PRÁCTICAS DE ESTADÍSTICA CON *R*

PRÁCTICA 1: INTRODUCCIÓN A *R*

1.1 Empezando a trabajar con *R*

El programa *R* (software libre) se puede descargar en la siguiente dirección de internet: http://cran.r-project.org/.

Para hacernos una idea de cómo trabaja *R* veamos un ejemplo de una sesión (tras teclear el texto en negrita pulsar **Enter**):

```
> x <- rnorm(11)              # Se generan 11 números
aleatorios de una distribución N(0,1)
> x
  [1]  1.248696794  1.270055408 -1.846021426
-0.878438503  0.278742064
  [6]       2.219470550    -0.028439195    -1.142189194
0.357516150 -0.873171993
 [11] -0.582792094
```

En las líneas anteriores se observa que después de introducir las instrucciones (en negrita) y tras pulsar **Enter** el programa nos va devolviendo las salidas correspondientes a cada sentencia. Es de resaltar que la primera sentencia

```
> x <- rnorm(11)
```

al igual que ocurre con otras, no da directamente ninguna salida; sencillamente almacena el valor de **x.** Si queremos visualizar ese valor debemos ejecutar la siguiente sentencia

```
> x
```

que nos da los 100 valores de **x** que se habían almacenado en la sentencia anterior.

Para empezar a trabajar con el programa realizaremos una de las tareas más sencillas que pueden ser llevadas a cabo en *R;* se trata de la introducción de una expresión aritmética y la obtención del resultado devuelto por el programa. Por ejemplo,

```
> 5*8
[1] 40
> exp(1)
[1] 2.718282
```

Para asignar un valor a una variable se utilizan los símbolos **<-**. Para insertar los comentarios que se deseen, basta anteponerles el símbolo #:

```
> # Ahora vamos a definir la variable x
> x<-5
> x
```

```
[1] 5
> x^3
[1] 125
> x*8
[1] 40
```

Cuando falta algo para completar una sentencia aparece el símbolo+:

```
> sqrt(3
+
```

Si completamos ahora la expresión, en este caso con el cierre del paréntesis, el programa escribe el resultado:

```
> sqrt(3
+ )
[1] 1.732051
```

1.2 Introducción de datos

Para construir un vector se utiliza la sentencia **c()**:

```
> pesoenkg<-c(5,2.8,3.7,4.6,8.1,3.2)
> pesoenkg
[1] 5.0 2.8 3.7 4.6 8.1 3.2
> pesoengr<-pesoenkg*1000
> pesoengr
[1] 5000 2800 3700 4600 8100 3200
```

Para construir series de valores, por ejemplo los números impares comprendidos entre 1 y 65, hacemos:

```
> seq(1,65,2)
[1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33
35 37 39 41 43 45 47 49
[26] 51 53 55 57 59 61 63 65
```

Mediante la función **rep** es posible repetir un patrón dado, incluso caracteres:

```
> rep(8,4)
[1] 8 8 8 8
> rep(98,5)
[1] 98 98 98 98 98
> rep(c("sí","no"),3)
[1] "sí" "no" "sí" "no" "sí" "no"
```

1.3 Lectura de datos de un archivo

Si se desea cargar datos desde un archivo externo, por ejemplo si se quiere leer el archivo “2-Tensión de rotura.txt” debemos especificar con total exactitud primero la ruta y finalmente el nombre del archivo, todo ello entre comillas. Es conveniente poner el símbolo \\ en lugar de \:

```
> tensión<-
read.table("C:\\Estadística\\Prácticas\\Tensión de
rotura.txt")
> tensión
     V1
1  4.05
2  4.58
3  4.42
...
49 3.54
50 4.84
```

También es posible crear un archivo en Excel (*data frame* o marco de datos) y pasarlo a *R*. Por ejemplo, creamos el archivo “DF.txt” y lo guardamos como *Texto (delimitado por tabulaciones).*

Costo.unit	Costo.mat	Costo.mano.de.obra
13.59	87	80
15.71	78	95
15.97	81	106
20.21	65	115
24.64	51	128

Los puntos que aparecen, en lugar de los espacios de los nombres de las variables, deben ser colocados para que funcione la sentencia **read.table**. Ahora leemos el archivo “DF.txt”:

```
>df<-
read.table("C:\\Estadística\\Prácticas\\DF.txt",header
=T)
> df
  Costo.unit Costo.mat Costo.mano.de.obra
1      13.59        87                 80
2      15.71        78                 95
3      15.97        81                106
4      20.21        65                115
5      24.64        51                128
```

El argumento **header=T** significa que la primera línea del marco de datos contiene los nombres de las variables.

Con **attach** las variables son accesibles por su nombre en la sesión de *R* y con **names** se consigue una lista de las variables:

```
> attach(df)
> names(df)
[1]   "Costo.unit"                    "Costo.mat"
"Costo.mano.de.obra"
> Costo.mat
[1] 87 78 81 65 51
```

Si queremos conocer de qué tipo es una variable concreta hacemos

```
> mode(Costo.mat)
[1] "numeric"
```

1.4 Cómo definir funciones

Para definir nuevas funciones en *R*, como por ejemplo $f(x) = 2x^5-3$, se hace:

```
> f<-function(x) 2*x^5-3
> f(3)
[1] 483
```

También se pueden definir funciones de dos o más variables y aprovechar la forma en que trabaja *R* con vectores para realizar todas las operaciones con una sola evaluación. Supongamos, por ejemplo, que queremos evaluar la función $z = \frac{x^2 + y^2}{x + y}$ en 8 puntos (x,y). Podemos comenzar creando los vectores que contienen los valores de interés:

```
> x<-1:8
> x
[1] 1 2 3 4 5 6 7 8
> y<-rep(5,8)
> y
[1] 5 5 5 5 5 5 5 5
```

Definimos ahora la función z, cuyo valor será evaluado en los puntos (x,y) definidos anteriormente:

```
> z<-(x^2+y^2)/(x+y)
> z
[1] 4.333333 4.142857 4.250000 4.555556 5.000000 5.545455 6.166667
6.846154
```

1.5 Cómo gestionar una sesión de *R*

Al iniciar una sesión de *R* hay un directorio de trabajo donde el programa busca, por defecto, cualquier archivo que sea solicitado y donde coloca los archivos que se crean durante la sesión. Es conveniente utilizar distintos directorios para distintos proyectos. Para ver en qué directorio se está trabajando se hace

```
> getwd()
[1] "C:/Archivos de programa/R/R-2.3.1"
```

Si queremos cambiar de directorio elegimos en el menú

File → Change dir…

Todos los *objetos* (variables) creados en *R* se almacenan en un *workspace* (puede haber varios) o espacio de trabajo común. Para ver qué objetos se han definido elegimos en el menú

Misc → List objects

y para eliminarlos todos

Misc → Remove all objects

Si sólo se quieren eliminar, por ejemplo, los objetos **pesoenkg** y **pesoengr** hacemos

```
> rm(pesoenkg,pesoengr)
```

Estas acciones y otras pueden ser llevadas a cabo también con los botones del menú, del mismo modo que hemos hecho para ver o eliminar los objetos del *workspace.*

Es posible guardar el *workspace* en un archivo en cualquier momento haciendo

File → Save Workspace

El *workspace* está formado sólo por objetos, y no por ninguna entrada obtenida durante la sesión. Si se desean guardar las salidas hay que utilizar

File → Save to File

o bien **Cortar y Pegar**.

La historia de los comandos introducidos en una sesión puede ser guardada y cargada mediante

File → Save/Load History

Para ver todos los comandos introducidos durante una sesión hacemos

```
> history(Inf)
```

Algunos paquetes (*packages*) para aplicaciones especiales son parte de la instalación básica de *R*; otros pueden ser obtenidos en la misma página web de donde se descarga el programa. Para ver qué paquetes están instalados hacemos

```
> library()
```

y, por ejemplo, para cargar el paquete **datasets**, hacemos

```
> library(datasets)
```

o bien en el menú usamos la herramienta **Packages.**

Para ver el contenido de **datasets** hacemos

```
> library(help=datasets)
```

Una forma muy práctica de gestionar una sesión de *R* consiste en lo siguiente: una vez terminada la sesión, copiamos y pegamos la salida de la instrucción **history(Inf)** en un editor de texto generando un archivo de extensión .txt. Este archivo es susceptible de ser manipulado, cambiando datos, introduciendo nuevas variables, etc. Posteriormente se copia y se pega en *R* obteniéndose las correspondientes salidas a los cambios efectuados. Veamos un ejemplo: Supongamos que partiendo de los datos del vector x queremos obtener sus cuadrados, sus cubos y la suma de todos los valores al cuadrado. Haríamos lo siguiente:

```
> x<-c(1,-2,3,-8.23,9.06,-2.8,6,-0.5,3,-4)
> x
 [1]  1.00 -2.00  3.00 -8.23  9.06 -2.80  6.00 -0.50
3.00 -4.00
> y<-x^2
> y
 [1]  1.0000  4.0000  9.0000 67.7329 82.0836  7.8400
36.0000  0.2500  9.0000
[10] 16.0000
> z<-x^3
> z
 [1]    1.0000   -8.0000   27.0000 -557.4418  743.6774
-21.9520  216.0000
 [8]   -0.1250   27.0000  -64.0000

> sum(y)
[1] 232.9065

> history(Inf)
```

La salida de la última instrucción la introducimos en un editor de texto obteniendo el archivo “Documento1.txt”:

```
x<-c(1,-2,3,-8.23,9.06,-2.8,6,-0.5,3,-4)
x
y<-x^2
y
z<-x^3
z
sum(y)
```

Supongamos ahora que el primer elemento del vector x lo cambiamos de 1 a 13 y que en vez de calcular las potencias cúbicas de x calculamos las potencias cuartas. Efectuamos esos cambios en el archivo “Documento1.txt”, después lo seleccionamos todo y lo pegamos en *R* obteniendo lo siguiente:

```
> x<-c(13,-2,3,-8.23,9.06,-2.8,6,-0.5,3,-4)
```

```
> x
 [1] 13.00 -2.00  3.00 -8.23  9.06 -2.80  6.00 -0.50
3.00 -4.00
> y<-x^2
> y
 [1] 169.0000    4.0000    9.0000   67.7329   82.0836
7.8400  36.0000   0.2500
 [9]   9.0000  16.0000
> z<-x^4
> z
 [1] 28561.0000      16.0000      81.0000   4587.7457
6737.7174    61.4656
 [7]  1296.0000     0.0625    81.0000   256.0000
> sum(y)
[1] 400.9065
```

PRÁCTICA 1: EJERCICIOS

■Ejercicio 1-1: Considérese la siguiente tabla de valores:

hora	1	3	5	7	9	11	13	15	17	19
nivel	-20,5	23,2	-88	-24,5	22	21	-57	34,8	33	-21,9

1º) Listar todos los objetos que estén en el *workspace* de *R* y eliminarlos.

2º) Con los datos de la tabla construir, mediante un editor de texto, el archivo de texto "entrada.txt" y guardarlo en el disco duro C.

3º) Leer el archivo anterior y generar un marco de datos, denominado "datos".

4º) Obtener el cuadrado de cada uno de los valores de las dos variables y guardar los resultados construyendo el archivo de texto "salida.txt".

■Ejercicio 1-2: Generar todos los números pares comprendidos entre 100 y 199. Realizar un gráfico en el que las abscisas sean estos valores y las ordenadas sus logaritmos naturales, y otro en el que las ordenadas sean sus cosenos.

■Ejercicio 1-3: Definir y representar gráficamente la siguiente función:

$$f(x)=\begin{cases}\dfrac{x}{2}-1 & si\ x\in(2,4)\\ 0 & \text{en otro caso}\end{cases}$$

■Ejercicio 1-4: Generar una lista aleatoria de 100 números naturales comprendidos entre 1 y 10. Ordenar los valores obtenidos en orden creciente y decreciente.

■Ejercicio 1-5: La función **mean** está integrada en *R* y sirve para calcular la media aritmética de un conjunto de datos. Construir una función que haga lo mismo que la función anterior. Ilustrarlo con un ejemplo.