VI. Bioinformatics in R (presentation)

Center for Health Data Science, University of Copenhagen

06 July, 2021

Bioconductor

Bioconductor provides tools for computational biology and bioinformatics analysis in R - it is open source and open development and it has an active user community.
Mostly when we install R-packages we use `install.packages('name_of_package')`. When we use this command we refer to the CRAN repository of packages, however sometimes we want a package from `Bioconductor` instead. For this we use the command `BiocManager::install('name_of_package')`. In order to use this installer, you need to download the R-package `BiocManager` e.g. `install.packages('BiocManager')`.

Gene Expression Analysis in R with `DEseq2`

DEseq2 is one of the many packages/frameworks which exists for analysis of bulk gene expression data in R. For more information on DEseq2, please have a look at the original publication here.

Other highly used packages for differential expression analysis *DEA* are:

- limma
- edgeR
- NOIseq

`DEseq2` has many advantages over classical models and post hoc tests, as it is specifically developed for handling common issues and biases in expression data, including differences in sequencing depth and highly variable dispersion of counts between genes.
In brief, `DEseq2` fits a generalized linear model (GLM) for each gene in the dataset. In the case where we compare two groups i.e. treatment vs control, the GLM fit returns coefficients indicating the overall expression strength of a gene, along with the log-2 fold change between groups. `DEseq2` adjusts variable gene dispersion estimates using an empirical Bayes approach which borrows information across genes and shrinks gene-wise dispersions towards a common dispersion trend to increase accuracy of differential expression testing.

About the Dataset

The dataset used for this presentation was acquired from the following github tutorial on RNAseq analysis: https://combine-australia.github.io/RNAseq-R/06-rnaseq-day1.html.

RNA sequencing data generated from luminal and basal cell sub-populations in the mammary gland of three groups of mice:
- Control
- Pregnant
- Lactating

The objective of the original study (found here) was to identify genes specifically expressed in lactating mammary glands, the gene expression profiles of luminal and basal cells from different developmental stages were compared.

Load R-packages:

```
# Data Wrangling
# install.packages("tidyverse")
# install.packages("readxl")
library(tidyverse)
library(readxl)

# For Plotting
# install.packages("ggplot2")
# install.packages("viridis")
library(ggplot2)
library(viridis)


# For DEA
# install.packages("BiocManager")
# BiocManager::install("DESeq2")
library(BiocManager)
library(DESeq2)


# For Enrichment Analysis
# BiocManager::install("clusterProfiler")
# BiocManager::install("org.Mm.eg.db")
library(clusterProfiler)
library(org.Mm.eg.db)
```

Importing Data

Reading in data:

```
exprDat <- read_excel("MouseRNAseq.xlsx")
exprInfo <- read_excel("MouseSampleInfo.xlsx")

# Look at the data:
head(exprDat, n=5)
```

```
## # A tibble: 5 x 14
##   EntrezGeneID GeneName MCL1.DG MCL1.DH MCL1.DI MCL1.DJ MCL1.DK MCL1.DL MCL1.LA
##   <chr>        <chr>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 497097       Xkr4         438     300      65     237     354     287       0
## 2 19888        Rp1            1       1       0       0       0       0      10
## 3 20671        Sox17        106     182      82     105      43      82      16
## 4 27395        Mrpl15       309     234     337     300     290     270     560
```

```
## 5 18777        Lypla1       652     515     948     935     928     791     826
## # ... with 5 more variables: MCL1.LB <dbl>, MCL1.LC <dbl>, MCL1.LD <dbl>,
## #   MCL1.LE <dbl>, MCL1.LF <dbl>
```

```r
dim(exprDat)
```

```
## [1] 23151    14
```

```r
head(exprInfo)
```

```
## # A tibble: 6 x 5
##   SampleName CellType Status   Status.Type    CellType.colors
##   <chr>      <chr>    <chr>    <chr>          <chr>
## 1 MCL1.DG    basal    control  control.basal  #79ADDC
## 2 MCL1.DH    basal    control  control.basal  #79ADDC
## 3 MCL1.DI    basal    pregnant pregnant.basal #79ADDC
## 4 MCL1.DJ    basal    pregnant pregnant.basal #79ADDC
## 5 MCL1.DK    basal    lactate  lactate.basal  #79ADDC
## 6 MCL1.DL    basal    lactate  lactate.basal  #79ADDC
```

Convert character columns to factor types:

```r
exprInfo <- exprInfo %>%
  mutate(CellType = as.factor(CellType),
         Status = factor(Status, levels = c("control", "pregnant", "lactate")),
         Status.Type = as.factor(Status.Type))

head(exprInfo)
```

```
## # A tibble: 6 x 5
##   SampleName CellType Status   Status.Type    CellType.colors
##   <chr>      <fct>    <fct>    <fct>          <chr>
## 1 MCL1.DG    basal    control  control.basal  #79ADDC
## 2 MCL1.DH    basal    control  control.basal  #79ADDC
## 3 MCL1.DI    basal    pregnant pregnant.basal #79ADDC
## 4 MCL1.DJ    basal    pregnant pregnant.basal #79ADDC
## 5 MCL1.DK    basal    lactate  lactate.basal  #79ADDC
## 6 MCL1.DL    basal    lactate  lactate.basal  #79ADDC
```

Initial Data Check & Filtering:

Firstly, we will give our count distributions a look. You could plot all gene counts together or even better sample a couple to get an idea of what they look like individually.

All together:

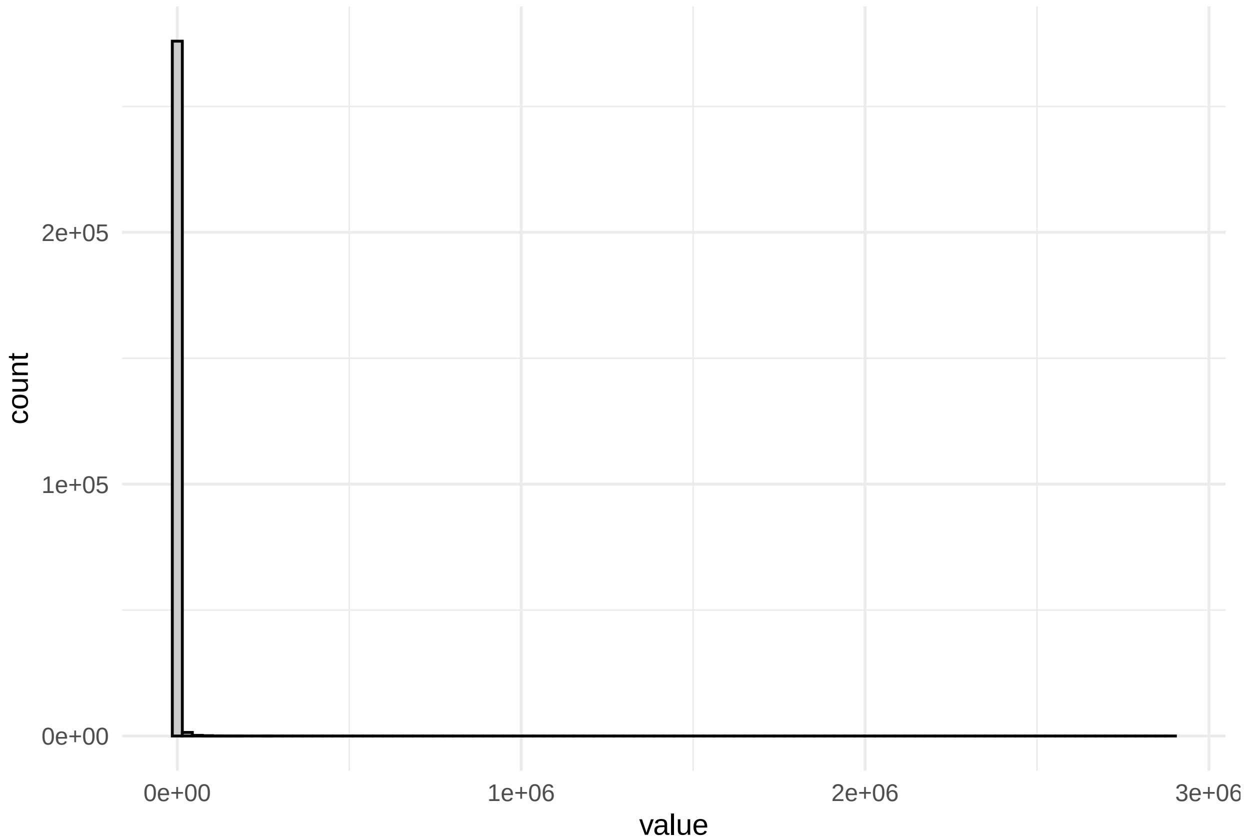
```r
expr1 <- exprDat %>%
  dplyr::select(-EntrezGeneID, -GeneName) %>%
  gather() %>%
  dplyr::select(value) %>%
  mutate(valuelog2 = log2(value+1))

head(expr1, n=5)
```

```
## # A tibble: 5 x 2
```

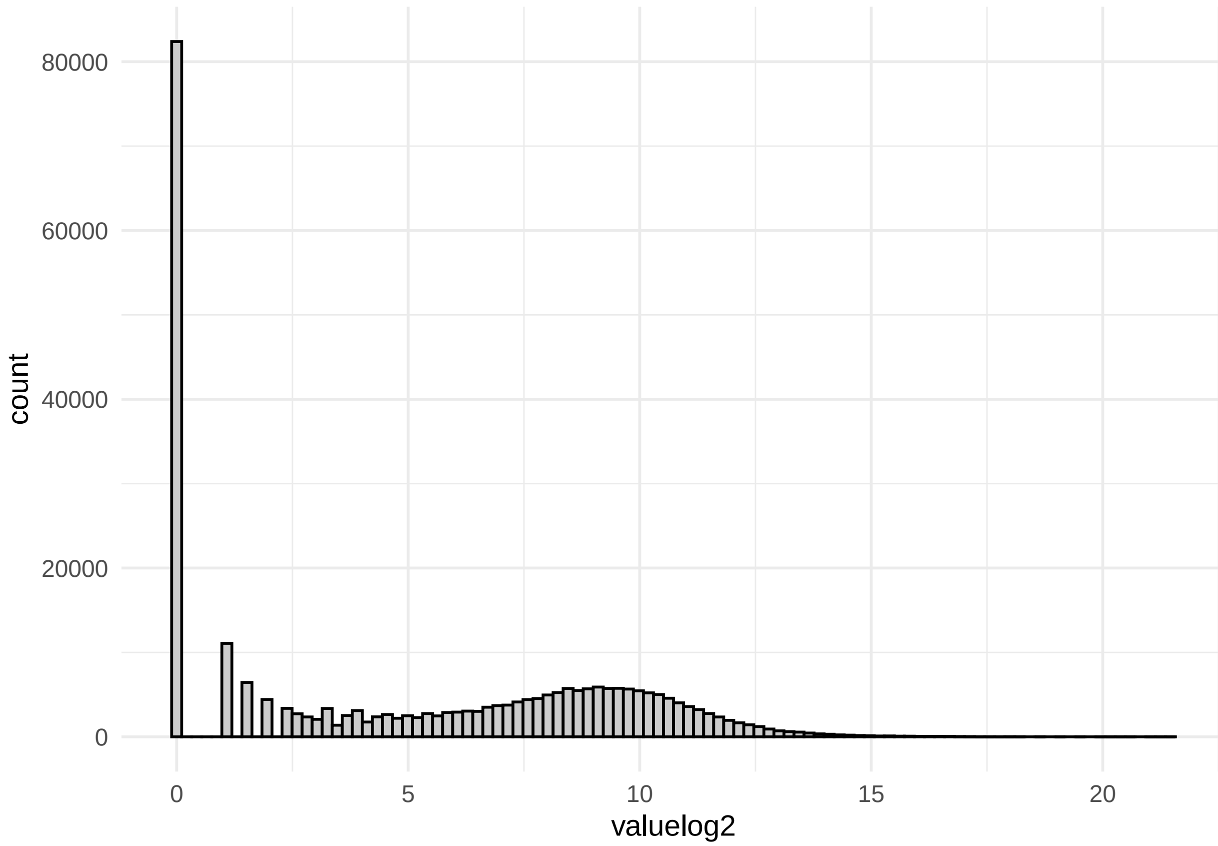
```
##   value valuelog2
##   <dbl>     <dbl>
## 1   438      8.78
## 2     1      1
## 3   106      6.74
## 4   309      8.28
## 5   652      9.35
```

Plot it with ggplot2:

```
p1 <- ggplot(expr1, aes(value)) +
  geom_histogram(color="black", fill="grey80", bins=100) +
  theme_minimal()
p1
```



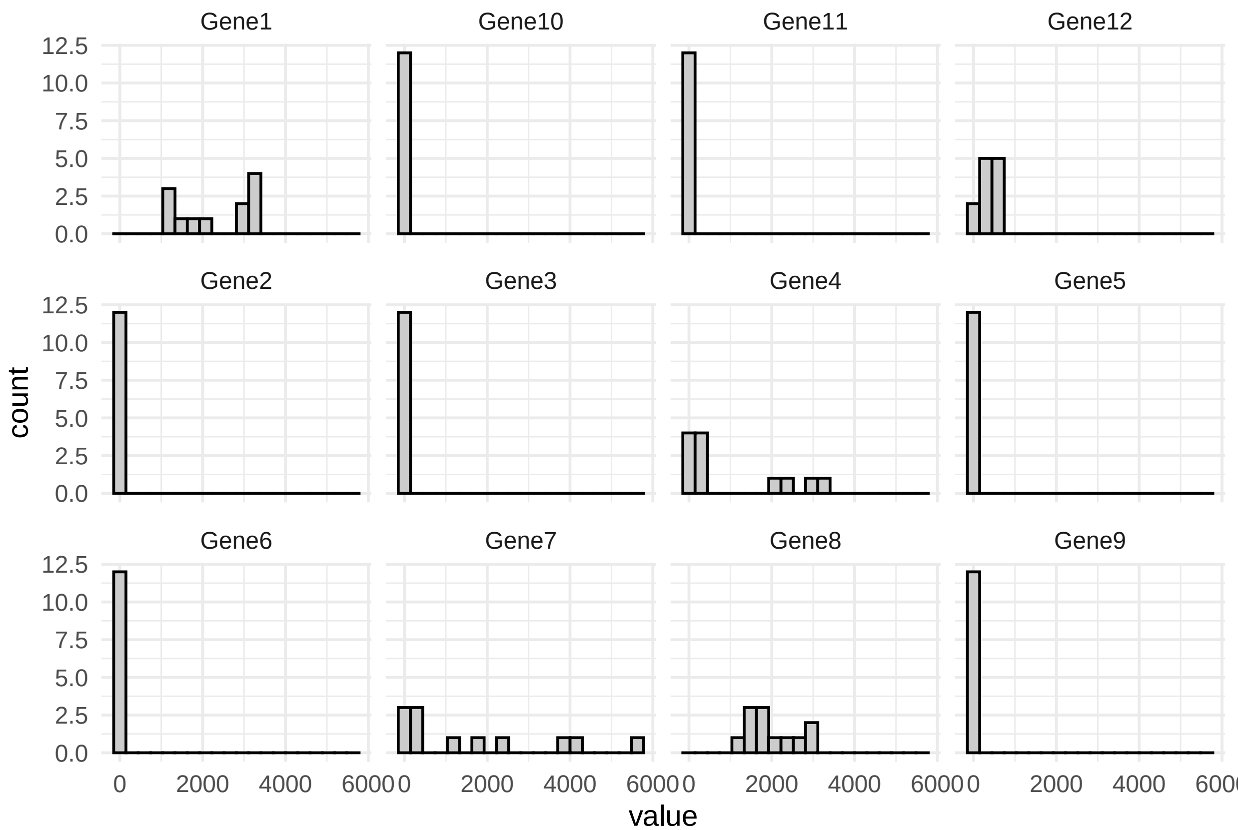

```
p2 <- ggplot(expr1, aes(valuelog2)) +
  geom_histogram(color="black", fill="grey80", bins=100) +
  theme_minimal()

p2
```



The first plot tells us that we have a lot of 0 counts. Let's try to sample n random genes and plot their count distribution.

```
expr10 <- exprDat %>%
  dplyr::select(-EntrezGeneID, -GeneName) %>%
  sample_n(.,12) %>%
  t() %>%
  as_tibble() %>%
  rename_at(vars(names(.)), ~paste0("Gene", seq(1:12))) %>%
  gather()
```

```
ggplot(expr10, aes(value)) +
  geom_histogram(color="black", fill="grey80", bins=20) +
  theme_minimal() +
  facet_wrap(~key)
```



We will filter out genes with too many zero counts. First, pull out GeneNames, then count number of 0s across samples for each gene and filter:

```
# Pull out GeneName
GeneNames <- exprDat %>%
  dplyr::select(EntrezGeneID, GeneName)

# Count number of 0s across samples. Filter samples where at least four samples has a count great than 
exprDat <- exprDat %>%
  dplyr::select(-EntrezGeneID, -GeneName) %>%
  mutate(nzeros = rowSums(.==0)) %>%
  bind_cols(GeneNames,.) %>%
  filter(nzeros <= 8) %>%
  dplyr::select(-nzeros)

#How many genes do we have left:
dim(exprDat)
```

```
## [1] 17308    14
```

Differential Expression Analysis- DESeq2

We will now make a DESeq2 object. For this we use the function `DESeqDataSetFromMatrix` from the DEseq2 package. As input we give our count matrix, our gene IDs and our meta data (exprInfo). Additionally we include a design for DE contrasts. In this case we add CellType (luminal or basal) and Status (control, pregnant or lactating).

Convert to exprDat to a dataframe and make GenNames column into rownames:

```
exprDat <- exprDat %>%
  dplyr::select(-EntrezGeneID) %>%
  column_to_rownames(., var = "GeneName")
```

Make a DESeq2 object:

```
exprObj <- DESeqDataSetFromMatrix(countData = exprDat,
                                  colData = exprInfo,
                                  design= ~CellType+Status)
exprObj
```

```
## class: DESeqDataSet
## dim: 17308 12
## metadata(1): version
## assays(1): counts
## rownames(17308): Xkr4 Rp1 ... Uty Gm47283,
## rowData names(0):
## colnames(12): MCL1.DG MCL1.DH ... MCL1.LE MCL1.LF
## colData names(5): SampleName CellType Status Status.Type
##   CellType.colors
```

Next, we use `DEseq()` to estimate dispersion, gene-wise and mean-dispersion, fitting model(s):

```
exprObj <- DESeq(exprObj)
```

Preliminary analysis:

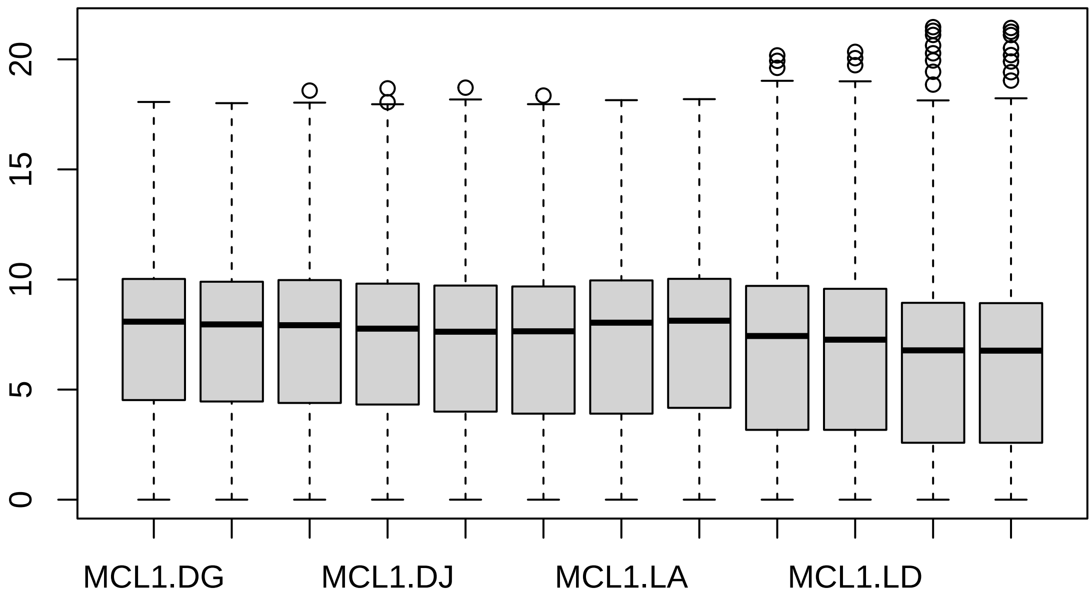
Let's have a look at the library sizes:

```
colSums(assay(exprObj))
```

```
##  MCL1.DG  MCL1.DH  MCL1.DI  MCL1.DJ  MCL1.DK  MCL1.DL  MCL1.LA  MCL1.LB
## 22634514 21155013 23488082 22100122 21057113 19583106 19698631 20944796
##  MCL1.LC  MCL1.LD  MCL1.LE  MCL1.LF
## 21675050 21457888 24419457 24366629
```

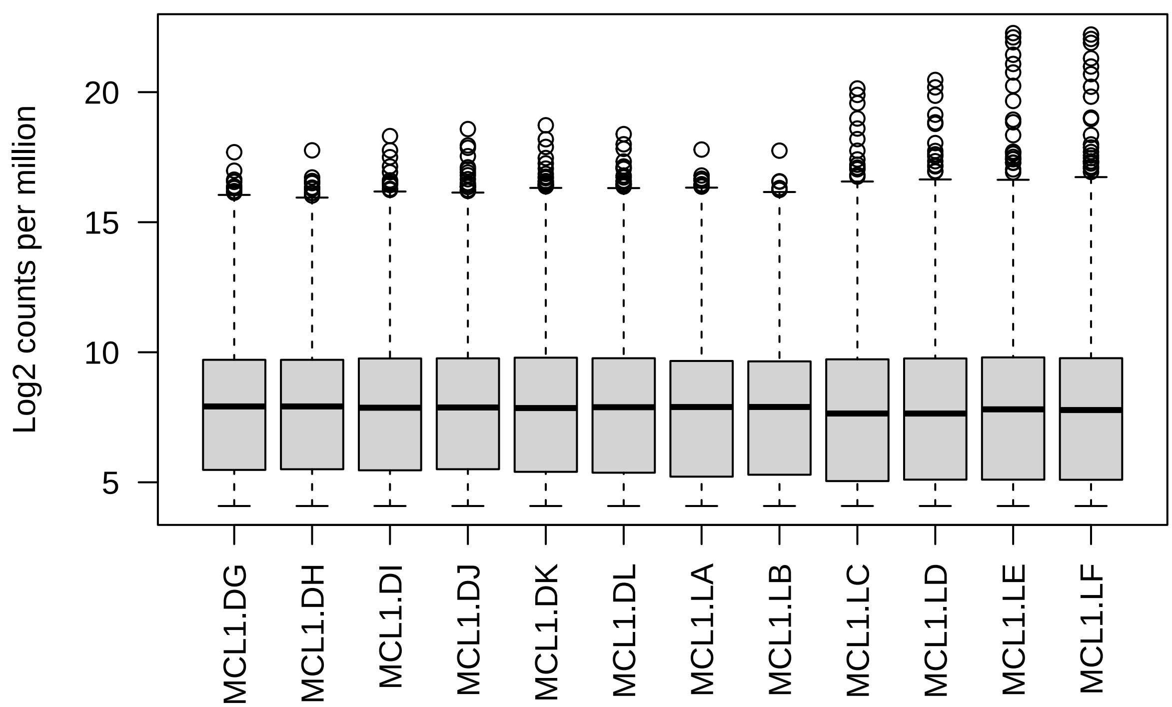
The count distributions may be dominated by a few genes with very large counts. These genes will drive plotting e.g. heatmaps, PCA analysis etc. Let's see if we have any genes with high large counts and in turn, dispersion in our dataset. For convenience I am using the base R boxplot function:

```
#boxplot(assay(exprObj))
boxplot(log2(assay(exprObj)+1))
```



We perform variance stabilizing transformation to obtain log2 counts per million read mapped, overcoming issues with outlier genes and sequencing depth:

```
exprObjvst <- vst(exprObj,blind=FALSE)
boxplot(assay(exprObjvst), xlab="", ylab="Log2 counts per million",las=2)
```

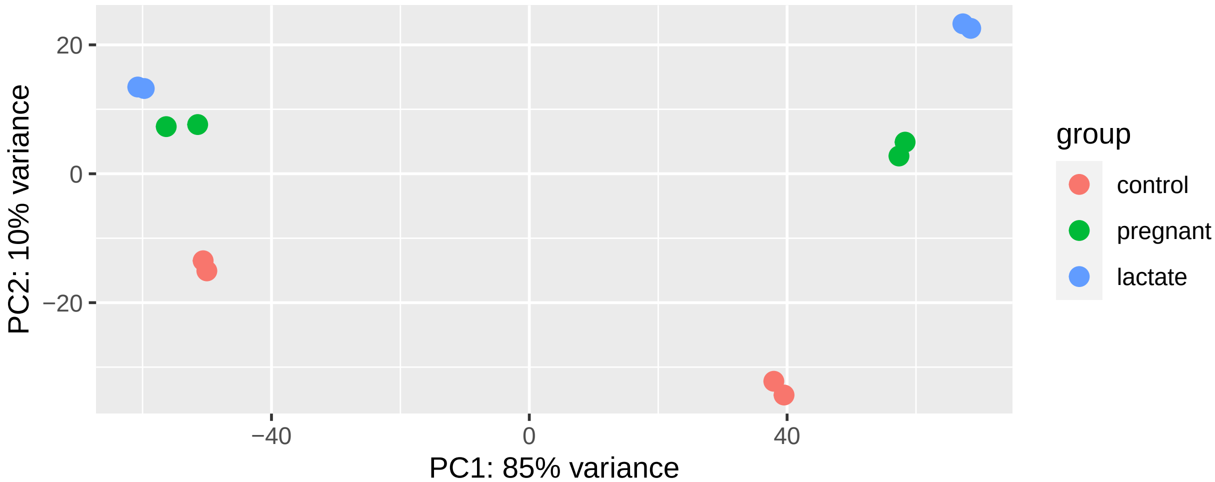


Principal Component Analysis

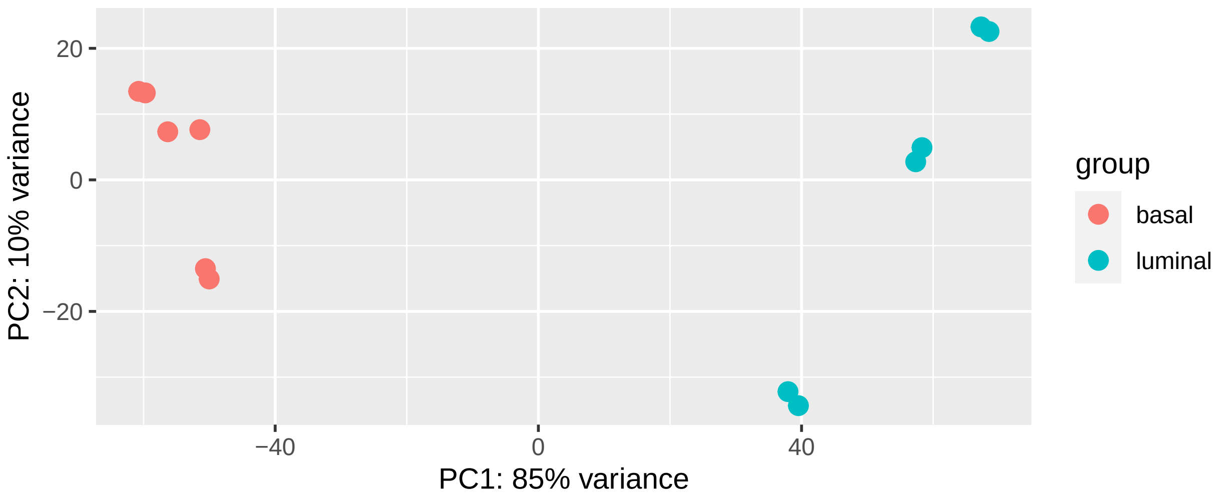
Before performing DEA it is a good idea to explore how samples cluster together based on there gene expression profile. The expectation here is that samples from the same group (treatment vs control, condition

A vs condition B, etc.) will cluster together. A principal component analysis (PCA) plot can also help us to identify outlier samples which might need to be removed from the analysis. We use our vst counts for principal component analysis:

```
plotPCA(exprObjvst,intgroup=c("Status"))
```




```
plotPCA(exprObjvst,intgroup=c("CellType"))
```




```
#plotPCA(exprObj,intgroup=c("TypeStatus"))
```

Testing

Have a look at the group comparisons:

```
resultsNames(exprObj)
```

```
## [1] "Intercept"                   "CellType_luminal_vs_basal"
## [3] "Status_pregnant_vs_control" "Status_lactate_vs_control"
```

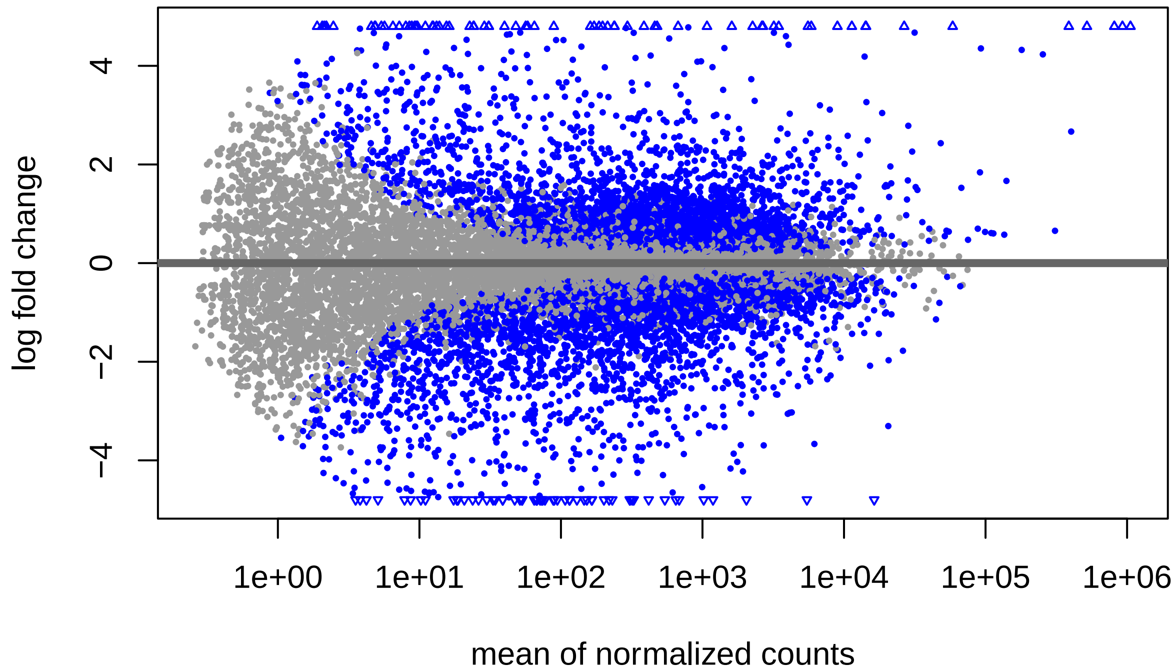
Test for DE genes between the three groups of mice, adjusted for cell type:

(I) lactating and control mice:

```
resLC <- results(exprObj, contrast = c("Status", "lactate", "control"), independentFiltering = FALSE)
```

Summary and plot of DE analysis results:

```
DESeq2::plotMA(resLC)
```




```
summary(resLC)
```

```
##
## out of 17308 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 3550, 21%
## LFC < 0 (down)     : 3474, 20%
## outliers [1]       : 0, 0%
## low counts [2]     : 0, 0%
## (mean count < 0)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Custom function to filter results of DEA. We make a function to save writing the same code three times in a row, one time for each comparison:

```
# SIGNIFICANT DE GENES:
# Takes as arguments:
# my.res = a dataframe of results from the DEseq results()function
# my.LFC = log fold change cutoff, default is 1.0
# my.cof = adjusted p-value cutoff, default is 0.01

SigDE <- function(my.res, my.LFC=1.0, my.cof=0.01) {
  my.res <- as.data.frame(my.res) %>%
    rownames_to_column(., var = "GeneName") %>%
```

```r
    as_tibble() %>%
    left_join(., GeneNames) %>%
    mutate(dir = ifelse(log2FoldChange >= 0, 'up', 'down')) %>%
    filter((log2FoldChange >= my.LFC | log2FoldChange <= -my.LFC) & padj <= my.cof) %>%
    arrange(padj, desc(abs(log2FoldChange)))
  return(my.res)
}
```

Filter DEA results from comparison of lactating vs control mice using custom function:

```r
resLC <- SigDE(resLC)

# Number of DE genes:
dim(resLC)
```

```
## [1] 2126    9
```

```r
# Give it a look
head(resLC, n=5)
```

```
## # A tibble: 5 x 9
##   GeneName baseMean log2FoldChange lfcSE  stat    pvalue      padj EntrezGeneID
##   <chr>       <dbl>          <dbl> <dbl> <dbl>     <dbl>     <dbl> <chr>
## 1 Wap       387497.           9.76 0.424  23.0 3.46e-117 5.98e-113 22373
## 2 Csn1s2a  1056557.           8.24 0.414  19.9 6.35e- 88 5.49e- 84 12993
## 3 Csn1s2b    26616.          10.4  0.580  18.0 2.69e- 72 1.55e- 68 12992
## 4 Glycam1   520318.           9.62 0.539  17.9 2.52e- 71 1.09e- 67 14663
## 5 Pigr       11356.           6.40 0.373  17.2 4.17e- 66 1.44e- 62 18703
## # ... with 1 more variable: dir <chr>
```

Below we perform the same steps as above to get the DE genes between (II) pregnant and control mice and (III) lactating and pregnant mice:

(II) pregnant and control mice:

```r
resPC <- results(exprObj, contrast = c("Status", "pregnant", "control"),  independentFiltering = FALSE)
#DESeq2::plotMA(resPC)
#summary(resPC)
resPC <- SigDE(resPC)

# Number of DE genes:
dim(resPC)
```

```
## [1] 1206    9
```

(III) lactating and pregnant mice:

```r
resLP <- results(exprObj, contrast = c("Status", "lactate", "pregnant"),  independentFiltering = FALSE)
#DESeq2::plotMA(resLP)
#summary(resLP)
resLP <- SigDE(resLP)

# Number of DE genes:
dim(resLP)
```

```
## [1] 790   9
```

Heatmap Visualization

To visually inspect if DE genes identified in our DESeq2 analysis successfully separate the three groups of mice (control, pregnant and lactating), we will make a heatmap. For this we use the `heatmap` function and package `viridis`.

It will not make sense to include all DE genes in this heatmap (3000 genes). Instead pick the top 50 most significant DE genes, based on adj. p-value and logFC.

Make a vector of unique EntrezGeneIDs (top50):

```
topDE <- bind_rows(resPC[1:50,], resLC[1:50,], resLP[1:50,]) %>%
  pull(GeneName) %>%
  unique()

length(topDE)
```

```
## [1] 124
```

The expression counts themselves (not logFC) are needed for the heatmap. We use the topDE vector to extract these from the vst normalized DESeq2 object.

```
head(assay(exprObjvst), n=5)
```

```
##         MCL1.DG  MCL1.DH  MCL1.DI  MCL1.DJ  MCL1.DK  MCL1.DL  MCL1.LA  MCL1.LB
## Xkr4   8.536754 8.157576 6.401025 7.987204 8.608154 8.350271 4.088346 4.088346
## Rp1    4.395113 4.408625 4.088346 4.088346 4.088346 4.088346 5.049024 4.605457
## Sox17  6.823999 7.536586 6.631003 7.018342 6.198459 6.849080 5.290970 5.528569
## Mrpl15 8.084835 7.843863 8.288204 8.288468 8.346705 8.271139 8.881522 8.550341
## Lypla1 9.068321 8.867045 9.679391 9.821810 9.921005 9.717343 9.409346 9.384315
##         MCL1.LC  MCL1.LD  MCL1.LE   MCL1.LF
## Xkr4   4.088346 4.088346 4.088346  4.088346
## Rp1    5.155437 4.603001 4.088346  4.088346
## Sox17  5.496313 5.102135 4.880034  5.487699
## Mrpl15 8.993366 8.614985 9.157331  9.288091
## Lypla1 9.418084 9.531917 9.945641 10.020675
```

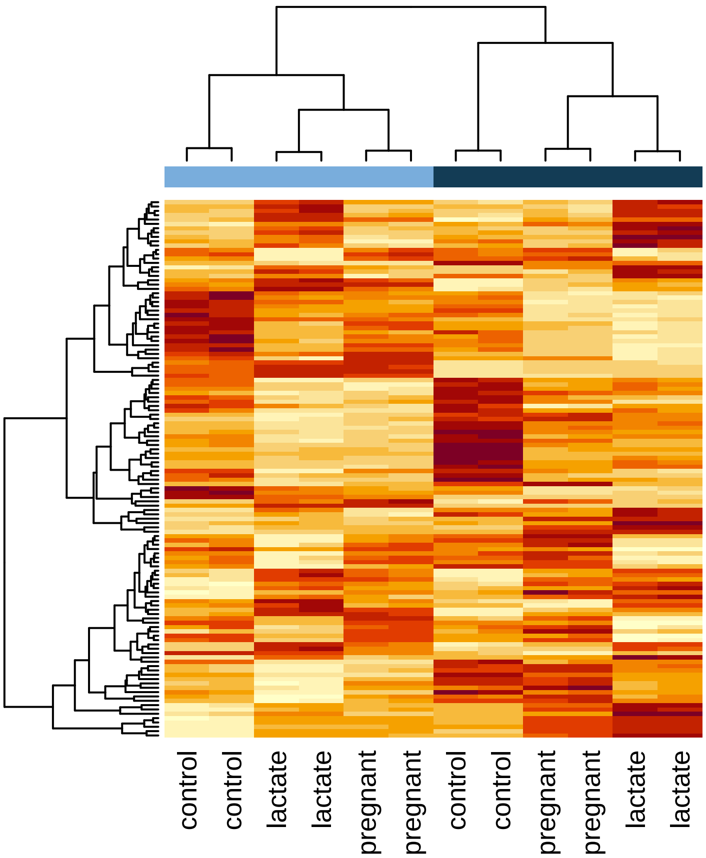
```
resVST <- assay(exprObjvst) %>%
  as.data.frame() %>%
  rownames_to_column(var = "GeneName")  %>%
  as_tibble() %>%
  filter(GeneName %in% topDE)
```

The heatmap function in base R wants gene expression data as a matrix (a dataframe with numeric values only). We extract the GeneNames column and convert the tibble into a matrix:

```
HPnames <- resVST %>%
  pull(GeneName)


HPdat <- resVST %>%
  dplyr::select(-GeneName) %>%
  as.matrix()
```

We use the `heatmap` function to generate a heatmap. We can modify the look of the heatmap as desired, e.g. add column colors, row labels, change color scheme etc.

```
heatmap(HPdat, labRow=FALSE,
        ColSideColors=exprInfo$CellType.colors,
        labCol=exprInfo$Status,
        cexCol=1.2,
        cexRow = 1.3)
```



GO & Pathway enrichment analysis with KEGG

For gene ontology enrichment and pathway enrichment analysis we will use the R-package `clusterProfiler` http://yulab-smu.top/clusterProfiler-book/index.html and the `KEGG` ontologies and pathways.

First, we check that we have pathway information for our species of interest, in this case mouse:

```
search_kegg_organism('mmu', by='kegg_code')
```

```
##    kegg_code scientific_name common_name
## 14       mmu    Mus musculus       mouse
```

Pathway enrichment analysis, using all genes from our expression dataset as the gene universe (e.g. background for enrichment):

```r
resLCpw <- enrichKEGG(gene = resLC$EntrezGeneID,
                      organism = 'mmu',
                      universe=unique(GeneNames$EntrezGeneID))
head(resLCpw, n=3)
```

```
##                ID             Description GeneRatio  BgRatio       pvalue
## mmu00100 mmu00100    Steroid biosynthesis    13/950  20/8733 1.032015e-08
## mmu01212 mmu01212   Fatty acid metabolism    22/950  61/8733 1.734092e-07
## mmu04514 mmu04514 Cell adhesion molecules    40/950 163/8733 4.805815e-07
##              p.adjust       qvalue
## mmu00100 3.292127e-06 2.781008e-06
## mmu01212 2.765876e-05 2.336460e-05
## mmu04514 5.110184e-05 4.316802e-05
##
## mmu00100
## mmu01212
## mmu04514 14663/12739/20344/20339/66797/20969/12740/18613/54419/12737/17123/94332/54420/12550/69524/2
##          Count
## mmu00100    13
## mmu01212    22
## mmu04514    40
```

Gene ontology enrichment analysis, using all genes from our expression dataset as the gene universe (e.g. background for enrichment). Here we enrich for *molecular function* (MF), but there are also other options for type of ontology. Other parameters set are: Benjamini holbech correction for multiple testing and cutoffs for significance.

```r
resLCgo <- enrichGO(gene = resLC$EntrezGeneID,
                    OrgDb = org.Mm.eg.db,
                    ont = "MF",
                    pAdjustMethod = "BH",
                    pvalueCutoff  = 0.01,
                    qvalueCutoff  = 0.05,
                    readable = TRUE,
                    universe = unique(GeneNames$EntrezGeneID))

head(resLCgo, n=3)
```

Below we perform the same steps as above to get the DE genes between (I) pregnant and control mice and (II) lactating and pregnant mice:

(I) pregnant and control mice:

```r
resPCpw <- enrichKEGG(gene = resPC$EntrezGeneID,
                      organism = 'mmu',
                      universe=unique(GeneNames$EntrezGeneID))
head(resPCpw)
```

```r
resPCgo <- enrichGO(gene = resPC$EntrezGeneID,
                    OrgDb = org.Mm.eg.db,
                    ont = "MF",
                    pAdjustMethod = "BH",
                    pvalueCutoff  = 0.01,
                    qvalueCutoff  = 0.05,
                    readable = TRUE,
                    universe=unique(GeneNames$EntrezGeneID))
```

```
head(resPCgo)
```

(II) lactating and pregnant mice:

```
resLPpw <- enrichKEGG(gene = resLP$EntrezGeneID,
                      organism = 'mmu',
                      universe=unique(GeneNames$EntrezGeneID))
head(resLPpw)
```

```
resLPgo <- enrichGO(gene = resLP$EntrezGeneID,
                    OrgDb = org.Mm.eg.db,
                    ont = "MF",
                    pAdjustMethod = "BH",
                    pvalueCutoff  = 0.01,
                    qvalueCutoff  = 0.05,
                    readable = TRUE,
                    universe=unique(GeneNames$EntrezGeneID))

head(resPCgo)
```