

Today we will try to do together the exercise we discussed last time.
The important is not the way you do it, the important is that you get the results.

When you start doing some analysis in R, the first point is understanding what you want. The second point is organizing all the things you have. Then, from there you can start. You could write on paper which are the steps to follow, and then start coding. You have to be very organized. In this case the organization will not be so complex.

```
1 #Install in your Rstudio Bioconductor DESeq2 package.
2 #Following DESeq2 vignette: detect the genes called differential expressed compar
3 #DMSO versus acute osi,
4 #Thresholds:  $|\log_2FC| \geq 1$  &  $\text{adj-p-val} \leq 0.05$ 
5 #DMSO versus cronic osi
6 #Thresholds:  $|\log_2FC| \geq 1$  &  $\text{adj-p-val} \leq 0.05$ 
7 #Filter osi_log2CPM.csv using DE from acute osi and plot UMAP
8 #There is any difference with respect to what can be observed using all genes?
9 #Filter osi_log2CPM.csv using DE from cronic osi and plot UMAP
10 #There is any difference with respect to what can be observed using all genes?
11 #Filter osi_log2CPM.csv using combined DE from cronic/acute osi and plot UMAP
12 #There is any difference with respect to what can be observed using all genes?
13
14 #created acute and cronic folders
15 #placing bulk RNAseq acute and cronic in the corresponding folder
16 #created sc_ctrl and sc_acute folders
17 #placing sc data in the correspondig folders. N.B. These data are log2CPM transfo
18
```

As you can see here, I have summarized the various steps we have to do in the exercise.

I am assuming that you managed to install DESeq2 package on your computer. You should also have umap. Now, the first point is that we would like to detect genes called differentially expressed comparing the DMSO (control) VS the acute treatment with Osimertinib or DMSO VS the chronic treatment with Osimertinib.

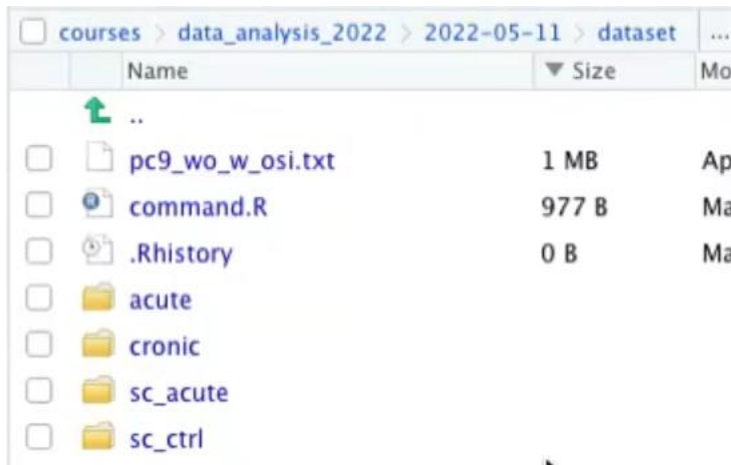
In both cases, we set an adjusted p-value below 0.05. This means that we are accepting the possibility that the 5% of the genes that are within my list of DEGs are there by chance.

Furthermore, we apply a filter on the absolute value of the log2FC, looking for values that are above 1 or below 0.1. Why this threshold? It is mainly related to the sensitivity and the numerosity of our samples. The sensitivity could refer to qPCR for example, because generally you try to validate the genes detected by this high throughput experiment with an independent technique, that could be qPCR. Since the number of replicates is little, the variability among samples is relatively high and qPCR will read what you have. If you start searching for genes having a variation on fold change that is smaller than 1, many of these genes cannot be validated because the noise is going to be so high to hide them. This threshold is reasonable unless you have a very large number of replicates. In that case it could be that your variation gets smaller, and you can decrease the threshold. There is no biological reason for the fact that log2FC must be equal or higher than 1. Many variations with a log2FC lower than 1 could still be significant. The point is that we have problems in validating these small variations, mainly due to the limits of the technology we are using. Once I have identified these elements, we would like to use only these genes that have been detected as differentially expressed to see if the map of the acute treatment with Osimertinib on single cell data will change the umap plot. In other words, we can see if using only these genes we get a better separation of the data.

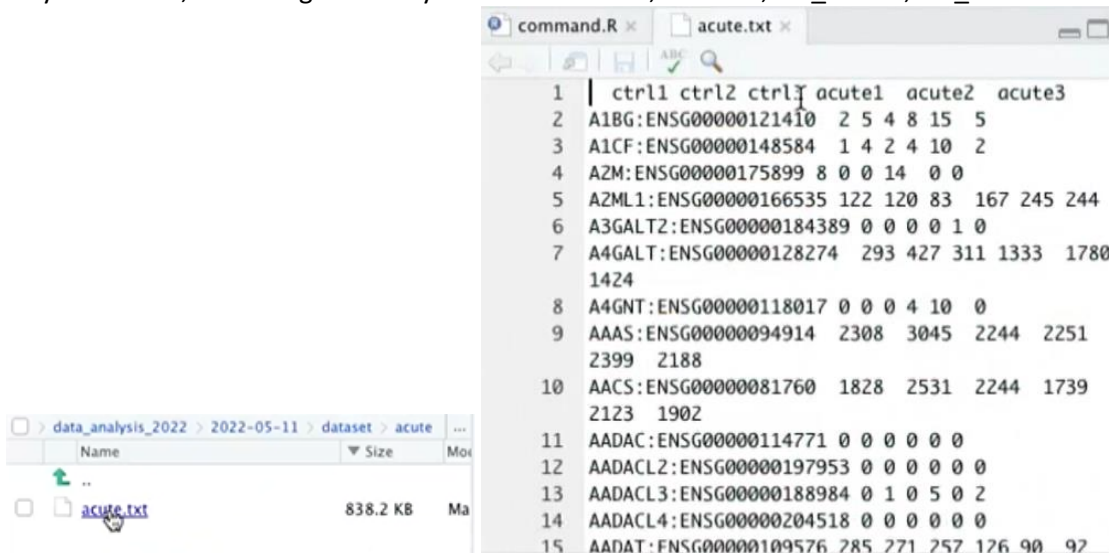
If we will have time, I'll show you also the use of a web page called omixnet (<https://www.omix-visualization.com/#sthash.azHBJ3Na.dpbs>), which is useful to give a biological meaning to the list of DEGs you have detected.

The first thing I did is reorganizing the data in a way that I have everything there. Since you are expecting to use these data for many years, it is important to keep those data altogether (the genome on which you

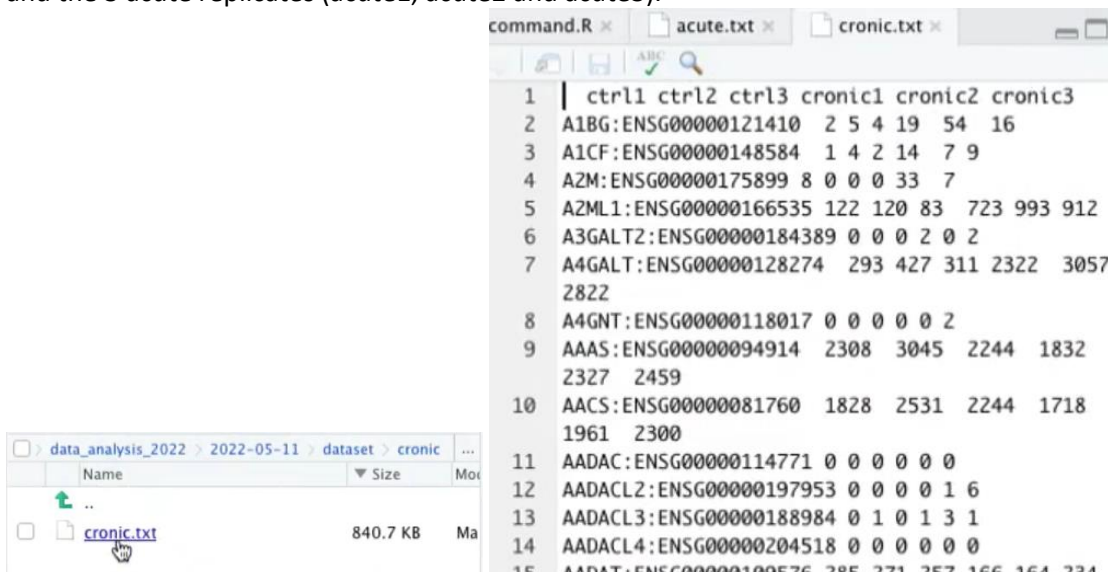
have done the mapping, the fastq files, the downstream results and so on all in the same hard drive). The ideal situation would be having a disk on which you do all the analysis, a backup disk containing all the fastq files, once your analysis is finished you can consider the system “frozen”, then you copy all your data on the backup disk and these 2 hard drives are stored somewhere so that you can recover their information in the future.



As you can see, I have organized my folders in “acute”, “cronic”, “sc_acute”, “sc_ctrl”.



In the acute folder I have made a file called acute.txt, which contains the 3 controls (ctrl1, ctrl2 and ctrl3) and the 3 acute replicates (acute1, acute2 and acute3).



In the cronic folder I have a cronic.txt file with the same organization.



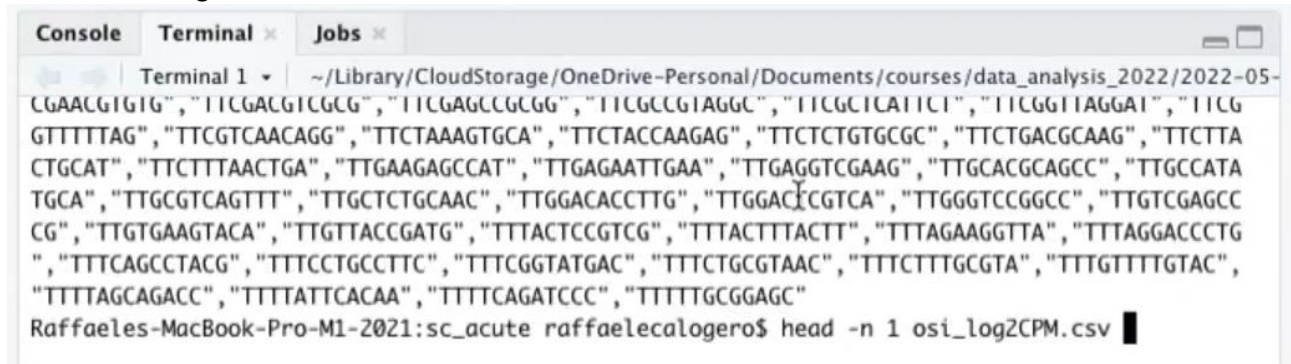
Then I have the sc_acute folder, that is the one we are interested in. here we have the file osi_log2CPM.csv.

If we type in the terminal:

```
$ cd sc_acute/
```

```
$ head -n1 osi_log2CPM.csv
```

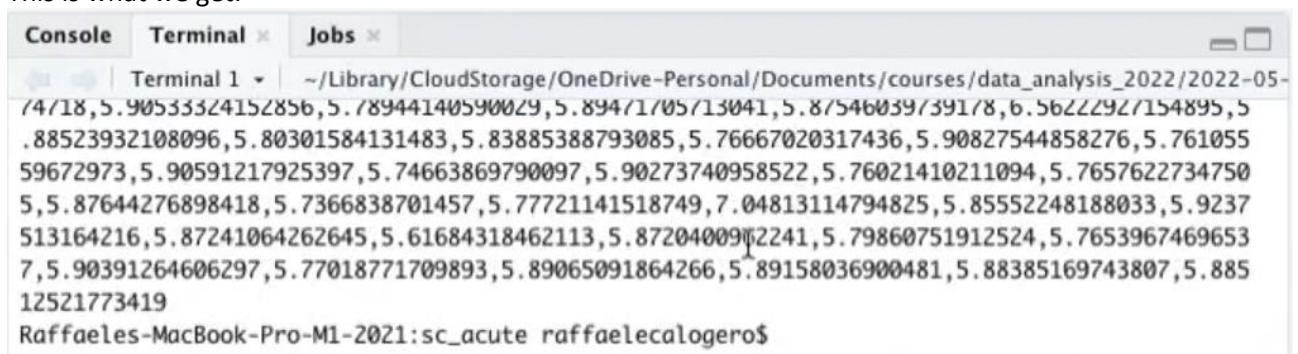
This is what we get:



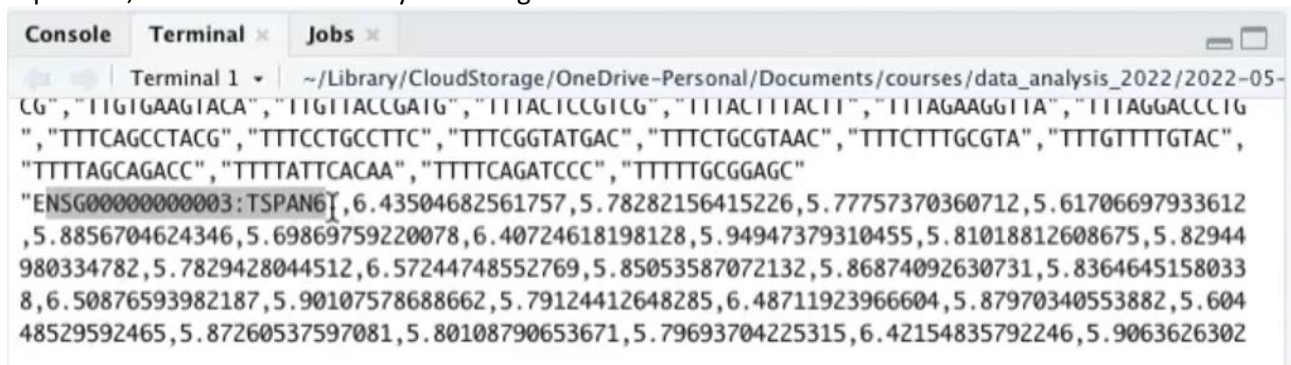
And if we tell the terminal to show also the second line by typing:

```
$ head -n2 osi_log2CPM.csv
```

This is what we get:



This is a comma-separated file. All the data coming from the differential expression analysis are tab-separated, but the main criticality is this organization here:



In the single cell I have the ENSEMBL ID semicolon and then the gene symbol. In bulk RNA-seq data it is the opposite (gene symbol : ensembl ID). So, we have to invert these identifiers.

So, this is the first thing I'm going to do. I'll reorganize the bulk RNA-seq data so that they are similar to the single cell ones. Why do I do this instead of the opposite operation? Because it is more convenient this way.



	Name	Size
↑	..	183.5 MB
□	osi_log2CPM.csv	183.5 MB

Indeed, as you can see, the single cell dataset is very big (183.5 MB).



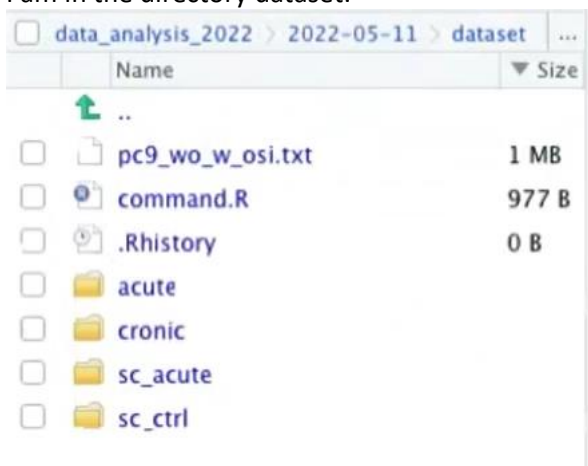
	Name	Size
↑	..	838.2 KB
□	acute.txt	838.2 KB

Instead, the bulk dataset is far smaller (838.2 KB), and so much easier to play with.

The first thing is checking where I am with the `getwd()` command.

```
> getwd()
[1] "/Users/raffaelecalogero/Library/CloudStorage/OneDrive-Personal/Documents/courses/data_analysis_2022/2022-05-11/dataset"
>
```

I am in the directory dataset.



	Name	Size
↑	..	1 MB
□	pc9_wo_w_osi.txt	977 B
□	command.R	0 B
□	.Rhistory	
□	acute	
□	chronic	
□	sc_acute	
□	sc_ctrl	

My data are in the subdirectories.

The first thing I'll do is uploading the `./acute/acute.txt` into R using the command `read.table()`.

"." means where I am (dataset).

"acute" is the subfolder containing my file.

"acute.txt" is my file.

```
> acute <- read.table("./acute/acute.txt", header=T, sep="\t", row.names=1)
> |
```

Now we want to modify the row names.

```
> rn <- rownames(acute)
> |
```

I create a variable called `rn` and I put there the row names.

`rownames()` is the command which extracts the row names from a data frame.

Instead, `names()` is the command which extracts the header from a data frame.

These commands only work on data frames (e. g. they do not work on matrices).

```
> head(rn)
[1] "A1BG:ENSG00000121410"
[2] "A1CF:ENSG00000148584"
[3] "A2M:ENSG00000175899"
[4] "A2ML1:ENSG00000166535"
[5] "A3GALT2:ENSG00000184389"
[6] "A4GALT:ENSG00000128274"
> |
```

To be sure that I have extracted what I wanted to, I check the content of the variable rn using the command head().

```
> tmp <- strsplit(rn, ":")
> |
```

Now I must separate the 2 elements of the row names using the command strsplit().

```
[[5]]
[1] "A3GALT2"      "ENSG00000184389"

[[6]]
[1] "A4GALT"      "ENSG00000128274"

> |
```

I check what I have done by typing:

```
> head(tmp)
```

We get a list in which every element is a vector containing 2 things (the 1st one is the gene symbol, the 2nd one is the ensembl ID).

```
> tmp1 <- sapply(strsplit(rn, ":"), function(x)x[1])
> |
```

apply() → this command can be used on data frames and matrices. You must give as parameters: the name of the data frame/matrix, 1 or 2 based on if you want to work on the rows or on the columns, the function you want to apply.

sapply() → this command can be used on vectors. It will return a vector.

lapply() → this command can be applied on whatever element you want. It will return a list.

We will use sapply() because we would like to have a vector as output.

Since I know that strsplit() works and provides a list as output, I can put in front of it the command sapply().

However, in order to swap my two identifiers I must create my own function and pass this function to sapply().

How to define a function?

```
function(x)x[1]
```

The parameter x will be the content of each element of our list.

The function we have defined will extract the first identifier from each element of my list.

Let's check the results with head().

```
> head(tmp1)
[1] "A1BG"      "A1CF"      "A2M"      "A2ML1"
[5] "A3GALT2"   "A4GALT"
> |
```

Since I want to use both identifiers, I can do a function which is a little more sophisticated.

```
tmp <- sapply(strsplit(rn, ":"), function(x){
  tmp <-
})
```

Remember that whatever happens in one function stays in the function. If I use the same name for a variable which is outside the function and another variable which is inside the function, we won't have any problem. The global variables and the local variables are independent.

```
rn1 <- sapply(strsplit(rn, ":"), function(x){
  tmp <- paste(x[2], x[1], sep=":")
  return(tmp)
})
```

However, I prefer to call the global one rn1 because it will contain the final row names I want.

paste() is a command useful to combine elements.

I assign to tmp the reorganized names, and then I return tmp. Since tmp is the last variable I have created, even if I do not add "return(tmp)", it will be returned anyways.

Se non inseriamo "return()" solo l'ultima variabile creata nella funzione sarà restituita.

Ok, now we run this function and check the results using the command head().

```
> head(rn1)
[1] "ENSG00000121410:A1BG"
[2] "ENSG00000148584:A1CF"
[3] "ENSG00000175899:A2M"
[4] "ENSG00000166535:A2ML1"
[5] "ENSG00000184389:A3GALT2"
[6] "ENSG00000128274:A4GALT"
> |
```

This is indeed what we wanted.

Now we'll create the swapping function.

```
swapping <- function(my.rownames){
}

tmp <- swapping(my.rownames=rn) OR tmp <- swapping(rn)
```

Since I only have one variable in the swapping function, I can also write the last line as follows. It doesn't matter. This is the way how I can pass the row names of my dataset to my function.

Now I can copy the generic function I had defined before in my swapping function and do some little adjustments.

```
swapping <- function(my.rownames){
  rn1 <- sapply(strsplit(my.rownames, ":"), function(x){
    tmp <- paste(x[2], x[1], sep=":")
    return(tmp)
  })
  return(rn1)
}

tmp <- swapping(my.rownames=rn)
```

This is one option. This is basically executing the previously defined function, but within another function. We first put in the memory our swapping function, and then execute it on rn.

```
> head(tmp)
[1] "ENSG00000121410:A1BG"
[2] "ENSG00000148584:A1CF"
[3] "ENSG00000175899:A2M"
[4] "ENSG00000166535:A2ML1"
[5] "ENSG00000184389:A3GALT2"
[6] "ENSG00000128274:A4GALT"
> |
```

We check then the results with the head() command. So, it works.

Let's complicate it a little more. I don't want to use the row names, but rather the whole data frame.

In this way I will simply need to load my data frame and get directly the data frame with the swapped row names.

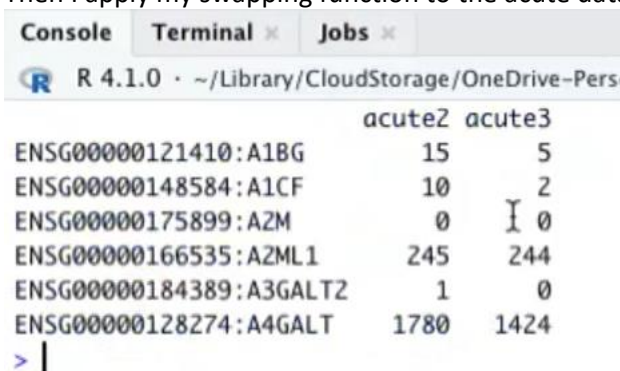
```
swapping <- function(my.df){
  my.rownames <- rownames(my.df)
  rn1 <- sapply(strsplit(my.rownames, ":"), function(x){
    tmp <- paste(x[2], x[1], sep=":")
    return(tmp)
  })
  rownames(my.df) <- rn1
  return(my.df)
}
```

I'm extracting the row names of my data frame directly within the function. Once I've generated rn1, I'm assigning it the row names of my data frame. At the end I'm returning my data frame.

The swapping function gets as input your data frame, assigns the row names in a variable called my.rownames, it strsplit the row names on the basis of the ":", reorganizes the data in a way so that the 1st identifier becomes the 2nd and the 2nd becomes the 1st, reassigns these new row names to the row names of my data frame, and at the end returns my data frame.

```
acute.swapped <- swapping(acute)
```

Then I apply my swapping function to the acute data frame. I call the output acute.swapped.



```
Console Terminal x Jobs x
R 4.1.0 ~ /Library/CloudStorage/OneDrive-Pers
acute2 acute3
ENSG00000121410:A1BG      15      5
ENSG00000148584:A1CF      10      2
ENSG00000175899:A2M        0      0
ENSG00000166535:A2ML1     245     244
ENSG00000184389:A3GALT2     1      0
ENSG00000128274:A4GALT    1780    1424
> |
```

I check what acute.swapped looks like by typing head(acute.swapped). it seems to be ok.

[questi sono i comandi finali scritti finora in command.R

```
19 #this function is used to swap the rownames elements in a dataframe
20 swapping <- function(my.df){
21   my.rownames <- rownames(my.df)
22   rn1 <- sapply(strsplit(my.rownames, ":"), function(x){
23     tmp <- paste(x[2], x[1], sep=":")
24     return(tmp)
25   })
26   rownames(my.df) <- rn1
27   return(my.df)
28 }
29
30
31 acute <- read.table("./acute/acute.txt", header=T, sep="\t", row.names=1)
32 acute.swapped <- swapping(acute) ]
```

I can repeat exactly the same operations on the cronic dataset.

```
cronic <- read.table("./cronic/cronic.txt", header=T, sep="\t", row.names=1)
cronic.swapped <- swapping(cronic)
```

And then I can check the result by using the head() command.

```
> head(cronic.swapped)
```

```

Console Terminal x Jobs x
R 4.1.0 · ~/Library/CloudStorage/OneDrive-Perso

      cronic2 cronic3
ENSG00000121410:A1BG      54      16
ENSG00000148584:A1CF       7       9
ENSG00000175899:A2M       33       7
ENSG00000166535:A2ML1    993     912
ENSG00000184389:A3GALT2     0       2
ENSG00000128274:A4GALT   3057    2822
>

```

The output seems to be fine also in this case.

The cronic.swapped and the acute.swapped data frames are those we will use for the next steps.

Next step is finding DEGs for acute.

```

#DE for acute
library(DESeq2)

```

First thing I must do is loading the library DESeq2.

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

I have to take my acute.swapped table as a matrix, because the type of file required by DESeq2 functions is matrix. The data frame doesn't change, only the data type changes into a numerical matrix.

A matrix is organized in a slightly different way compared to a data frame. The matrix contains data, and the rows and the columns are a list where the element 1 is the row name and the element 2 is the column name. If you set `rownames.force=T`, makes the matrix able to read the row names. However, we don't care at this level.

```

> dimnames(acute.swapped)[2]
[[1]]
[1] "ctrl1" "ctrl2" "ctrl3" "acute1" "acute2" "acute3"

```

Now, from the acute.swapped, I have to take my covariate elements, which are related to the column names. We can see that the output is a list.

```

> dimnames(acute.swapped)[[2]]
[1] "ctrl1" "ctrl2" "ctrl3" "acute1" "acute2" "acute3"

```

Instead, if I do it this way, the output is a vector. The first squared bracket indicates the list, the second squared bracket indicates the inside of the list.

```

coldata <- data.frame(samples=dimnames(acute.swapped)[[2]],
                      condition=factor(c(rep("ctrl", 3), rep("trt",3))))

```

Now I create my coldata data frame.

```

> coldata
  samples condition
1  ctrl1      ctrl
2  ctrl2      ctrl
3  ctrl3      ctrl
4 acute1      trt
5 acute2      trt
6 acute3      trt
>

```

I check what my coldata looks like.

```

> coldata$condition
[1] ctrl ctrl ctrl trt trt trt
Levels: ctrl trt
>

```


I verify that the column condition is a factor.

The reason why I did it this way is that you need a data frame. You cannot add just one column (the condition one) because in that case the data frame would be rather a vector and the system gets mad, and at a certain point you'll get some issue.

```
rownames(coldata) <- dimnames(acute.swapped)[[2]]
```

Let's also set the row names of coldata.

	samples	condition
ctrl1	ctrl1	ctrl
ctrl2	ctrl2	ctrl
ctrl3	ctrl3	ctrl
acute1	acute1	trt
acute2	acute2	trt
acute3	acute3	trt

Now, that's how coldata looks like.

```
dds <- DESeqDataSetFromMatrix(countData = acute.swapped,  
                              colData = coldata,  
                              design = ~ condition)
```

Now I have to create this DESeq2 object containing the count data, the column data, and the design (indicating which comparison you are interested to do).

```
> dds  
class: DESeqDataSet  
dim: 18774 6  
metadata(1): version  
assays(1): counts  
rownames(18774): ENSG00000121410:A1BG ENSG00000148584:A1CF ...  
                ENSG00000159840:ZYX ENSG00000074755:ZZEF1  
rowData names(0):  
colnames(6): ctrl1 ctrl2 ... acute2 acute3  
colData names(2): samples condition  
> |
```

The output is an object of the class DESeqDataSet. You have the metadata and so on. The assay is counts and is the matrix you are actually working with. The rownames are the names that are going to be assigned to each of the rows. The column names are the 2 groups we had defined (ctrl and acute). colData names contains the comparison we consider.

I try to keep my way of doing the experiment always the same. The control is always the 1st covariate, while the condition is always the 2nd covariate. The first variable will be used as reference. The ratio is between the variable 1 (control) and the variable 2 (control). When you detect upregulated genes it means that those genes are more expressed in the treated sample, while when I detect downregulated genes, it means that those genes are more expressed in the control sample.

If you keep everything constant, whatever data you have generated, you know how to read them. If you do not keep everything constant, then you could get confused.

Ideally the first group is always the control one.

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

Now, I must run this command via the DESeq() command.

```
> acute.res <- results(dds)
```

Let's go to the last extraction, via the results() command.

We can re-write the commands as follows:

```

#DE for acute
library(DESeq2)

swapped <- acute.swapped
swapped <- as.matrix(swapped)
coldata <- data.frame(samples=dimnames(swapped)[[2]],
                      condition=factor(c(rep("ctrl", 3), rep("trt",3))))
rownames(coldata) <- dimnames(swapped)[[2]]
dds <- DESeqDataSetFromMatrix(countData = swapped,
                              colData = coldata,
                              design = ~ condition)
dds <- DESeq(dds)
res <- results(dds)
> dim(res)
[1] 18774      6

```

Then I can check the dimensions of the output data. First dimension= all the genes. There is no filter. If you wish to filter your data you can do it later on.

Why is it useful to rewrite everything in this way?

Because now we can include all these elements in a function, called mydeseq.

```

#executing deseq on dataframe made of 3 ctrl and 3 trt
mydeseq <- function(my.df){
  swapped <- my.df
  swapped <- as.matrix(swapped)
  coldata <- data.frame(samples=dimnames(swapped)[[2]],
                        condition=factor(c(rep("ctrl", 3), rep("trt",3))))
  rownames(coldata) <- dimnames(swapped)[[2]]
  dds <- DESeqDataSetFromMatrix(countData = swapped,
                                colData = coldata,
                                design = ~ condition)

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

```

Since we have two similar experiments organized the same way and that have to be run the same way, we can apply this function on both.

```

acute.res <- mydeseq(acute.swapped)
chronic.res <- mydeseq(chronic.swapped)
> chronic.res <- mydeseq(chronic.swapped)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
> |

```

As soon as it finishes running, I check the initial lines of the outputs generated.

```
> head(acute.res)
```

```
log2 fold change (MLE): condition trt vs ctrl
Wald test p-value: condition trt vs ctrl
DataFrame with 6 rows and 6 columns
```

	baseMean	log2FoldChange	lfcSE	stat
	<numeric>	<numeric>	<numeric>	<numeric>
ENSG00000121410:A1BG	6.247015	1.289042	1.037968	1.241890
ENSG00000148584:A1CF	3.623808	1.147181	1.357104	0.845316
ENSG00000175899:A2M	3.910147	0.699868	2.469053	0.283456
ENSG00000166535:A2ML1	160.971654	0.951044	0.257614	3.691746
ENSG00000184389:A3GALT2	0.147655	0.905096	4.080473	0.221812
ENSG00000128274:A4GALT	904.340951	2.088896	0.119913	17.420099

	pvalue	padj
	<numeric>	<numeric>
ENSG00000121410:A1BG	2.14277e-01	3.01061e-01
ENSG00000148584:A1CF	3.97935e-01	5.06552e-01
ENSG00000175899:A2M	7.76827e-01	8.46734e-01
ENSG00000166535:A2ML1	2.22720e-04	5.20973e-04
ENSG00000184389:A3GALT2	8.24461e-01	8.68780e-01
ENSG00000128274:A4GALT	5.80771e-68	3.20460e-66

```
> head(cronic.res)
```

```
log2 fold change (MLE): condition trt vs ctrl
Wald test p-value: condition trt vs ctrl
DataFrame with 6 rows and 6 columns
```

	baseMean	log2FoldChange	lfcSE	stat
	<numeric>	<numeric>	<numeric>	<numeric>
ENSG00000121410:A1BG	16.100904	2.93578	0.854002	3.437672
ENSG00000148584:A1CF	6.085606	2.07897	1.150027	1.807758
ENSG00000175899:A2M	7.707700	2.02533	3.172755	0.638351
ENSG00000166535:A2ML1	473.347964	2.89967	0.183969	15.761685
ENSG00000184389:A3GALT2	0.648143	2.76581	3.955455	0.699240
ENSG00000128274:A4GALT	1479.819804	2.89216	0.116323	24.863215

	pvalue	padj
	<numeric>	<numeric>
ENSG00000121410:A1BG	5.86739e-04	1.25090e-03
ENSG00000148584:A1CF	7.06441e-02	1.09023e-01
ENSG00000175899:A2M	5.23245e-01	6.20371e-01
ENSG00000166535:A2ML1	5.70907e-56	1.40683e-54
ENSG00000184389:A3GALT2	4.84402e-01	5.83919e-01
ENSG00000128274:A4GALT	1.86088e-136	2.74323e-134

```
>
```

Between acute.res and chronic.res the names should be the same, while the numbers should be different.

So, what do we have in these outputs?

- baseMean = number of mean counts associated to each gene (not log-transformed)
- log2FoldChange
- lfcSE = standard error of the model
- stat = it's the statistic from which you derive the p-value
- pvalue = p-value
- padj = the adjusted p-value (after Bonferroni correction)

We are interested in the adjusted p-value and the log2FC.

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

Now it is the moment to add filters. We can apply the filters directly below the results() command. I have to filter the rows to keep only the genes having an absolute log2FC greater or equal to 1 and an adjusted p-value lower or equal to 0.05. In order to do it, we must use the which function.

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

This is another possibility to do the same stuff intersecting two which commands.

So, I can again run my function, which now includes my filters, and re-run the acute.res and the cronic.res.

There is a problem with the NA values.

In which((res\$padj <= 0.05)) & (abs(res\$log2FoldChange) >= 1) :

Error: logical subscript contains NAs

Show Traceback

Rerun with Debug

Since I got this error, the acute.res is just the old one.

> acute.res[1:10,]

ENSG00000148384:A1CF	3.023800	1.147181	1.3371037	0.843310
ENSG00000175899:A2M	3.910147	0.699868	2.4690527	0.283456
ENSG00000166535:A2ML1	160.978654	0.951044	0.2576136	3.691746
ENSG00000184389:A3GALT2	0.147655	0.905096	4.0804729	0.221812
ENSG00000128274:A4GALT	904.340951	2.088896	0.1199130	17.420099
ENSG00000118017:A4GNT	2.166762	4.542241	2.5979258	1.748410
ENSG00000094914:AAAS	2382.476385	-0.195822	0.0866367	-2.260263
ENSG00000081760:AACS	2041.833207	-0.247580	0.0979920	-2.526538
ENSG00000114771:AADAC	0.000000	NA	NA	NA

pvalue
<numeric> padj
<numeric>

ENSG00000121410:A1BG	2.14277e-01	3.01061e-01
ENSG00000148584:A1CF	3.97935e-01	5.06552e-01
ENSG00000175899:A2M	7.76827e-01	8.46734e-01
ENSG00000166535:A2ML1	2.22720e-04	5.20973e-04
ENSG00000184389:A3GALT2	8.24461e-01	8.68780e-01
ENSG00000128274:A4GALT	5.80771e-68	3.20460e-66
ENSG00000118017:A4GNT	8.03930e-02	1.26413e-01
ENSG00000094914:AAAS	2.38049e-02	4.17503e-02
ENSG00000081760:AACS	1.15193e-02	2.13381e-02
ENSG00000114771:AADAC	NA	NA

> |

As you can see in line 10 there are NA values everywhere. This means that this gene was not considered to be significant because its mean expression value is near to 0.

```
dds <- DESeq(dds, is.na(x))
res <- results(dds)
res <- res[!is.na(res$log2FoldChange),]
res <- res[which((res$padj <= 0.05)) & (abs(res$log2FoldChange) >= 1),]
return(res)
}
```

So in order to make this stuff work, you have to discard all the lines where one of the parameters (let's say log2FoldChange is NA). I can do it adding a line where I use the is.na() command. The "!" means "discard". On other word it means that, if the res\$log2FoldChange is NA, that line is discarded. I'm keeping only the

lines giving FALSE as output to the is.na() command. The question is: is this NOT NA? If yes, I keep it; if not, I discard it.

Remember that “==” means “equal”, while “!=” means “different”.

```
#executing deseq on dataframe made of 3 ctrl and 3 trt
mydeseq <- function(my.df){
  swapped <- my.df
  swapped <- as.matrix(swapped)
  coldata <- data.frame(samples=dimnames(swapped)[[2]],
                        condition=factor(c(rep("ctrl", 3), rep("trt", 3))))
  rownames(coldata) <- dimnames(swapped)[[1]]
  dds <- DESeqDataSetFromMatrix(countData = swapped,
                                colData = coldata,
                                design = ~ condition)

  dds <- DESeq(dds)
  res <- results(dds)
  res <- res[!is.na(res$log2FoldChange),]
  res <- res[which((res$padj <= 0.05)) & (abs(res$log2FoldChange) >= 1),]
  return(res)
}
```

So, with this extra step, our function should work.

```
final dispersion estimates
fitting model and testing
Warning message:
In which((res$padj <= 0.05)) & (abs(res$log2FoldChange) >= 1) :
  longer object length is not a multiple of shorter object length
> cronic.res <- mydeseq(cronic.swapped)
estimating size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
Warning message:
In which((res$padj <= 0.05)) & (abs(res$log2FoldChange) >= 1) :
  longer object length is not a multiple of shorter object length
> |
```

If we re-run acute.res and cronic.res we still get a new warning. The error was an error of the brackets of the which() command.

```
res <- res[which((res$padj <= 0.05) & (abs(res$log2FoldChange) >= 1)),]
```

This way it should be correct.

So, let's run the function again. Then we run acute.res and chronic.res too, and we get no warning nor error.

```
> dim(acute.res)
[1] 4027 6
> dim(cronic.res)
[1] 4276 6
> |
```

We can check the dimensions of acute.res and chronic.res. The two row numbers are quite similar.

```
acute.res <- mydeseq(acute.swapped)#4027
cronic.res <- mydeseq(cronic.swapped)#4276
```

I can write the row number in the script file in order to remember it.

When you do differential expression in this way, it can happen that you are discarding a gene that is called as DEG in one group simply because your thresholding (for the adjusted p-value or the log2FC) is changing. Let's imagine a gene having a log2FC of 0.999 in chronic but 1.011 in the acute. It is going to be lost in the chronic but kept in the acute. This is simply because I've put this threshold.

The best way would be that I take the full list of the log2FC, I do something in order to treat them in the same way. I would need a system of ranking them together to see what is actually going on. At this present time, we don't mind. We just want to take the 4027 genes coming from the differential expression in DMSO VS acute treatment and use them in single cell analysis to see if I get a better separation of the data. We currently don't care if the list of DEGs for chronic and acute are overlapping or not.

Now, we can go to the 2nd step. We have to filter the treated single cell samples using the list of DEGs recalled from the acute or the chronic condition.

```
#working on scRNA
sc <- read.csv("../sc_acute/osi_log2CPM.csv", header=T, row.names=1)
```

First of all we must load the osi_log2CPM.csv file (located in the folder sc_acute) using the command read.csv().

It takes quite a lot of time to be loaded.

```
> dim(sc)
[1] 13968  979
> |
```

I look at the dimensions of the loaded data.

```
sc.acute <- sc[which(rownames(sc) %in% rownames(acute.res)),]
```

Now I can filter the row names on based on the DEGs of the acute experiment. I must do it using the which() command or intersect().

```
> dim(sc.acute)
[1] 3523  979
> |
```

As expected, if I check the dimensions of this filtered output, the number of rows is lower with respect to that of the list of DEGs. This means that some of the genes that are in the list of the DEGs are not in the list of the single cell. This is perfectly possible, because these 2 experiments are done in different times. Furthermore, the way you are assigning the gene symbol can also be a little different between the 2 datasets. However, the overlap is pretty good (3523 over 4027).

```
sc.chronic <- sc[which(rownames(sc) %in% rownames(chronic.res)),]
```

We do the same stuff with the chronic DEGs.

```
> dim(sc.chronic)
[1] 3636  979
> |
```

We check the dimensions. Also in this case, the overlap is pretty good (3636 over 4276).

```
library(umap)
```

Now, we must plot. The library we need to load is umap (or tSNE).

```
umap.out <- umap(t(sc.acute), random_state=111, n_epochs = 1000, min_dist=0.05, n_neighbors=15)
```

Let's start with the acute. We do the transposition of the matrix because the rows must be the cells. The umap is done on the cells, so the rows should be the cells. The random_state is just the seed you start with. I set the n_epochs to 1000. Then I set the basic min_dist=0.05 and n_neighbors=15 parameters.

```
f=data.frame(x=as.numeric(umap.out$layout[,1]),y=as.numeric(umap.out$layout[,2]))
```

Then I create a data frame containing the umap output (dimension 1 (called x) VS dimension 2 (called y)).

```
library(ggplot2)
```

From that data frame I'll do the ggplot. Therefore, I need to load the library ggplot2.

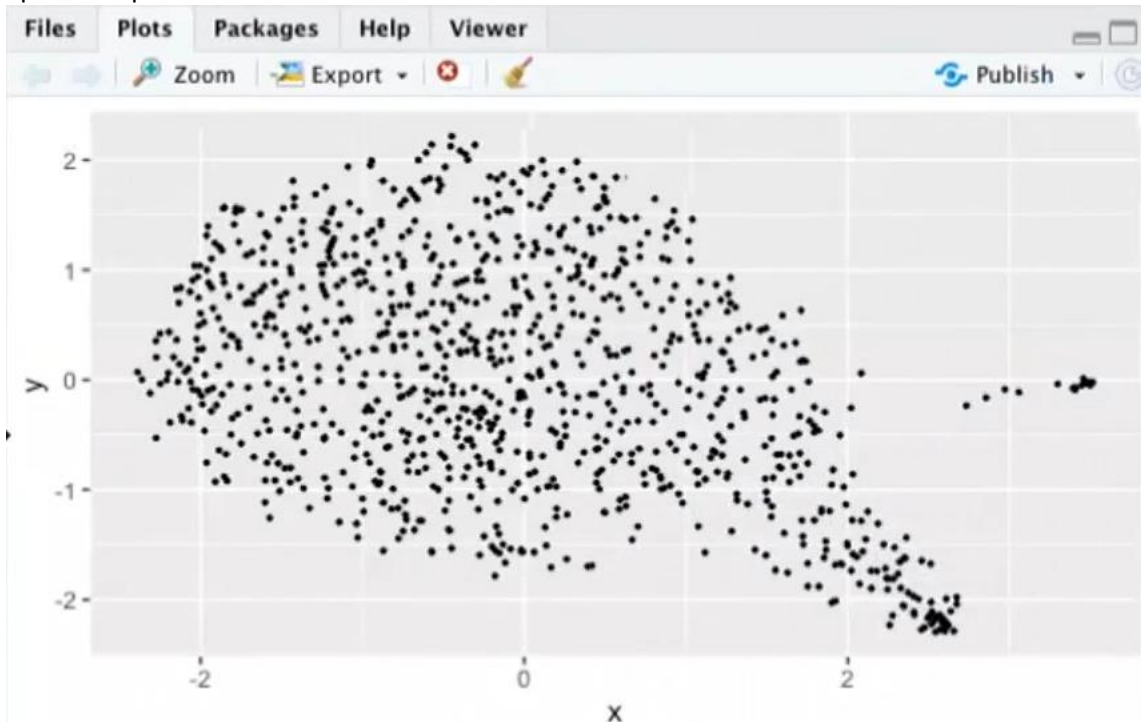
```
#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.3)
```

The "aes" represents the objects you want to plot (x VS y). The geometry is that of a dot plot. pch=19 means that the symbol used is a full circle. We set cex=0.5.

I run the plotting.

```
print(sp)
```

I print the plot.



It is not too bad. If you remember the plot we had before, this plot seems to be a little bit better.

```
pdf("sc_acute_umap_md0.05_nn15.pdf")
print(sp)
dev.off()
```

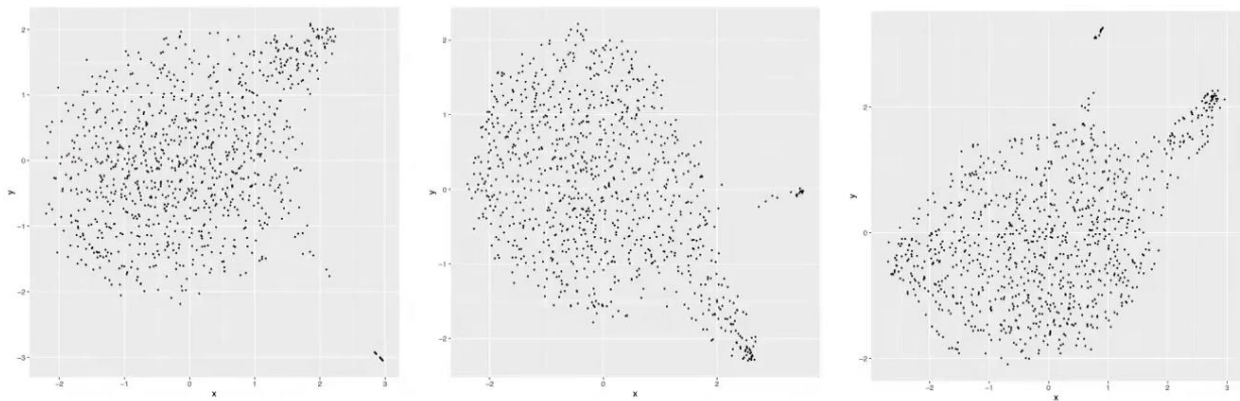
I can generate a pdf from this plot.

```
umap.out <- umap(t(sc.cronic), 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]))
#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.5)
pdf("sc_cronic_umap_md0.05_nn15.pdf")
print(sp)
dev.off()
```

Then, we do the same stuff on the chronic data.

```
umap.out <- umap(t(sc), 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]))
#plotting UMAP
sp <- ggplot(f, aes(x=x,y=y)) + geom_point(pch=19, cex=0.5)
pdf("sc_umap_md0.05_nn15.pdf")
print(sp)
dev.off()
```

We also plot the single cell full data (without filtering for the DEGs), as the question is if these reduced plots are the same or different from the full data plot.



Then I can open an empty power point, take these 3 pdf plots and paste them in a power point slide, so that we can easily compare them. The plot on the left was obtained using the full data, the one in the center is obtained with the acute DEGs and the one on the right with the chronic DEGs.

It is possible that we can improve a little bit the plots by changing the `min_dist` and `n_neighbors` parameters.

When you use the set of the data for the acute and the chronic, we get a slightly better separation compared to the full data plot.

I think that we can improve more the plots by applying a clustering method (we will do it next time).

Instead of using the sets of genes that are chronic or acute, we could try other 2 things (TRY TO DO THESE THINGS FOR THE NEXT TIME):

- take the genes in common between the acute and chronic DEGs lists (the intersection)
- take the genes that are present in the acute DEGs list but NOT in the chronic one and vice versa (genes that are only acute and genes that are only chronic).

We want to know if in these situations the separation gets better or worse.

This is to rule out if the set of cells we see that is detaching from the rest has some meaning or is slightly different from the other cells with respect to looking at the full dataset.

So, this is a way in which we can combine the information coming from bulk RNA seq on a dataset that is noisier, but from which we can extrapolate some extra information.

When you reach a certain level of analysis, it is useful to save the results in a power point. Add in the notes the description of the figures you have obtained.

So, when we are at this point, we still need to do something. Everything we have done do far was done in RStudio, so we didn't save anything.

You can save the old variables using the `save()` command. This is the same it asks you when you try to close RStudio (when it asks you this, it always re-write the same .rmd environment). Ideally in order to save the old variables you should change the name of the environment image. Normally I call the data I want to save as `rdata_date_of_today.rmd`.

```
> rownames(acute.res)
```

```
ENSG00000121410:A1BG
ENSG00000148584:A1CF
ENSG00000175899:A2M
ENSG00000166535:A2ML1
ENSG00000184389:A3GALT2
ENSG00000128274:A4GALT
```

If you remember, the row names of `acute.res` were organized this way.

Now I need only the symbol.

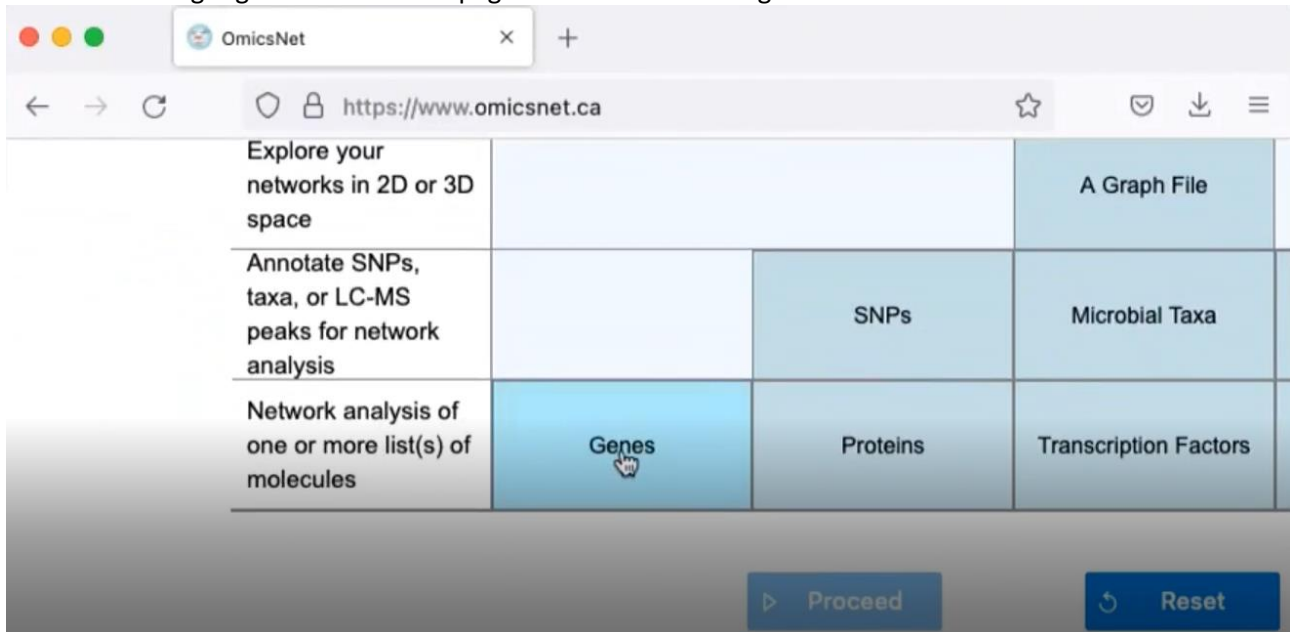
```
#de results
acute.res <- mydeseq(acute.swapped)#4027
tmp <- sapply(strsplit(rownames(acute.res), ":"), function(x)x[2])
```

In tmp I will have only the gene symbols.

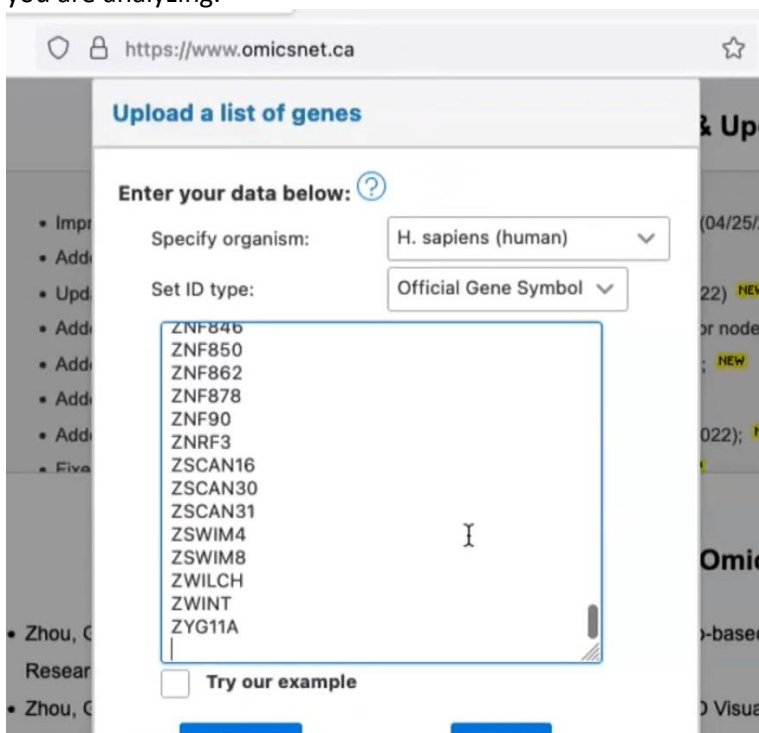
```
writeln(tmp, "acute_sym.txt")
```

Since it is a vector, you can use the writeLines() command. This will write on each row the gene symbols we have extracted. I've just created a file containing the list of the DEGs of the DMSO VS acute treatment.

Let's search on google OmicsNet web page. This is a data mining tool.



It offers various tools to visualize interactions related to functional relationships among the various genes you are analyzing.



In this page we can load our list of genes (in the form of official gene symbols). I simply select all, copy and paste here. I press upload.

After uploading the data, we click on proceed.

Here on the left you can see I have uploaded more than 3000 genes.

We have various options on the right (you can investigate protein-protein, miRNA-gene, metabolite-protein and TF-gene interactions). For the type of data we have, TF-gene interactions is the best option.

There are various databases you can use. I really like TRRUST because it only uses experimentally validated data for the generation of the relationships among TFs and gene targets. I select it and press submit.

Individual Omics Networks

Each network is created independently by searching input list against a selected database. The network usually c

Input Type	Network Type	Sizes (node# - edge# - seed#)	Br
Gene	TF-gene	1126 - 2742 - 730	B

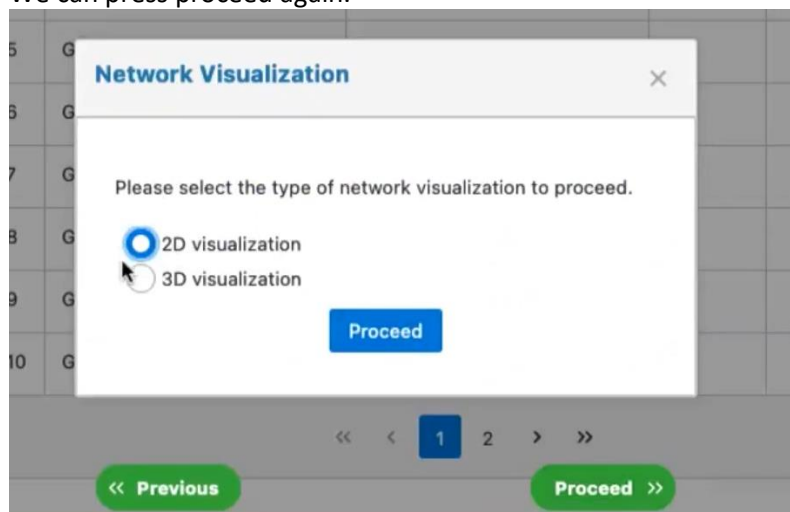
We managed to create a network. I press proceed again.

Multi-omics Network Building

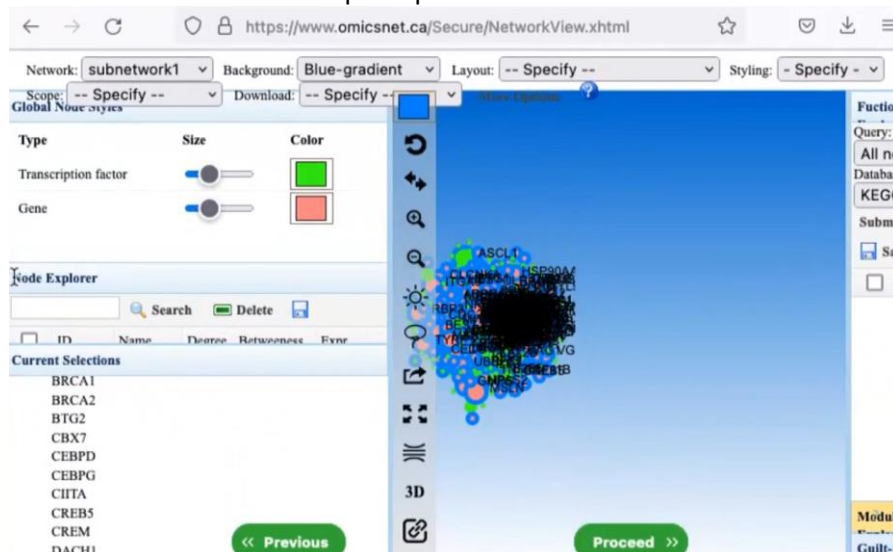
If more than one network was generated in the previous page, they will be merged together to form multi-omics network shared nodes. The network is then decomposed into connected subnetworks available for visual analysis in the next step. If the resulting subnetwork1 is too large, you can trim the network to be suitable for visual analytics (< 2000 nodes) using the **Tools** on the left.

Subnetworks	Seed nodes	All nodes	Edges	Topology	Download
subnetwork1	TF: 129; Gene/protein: 708;	TF: 505; Gene/protein: 708;	2700	View	Download
subnetwork2	TF: 1; Gene/protein: 3;	TF: 2; Gene/protein: 3;	3	View	Download
subnetwork3	Gene/protein: 2;	TF: 1; Gene/protein: 2;	2	View	Download
subnetwork4	Gene/protein: 1;	TF: 2; Gene/protein: 1;	2	View	Download
subnetwork5	Gene/protein: 1;	TF: 2; Gene/protein: 1;	2	View	Download
subnetwork6	Gene/protein: 2;	TF: 1; Gene/protein: 2;	2	View	Download

Here we can see that we have various subnetworks. The first one is the biggest one, accounting for 505 TFs and 708 gene targets. Then there are many other smaller ones. We can press proceed again.



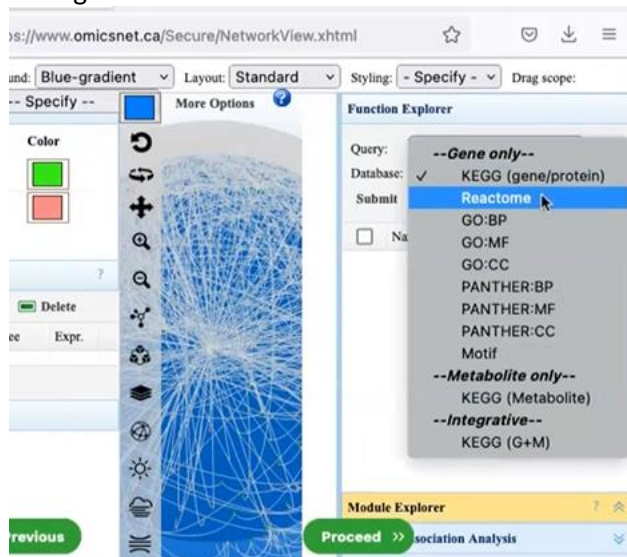
I select 2D visualization and press proceed.



In the node explorer there is the number of nodes that are associated to each gene.

Unfortunately, the visualization is not the best one. The point is that for example SP1 has 144 connections. This means that I can start to look at the TFs having the higher connectivity and look at them as potentially intrinsic TFs associated to our experiment. SP1 actually is not the best one, because it is a very general TF. However, in principle, the visualization coming out from this page could be useful to extract info about the regulation of the genes we detected as DEGs.

OmicsNet helps you a lot in identifying potential upstream regulators with respect to the list of genes you have generated.



The nodes you have can be for example investigated as enrichment for subgroups of data. For example I could run the reactome.

Name	Hits	P-val	P-val(adj.)	Color
Mitotic G1-G1/S phases	28	9.56e-10	0.00000119	
G2/M Checkpoints	16	1.7e-9	0.00000119	
G1/S Transition	24	4.48e-9	0.00000209	
Activation of ATR in response to repli	14	1.27e-8	0.00000445	
DNA strand elongation	12	2.8e-8	0.00000785	
Activation of the pre-replicative compl	12	4.26e-8	0.00000996	
Cell Cycle, Mitotic	49	7.44e-8	0.0000149	
Cell Cycle Checkpoints	24	9.33e-8	0.0000163	
Unwinding of DNA	7	3.42e-7	0.0000505	
DNA Replication	20	3.6e-7	0.0000505	

For example, you can see that 28 hits are related to the mitosis and also more hits are generally related to the growth of the cells.

However, we are looking for genes that are related to the treatment with the drug. This means that the cell cycle genes are somehow affected by the drug.

So, OmicsNet helps you extracting very important information.

In the exam you'll need to use your small R vocabulary to extract what you want.