



BIOC | BioInformatics Competence Center

# Bioinformatic Analysis of RNA-sequencing

## R Introduction: Files & usefull functions

Allison Burns, Maxime Jan, Linda Mhalla, Christian Iseli, Nicolas Gueux

# Usefull functions

# Operation on vector

Repeat a variable or vector X time

```
MyVector<-c(1,2,3,4,5)  
rep(MyVector, times=2)
```

```
## [1] 1 2 3 4 5 1 2 3 4 5
```

Or you can repeat each element of a vector X time

```
rep(MyVector, each=2)
```

```
## [1] 1 1 2 2 3 3 4 4 5 5
```

You can use paste to add some character (e.g. days)

```
paste("Day", MyVector, sep="_")
```

```
## [1] "Day_1" "Day_2" "Day_3" "Day_4" "Day_5"
```

Or collapse all character into a single character

```
paste("Day", MyVector, sep="_", collapse = ".")
```

```
## [1] "Day_1.Day_2.Day_3.Day_4.Day_5"
```

# The *apply* functions

the apply functions can be usefull to use a function on every item of a vector, list, matrix

- apply to use a function on item in a vector
- apply on a matrix

```
MyGenes<-c("Arnt1", "Per2", "Cry1", "per1")
sapply(MyGenes, toupper)
```

```
##      Arnt1      Per2      Cry1      per1
## "ARNTL"    "PER2"    "CRY1"    "PER1"
```

- lapply can be used on a list and will return list

```
lapply(MyGenes, toupper)
```

```
## [[1]]
## [1] "ARNTL"
##
## [[2]]
## [1] "PER2"
##
## [[3]]
## [1] "CRY1"
##
## [[4]]
## [1] "PER1"
```

```
MyMatrix<-matrix(c(1,2,3,4,5,6),ncol=2)
MyMatrix
```

```
##      [,1] [,2]
## [1,]  1   4
## [2,]  2   5
## [3,]  3   6
```

```
# Compute mean value for each column
apply(MyMatrix, 2, mean)
```

```
## [1] 2 5
```

To compute for each row, use: apply(MyMatrix, 1, mean)

For sum of matrix, you can simply use RowSum or ColSum as well as rowMeans and colMeans

```
rowSums(MyMatrix)
```

```
## [1] 5 7 9
```

```
rowMeans(MyMatrix)
```

```
## [1] 2.5 3.5 4.5
```

```
colSums(MyMatrix)
```

```
## [1] 6 15
```

```
colMeans(MyMatrix)
```

```
## [1] 2 5
```

# aggregate function

You may want to compute some mean or sd over same samples or genes. You can use the aggregate function

```
MyData<-data.frame(X=1:4,  
                   groups=rep(c("A", "B"), each=2))
```

MyData

```
##      X groups  
## 1 1      A  
## 2 2      A  
## 3 3      B  
## 4 4      B
```

```
aggregate(x = MyData$X, by = list(MyData$groups), FUN = mean)
```

```
##      Group.1  x  
## 1          A 1.5  
## 2          B 3.5
```

# Create Functions

You can create your own function with some arguments

```
MyFunction<-function(arg1,arg2){  
  SumOfArgs<-arg1 + arg2  
  return(SumOfArgs)  
}  
MyFunction(arg1 = 1, arg2 = 10)
```

```
## [1] 11
```

You can have default arguments

```
MyOtherFunction<-function(arg1=3, arg2="Cortex"){  
  RepArg<-rep(arg2, arg1)  
  return(RepArg)  
}  
MyOtherFunction()
```

```
## [1] "Cortex" "Cortex" "Cortex"
```

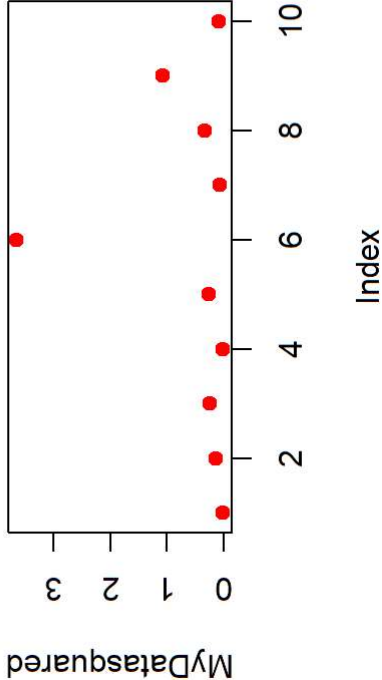
You can pass any named or unnamed argument using the '...'

```
MyPlotFunction<-function(x,...){  
  MyDataSquared<-x^2  
  plot(MyDataSquared,...)  
}
```

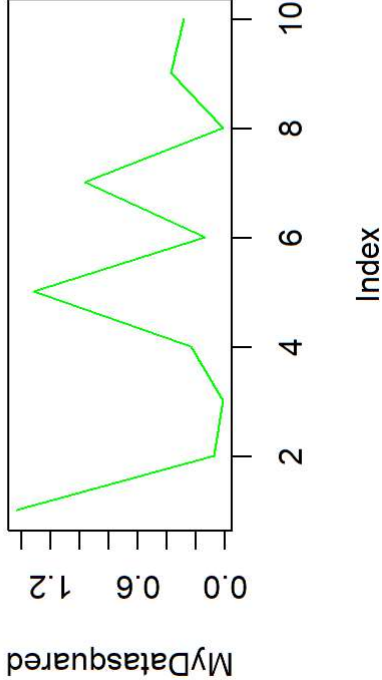
The additional arguments are passed to the *plot()* function

```
MyPlotFunction(x = rnorm(10) , pch=19,  
  col = "red", main="Dots")
```

Dots



line





# Local and global variables

Within a function, your variable are local.

I can access within a function a variable defined outside a function (e.g. *Variable\_A*)

```
Variable_A<-1

MyFunction<-function(){
  Variable_B<-2
  print(Variable_A)
}

MyFunction()
```

## [1] 1

however, Variable\_B do not “exist” outside the function

```
exists("Variable_B")
```

## [1] FALSE

Also you cannot change Variable\_A inside your function with a simple assignment operator '<-'

You need a superassignment operator '<<-'

```
Variable_A<-1
Variable_B<-1
MyFunction<-function(){
  Variable_A<-2
  Variable_B<<-2
  return(invisible(NULL))
}
MyFunction()

Variable_A
```

## [1] 1

```
Variable_B
```

## [1] 2

# Loops

You can use for loop

```
for (i in 1:10){  
  print(i)  
}
```

```
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7  
## [1] 8  
## [1] 9  
## [1] 10
```

While loop

```
Num<-4  
while(Num > 0){  
  Num<-Num - 1  
  print(Num)  
}
```

```
## [1] 3  
## [1] 2  
## [1] 1  
## [1] 0
```

# if else

```
Variable_A<- TRUE
x<-1

if (Variable_A==TRUE){

  x<-x+1

  y<-10

}else{
  x<-x-1
}

x
```

## [1] 2

y

## [1] 10

# gsub

If you want to replace some character into a string, use gsub (see regular expression)

*gsub(pattern, replacement, x)*

```
MyData<-c("Mus Musculus", "Homo Sapiens")
gsub(" " " " " ", MyData)
```

```
## [1] "Mus_Musculus" "Homo_Sapiens"
```

e.g. a more complex example

```
Pathways<-c("Glycolysis - Rattus norvegicus (rat)")
```

```
gsub(" - Rattus norvegicus \\(rat\\)", "", Pathways)
```

```
## [1] "Glycolysis"
```

# reshape2 package

Often, you will recieve data in the form of a data.frame or matrix, and you want to transform it into a data.frame with a single value per row.

A usefull package is *reshape2* that will help you to do this:

```
MyDataFrame

##      Genes Sample1 Sample2 Sample3
## 1 Cry1      6      5      3
## 2 Per2      3      5      9
## 3 Arnt1     2      8      5
## 4 Cdk4      3      3      5

library(reshape2)
MyNewDF<-melt(MyDataFrame,id.vars = "Genes",
              measure.vars = c("Sample1", "Sample2", "Sample3"),
              variable.name = "Sample",value.name = "ReadCounts"
            )
MyNewDF
```

We can *melt* this

##	Genes	Sample	ReadCounts
## 1	Cry1	Sample1	6
## 2	Per2	Sample1	3
## 3	Arnt1	Sample1	2
## 4	Cdk4	Sample1	3
## 5	Cry1	Sample2	5
## 6	Per2	Sample2	5
## 7	Arnt1	Sample2	8
## 8	Cdk4	Sample2	3
## 9	Cry1	Sample3	3
## 10	Per2	Sample3	9
## 11	Arnt1	Sample3	5
## 12	Cdk4	Sample3	5

Statistical test

Many statistical test are available in R:

- student test: *t.test()*
- non-parametric t-test: *wilcox.test()*
- correlation: *cor.test()* with pearson,spearman or kendall methods
- ANOVA: *aov()*, for type II or type III ANOVA you can use the the car package (*car::Anova()*)
- ANOVA post-hoc tukey using *TukeyHSD()*
- Linear model using *lm()* (we will see how to write your model !)
- Mixed effect model using lme4 package

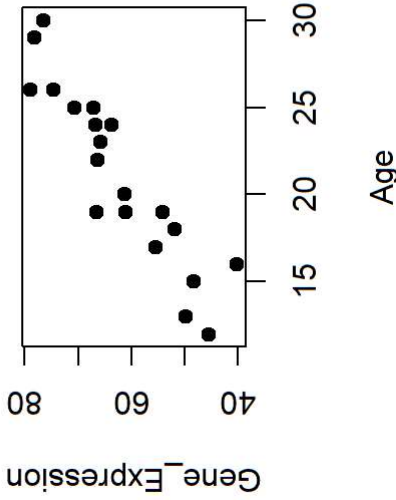
# Linear Regression

Linear model is written in this form:

$$y = \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 \dots$$

- where y is your variable of interest (dependent variable)
- X your predictors (independent variables)
- $\beta$  values are the weights of your predictors (what you generally want to test)

```
plot(Age, Gene_Expression, pch=19)
```



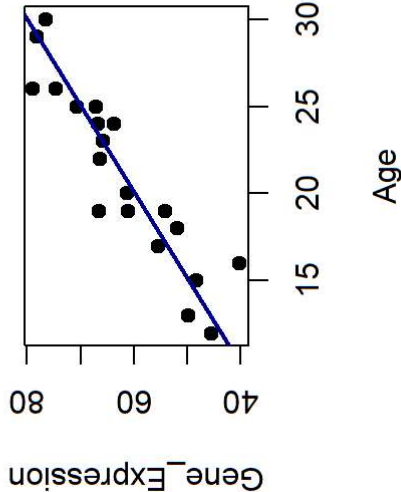
We test for an effect of age on gene expression

```
summary(lm(Gene_Expression~Age))[[4]]
```

##		Estimate	Std. Error	t value	Pr(> t )
##	(Intercept)	19.592825	4.4537217	4.399203	3.460734e-04
##	Age	2.016758	0.2054567	9.815975	1.187690e-08

And found a significant effect of age:

$$GeneExpression = \beta_0 + \beta_1 * Age + \epsilon; \beta_1 = 2.016758`$$





# Design Matrix

Depending on your design, DEseq2 may require a design matrix and which coefficient you want to test.

Your design may look like this:

##	Gene_Expression	Treatment	Batch
## 1	0.16022185	Control	B1
## 2	-1.34087504	Control	B2
## 3	-0.50945013	Control	B3
## 4	-3.49290641	A_vehic	B1
## 5	1.49877604	A_vehic	B2
## 6	0.62524329	A_vehic	B3
## 7	0.03776714	Treatment	B1
## 8	0.30327340	Treatment	B2
## 9	-0.78182847	Treatment	B3

You can write a model matrix that will look like this, but your Treatment A\_vehic becomes the reference (your coefficient 4 represent the difference of *Control* – A\_vehic, and coefficient 5: *Treatment* – A\_vehic)

```
head(model.matrix(~Batch + Treatment, data=MyMetadata),9)
```

##	(Intercept)	BatchB2	BatchB3	TreatmentControl	TreatmentTreatment
## 1	1	0	0	1	0
## 2	1	1	0	1	0
## 3	1	0	1	1	0
## 4	1	0	0	0	0
## 5	1	1	0	0	0
## 6	1	0	1	0	0
## 7	1	0	0	0	1
## 8	1	1	0	0	1
## 9	1	0	1	0	1

To solve this problem, write Treatment as factor and relevel it:

```
MyMetadata$Treatment<-factor(MyMetadata$Treatment,
                              levels = c("Control", "A_vehic", "Treatment"))
```

Now your coefficient 4 represent the difference of  $A_{vehic}$  –  $Control$ , and coefficient 5:  $Treatment$  –  $Control$

```
head(model.matrix(~Batch + Treatment, data=MyMetadata), 9)
```

##	(Intercept)	BatchB2	BatchB3	TreatmentA_vehic	TreatmentTreatment
## 1	1	0	0	0	0
## 2	1	1	0	0	0
## 3	1	0	1	0	0
## 4	1	0	0	1	0
## 5	1	1	0	1	0
## 6	1	0	1	1	0
## 7	1	0	0	0	1
## 8	1	1	0	0	1
## 9	1	0	1	0	1

# Files

# Read files

- You can read many different files in R (csv, txt, xlsx etc....)
- If you file has a header, add *header=T*
- You can use the built-in function like *read.table()* or *read.csv()* for text files
- The package openxlsx will read excel format files
- You can also use the *Import Dataset* in Rstudio (upper-right corner)
- You can press *tab* while writing the path to a file to help you
- You can also skip some line in a file using the *skip* option in *read.table()*

# Write files

- To write files use *write.table()* function.
- Some option are very usefull, like:

```
write.table(MyDataFrame,  
            quote=F, # Avoid adding "" to character  
            sep="\t", # Use tab to separate data  
            col.names = T, # Add colnames to the file  
            row.names = T # Add rownames to the file  
            )
```

or save directly your variable in a Rdata object

```
save(MyDataFrame, MyDataFrame2, file="MyDataFrames.Rdata")
```

and load the file using *load()*

```
load("MyDataFrames.Rdata")
```