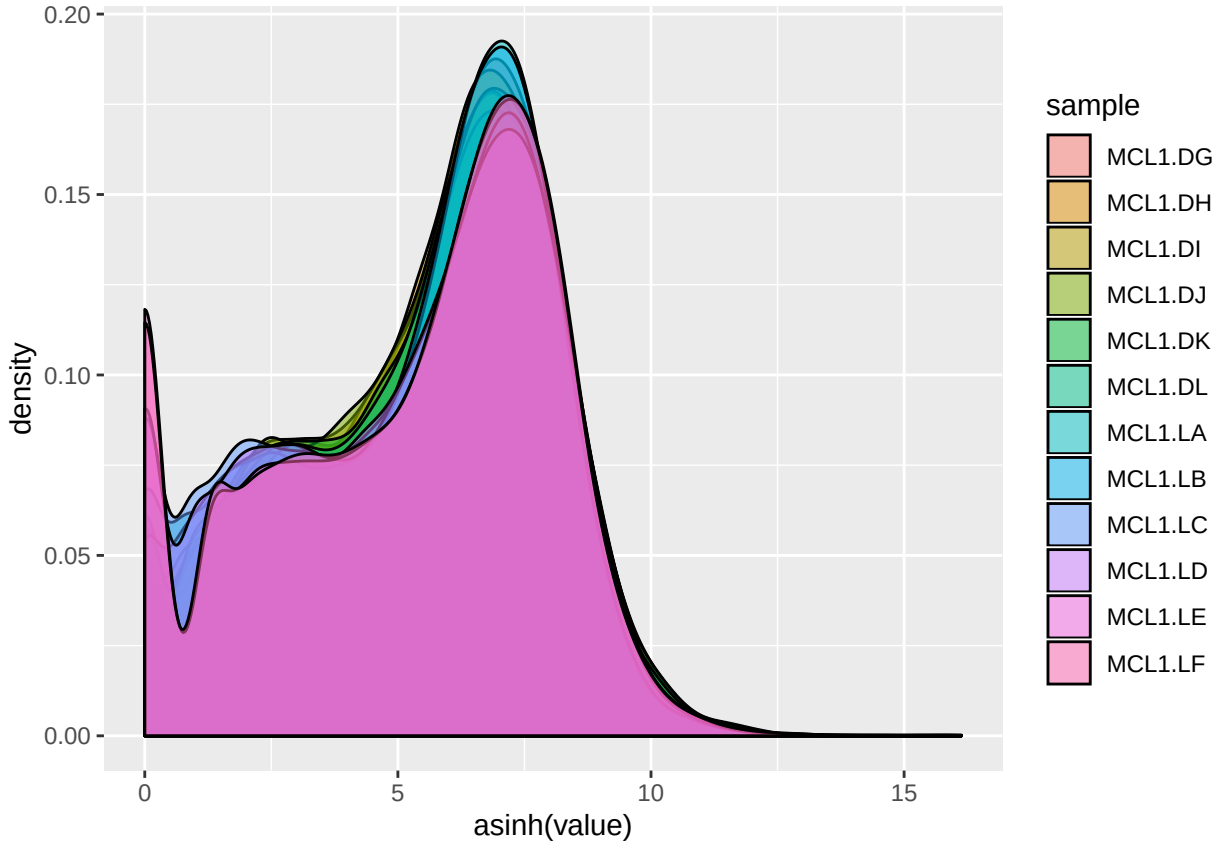
Normalized data can now be retrieved using `counts(dds.norm, normalized=TRUE)`. Let's see what the normalized data looks like:

```
library(reshape2)
library(ggplot2)
df_norm  <- as.data.frame(counts(dds.norm, normalized=TRUE))
df_norm [["GeneID"]] <- rownames(df_norm )
df_melt_norm <- melt(df_norm , id.vars = "GeneID")
df_melt_norm <- dplyr::rename(df_melt_norm,  sample = variable)
```

```
ggplot(df_melt_norm, aes(x=sample, y = asinh(value), fill = sample)) +
  theme(axis.text.x = element_text(angle = 90)) +
  geom_boxplot(alpha = 0.5)
```



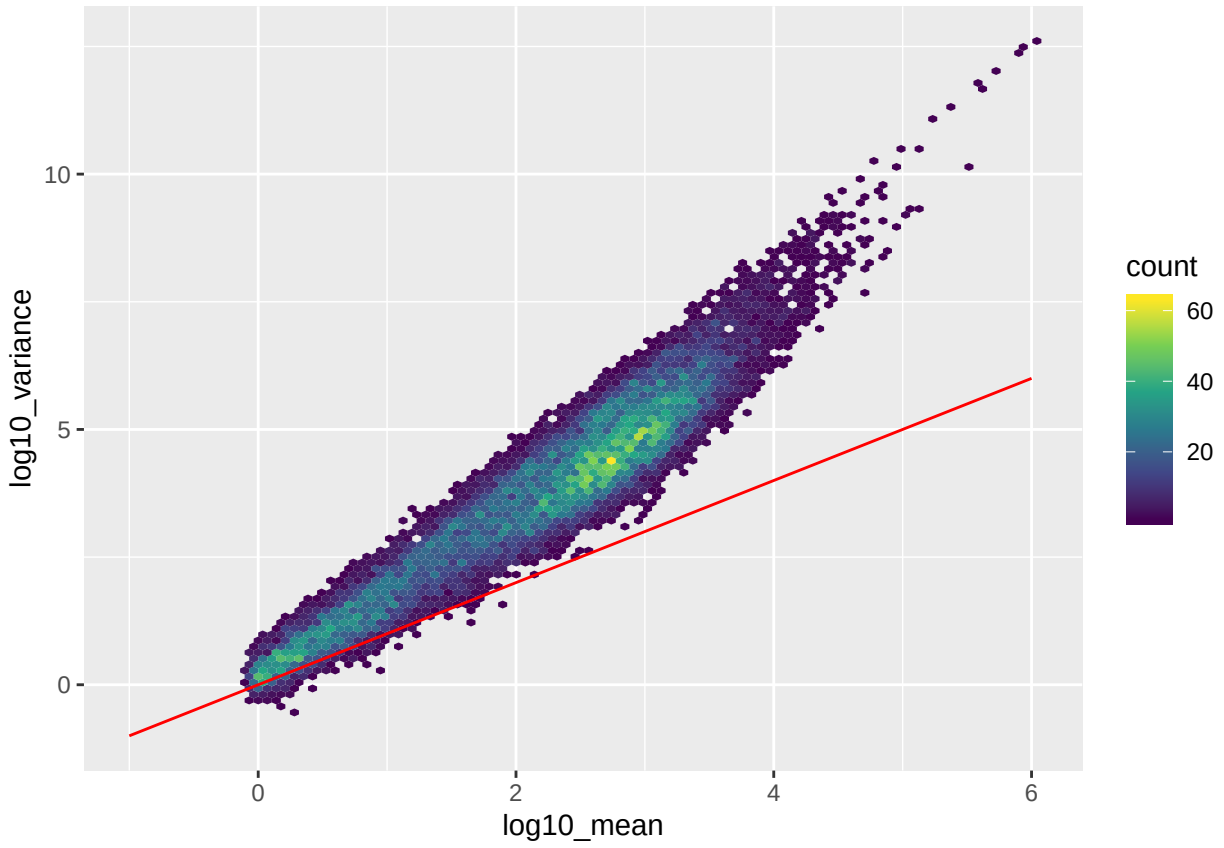

```
ggplot(df_melt_norm, aes(x = asinh(value), fill = sample)) +
  geom_density(alpha = 0.5)
```



Mean-variance relationship

As you can see from the following plot the relationship between variance and mean is not linear as would be the case for a Poisson distribution (red line).

```
## Computing mean and variance
df_norm <- as.data.frame( counts(dds.norm, normalized=TRUE) )
mean <- apply(df_norm, 1, mean, na.rm = TRUE)
variance <- apply(df_norm, 1, var, na.rm = TRUE)

df_norm$log10_mean <- log10(mean)
df_norm$log10_variance <- log10(variance)


p1<-ggplot(df_norm, aes(x=log10_mean, y=log10_variance)) +
  geom_hex(bins=100) +
  scale_fill_viridis_c()+
  annotate("segment", x = -1, y = -1, xend = 6, yend=6, color = "red")

p1
```

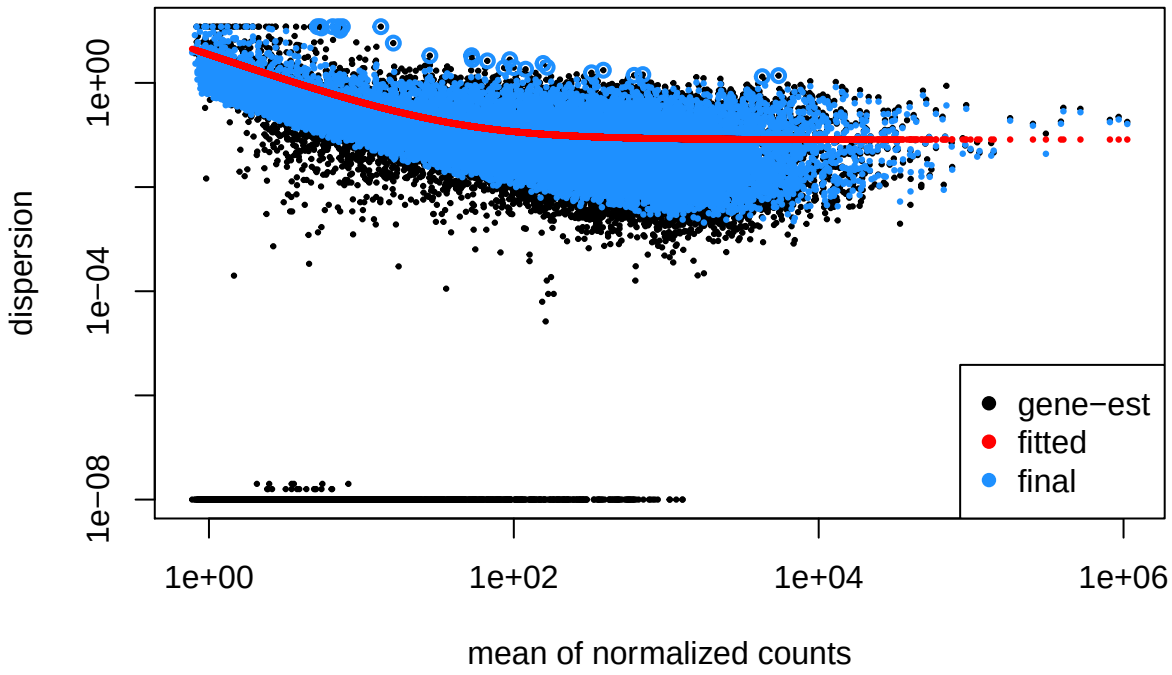


Modelling read counts through a negative binomial

To perform diffential expression call DESeq will assume that, for each gene, the read counts are generated by a negative binomial distribution. One problem here will be to estimate, for each gene, the two parameters of the negative binomial distribution: mean and dispersion.

-The mean will be estimated from the observed normalized counts in both conditions. -The first step will be to compute a gene-wise dispersion. When the number of available samples is insufficient to obtain a reliable estimator of the variance for each gene, DESeq will apply a shrinkage strategy, which assumes that counts produced by genes with similar expression level (counts) have similar variance (note that this is a strong assumption). DESeq will regress the gene-wise dispersion onto the means of the normalized counts to obtain an estimate of the dispersion that will be subsequently used to build the binomial model for each gene. Performing estimation of dispersion parameter and display a diagnostic plot

```
## Performing estimation of dispersion parameter
dds.disp <- estimateDispersions(dds.norm)

## A diagnostic plot which
## shows the mean of normalized counts (x axis)
## and dispersion estimate for each genes
plotDispEsts(dds.disp)
```



####Performing differential expression

Now that a negative binomial model has been fitted for each gene, the nbinomWaldTest can be used to test for differential expression.

```
dds.wald <- nbinomWaldTest(dds.disp)
resultsNames(dds.wald)
```

```
## [1] "Intercept"                  "CellType_luminal_vs_basal"
## [3] "Status_pregnant_vs_lactate" "Status_virgin_vs_lactate"
```

As before, we can see the result table. We keep only results with adjusted p-values (correction for multiple tests computed with the Benjamini–Hochberg procedure) below 0.01 using `alpha`.

```
res.wald <- results(dds.wald, alpha=0.01, pAdjustMethod="BH", name = "CellType_luminal_vs_basal")
summary(res.wald)
```

```
##
## out of 18178 with nonzero total read count
## adjusted p-value < 0.01
## LFC > 0 (up)       : 4946, 27%
## LFC < 0 (down)     : 5579, 31%
## outliers [1]       : 0, 0%
## low counts [2]     : 0, 0%
## (mean count < 1)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

```
head(res.wald)
```

```
## log2 fold change (MLE): CellType luminal vs basal
## Wald test p-value: CellType luminal vs basal
## DataFrame with 6 rows and 6 columns
##                baseMean     log2FoldChange             lfcSE                stat
##               <numeric>          <numeric>         <numeric>           <numeric>
## 497097 126.481585433975  -10.1820437902142  0.930694407250738 -10.9402653662569
```

```
## 19888  1.95697709143844   3.06715205617399   1.23908077239746  2.47534472691353
## 20671  50.5979096011351  -2.71203870035567  0.337521822904564 -8.03515066675411
## 27395  358.719720532762  0.799404687906072  0.127971895259294   6.2467207060295
## 18777  744.356910502922  0.106722300368205  0.134155732952959 0.795510545983351
## 21399  1097.91010559132 -0.361177585114154 0.0647349105685968 -5.57933241803787
##                      pvalue                padj
##                   <numeric>           <numeric>
## 497097 7.39830424216018e-28 1.00288124171505e-26
## 19888    0.0133107627219475   0.0215288766580267
## 20671  9.34639816135234e-16 6.19842487329671e-15
## 27395  4.19159587750374e-10  1.7408003166841e-09
## 18777     0.426316578845322    0.488097690960623
## 21399  2.41443432722022e-08 8.38387530089954e-08
```

References

Fu, Nai Yang, Anne C Rios, Bhupinder Pal, Rina Soetanto, Aaron T L Lun, Kevin Liu, Tamara Beck, et al. 2015. "EGF-mediated induction of Mcl-1 at the switch to lactation is essential for alveolar cell survival." *Nature Cell Biology* 17 (4): 365–75. https://doi.org/10.1038/ncb3117.