Primera Entrega: Curso R

UNIVERSIDAD DE GRANADA

Estadística Computacional (2019-2020)

Daniel Bolaños Martínez
danibolanos@correo.ugr.es
Grupo 2 - Viernes 11:00h

Índice

1. Introducción.

En el siguiente documento se mostrará el trabajo realizado para la primera tarea de la asignatura Estadística Computacional durante el periodo no presencial.

Se adjuntarán el código de los ejercicios propuestos y las gráficas generadas a partir de los datos proporcionados.

2. Simulación de una extracción de cartas de una baraja.

Se hará una simulación del siguiente experimento: Realizar un muestreo con reemplazamiento de una baraja de 52 cartas, hasta obtener los cuatro ases e indicar el número de extracciones necesarias.

2.1. Con reemplazamiento.

A continuación cargaremos la función **CuatroAses** y veremos su resultados para diferentes parámetros.

Ejemplo mostrando los comentarios y con máximo 20 extracciones:

```
> CuatroAses(T,20)

No he podido obtener cuatro ases en  20 extracciones.
$E
[1] NA

$R
[1] 48 52 47 48 27 52 14 36  9 17 36 24 38 23  7 11 20 17  8 15

$Conseguido
[1] FALSE
```

Ejemplo no mostrando los comentarios y con máximo 50 extracciones:

```
> CuatroAses(F,50)

$E
[1] NA

$R
```

```
 [1] 11 16  6 13 24 28 23  5 34 30 39 47  8 35 17 50  5 37  4 38 11 16 30
    50 19
[26] 28 26 24 32 52  1 48 25  3 45 27 29 32 39 30 22  8 36 16  4 49 16 33
     19  9

$Conseguido
[1] FALSE
```

Se puede utilizar el resultado de la función como entrada de otra función, para repetir varias veces la simulación y estudiar la distribución de la misma.

Mostraremos diversas ejecuciones para la función **DistriAses**. En el primer ejemplo se realizan 4 experimentos con un máximo de 100 extracciones y en el segundo 6 experimentos con un máximo de 80.

```
> DistriAses(4, 100)
[1]  63  77  53 100

> DistriAses(6, 80)
[1] 80 NA 36 29 NA NA
```

2.2. Sin reemplazamiento.

En el guión también podemos observar una función para realizar el experimento sin reemplazamiento con cada extracción.

Leemos la función **CuatroAses.Sin** y procedemos a ejecutar varias pruebas. Podemos ver que en esta ejecución tarda 33 extracciones en obtener los 4 ases de las 52 posibles.

```
> CuatroAses.Sin()

$E
[1] 33

$R
 [1] 41 26 25 44  4 43 46 32 40  8 16 37 34 33 19 20 21 12 38 28 27 42 30
    50 31
[26] 51 22 14 49 36 29 47  1
```

2.3. Con nombre.

En este ejemplo se añaden los nombres de las cartas a cada una, de esta forma en vez de asignar un número a cada carta se le asigna el número y el palo correspondientes.

```
> CuatroAses.Nombres(T,10)

No he podido obtener cuatro ases en 10 extracciones.
$E
[1] NA

$R
[1] 49 21 30 10 46 43 17 11 25

$N
[1] "Diez de Bastos" "Ocho de Copas" ...
[9] "Caballo de Copas"

$Conseguido
[1] FALSE
```

2.4. Ejemplo propuesto: Dados.

A continuación se presenta un ejemplo similar al de las cartas donde jugaremos con dos dados. La idea del experimento está basada en un juego de dados llamado *'Craps'* donde se juega con el valor obtenido en el lanzamiento de dados.

En nuestro caso estudiaremos las probabilidades de obtener los denominados *'ojos de serpiente'* con los dados, es decir, un 2 como suma de dos dados o un 1 en cada dado. Este hecho en el juego puede ser muy bueno o muy malo dependiendo del rol del jugador por lo que resulta de interés estudiarlo.

Hemos definido la función **DosDados** para controlar el experimento:

```
DosDados = function(Mostrar=F, Maximo=1000, Total=1)
{

#####################################################################
#
# Experimentamos con el numero de lanzamientos necesarios
# para obtener un dos con dos dados
#
# Pone el contador de aciertos a 0
# Hacemos 2 lanzamientos y sumamos el resultado
# Incrementamos el numero de lanzamientos
```

```
# Si no es un 2 vuelve al principio del bucle
# Si es un 2 se incrementa el contador de aciertos
# Si conseguimos el total de aciertos requeridos salimos del bucle
#
#################################################################

Lanzamientos = 0
Resultado = 1:Maximo
Aciertos = 0
repeat{
  Lanzamientos = Lanzamientos + 1
  if (Maximo <= Lanzamientos ){
    if(Mostrar){
       cat("No he podido obtener ", Total , "acierto(s) en ",
           Lanzamientos, "lanzamientos. \n")
    }
  return(list(E = NA, R = Resultado, Conseguido=F))
  }
  TiraPrimerDado = sample(6, 1, T)
  TiraSegundoDado = sample(6, 1, T)

  Resultado[Lanzamientos] = TiraPrimerDado + TiraSegundoDado
  if(Resultado[Lanzamientos] != 2){next}

  Aciertos = Aciertos + 1

  if(Aciertos == Total){break}
}
length(Resultado) = Lanzamientos

if(Mostrar){
  cat("He necesitado", Lanzamientos, "lanzamientos para obtener", Total,
      "acierto(s) en el experimento\n")
}

return(list(E = Lanzamientos, R = Resultado, Conseguido=T))
}
```

Realizaremos algunas pruebas con la función definida para realizar 1 éxito en nuestro experimento para 20 lanzamientos.

```
> DosDados(T,20,1)

He necesitado 8 lanzamientos para obtener 1 acierto(s) en el experimento
$E
[1] 8

$R
```

```
[1]  9 10  7 11  8  6 12  2

$Conseguido
[1] TRUE
```

También probaremos para realizar 2 éxitos en nuestro experimento para 50 lanzamientos.

```
> DosDados(T,50,2)

He necesitado 12 lanzamientos para obtener 2 acierto(s) en el experimento
$E
[1] 12

$R
[1]  4 10  7  7  2  9  6  7  8  5  8  2

$Conseguido
[1] TRUE
```

Al igual que hicimos con las cartas, podemos definir una función que realice varios experimentos y muestre su distribución. La llamaremos **DistriDados**.

```
DistriDados = function(n=10, Maximo=100, Total=1)
{
Lanzo = vector(length=n)
  for(i in 1:n)
    Lanzo[i] = DosDados(F,Maximo,Total)$E
  Lanzo
}
```

Podemos observar un resumen de los datos y observar el histograma de la distribución con las órdenes **summary()** y **hist()**.

```
> Distribucion = DistriDados()

> summary(Distribucion)

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
3.00    5.00   15.00   22.78   39.00   59.00       1

> hist(Distribucion)
```

A continuación mostraremos el histograma generado para 10 experimentos realizados con un máximo de 100 lanzamientos para obtener un total de 1 acierto en cada uno de ellos.

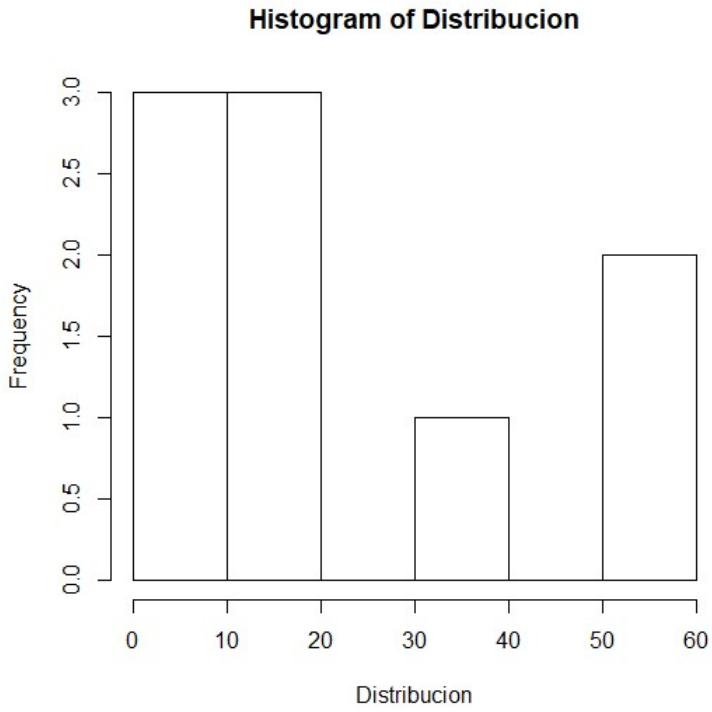

Figura 1: Histograma de la distribución.

2.5. Ejemplo propuesto: Moneda.

Esta vez el experimento consiste en lanzar una moneda y ver el número de caras (valor 1) seguidos que obtenemos en un número máximo de lanzamientos.

Para el experimento valoraremos el número de 1 que obtenemos seguidos. Si durante los lanzamientos obtenemos una cruz (valor 0) reseteamos el experimento manteniendo el recuento de de lanzamientos que llevemos hasta el momento.

Hemos definido la función **LanzaMoneda** para controlar el experimento:

```
LanzaMoneda = function(Mostrar=F, Maximo=10, Total=1)
{

#################################################################
#
# Experimentamos con el numero de caras seguidas
# obtenidas en el lanzamiento de una moneda
#
# Pone el contador de aciertos a 0
# Hacemos 1 lanzamiento 0 cruz 1 cara
# Incrementamos el numero de lanzamientos
# Si sale 1 cara se incrementa el contador de aciertos
# Si sale 1 cruz los aciertos pasan a 0
# Si conseguimos el total de aciertos seguidos salimos del bucle
#
#################################################################

Lanzamientos = 0
Resultado = 1:Maximo
Aciertos = 0
repeat{
  Lanzamientos = Lanzamientos + 1
  if (Maximo <= Lanzamientos){
    if(Mostrar){
      cat("No he podido obtener", Total , "caras en", Lanzamientos, "
          lanzamientos. \n")
    }
  return(list(E = NA, R = Resultado, Conseguido=F))
}
LanzaMoneda = sample(c(0,1), 1, T)

Resultado[Lanzamientos] = LanzaMoneda
if(Resultado[Lanzamientos] == 1){
  Aciertos = Aciertos + 1
}

else{
  Aciertos = 0
}

if(Aciertos == Total){break}
}
length(Resultado) = Lanzamientos

if(Mostrar){
  cat("He necesitado", Lanzamientos, "lanzamientos para obtener", Total,
      "caras seguidas\n")
```

```
}

return(list(E = Lanzamientos, R = Resultado, Conseguido=T))
}
```

Realizaremos algunas pruebas con la función definida para conseguir 2 caras seguidas para 10 lanzamientos.

```
> LanzaMoneda(T,10,2)

He necesitado 6 lanzamientos para obtener 2 caras seguidas
$E
[1] 6

$R
[1] 0 1 0 0 1 1

$Conseguido
[1] TRUE
```

También probaremos para conseguir 3 caras seguidas en nuestro experimento para 30 lanzamientos.

```
> LanzaMoneda(T,30,3)

He necesitado 9 lanzamientos para obtener 3 caras seguidas
$E
[1] 9

$R
[1] 0 0 1 0 0 0 1 1 1

$Conseguido
[1] TRUE
```

Al igual que hicimos con las cartas y los dados, podemos definir una función que realice varios experimentos y muestre su distribución. La llamaremos **DistriMoneda**.

```
DistriMoneda = function(n=10, Maximo=20, Total=2)
{
Lanzo = vector(length=n)
for(i in 1:n)
  Lanzo[i] = LanzaMoneda(F,Maximo,Total)$E
Lanzo
```

```
}
```

Podemos observar un resumen de los datos y observar el histograma de la distribución con las órdenes **summary()** y **hist()**.

```
> Distribucion = DistriMoneda()

> summary(Distribucion)

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
2.00    2.25    3.00    4.20    5.75   10.00

> hist(Distribucion)
```

A continuación mostraremos el histograma generado para 10 experimentos realizados con un máximo de 20 lanzamientos para obtener un total de 2 aciertos en cada uno de ellos.

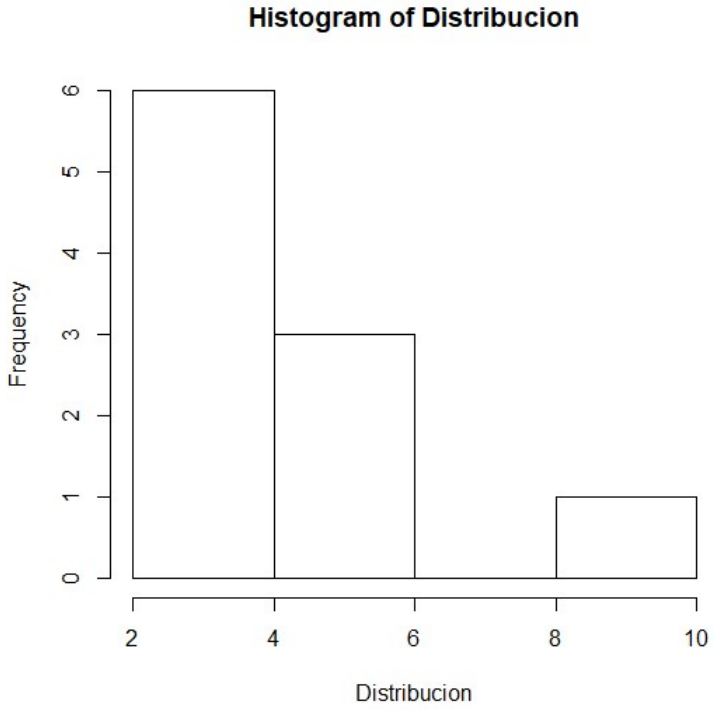

Figura 2: Histograma de la distribución.

3. Epígrafes 3.5.6 al 3.5.8

3.1. Función Suma de potencias.

En primer lugar aparece una función que calcula la suma de los cuadrados y cubos de las componentes de un vector, y los devuelve en un vector.

A continuación realizaremos algunos ejemplos sobre la función **sumpot** definida en el guión.

```
> sumpot(1:5)

$cuadrados
[1] 55

$cubos
[1] 225
```

A continuación se modifica la definición, añadiendo un segundo argumento que, predeterminadamente, da el mismo resultado, pero que permite obtener las sumas de otras potencias. A esta función la llamaremos **sumpot2**.

```
>  sumpot2(1:5,c(2,3))

[[1]]
[1] 55

[[2]]
[1] 225
```

3.2. Función en una cuadrícula.

La siguiente función genera una cuadrícula sobre un conjunto de ordenadas (x) y abscisas (y) y, para cada punto de esa cuadrícula, calcula la distancia hasta el origen de coordenadas.

```
> distancia.origen(1:4,-3:3)

     [,1]     [,2]     [,3] [,4]     [,5]     [,6]     [,7]
[1,] 3.162278 2.236068 1.414214    1 1.414214 2.236068 3.162278
[2,] 3.605551 2.828427 2.236068    2 2.236068 2.828427 3.605551
[3,] 4.242641 3.605551 3.162278    3 3.162278 3.605551 4.242641
[4,] 5.000000 4.472136 4.123106    4 4.123106 4.472136 5.000000
```

También podemos escribir una definición alternativa utilizando la forma general del producto exterior. Funciona de igual manera ofreciendo los mismos resultados.

```
> distancia.ORIGEN=function(x,y){
  outer(x,y,function(x,y) sqrt(x^2+y^2))
}
```

3.3. Procesos no lineales.

Las tres funciones siguientes corresponden a procesos no lineales descritos en **Dong** que, de forma sencilla, definen estructuras complejas.

Realizaremos una prueba para cada función **dong1, dong2, dong3**.

```
> dong1(10)

$x
[1]  0.0000  0.5000  1.5000  2.2500  0.5000 -1.6250 -2.8750 -3.0625
    -0.6250

$y
[1] -1.0000  0.5000  1.0000  0.7500 -1.7500 -2.1250 -1.2500 -0.1875
    2.4375

> dong2(10)

$x
[1] 0.5000000 0.5000000 0.5000000 0.7500000 0.8750000 0.6875000 0.3437500
[8] 0.4218750 0.7109375

$y
[1] 0.5000000 0.7500000 0.8750000 0.4375000 0.2187500 0.6093750 0.3046875
[8] 0.6523438 0.3261719

> dong3(10)

$x
[1]  0.89000000 -0.44860000 -0.35084600 -0.20760694 -0.03488421
    0.15502503
[7]  0.35280221  0.55151013  0.74610314

$y
[1] 2.410000 0.761600 2.265304 3.539542 4.616915 5.525773 6.290706
    6.932988
[9] 7.470980
```

3.4. Lista de búsqueda y entornos.

Podemos encontrar todos los lugares donde se encuentran objetos con las funciones **search** y **searchpaths**.

```
> search()

[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"

> searchpaths()

[1] ".GlobalEnv"
[2] "C:/Program Files/R/R-3.6.2/library/stats"
[3] "C:/Program Files/R/R-3.6.2/library/graphics"
[4] "C:/Program Files/R/R-3.6.2/library/grDevices"
[5] "C:/Program Files/R/R-3.6.2/library/utils"
[6] "C:/Program Files/R/R-3.6.2/library/datasets"
[7] "C:/Program Files/R/R-3.6.2/library/methods"
[8] "Autoloads"
[9] "C:/PROGRA~1/R/R-36~1.2/library/base"
```

Estas funciones devuelven un vector con la lista de libros y objetos de R que están en uso actualmente.

Con las funciones **ls** y **get** podemos buscar y recuperar objetos de las sesiones anteriores.

```
> ls(1,pa="Dados")

[1] "DistriDados" "DosDados"

> get("DosDados",) -> MM
```

4. Uso de archivos externos.

4.1. Ejercicio: Lectura de hojas de datos.

Existen diversas funciones que permiten leer datos de un archivo y almacenarlos directamente en una hoja de datos. Los datos del archivo deben estar en forma de tabla y se creará una hoja de datos con el mismo número de variables que campos o columnas haya en el archivo y con el mismo número de filas que líneas haya en el archivo.

Nos centraremos en las funciones **read** y utilizaremos los datos aportados en los siguientes enlaces proporcionados:

http://www.ugr.es/local/andresgc/Datos.txt , *http://www.ugr.es/local/andresgc/Datos2.txt.*

En primer lugar, leemos los datos con la función **read.csv** y **read.table** almacenamos su valor en una matriz.

```
datos1 = read.csv("http://www.ugr.es/local/andresgc/Datos.txt",header=T,
    sep=",")

datos2 = read.table("http://www.ugr.es/local/andresgc/Datos2.txt",header=
    T)
```

Podemos ver un resumen de los datos con la función **summary** utilizada anteriormente.

```
> summary(datos1)

      Peso            Altura           Edad           Sexo
 Min.   :55.00   Min.   :155.0   Min.   :21.00   Hombre:20
 1st Qu.:62.00   1st Qu.:164.5   1st Qu.:21.00   Mujer :16
 Median :71.50   Median :174.0   Median :22.00
 Mean   :70.72   Mean   :172.9   Mean   :22.14
 3rd Qu.:78.25   3rd Qu.:181.0   3rd Qu.:23.00
 Max.   :89.00   Max.   :190.0   Max.   :25.00

> summary(datos2)

     Nombre        Peso          Altura          Edad          Sexo
 Adela          :1   Min.   :47.0   Min.   :1.600   Min.   :22.00   H:7
 Alberto Garcia :1   1st Qu.:58.0   1st Qu.:1.630   1st Qu.:23.00   M:7
 Andres Garces  :1   Median :65.0   Median :1.675   Median :25.50
 Diego Moreno   :1   Mean   :67.5   Mean   :1.699   Mean   :25.93
 Juan Trucha    :1   3rd Qu.:76.5   3rd Qu.:1.762   3rd Qu.:26.00
```

```
Juana Garcia   :1    Max.    :89.0    Max.    :1.850    Max.    :42.00
(Other)        :8
```

También podemos mostrar el histograma por variable para cada conjunto de datos utilizando la función **hist** de la siguiente forma:

```
> hist(datos1$Altura, main="Altura datos 1")
```

```
> hist(datos2$Peso, main="Peso datos 2")
```

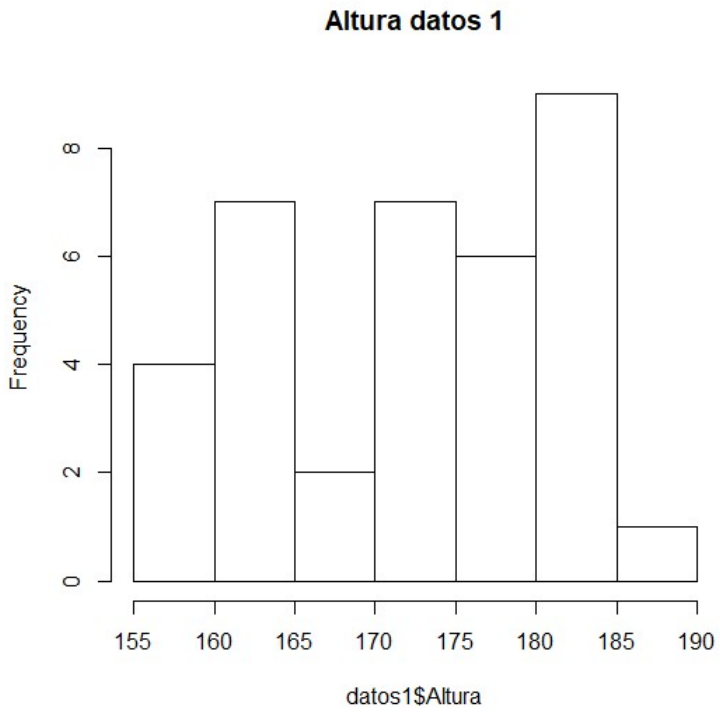

Figura 3: Histograma generado para Altura sobre datos1.

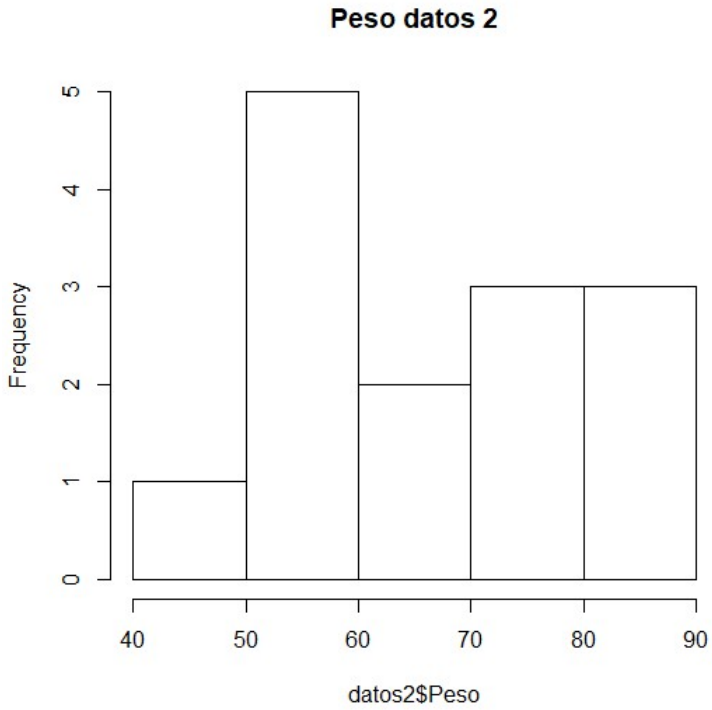

Figura 4: Histograma generados para Peso sobre datos2.

También utilizaremos algunas funciones derivadas de **plot** como **barplot** y **boxplot** que generan gráficos diferentes para cada conjunto de datos.

```
> barplot(datos1$Altura)
> plot(datos1)
> boxplot(datos1)
> barplot(datos2$Peso)
> plot(datos2)
> boxplot(datos2)
```

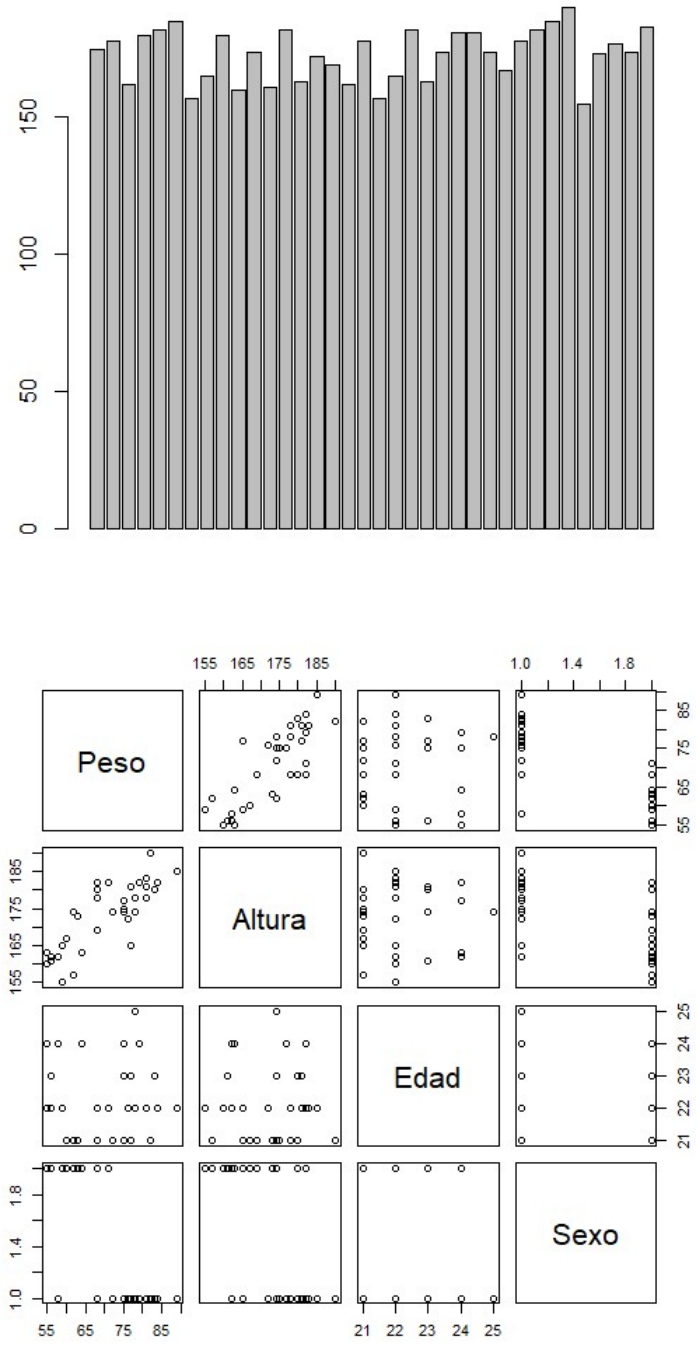

Figura 5: Gráfica para barplot(datos1$Altura) y plot(datos1).

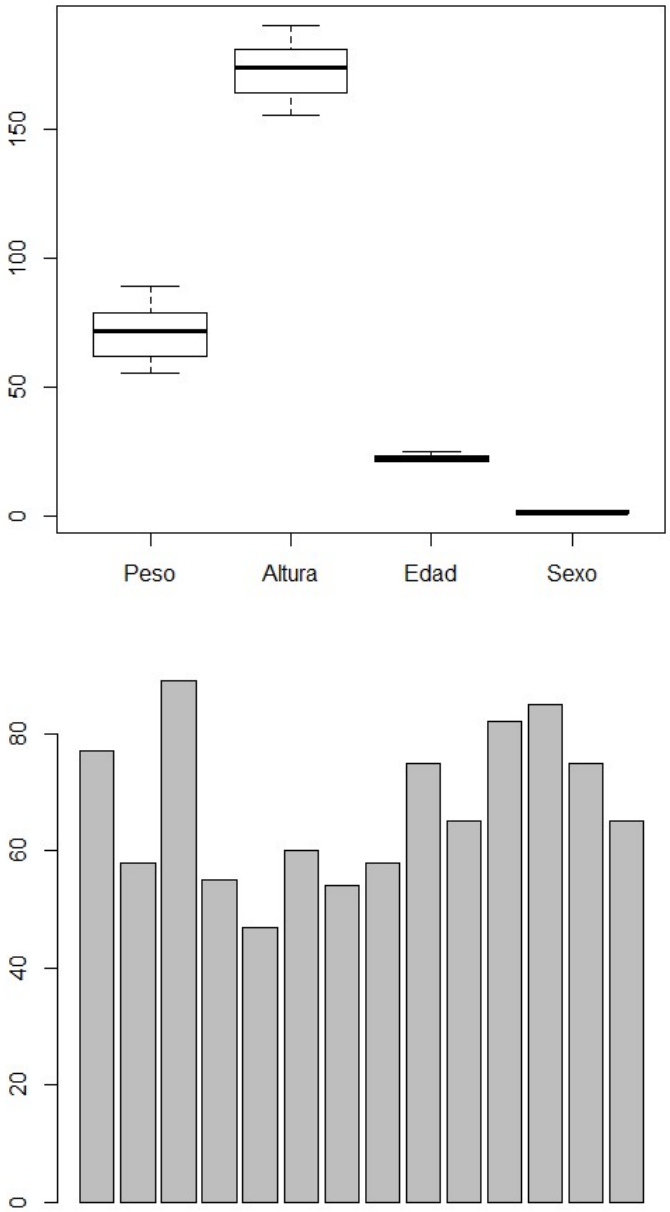

Figura 6: Gráficas para boxplot(datos1) y barplot(datos2$Peso).

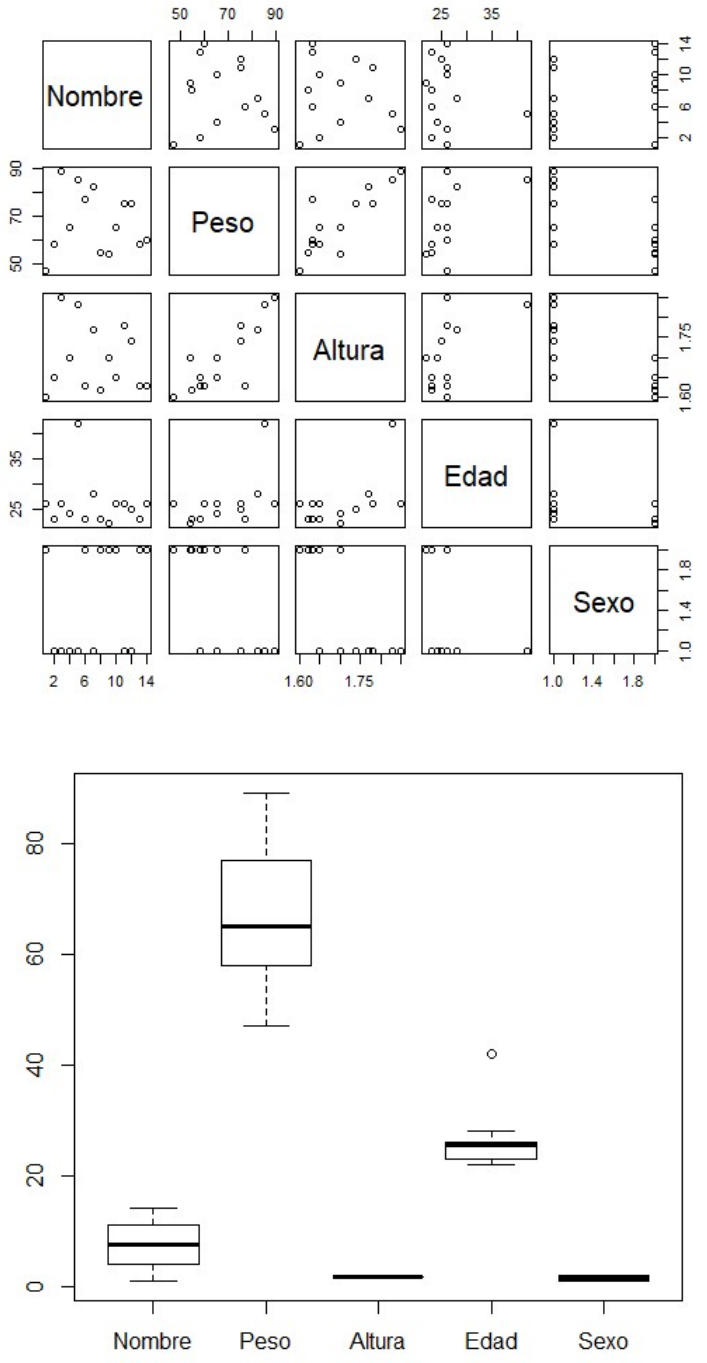

Figura 7: Gráficas para plot(datos2) y boxplot(datos2).

4.2. Gestión de objetos.

Existen diferentes funciones que permiten almacenar y recuperar objetos, unas son binarias (rápidas y exactas) y otras textuales (legibles por el usuario y compatibles entre plataformas y versiones del lenguaje).

En el siguiente ejemplo creamos un vector de números y los guardamos en una variable, seguidamente escribimos su contenido en un archivo de texto y borramos la variable que almacena el vector. Finalmente lo recuperamos leyendo desde el archivo txt.

```
> x <- (1:10)^2
> x
 [1]  1  4  9 16 25 36 49 64 81 100
> write(x, file="x.txt")
> rm(x)
> x
Error: Object "x" not found
> x <- scan("x.txt")
> x
 [1]  1  4  9 16 25 36 49 64 81 100
```

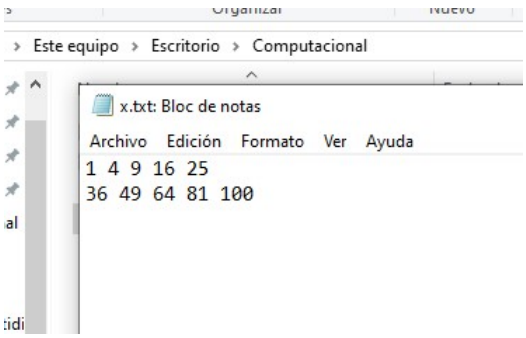

Figura 8: Archivo *x.txt* creado en el path de **getwd()**.

4.3. Ejercicio: Función library.

La función **library** gestiona los libros de la biblioteca, dando información sobre los existentes y cargándolos en memoria o descargándolos de la misma.

Probaremos a cargar y descargar libros tal y como se nos indica en el guión:

```
> search()
```

```
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

```
> library()
```

```
> library("parallel")
```

```
> search()
```

```
 [1] ".GlobalEnv"        "package:parallel"  "package:stats"
[4] "package:graphics"  "package:grDevices" "package:utils"
[7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

```
> library(help="parallel")
```

```
> help("mclapply")
```

```
> detach(pos=2)
```

```
KernSmooth              Functions for Kernel Smoothing Supporting Wand
                        & Jones (1995)
lattice                 Trellis Graphics for R
MASS                    Support Functions and Datasets for Venables and
                        Ripley's MASS
Matrix                  Sparse and Dense Matrix Classes and Methods
methods                 Formal Methods and Classes
mgcv                    Mixed GAM Computation Vehicle with Automatic
                        Smoothness Estimation
nlme                    Linear and Nonlinear Mixed Effects Models
nnet                    Feed-Forward Neural Networks and Multinomial
                        Log-Linear Models
parallel                Support for Parallel computation in R
rpart                   Recursive Partitioning and Regression Trees
spatial                 Functions for Kriging and Point Pattern
                        Analysis
splines                 Regression Spline Functions and Classes
stats                   The R Stats Package
stats4                  Statistical Functions using S4 Classes
survival                Survival Analysis
tcltk                   Tcl/Tk Interface
tools                   Tools for Package Development
translations            The R Translations Package
utils                   The R Utils Package
```

Figura 9: Libros mostrados por **library()**.

```
                    generation.
License:            Part of R 3.6.2
Imports:            tools, compiler
Suggests:           methods
Enhances:           snow, nws, Rmpi
NeedsCompilation:   yes
Built:              R 3.6.2; x86_64-w64-mingw32; 2019-12-12 08:50:03
                    UTC; windows

Index:

clusterApply            Apply Operations using Clusters
detectCores             Detect the Number of CPU Cores
makeCluster             Create a Parallel Socket Cluster
mclapply                Serial versions of 'mclapply', 'mcmapply' and
                        'pvec'
nextRNGStream           Implementation of Pierre L'Ecuyer's RngStreams
parallel-package        Support for Parallel Computation
splitIndices            Divide Tasks for Distribution in a Cluster

Further information is available in the following vignettes in
directory 'C:/PROGRA~1/R/R-36~1.2/library/parallel/doc':

parallel: Package 'parallel' (source, pdf)
```

Figura 10: Ayuda del libro instalado **parallel**.

```
mc.cores: The number of cores to use, i.e. at most how many child
          processes will be run simultaneously.  Must be exactly 1 on
          Windows (which uses the master process).

Details:

     'mclapply' calls 'lapply' and 'pvec' makes a single call 'FUN(v,
     ...)'.  On Unix-alikes 'mc.cores > 1' is allowed and uses parallel
     operations.

Value:

     For 'mclapply', a list of the same length as 'X' and named by 'X'.

     For 'mcmapply', a list, vector or array: see 'mapply'.

     For 'mcMap', a list.

     For 'pvec', a vector of the same length as 'v'.

See Also:

     'parLapply', 'clusterMap'.
```

Figura 11: Información de la funcionalidad *mclapply* del libro **parallel**.

Finalmente para descargar el libro hacemos uso de la función **detach()**.

También podemos instalar paquetes que descarguemos de la web e instalarlos de la misma forma que los paquetes vistos con la orden **library()**.

```
> install.packages("package's name")
> library("package's name")
```

5. Bibliografía.

Referencias

[1] Documento *'Curso R'* de la asignatura.

[2] https://www.r-bloggers.com/box-plot-with-r-tutorial/