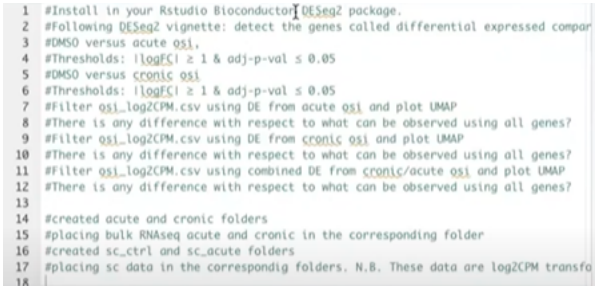
9 -Correction exercise lesson 7 and Omics.net

TOPICS	
DATE	@May 11, 2022 2:00 PM
LAST EDITED BY	
LAST EDITED TIME	@May 15, 2022 5:33 PM
MADE BY	
PROF	Calogero
Recording	
STATUS	☑

In this lesson we will do the exercise of lesson 7 step by step together.

When you have to do this kind of complex analysis, you have first to understand what you want and write conceptually the steps you need to obtain it. You have also to organize all the data you have in a clever manner (usually you have to make folders, one for the acute and one for the chronic part, but this is up to you). This is an example of what Calogero did.



```
#Install in your Rstudio Bioconductor DESeq2 package.
#Following DESeq2 vignette: detect the genes called differential expressed compar
#DMSO versus acute osi,
#Thresholds: |logFC| ≥ 1 & adj-p-val ≤ 0.05
#DMSO versus cronic osi
#Thresholds: |logFC| ≥ 1 & adj-p-val ≤ 0.05
#Filter osi_log2CPM.csv using DE from acute osi and plot UMAP
#There is any difference with respect to what can be observed using all genes?
#Filter osi_log2CPM.csv using DE from cronic osi and plot UMAP
#There is any difference with respect to what can be observed using all genes?
#Filter osi_log2CPM.csv using combined DE from cronic/acute osi and plot UMAP
#There is any difference with respect to what can be observed using all genes?

#created acute and cronic folders
#placing bulk RNAseq acute and cronic in the corresponding folder
#created sc_ctrl and sc_acute folders
#placing sc data in the correspondig folders. N.B. These data are log2CPM transfo
```

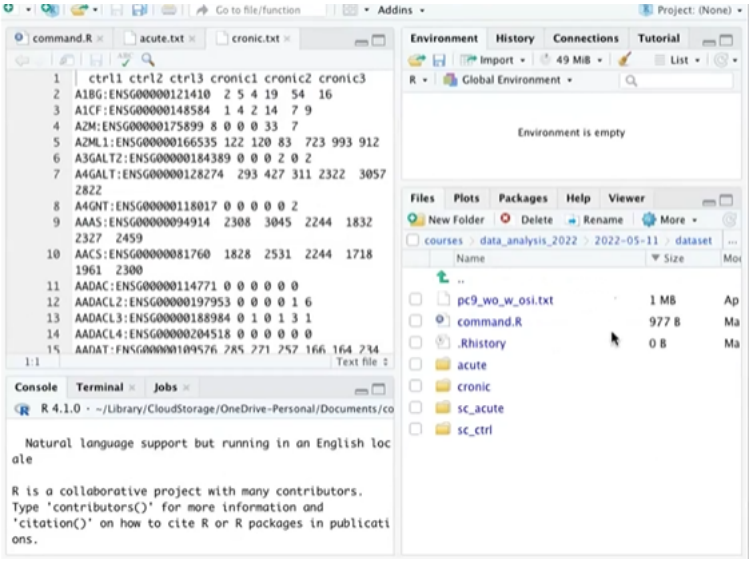
We'd like to find differentially expressed (DE) genes comparing DMSO vs acute and chronic osimertinib treatment on RNAseq data. We are looking for adjusted p-value (Benjamini-Hochberg corrected p-value) ≤ 0.05 (5% of the DE genes there by chance) and |log2FC| ≥ 1 (this threshold has to do with the sensitivity and the numerosity of our sample. Sensitivity is related to the validation of the DE genes we found: after DE analysis you have to validate the genes you found with qPCR for example, or other independent techniques. Since, the number of replicates is little and the variability is high, we have to set the logFC of our analysis accordingly. If this threshold were lower, we wouldn't be able to validate many of the DE genes we have selected with qPCR. Still this threshold will not allow validation of 100% of the DE genes found with the computational analysis, but this threshold is reasonable).

> ❗ There is no biological reason behind the |log2FC| ≥ 1, since in biology also small transcriptional changes may drive significant consequences.

After having found the DE genes, we will filter the single cell dataset and look if using only the DE genes, the plot of the acute single cell data is able to separate better the data.

Since we are expecting to revise the code after some time, it is important to keep everything together (the genome on which you've done the mapping, the fastQ files and all the downstream analysis you've done) and comment your code.

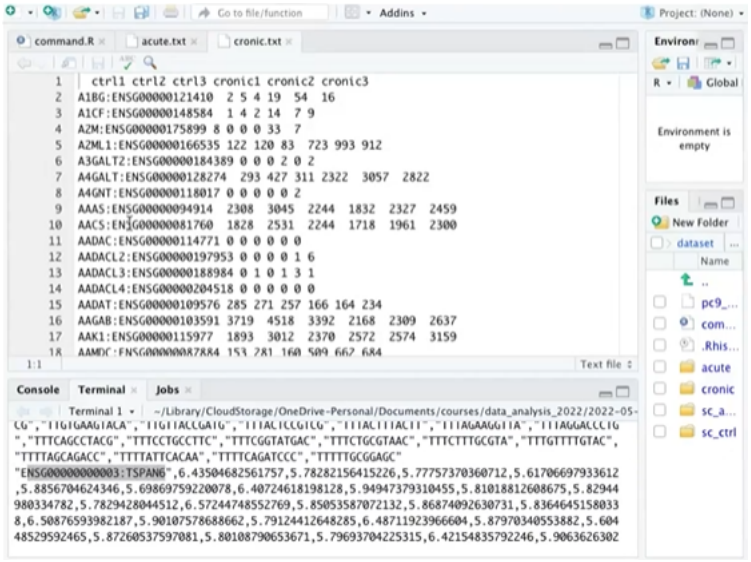
The professor has organized a folder for each experiment, with a txt file for each bulk and a csv for the single cell experiments with DMSO or acute osimertinib.



You can visualize the txt simply by clicking on them, while for the csv, you can do

```
head -n 1 "nameoffile" #to see only the first line of the dataset
head -n 2 "nameoffile" #to see only the first and second line
```

The first criticality to solve is the fact that the bulk experiment have the rownames with the symbol and the ensemble ID switched:



The first thing to do is to reorganize the bulk data so that it will look as the single cell data: you could also do the opposite, but working with large datasets is always long and annoying, so you rearrange the smallest one (annotation problem).

1. Load the files

```
acute <- read.table("./folder/acute.txt", header = T, sep="\t", row.names=1)
# The path is to load the files that are not in our wd
chronic <- read.table("./folder/chronic.txt", header = T, sep ="\t", row.names=1)
```

2. To do the step of changing the rownames by inverting the ensemble ID and the symbol we can do as we did to use `strsplit(data_to_separate, character_of_separation)` and create a list of the rownames splitted by the : , transform it into a matrix with `matrix` and then invert the symbol and the ensemble ID with `paste`. Actually, there is a also a more sophisticated way to do this **by creating a function**:

```
rn <- rownames(acute)
# remember that row.names is to set the rownames, while rownames without dot is to retrieve them

# sapply applies the function we want on anything that is a vector and returns a vector
# lapply instead can be applied on everything and returns a list (not used in this case)
# in this case we have a list and sapply will pass through every element (by rows) on the list and do the function

tmp <- sapply(strsplit(rn, ":"), function(x)x[1])
# function(x) means that there is a parameter x which is passed to the function.
# here the parameter is "take the first element of the list and save on tmp
# by doing head we can see that there are only the symbols saved in tmp

# I actually want something more complicated: more complex parameters are specified in between curly brackets
tmp <- sapply(strsplit(rn, ":"), function(x) {
  tmp <- paste(x[2], x[1], sep = ":")
  # not a problem to have the same names: the elements in a function will survive only until the function is on
  # the global variable and the one inside the function are completely independent
  return(tmp) # not necessary since tmp is the last variable used, but good practice to put it
}

# with paste function we are assigning tmp for the value x the correct order of ensemble ID (x[2]) and the symbol (x[1]).
```

Now with `head(tmp)` you can see that we have a vector with the correct order of ensembleID and symbol of the gene.

Actually we can do this even better: we can create a swap function to do this in a better way.

```
swapping  <- function(my.rownames){
rn1 <- sapply(strsplit(my.rownames, ":"), function(x){
  tmp <- paste(x[2], x[1], sep = ":")
  return(tmp)
  }
return(rn1)
}
# we putting all the stuff we did before inside another function
# in this way we are creating something that then requires just the input file and then gives you the output

# after having created your function you can pass your data to it
tmp <- swapping(my.rownames=rn) # tmp <- swapping(rn) would have worked just fine since the function has only one parameter
```

The next step is that we want to give the dataframe as input and receive the swapped dataframe as output (without doing anythig else):

```
swapping  <- function(my.df){
  my.rownames <- rownames(my.df) # extract the rownames and put inthe same variable we used before so that we do not have to change anythin
  rn1 <- sapply(strsplit(my.rownames, ":"), function(x){
    tmp <- paste(x[2], x[1], sep = ":")
    return(tmp)
  }
rownames(my.df) <- rn1 # reassign the new rownames contained in rn1 to the original dataframe (which is this function is called my.df)
return(my.df)
}

# after loading the function you can run it directly on your dataframes
acute.swapped <- swapping(acute)
chronic.swapped <- swapping(chronic)
```

Repeat everything for chronic: this is very simple now that we have a function and we are not doing everything by hand step by step.

3. Now we want to do the diff expr analysis

```
#load the library
library(DESeq2)

# first put acute.swapped and chronic.swapped in a matrix (before they were a vector)
```

```
acute.swapped <- as.matrix(acute.swapped)
chronic.swapped <- as.matrix(chronic.swapped)

# then create the coldata for acute
# we have to take the covariate elements = elements inside the colnames.
dimnames(acute.swapped)[2]
# this will print a list of the colnames of the acute swapped matrix
dimnames(acute.swapped)[[2]]
# with double brackets we are writing inside a list

# we want to create a dataframe with samples and condition where
# samples = dimnames
# conditions = ctrl ctrl ctrl treat treat treat (this is always the order of the replicates)
coldata <- data.frame(samples=dimnames(acute.swapped), condition=factor(c(rep("ctrl",3), rep("treat",3))))
# here we are using dimnames because of the structure of a matrix

rownames(coldata) <- dimnames(acute.swapped)[[2]]
# this step is needed to reassign also the correct rownames to the coldata dataframe

# now we have to create the dds object to run the DE analyis
dds <- DESeqDatasetFromMatrix(countData = acute.swapped, colData = coldata, design = ~ condition)
dds <- DESeq(dds)
res <- results(dds)

dim(res) #18774     6
```

> ❗ When doing this kind of analysis we are always trying to stick to the same organization: the ctrl is always the first covariate while the treatment is the second. In this way the first variable will be always the reference, and the results will always have the DE genes in the treated samples compared to the control.
>
> In this way we know how to read the results without referring back to the code.

All this can go in a function (but we have first to change the `acute.swapped` variable in `swapped` only so that it becomes the parameter of a function.

```
#this is a fucntion that giving the input data frame perfom the dE analysis
mydeseq <- function(my.df){
  swapped <- my.df
  swapped <- as.matrix(acute.swapped)
  coldata <- data.frame(samples=dimnames(swapped), condition=factor(c(rep("ctrl",3), rep("treat",3))))
    rownames(coldata) <- dimnames(acute.swapped)[[2]]
  dds <- DESeqDatasetFromMatrix(countData = swapped, colData = coldata, design = ~ condition)

dds <- DESeq(dds)
  res <- results(dds)
  return(res)
}

acute.res <- mydeseq(acute.swapped)
chronic.res <- mydeseq(chronic.swapped)
```

In the results (seen with `head`), we have adjusted mean counts, then log2FC, then the intermediate parameters (std error of the model, statistics for deriving the pvalue and the pvalue itself). We are interested in the adj p-value and log2FC.

4. We now have to filter: all goes inside the function

```
#this is an unique function. So, only giving the dataframe input file give as output the DESeq analysis after the filtering. .
mydeseq <- function(my.df){
  swapped <- my.df
  swapped <- as.matrix(swapped)
  coldata <- data.frame(samples=dimnames(acute.swapped), condition=factor(c(rep("ctrl",3), rep("treat",3))))
    rownames(coldata) <- dimnames(acute.swapped)[[2]]
  dds <- DESeqDatasetFromMatrix(countData = acute.swapped, colData = coldata, design = condition)
#this is a unique function that generate the DESeq result and already filter them.
dds <- DESeq(dds)
  res <- results(dds)
  res <- res$log2FoldChange
  # is NA res%log2FoldChange? If not, I take it otherwise I discard it
  # the ! is the negative (!= == different)
```

```
  # needed to avoid genes with 0 expression
  res <- res[intersect(which(res$padj <= 0.05,
                       which(abs(res$log2FoldChange) >= 1)),]
  return(res)
}

acute.res <- mydeseq(acute.swapped)
chronic.res <- mydeseq(chronic.swapped)
```

```
#alternative way only changing the filtering code
mydeseq <- function(my.df){
  swapped <- my.df
  swapped <- as.matrix(swapped)
  coldata <- data.frame(samples=dimnames(acute.swapped), condition=factor(c(rep("ctrl",3), rep("treat",3))))
    rownames(coldata) <- dimnames(acute.swapped)[[2]]
  dds <- DESeqDatasetFromMatrix(countData = acute.swapped, colData = coldata, design = condition)

dds <- DESeq(dds)
  res <- results(dds)
  res <- res[intersect(which(res$padj <= 0.05) & (abs(res$log2FoldChange) >= 1)),]
  return(res)
}

acute.res <- mydeseq(acute.swapped)
chronic.res <- mydeseq(chronic.swapped)
```

The filtering output should have 4027 genes in the acute and 4276 in the chronic.res.

5. Now we have to intersect the data with the sc data from lesson 6

```
sc <- read.csv("L6osi_log2CPM.csv", header=T, row.names=1)
# same as readtable but assumes that the separator is a comma and you do not have to specify it
# be careful: CHANGE THE NAME OF THE FILE AND PUT THE ONE IN YOUR COMPUTER


# perfom the filtering of the rownames on the basis of the acute data
sc.acute <- sc[which(rownames(sc) %in% rownames(acute.res)),]
sc.chronic <- sc[which(rownames(sc) %in% rownames(chronic.res)),]
dim(sc.acute) #3524
dim(sc.chronic) #3636
```

6. Now do UMAP

```
#perfom UMAP on the outdata
umap.out <- umap(t(sc.acute), random_state=111, n_epochs = 1000, min_dist=0.05, n_neighbors=15)
f=data.frame(x=as.numeric(umap.out$layout[,1]),y=as.numeric(umap.out$
                                                            layout[,2]))
# do the transposed because the rownames should be a row

# then plot with ggolot
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.5) # we used 0.3 but 0.5 better
pdf("sc_acute_UMAP.pdf")
print(sp_acute)
dev.off()

#add for chronic
```

We can do additional things: take the genes chronic + acute or only chronic or only acute to try to rule out if the cells we see as outliers are really outliers or just noise.

> ❗ FOR THE NEXT TIME
>
> Use the intersection or the subset only chr / only acute to plot UMAP → does the separation get better or worse?

We still need to have the acute swapped and chr swapped to be saved as files for next analysis. To save this stuff, we can save all the variables using the save command.

```
rownames(acute) #now it contains symbol-ensembleID
```

Write for each row the symbol of the differentially expressed genes for acute and chronic.

save in file .txt

Omics.net

Datamining tool to identify potential upstream regulators to the list of DE genes you have generated, to identify the TF that are inducing the changes in transcriptions of the genes.

Go to omics.net (https://www.omicsnet.ca/)

Load the list of the gene symbols and upload. Then proceed. After you can select among different type of analysis. We are interested in the TF-gene interaction, which is is the one we are interested in. There are then various datasets we can use: use TRRUST which is the most reliable one. There are various sub networks of TF. To visualize them, click proceed and then save the 2D visualization

In the node explorer there is the number of nodes associated to each gene = the genes associated to that TF, the genes regulated by that TF.

We can start to have a look at the TF with the higher connectivity. We see that SP1 has 144 connections: this means that we can select the TF that are associated with my DE genes of my experiment (i.e. the TF that are regulating the changes in transcription we induce with our treatment).

the node that you have can be investigated with different database:

We can also investigate the different nodes with different databases: for example with the reactome database we can see what are the paths in which our DE genes controlled by that TF are involved.

We can also color the TF on the basis of their function: here by watching the DE genes affected by SP1 we see that there are many cell cycle genes that are affected by our treatment for example, but this is not really informative since it is obvious that our treatment impedes cell replication. Indeed, SP1 is a general TF. We are offered the possibility to extract useful info for our analysis, but we should look deeper in the other TF trying to find something specific for this treatment.

❗ EXAM

- correct the code
- write new code (something we already did or we have to use the R vocabulary we have to do new things step by step)
- be careful of warnings → may be important