# RNA-seq analysis in R - First steps

## Contents



### Disclaimer

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

### Setup

If you haven't done so already, you should 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.

### 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.

### Load data

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

```
seqdata <- read.delim("./data/GSE60450_Lactation-GenewiseCounts.txt", stringsAsFactors = FALSE)
sampleinfo <- read.delim("./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"
##  [2] "Length"
##  [3] "MCL1.DG_BC2CTUACXX_ACTTGA_L002_R1"
##  [4] "MCL1.DH_BC2CTUACXX_CAGATC_L002_R1"
##  [5] "MCL1.DI_BC2CTUACXX_ACAGTG_L002_R1"
##  [6] "MCL1.DJ_BC2CTUACXX_CGATGT_L002_R1"
##  [7] "MCL1.DK_BC2CTUACXX_TTAGGC_L002_R1"
##  [8] "MCL1.DL_BC2CTUACXX_ATCACG_L002_R1"
##  [9] "MCL1.LA_BC2CTUACXX_GATCAG_L001_R1"
## [10] "MCL1.LB_BC2CTUACXX_TGACCA_L001_R1"
## [11] "MCL1.LC_BC2CTUACXX_GCCAAT_L001_R1"
## [12] "MCL1.LD_BC2CTUACXX_GGCTAC_L001_R1"
## [13] "MCL1.LE_BC2CTUACXX_TAGCTT_L001_R1"
## [14] "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 10 reads total.

```
keep <- rowMeans(countdata) >= 10
countdata <- countdata[keep, ]
```

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

## Adding 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"
##  [5] "ENSEMBLTRANS" "ENTREZID"     "ENZYME"       "EVIDENCE"
##  [9] "EVIDENCEALL"  "GENENAME"     "GO"           "GOALL"
## [13] "IPI"          "MGI"          "ONTOLOGY"     "ONTOLOGYALL"
## [17] "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
## 14802
```

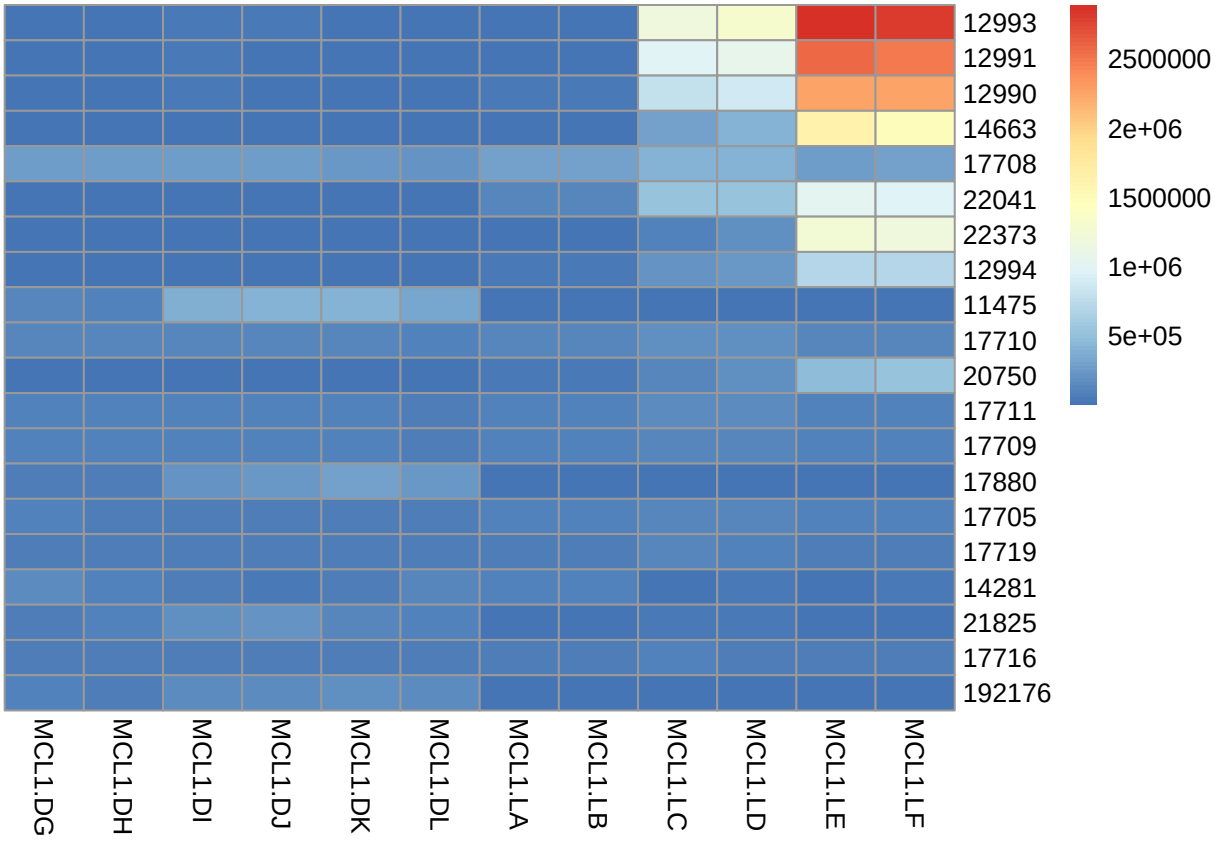
## Visualization

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

### Heatmap

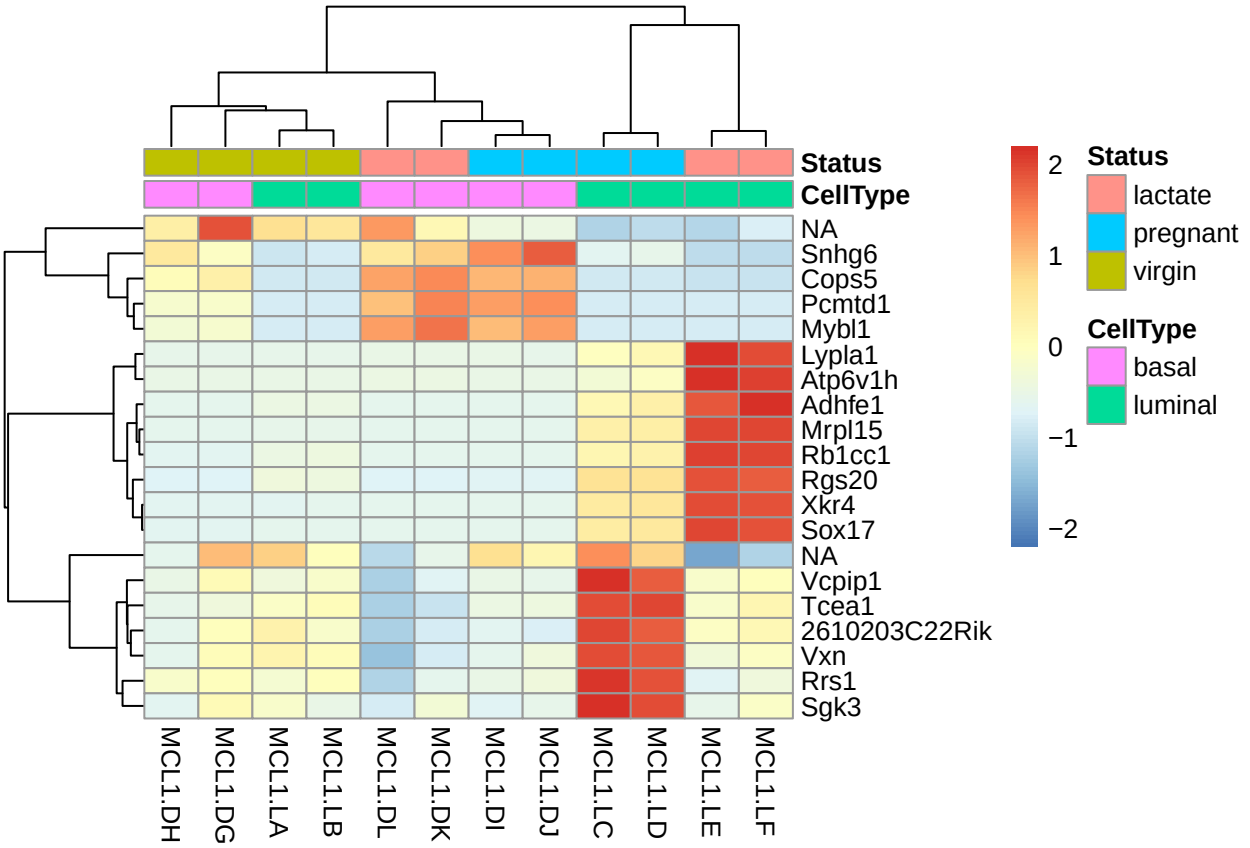
To explore a count matrix, it is often instructive to look at it as a heatmap. Below we show how to produce such a heatmap. We focus on the 20 genes with the highest average counts.

```
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.

```
df <- sampleinfo[ , c("CellType","Status")]

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




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

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

### 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     78.0771 67.7731 35.84184 34.65881 26.68514 17.39414
## Proportion of Variance  0.4118  0.3103  0.08679  0.08115  0.04811  0.02044
## Cumulative Proportion   0.4118  0.7221  0.80894  0.89009  0.93820  0.95864
##                             PC7      PC8     PC9     PC10    PC11
## Standard deviation     13.19289 11.90864 10.7471 10.50096 8.40331
## Proportion of Variance  0.01176  0.00958  0.0078  0.00745 0.00477
## Cumulative Proportion   0.97040  0.97998  0.9878  0.99523 1.00000
##                            PC12
## Standard deviation     9.726e-14
## Proportion of Variance 0.000e+00
```

```
## Cumulative Proportion  1.000e+00
```

You obtain 12 principal components, each one explaining a percentage of the total variation in the dataset (PC1 explains 41%, PC2 31% and so on). The relationship (correlation or anticorrelation, a.k.a loadings) between the initial variables and the principal components is in `$rotation` while the values of each sample in terms of the principal components is in `$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 -85.4128676  -15.324399
## MCL1.DH -70.8062008    8.038389
## MCL1.DI -75.4682832   14.458654
## MCL1.DJ -59.8570496   41.542638
## MCL1.DK -54.1608560   67.168739
## MCL1.DL -48.4729401   67.166409
## MCL1.LA   7.3383158 -110.534319
## MCL1.LB   0.8444046 -127.360495
## MCL1.LC  66.3270680  -49.601744
## MCL1.LD  73.1942669  -21.354719
## MCL1.LE 123.0025302   62.774510
## MCL1.LF 123.4716118   63.026338
```

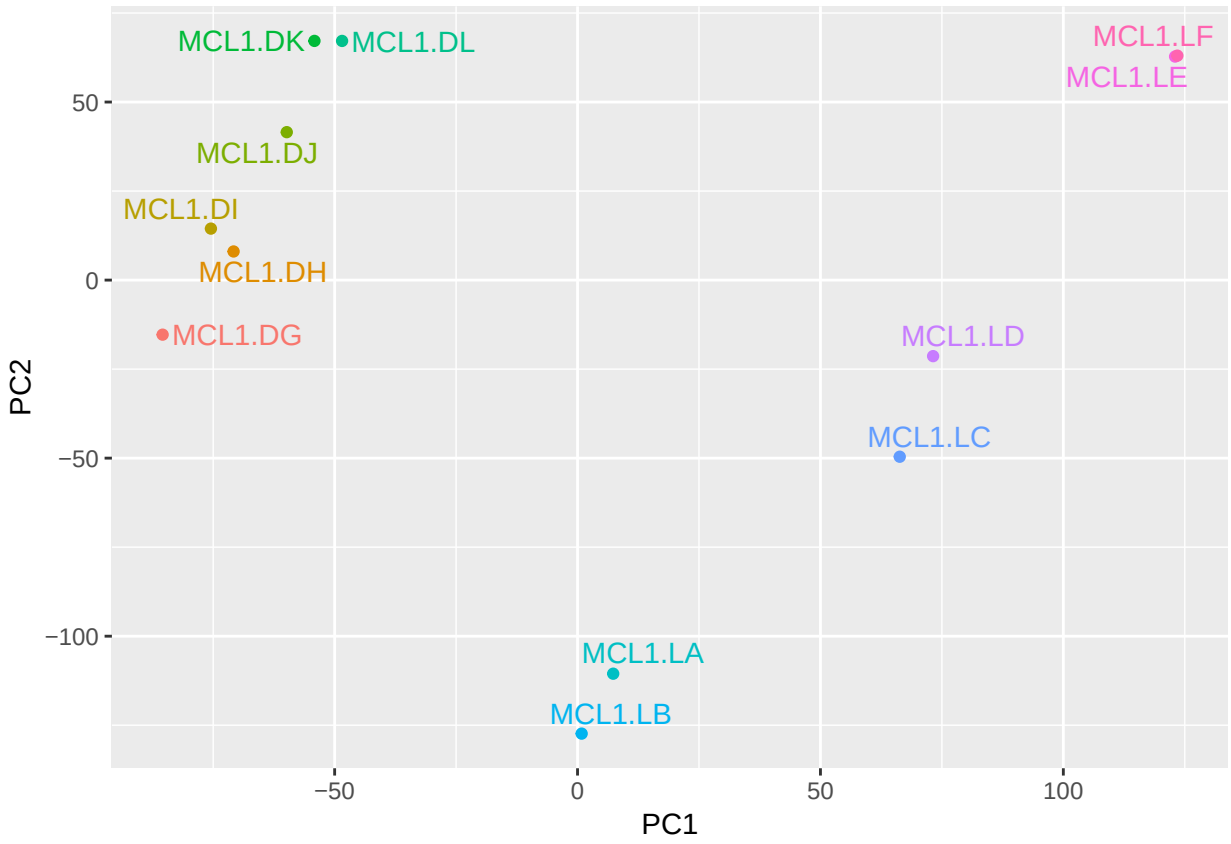
We'll use the `ggplot2` package to plot the results of the PCA. See the R for data science book for an introduction to ggplot2. We first need to create a data frame from the pca results.

```
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"
##  [8] "PC8"    "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)
```

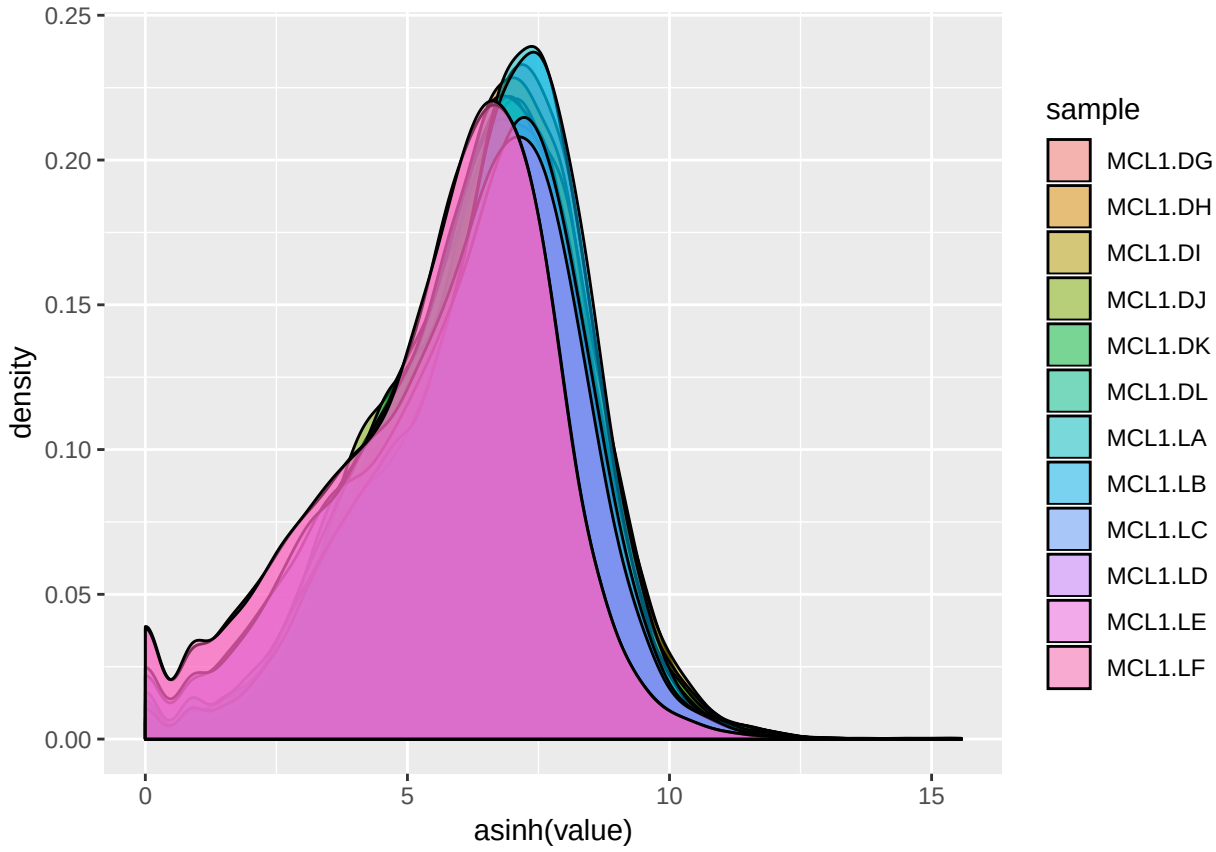
### Manipulating the data

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

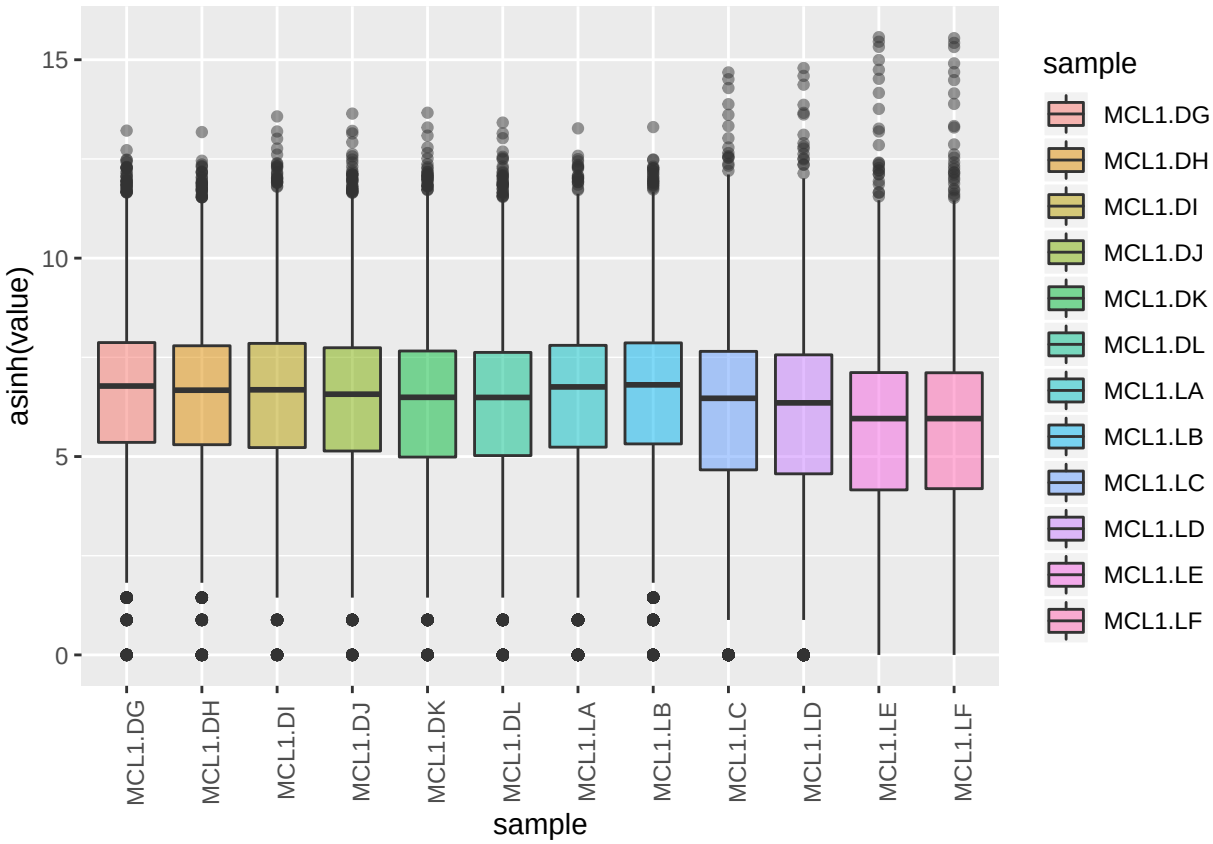
As an example, we can now plot the distribution of asinh transformed count values for each sample.

```
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.

```
library(ggridges)

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.

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.

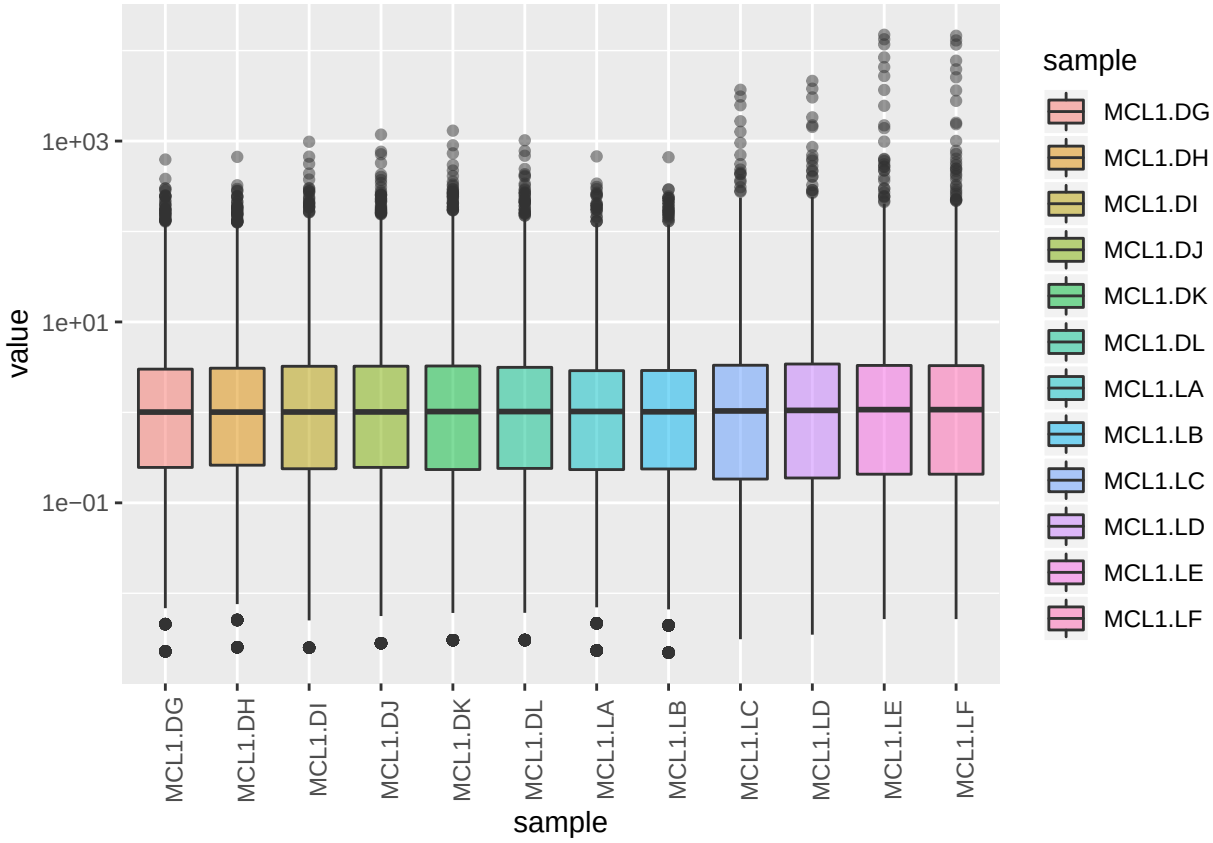
```
df_median <-
  df_melt %>%
  group_by(sample) %>%
  summarise(median = median(value, na.rm = TRUE))

df_median
```

```
## # A tibble: 12 x 2
##    sample  median
##    <fct>    <dbl>
##  1 MCL1.DG   438.
##  2 MCL1.DH   395
##  3 MCL1.DI   399
##  4 MCL1.DJ   357
##  5 MCL1.DK   330
##  6 MCL1.DL   329
##  7 MCL1.LA   429
##  8 MCL1.LB   452
##  9 MCL1.LC   322
## 10 MCL1.LD   287
## 11 MCL1.LE   193
## 12 MCL1.LF   193
```

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

See how we can normalize the data using the median:

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

plot <- ggplot(df_melt_norm, aes(x=sample, y = value, fill = sample)) +
  theme(axis.text.x = element_text(angle = 90)) +
  geom_boxplot(alpha = 0.5) +
  scale_y_log10()
plot
```



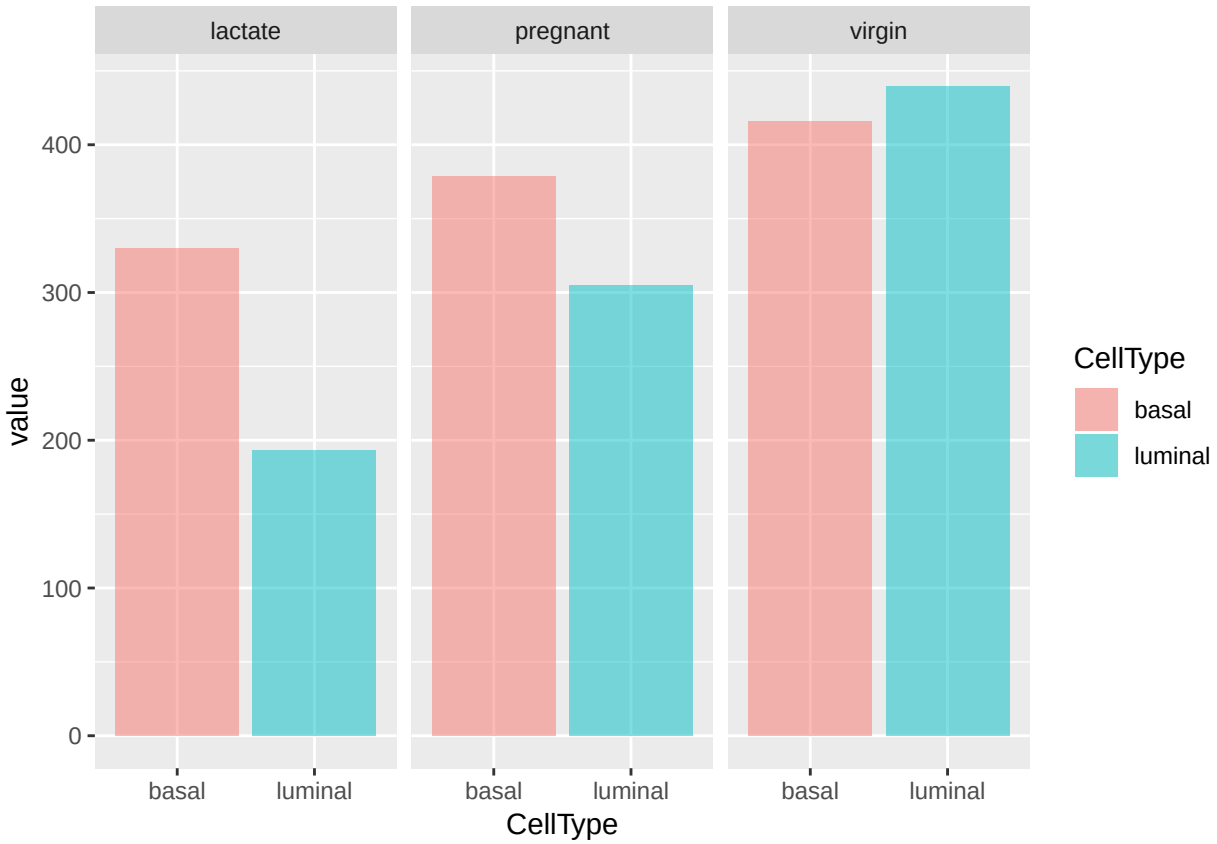

### Using metadata

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, ])
```

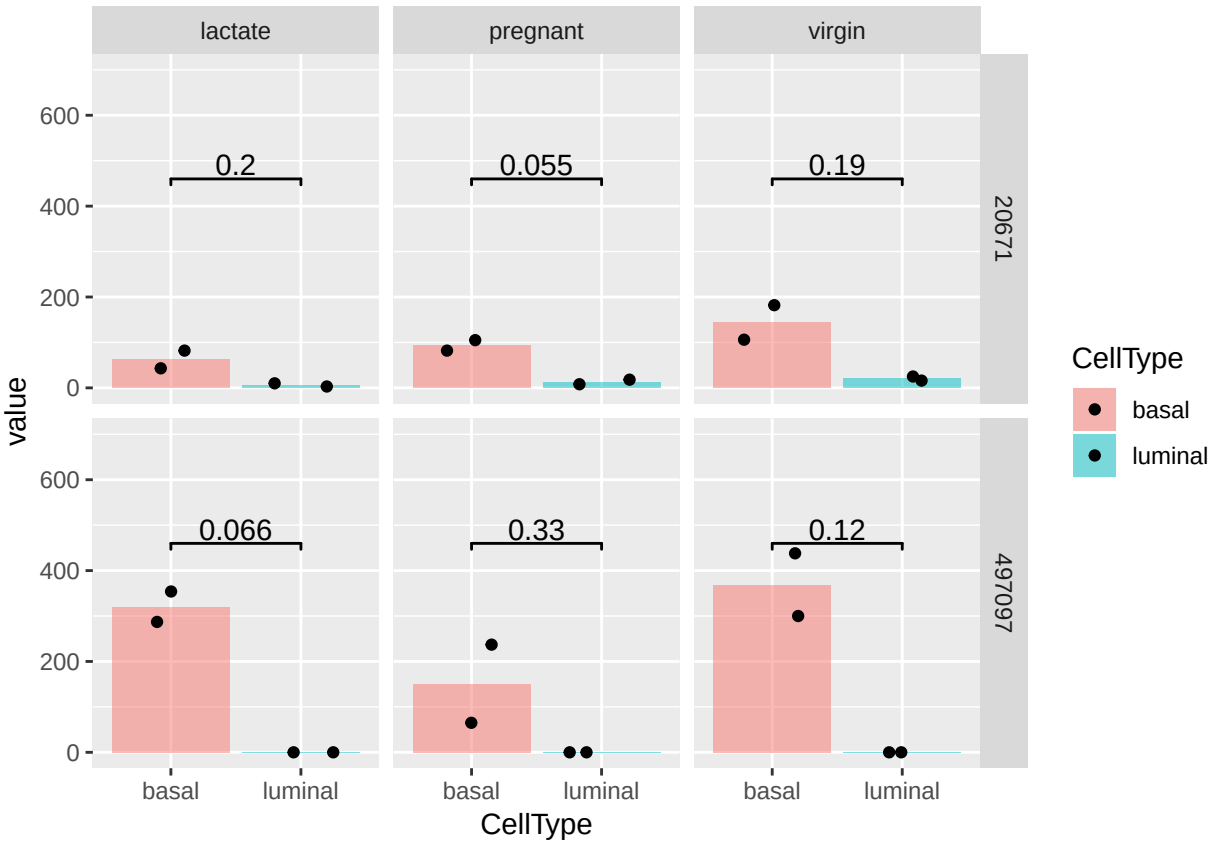
Now we have access to two more variables, `CellType` and `Status`, that we could use to form groups.

```
ggplot(df_melt, aes(x = CellType, y = 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 an dsplit 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
  ylim(c(0,700))+
  facet_grid(GeneID~Status)
```




## Creating a `DESeqDataSet`

We will use the `DESeq2` package. 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 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 `count(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. We'll see that it becomes quite straightforward to normalize data and perform differential expression analysis.

## 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. doi:10.1038/ncb3117.