PARTE 1

PROGRAMACIÓN en R

Recordemos que R puede funcionar como una calculadora, si bien parece algo sencillo, será sumamente útil para desarrollar nuestros scripts.

Por convención utilizaremos el símbolo “>” para referirnos al prompt que nos muestra R para ingresar nuestras sentencias.

```
> 1 + 2
[1] 3
```

Notar que la salida por pantalla, no nos muestra el símbolo “>”

La notación [1] Nos indica, en este caso por ejemplo, que “3”, es el primer elemento del resultado

```
> x <- (1:70)  ## Con el operador “ : “  Puedo crear secuencias de números, en este caso de 1 a 70
> x
>[1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14
 [15]  15  16  17  18  19  20  21  22  23  24  25  26  27  28
 [29]  29  30  31  32  33  34  35  36  37  38  39  40  41  42
 [43]  43  44  45  46  47  48  49  50  51  52  53  54  55  56
 [57]  57  58  59  60  61  62  63  64  65  66  67  68  69  70
```

En este ejemplo: 1, es el primer elemento, 15, es el decimoquinto elemento y asi siguiendo la misma lógica.

Otros símbolos para realizar operaciones:

```
+   Suma
-   Resta
*   Multipliación
/   División
^   Potencia
sqrt   Raíz
```

La asignación de variables se puede realizar mediante el operador “=“ o de la forma clásica en R :“<- “

Por ejemplo, es lo mismo asignarle a la variables “x” el valor de 10, de las siguientes maneras:

```
x = 10
x <- 10
10 -> x
```

Nosotros usaremos a lo largo del curso, la notación tradicional con el operador de asignación “<-”
Con las variables, podremos realizar cualquier operación matemática, por ejemplo:

```
> a <- 2
> b <- 3
> a^b
[1] 8
```

TIPOS DE DATOS

Todo en R es un objeto.

Para construir el lenguaje, empezaremos con los elementos más pequeños, que denominaremos “clases atómicas”, los cuales aún no llegan a ser un objeto.

```
-Character (textos, caracteres)
-Numeric (números reales)
-Integer (números enteros)
-Complex (números complejos)
-Logical (Verdadero/Falso)
```

El objeto más básico en R es un VECTOR

Los vectores solo pueden contener elementos de la misma clase. IMPORTANTE

Un vector, con elementos de diferentes clases, lo llamaremos LISTA.

VECTORES

Los vectores puede crearse con la función c()

```
> x <- c(0.1, 0.2)                ##
Numérico
> y <- c(TRUE, FALSE)      ## Lógico
> t <-  c(“a”, “b”, “c”)          ##
Character
> z <- c(1+0i, 2+4i)              ##
Complejo
```

Otra función muy importante es la función “class”, la cual nos indica que clase de objeto tenemos

```
> x <- c(0.1, 0.2)
> class(x)
[1] "numeric“

> y <- c(TRUE, FALSE)
> class(y)
[1] "logical"

> t <- c("a", "b", "c")
> class(t)
[1] "character”
```

OPERACIONES CON VECTORES

Entre vectores, podemos realizar diversas operaciones matemáticas

Por ejemplo suma:

```
> aa <- c(1, 2, 3)
> bb <- c(10, 1, 0)
> cc <- (aa + bb)
> cc
[1] 11  3  3
```

Otro aspecto sumamente útil y necesario en muchas ocasiones, es el ordenamiento de vectores.

```
> v <- c(11,2,0,1,10,43,3)
> v1 <- sort(v)
> sort(v,decreasing = TRUE)
```

También podemos recorrer un vector:

```
> v[4]
```

Nos muestra el cuarto elemento, del vector “v”

```
> length(v) # Nos muestra la cantidad de elementos en el vector
```

CONVERTIR TIPOS

```
> a <- c(1:10)
> a
 [1]  1  2  3  4  5  6  7  8  9
10
> class(a)
[1] "integer"

>b <- as.numeric(a)
>b
>class(b)
[1] "numeric“
```

Estas conversiones son muy importantes, ya que algunos algoritmos con los que trabajaremos a lo largo del curso solo aceptarán objetos de una clase de datos específica.

MATRICES

Las matrices en R pueden ser vistas como vectores con un atributo de dimensión.

```
> matriz <- matrix(nrow = 2, ncol = 3)

> dim(matriz)
[1] 2 3
```

Si las matrices las cargamos a mano, hay que tener en cuenta que se van llenando por COLUMNAS primero.

```
> matriz <- matrix(1:6, nrow = 2, ncol = 3)


> matriz
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Dos funciones que serán sumamente útiles a la hora de crear matrices son:
“cbind” y “rbind”, las cuales construyen matrices a partir de vectores, ya sea por columnas (cbind) o filas (rbind)

```
> x <- c(1:4)
> y <- c(20, 22, 24, 26 )

> cbind(x, y)
> rbind(x, y)
```

Ej: ¿Qué pasa en el siguiente caso?

```
> x <- c(1:4)
> y <- c(20, 22)
> cbind(x, y)
```

MÁS SOBRE TIPO DE DATOS

Anteriormente dijimos que todos los vectores debían poseer datos de la misma clase?

Ej: ¿Qué conclusiones sacan en esta sentencia?

```
> vector01 <- c("a", 1)
```

Las listas nos permitirán generar vectores con diferentes tipo de datos

```
> vector02 <- list(1, "a", TRUE, 0.4)
> vector02
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 0.4

> class(vector02)
```

Es muy importante comprender el funcionamiento de las listas, ya que muchos sets de datos con los que trabajaremos, tendrán mezclados diferentes tipo de datos

Los elementos de la lista son mostrados entre doble corchete [[i]]
Ahora, cada elemento de la lista, es tratado como un vector independiente

DATA FRAMES

Es el tipo de datos más importante de R.
Son usados para guardar tablas formadas por vectores de la misma longitud.

```
 a <- c(1, 2, 3)
 b <- c("a", "b", "c")
 c <- c(TRUE, FALSE, FALSE)

 data_set <- data.frame(a, b, c)
 data_set

  a b     c
1 1 a  TRUE
2 2 b FALSE
3 3 c FALSE
```

Como vimos, R trae en su core diversos sets de datos. Entre ellos hay un data frame muy estudiado, llamado mtcars

```
mtcars
class(mtcars)
mtcars[1, 2]

mtcars["Merc 280", "cyl"]
```

Útiles comandos para explorar un data frame

```
> nrow(mtcars)
> ncol(mtcars)

> head(mtcars)
> tail(mtcars)

> summary(mtcars)
```

Ej: ¿Cuál creen que es la diferencia más importante entre Matrices y Data Frames?

NOTA: Generalmente cuando importamos un set de datos, vía read.table o read.csv
R nos convierte automáticamente los sets de datos a data frames.

1) Construyendo un dataset:

```
nombre <- c("Juan","Maria","Pedro","Ana")
edad <- c(21,34,42,23)
genero <- c("Masculino","Femenino","Masculino","Femenino")

data_set <- as.data.frame(cbind(nombre,edad,genero))
```

Subset un dataset:

Crear un objeto llamado: “newdata”
Que contenga: ”Todos los hombres mayores a 21 años”

RESPUESTA: `newdata <- subset(nombre, edad > 21 & genero == "Masculino")`

2) Dos casos al azar

3)

Dado el siguiente vector:

vector_1 <- c(1:10,14,16,20,40)

Escribir un script que sume sus ultimos dos valores y los vuelque en un objeto llamado: "resultado_suma"

Ayuda: utilizar la funcion "length"

RESPUESTA:

```
v_suma_1 <- v_new[length(vector_1)]
v_suma_2 <- v_new[length(vector_1)-1]

resultado_suma  <- v_suma_1 + v_suma_2
```

PARTE 2

ESTRUCTURAS DE CONTROL en R

ESTRUCTURAS DE CONTROL

1.For

Estructura:

```
for (var in seq)
{
 expr
}
```

Ejemplo:

```
x <- c(5,12,13)

for (n in x)
{
print(n^2)
}
```

Hay una iteración del loop para cada componente del vector “x”.

En la primera iteración, n toma el valor de 5, en la segunda el valor de 12 y finalmente el valor de 13.

En cada iteración eleva esa componente al cuadrado y la imprime por pantalla.

2. if – else

Estructura:

```
if (cond){
expr
} else {
expr
}
```

Ejemplo:

```
x <- 9

if(x > 9) {
        print("x is larger than 9")
} else {
            print("x es menor o igual a 9")
}
```

Los “if” anidados son probablemente una de las estructuras de control más utilizadas.
Esto sucede cuando hay dos o más condiciones posibles.

Ejemplo:

```
x <- 1

if(x > 9) {

print("x es mayor a 9")

} else if(x > 7) {

print("x es mayor a 7, pero no mayor a  9")

} else {

print("x es menor a 7") }
```

3. while

Estructura:

```
while(cond) {
Expr
}
```

Ejemplo:

```
limite <- 20

n <- 0
s <- 0

while(s < limite) {

n <- n + 1
s <- s + n

}
resultado <- c(n, s)
resultado
```

FUNCIONES

Aquí presentamos una breve introducción a Funciones en R.

Las funciones en R son guardadas como Objetos como cualquier otro que hemos estado utilizando hasta el momento. Por ejemplo “lm”.

Las funciones se crean con el comando: “function”

Estructura:

```
f <- function(<argumentos>) {

expr

}
```

FUNCIONES

Ejemplo

```
f <- function(a) {
a^2
}
f(2)
```

¿Cuál es el resultado?
¿La siguiente sentencia, funciona? ¿Por qué?

```
f <- function(a, b) {
a^2
}
f(2)
```

¿Y la siguiente, funciona? ¿Por qué? Escríbala de tal manera, que funcione correctamente.

```
f <- function(a, b)
{
print(a)
print(b)
}
f(45)
```

EJERCICIOS

TRUE o FALSE:

(1)

```
x <-  12
x > 5 & x < 15
```

(2)

```
x <-  12
x > 5 | x > 15
```

(3)

```
a <- c(1,4,77,11,21,0,2,8)
```

Cuantos valores mayores a 11 hay?

(4)

```
!FALSE
is.numeric(5)
```

Que devuelven?

PARTE 3

LIBRERÍAS FINANZAS QUANTITATIVAS EN R

quantmod - Quantitative Financial Modelling & Trading Framework for R

Teniendo instalado R correctamente, seguir los proximos pasos:

---

(1) DOWNLOAD & INSTALL Rtools

Estas son una serie de herramientas de R, que hay que bajar e instalarlas. Aqui les dejo el link de la ultima version.

https://cran.r-project.org/bin/windows/Rtools/Rtools34.exe

==========

(2) INSTALL "curl"

Instalar librerias en R es muy sencillo. Aqui hay que abrir la aplicacion R.

Una vez que abrimos la consola de R, con el comando "install.packages" se instalan las librerias, por ejemplo en este caso "curl". Siempre respetar comillas y mayusculas

```
> install.packages("curl")
```

==========

(3) INSTALL "devtools"

Estas son varias funciones para desarrollo y programacion

```
> install.packages("devtools")
```

==========

(4) Install Last QUANTMOD with Yahoo Finance patch

Cargar en memoria la libreria “devtools”

```
> library(devtools)
```

Finalmente ahora si podemos instalar QUANTMOD. Podria ser tan sencillo como:
install.packages("quantmod")

quantmod - Quantitative Financial Modelling & Trading Framework for R

{ quantmod }

The quantmod package for R is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models.

A rapid prototyping environment, where quant traders can quickly and cleanly explore and build trading models.

a). Cargar libreria

```
> library(quantmod)
```

b) Traer datos

```
> getSymbols("AAPL")
> class(AAPL)
> head(AAPL)
> tail(AAPL)
```

AAPL es un objeto xts zoo relacionado con Series de Tiempo.

```
> symbs <- c("AAPL","YPFD.BA","^MERV") # Declaro varios symbols al mismo tiempo
> getSymbols(symbs)
```

Notar que para indice Merval se usa ^ para traer datos, y MERV es el objeto creado: > head(MERV)

Bajar datos por rangos de fechas:

```
> from.dat <- as.Date("01/01/17", format="%m/%d/%y")
> to.dat <- as.Date("05/30/17", format="%m/%d/%y")
> getSymbols("GGAL.BA", src="yahoo", from = from.dat, to = to.dat)

> head(GGAL.BA)
> tail(GGAL.BA)
```

Convertir data a mensual o semanal

```
> ggal.month <- to.monthly(GGAL.BA)
> ggal.week <- to.weekly(GGAL.BA)
```

Una buena practica es renombrar el nombre de las variables del set de datos:

```
> names(AAPL) <- c("Open","High","Low","Close","Vol","Adj") # Renombrar Variables
> head(AAPL)
```

Guardar datos en Tiempo Real – (Recordar cambiar el PATH)

```
getQuote("AAPL") #for Real Time
getQuote("AAPL")$Last

realtime_AAPL <- getQuote("AAPL")$Last

write.csv(AAPL,file="/Data/aapl.csv") # Guardo datos en formato CSV - Reemplazar por su PATH
write.table(realtime_AAPL, "/Data/aapl_price.csv", sep=",", col.names=F, append=T)
```

quantmod - Quantitative Financial Modelling & Trading Framework for R

Indicadores Tecnicos – Ejemplo RSI 21

```
> aapl_rsi_21 <- RSI(Cl(AAPL),21) # RSI 21 sobre el Valor de Cierre(Cl)
```

EJERCICIO:

Conformar un dataset que tenga como variables:

** Al menos 4 parametrizaciones diferentes del Indicador RSI

** Al menos 2 Parametrizaciones del Indicador SMA

** Al menos 2 Parametrizaciones del Indicador EMA

** Estudiar el indicador MACD y pensar como incorporarlo al set de datos.

** Como variable Target el “Precio de una acción”

** Pensar como poner como Target el Rendimiento diario

Luego, estudiar correlaciones entre variables.