VI. Bioinformatics in R (exercises)

Center for Health Data Science, University of Copenhagen

16 March, 2022

In this exercise you will perform differential expression analysis using the `DESeq2` package for R on an RNAseq expression dataset from airway smooth muscle cells. You will effectively be performing the same work flow of analysis as the one provided in **presentation 6**, with some variation along the way.

About the Dataset
Common medicine for treatment of astma are beta2-agonists and glucocorticosteroids, which mainly target the airway smooth muscle.
The dataset `airway` contains mRNA profiles from bulk RNAseq of smooth muscle cells from four male donors. The cell lines were treated with dexamethasone and albuterol, or were left untreated (controls). See original data and summary of intent here.

PART 1 - Preliminary Analysis

For this exercise you will need five R-packages: `tidyverse`, `ggplot2`, `readxl`, `BiocManager`, `DESeq2` and `EDASeq`. You should already have these installed!
Copy the chunk below to your .R or .Rmd file to load packages you need.
N.B If you do get an error when trying to load one of the packages below, you can try to install the missing package using `install.packages('my.package')`:

```
library(tidyverse)
library(ggplot2)
library(readxl)
library(BiocManager)
library(DESeq2)
library(EDASeq)
```

---

1. In the exercises folder you will find two files: `airway_counts.xlsx` (RNAseq count data) and `airway_metadata.xlsx` (sample information). Read in these two files and name them ***airDat*** and ***airMet***, respectively.
2. What kind of information do you have in the first four columns of the airDat? How many samples and genes are there in your count data?
3. In your **airMet** tibble you have four variables, all characters, convert *condition* and *celltype* into factors for further analysis.

Copy and run the code below. This line of code will give you a new column in your **airDat** tibble named *nzeros* (sum of 0 across samples for each gene).
First we temporarily de-select the columns with gene information, then we count 0s across samples.

```
airDat <- airDat %>%
  mutate(nzeros = rowSums(dplyr::select(., -Ensgene, -GeneSymbol, -GC, -Length)==0))
```

4. We do not want any genes where less than half of the samples have a count above 0. Filter out the genes for which this is the case, and remove the column *nzeros* from your tibble after filtering. Check how many genes you are left with. *N.B* Name the filtered dataset *airDatFilt*.

5. To get an idea of sample library sizes make a boxplot of gene counts, one for each sample. You will need to `gather()` the gene counts across samples into one column and the sample IDs into another column. Assign this output to a new variable named `airPlot`.
   HINT 1: First remove the columns with information on genes (there are four) and next use this: `gather(key = ID, value=geneCount)` to gather the counts into one column.
   HINT 2: Use `geom_boxplot()` to make a boxplot with ggplot2. Extra: To tilt your x-axis labels 90 degrees, add `theme(axis.text.x = element_text(angle = 90))` to your ggplot code.

6. You will see that boxplots are squeezed due to difference in count range between genes. Which type of data transformation could you use to overcome this problem?
   Remake the boxplot with transformed counts.
   HINT remember to add 1.0 pseudo count to all counts before transformation to handle counts which are 0.

Adjusting for GC content and gene length: It is well-established that gene length and GC-content will affect the number of reads obtained. It is therefore standard practice to check for such biases and correct accordingly. You already have information on length and GC content in the **airDatFilt** tibble.

The `EDAseq` package can help you correct for gene length and GC content. `EDAseq` provides a `biasPlot` showing you if GC content and gene length are a problem. To use this function your data must be in a format that `biasPlot` accepts, e.g. an `EDAseq` object.

7. Copy the code from both chunks below and run it. Have a look at the object you created, named *EDAobj*, and importantly, look at the plots created, does there appear to be a GC and/or gene length bias in your data? **HINT 1:** An unbias dataset should not show any trend in GC content or gene length across gene counts.

```
# Converts tibble to matrix of expression counts and removes gene information:
airDatM <- airDatFilt %>%
  dplyr::select(-Ensgene, -GeneSymbol, -GC, -Length) %>%
  as.matrix() %>%
  unname()

# Make EDASeq object
EDAobj <- newSeqExpressionSet(airDatM,
                              phenoData=AnnotatedDataFrame(data.frame(condition=airMet$condition,
                                                                      celltype=airMet$celltype)),
                              featureData=AnnotatedDataFrame(data.frame(GC=airDatFilt$GC,
                                                                        Length=airDatFilt$Length)))
```

```
# Bias plots
par(mfrow=c(1,2))
```

```
biasPlot(EDAobj, "GC", log=TRUE, ylim=c(0,10))
biasPlot(EDAobj, "Length", log=TRUE, ylim=c(0,10))
```

If you observe a bias in your data, you can correct for this with the function `withinLaneNormalization` from `EDASeq`. You can have a look at the EDASeq documentation under point 5 for more info on this function.

8. Copy and run the code in below to normalize your *EDAobj* object for GC content and gene length. Remake the `biasPlot` from above with the *EDAobjNorm* object. Are the biases removed?

```
EDAobjNorm <- withinLaneNormalization(EDAobj, "GC", which="full", offset=TRUE) %>%
  withinLaneNormalization(., "Length", which="full", offset=TRUE)
```

PART 2 - Differential Expression Analysis with DESeq2

Now we we begin our differential expression analysis in `DESeq2`. First, we use the function `DESeqDataSetFromMatrix` to make a `DESeq2 object` (just as shown in the presentation).

9. Fill in the DESeq2 object below. We want our design matrix to include *celltype* and *condition* from the metadata, (**airMet**). Fill in the *countData*, *colData* and *design* (rowData is optional).
   HINT Your countData must only contain counts, no gene IDs etc., so you should filter/slice these these columns out before adding the counts to the object.

```
exprObj <- DESeqDataSetFromMatrix(countData = ,
                                  rowData = ,
                                  colData = ,
                                  design=~)
```

10. If we are to be correct, you should "feed" the function `DESeqDataSetFromMatrix` the normalized counts from the `EDASeq` analysis, where you corrected for GC content and gene Length. This is tricky, so we have provided you with the code needed for this. Copy and run the code below.

```
exprObj <- DESeqDataSetFromMatrix(countData = counts(EDAobjNorm),
                                  colData = pData(EDAobjNorm),
                                  design = ~celltype+condition)

normFactors <- exp(-1 * offst(EDAobjNorm))
normFactors <- normFactors / exp(rowMeans(log(normFactors)))
normalizationFactors(exprObj) <- normFactors
```

In question 5., we made a ggplot2 boxplots to see the difference in library sizes and variances of samples. Based on this plot it seems like variance stabilizing transformation might improve our dataset for DEA analysis. Copy and run the code below to perform vst transformation.
Extra (optional): Make a boxplot with the vst counts.

```
exprObjvst <- vst(exprObj,blind=FALSE)
```

We would like to use a PCA plot to inspect if the vst transformation has improved the clustering of our samples by group. For this we will first perform principal component analysis using both the 'raw' counts and vst-transformed counts. We extract counts from the DESeq2 objects using the function `assay()`.

11. Copy and run the two chunks of code below.
    In the second chunk we are using three different functions: `t()`, `dist()` and `cmdscale()`. Use `?` to figure out what each of them do! What inputs do they take? and what are the default values?

```
# un-transformed counts
unTrf <- assay(exprObj) %>%
  t() %>%
  dist() %>%
  cmdscale(., eig=TRUE, k=2)

unTrf <- tibble(PC1=unTrf$points[,1],PC2=unTrf$points[,2])


# vst transformed counts

vstTrf <- assay(exprObjvst) %>%
  t() %>%
  dist() %>%
  cmdscale(., eig=TRUE, k=2)

vstTrf <- tibble(PC1=vstTrf$points[,1],PC2=vstTrf$points[,2])
```

You now have two datasets containing the first two principal components (PC1 & PC2) for both 'raw' (un-transformed) counts, *unTrf*, and vst-transformed counts, *vstTrf*.

12. Make a PCA plot for each set using ggplot2. Color the samples by condition and label them by *celltype* (**airMet** has this information).
    HINT A PCA plot is just a `geom_point()` plot with x=PC1 and y=PC2. Would you say that the transformation improved the partitioning of control and treated samples?

Next, we use `DEseq()` to estimate dispersion, gene-wise and mean-dispersion, fitting model(s). Copy the code below and run it.

```
# Fitting gene-wise glm models:
exprObj <- DESeq(exprObj)
```

We now have our model(s) ready and we want to contrast our condition groups, e.g. treated vs control.

13. Use the `DESeq2` function `results()` to do the post hoc test (just like we did in the presentation). Figure out what arguments it takes. As a minimum you will have to specify a DESeq2 model object (denoted by '.' below) and a contrast of interest. When you have run the function, have a look at the output.

```
resTC <- results(. , contrast = c(), independentFiltering = FALSE)
```

14. Use the function `summary()` to see the number of identified differentially expressed gene. Use the `plotMA()` function to visualize these.

15. Convert your results from `DESeq2` to a tibble and do the following:

    (a) Add two new columns to it (using `mutate`): *dir* indicating directionality of the logFC. The column *dir* can be made using the following syntax: `dir=factor(ifelse(log2FoldChange >= 1.0, "Up", "Down"), levels=c("Up", "Down"))` and one named *geneSymbols* with gene symbols from *airDatFilt*.

(b) Filter your results to only include genes with log2FoldChange of more than 1 or less than -1 and a *padj* (adjusted p-value) of less than 0.05.
(c) Arrange by *padj* (ascending) and the **absolute** *log2FoldChange* (descending). **HINT:** use the function `abs()` to get the absolute log2FoldChange before arranging.
(d) Extract the top 50 most significant DE genes based on *log2FoldChange* and *padj*.

16. Make a lollipop plot (https://www.r-graph-gallery.com/lollipop-plot.html) of the top 50 most significant DE genes:
    (a) You will need two ggplot geoms for this: `geom_segment()` and `geom_point()`.
    (b) The size of the point should reflect the log2FoldChange and the shade of the point should reflect the significance (e.g. the padj). *N.B* you may have to scale one or both of these to get a nicer plot.
    (c) Change the color scheme to one you would like, try `scale_colour_gradient2()`.
    (d) Play around with the look/theme of the plot, remove legends you dont need, grid lines etc.
    (e) Based on your plot, which genes seem to be most effected by treatment with dexamethasone and albuterol?