

Programando con R (Primera Entrega)

Alberto Jesús Durán López
DNI: 54142189-M
albduranlopez@correo.ugr.es

25 de marzo de 2020

Índice

1	Introducción	3
2	Simulación de una extracción de cartas de una baraja	3
2.1	Con Reemplazamiento	3
2.2	Sin reemplazamiento	5
3	Simulación de lanzamiento de una moneda	7
4	Simulación del lanzamiento de dos dados	10
5	Estudio de Datos.txt	13
6	Estudio de Datos2.txt	18

1. Introducción

En esta práctica realizaremos diferentes experimentos y probaremos los comandos y funcionalidades que nos ofrece R.

Como se hizo en clase, en cada sesión de trabajo establecemos nuestro directorio de trabajo con la orden *setwd(...)*

```
getwd() #Sirve para saber el directorio de trabajo actual
setwd("C:/Users/Alberto/Desktop/Computacional/Entrega") #nuevo
```

Además, podemos invocar a la función *help*, que nos muestra información de los diferentes parámetros de la función que se le pase.

2. Simulación de una extracción de cartas de una baraja

2.1. Con Reemplazamiento

Escribimos en nuestro programa la función que simula la extracción con reemplazamiento de nuestra baraja.

```
CuatroAses<-function(Mostrar=F, Maximo=1000){
  Extracciones=0
  Resultado=1:Maximo
  Ases=c(0,0,0,0)
  repeat{
    if(Maximo<=Extracciones){
      if(Mostrar)
        cat("No se obtienen 4 ases en ", Extracciones, "extrac. \n")
      return(list(E=NA, R=Resultado, Conseguido=F))
    }
    Extracciones = Extracciones+1
    SacoUna = sample(52,1)
    Resultado[Extracciones]=SacoUna

    if(SacoUna%%13!=1) #modulo
      next

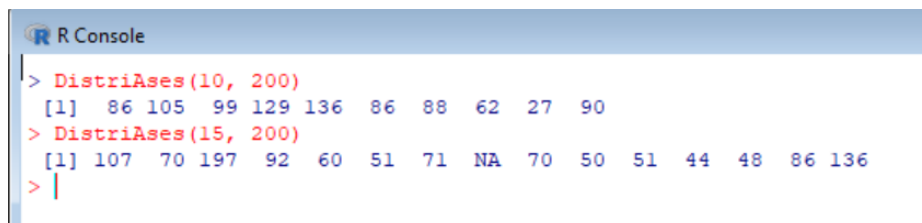
    Ases[(SacoUna-1)%/13+1]=1 #div. entera
    if(sum(Ases)==4)
      break
  }
  length(Resultado)=Extracciones
  if(Mostrar)
    cat("He necesitado ",Extracciones," extrac para sacar los 4 AS\n")

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

Sin embargo, la función anterior está determinada por el azar ya que si la ejecutamos con los mismos parámetros se obtienen diferentes resultados. Por ello, para estudiar su distribución, hacemos uso de la siguiente función:

```
DistriAses = function(n=5, Maximo=1000){
  Saco = vector(length=n)
  for(i in 1:n)
    Saco[i] = CuatroAses(F,Maximo)$E
  return(Saco)
}
```

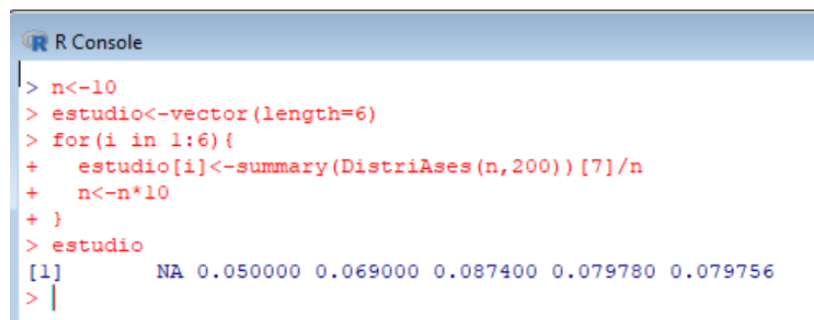
Ahora bien, llamaremos a la función anterior y estudiaremos la frecuencia con la que no se pueden conseguir los 4 ases con 200 extracciones, es decir el número de NA obtenidos en la llamada a la función *summary*.



```
R Console
> DistriAses(10, 200)
[1] 86 105 99 129 136 86 88 62 27 90
> DistriAses(15, 200)
[1] 107 70 197 92 60 51 71 NA 70 50 51 44 48 86 136
> |
```

Como vemos en el ejemplo anterior, con 10 experimentos sí se han obtenido los 4 Ases antes de llegar a las 200 extracciones, mientras que con $n=15$ hay un caso donde no ha sido posible, es decir, a mayor número de experimentos existe una mayor probabilidad de no poder extraer los 4 Ases en 200 extracciones

Vemos que proporción de NA conseguimos respecto al valor inicial de n . Realizamos un bucle **for** donde llamaremos a la función anterior con diferentes valores del parámetro n , desde 10 hasta un millón.



```
R Console
> n<-10
> estudio<-vector(length=6)
> for(i in 1:6){
+   estudio[i]<-summary(DistriAses(n,200)) [7]/n
+   n<-n*10
+ }
> estudio
[1] NA 0.050000 0.069000 0.087400 0.079780 0.079756
> |
```

Una vez ejecutado, mostramos los resultados obtenidos. Con $n=10$ se tenemos que no se ha obtenido ningún NA, con $n=100$ la proporción ha sido de 0.05. Así hasta llamar a la función con $n=1.000.000$, donde vemos que se estabiliza y la proporción obtenida de NA es de 0.079.

2.2. Sin reemplazamiento

Ahora bien, veremos la siguiente función, igual que la anterior pero en la que las cartas se extraen sin reemplazamiento.

```
CuatroAses_sin = function(Mostrar=F){
  Ases=0
  Resultado=sample(52)
  for(i in 1:52){
    if(Resultado[i]%%13!=1)
      next
    Ases=Ases+1
    if(Ases==4)
      break
    if(Mostrar)
      cat("He necesitado ", i, " extracciones para obtener cuatro ases\n")

    return(list(E=i, R=Resultado[1:i]))
  }
}
```

Al igual que pasaba con la función con reemplazamiento, si llamamos la función con diferentes valores de *n*, ésta obtiene diferentes resultados. Por ello, definimos la siguiente función para estudiar su distribución

```
DistriAses_sin = function(n=5){
  Saco=vector(length=n)
  for(i in 1:n)
    Saco[i]=CuatroAses_sin()$E
  return(Saco)
}
```

Estudiaremos ahora los diferentes parámetros de carácter estadístico.

Llamamos a la función y guardamos el resultado en la variable *exp*.

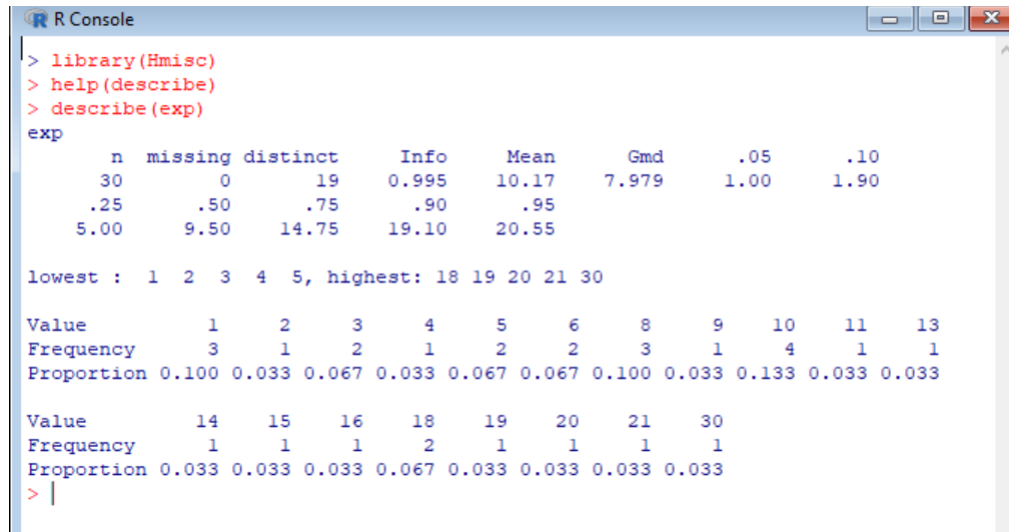
Hacemos llamadas de funciones como *summary* o *fivenum*, recordando que podemos usar el comando *help* para informarnos y saber qué hacen.

- **Summary:** Es una función genérica que devuelve un resumen de los resultados.
- **Fivenum:** Devuelve los 5 números resultantes del test de Tukey (minimum, lower-hinge, median, upper-hinge, maximum).

```
> exp<-DistriAses_sin(30)
> summary(exp)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.00   5.00   9.50  10.17  14.75   30.00
> summary(exp)[5]
3rd Qu.
14.75
> help(fivenum)
> fivenum(exp)
[1] 1.0 5.0 9.5 15.0 30.0
> |
```

Por otro lado, podemos añadir el libro *Hmisc* (instalarlo previamente) y usar su función *describe* que mostramos a continuación:

```
> library(Hmisc)
Error in library(Hmisc) : there is no package called 'Hmisc'
> install.packages("Hmisc")
Installing package into 'C:/Users/Alberto/Documents/R/win-library/3.6'
(as 'lib' is unspecified)
```



```
> library(Hmisc)
> help(describe)
> describe(exp)
exp
  n missing distinct    Info    Mean     Gmd     .05     .10
  30      0       19  0.995  10.17   7.979   1.00   1.90
 .25   .50   .75    .90    .95
 5.00  9.50 14.75 19.10 20.55

lowest : 1 2 3 4 5, highest: 18 19 20 21 30

Value      1      2      3      4      5      6      8      9     10     11     13
Frequency    3      1      2      1      2      2      3      1      4      1      1
Proportion 0.100 0.033 0.067 0.033 0.067 0.067 0.100 0.033 0.133 0.033 0.033

Value      14      15      16      18      19      20      21      30
Frequency    1      1      1      2      1      1      1      1
Proportion 0.033 0.033 0.033 0.067 0.033 0.033 0.033 0.033

> |
```

Ahora bien, volvemos a sobrescribir la instancia `exp`, pero esta vez con `n=10`. Podemos ver los resultados invocando al comando `exp`. Además, existen funciones útiles como `range`, `median`, `mean` o algunos más sofisticados como `sapply`

```
> exp<-DistriAses_sin(10)
> exp
[1] 4 4 13 18 12 12 7 23 8 28
> range(exp)
[1] 4 28
> median(exp)
[1] 12
> mean(exp)
[1] 12.9
> help(sapply)
> sapply(exp,quantile)
```

Para probar una última función, añadimos el libro `pastecs` e invocamos su función `stat.desc`, que nos muestra un resumen de los resultados con los que se llama a la función.

```
> library(pastecs)
> help(stat.desc)
> stat.desc(exp)
  nbr.val  nbr.null  nbr.na      min      max      range
10.0000000 0.0000000 0.0000000 4.0000000 28.0000000 24.0000000
      sum      median      mean  SE.mean CI.mean.0.95      var
129.0000000 12.0000000 12.9000000 2.5274053  5.7173881 63.8777778
  std.dev  coef.var
7.9923575 0.6195626

> |
```

3. Simulación de lanzamiento de una moneda

Programamos la Simulación de lanzamiento de una moneda. Como norma general, se suele pensar que la probabilidad de salir cara o cruz en una moneda es del 50 %. Sin embargo, para jugar un poco con las variables, introducimos otra nueva opción, que la moneda caiga de canto, que según estudios del matemático **Persi Diaconis** establecen que en torno a 1 de 6000 veces (en tiradas perfectas, sin vibraciones ni viento) saldrá de canto. Bien es cierto que existen otros factores que pueden influir, como que la moneda esté en el momento inicial con la cara hacia arriba o no, pero esos factores no se incluirán en nuestro experimento.

Enlace curioso en el que el lanzamiento de una moneda salió de canto. Pulsar [Aquí](#)

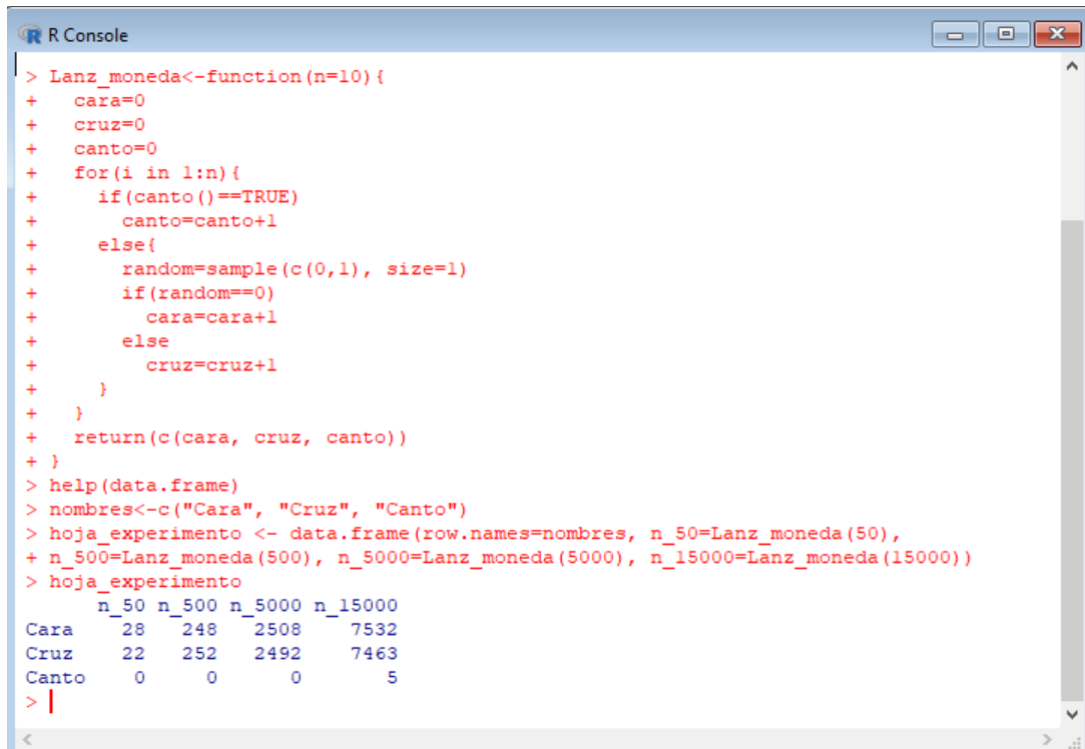
Creamos una pequeña función que calcula la probabilidad de 1/6000
Generamos un número aleatorio, si ese número es 1 devolvemos TRUE, realmente el valor 1 es indiferente, funcionaría igual sustituyendo el 1 por cualquier valor entre el 1 y 6000.

```
canto<-function(){
  v<-sample(6000,1)
  if(v==1)
    return(TRUE)
  return(FALSE)
}
```

Añadimos el código del lanzamiento de moneda. En el cual primero se comprobará si la moneda ha salido de canto y , en caso contrario, asignará la misma probabilidad a salir cara o cruz, llamando a la función `sample`.

```
Lanz_moneda<-function(n=10){
  cara=0
  cruz=0
  canto=0
  for(i in 1:n){
    if(canto()==TRUE)
      canto=canto+1
    else{
      random=sample(c(0,1), size=1)
      if(random==0)
        cara=cara+1
      else
        cruz=cruz+1
    }
  }
  return(c(cara, cruz, canto))
}
```

Ahora, llamamos a la función anterior con diferentes valores de n . Con los resultados podemos realizar infinidad de experimentos. Creamos, por ejemplo, un `data.frame` que mostramos a continuación:



```

> Lanz_moneda<-function(n=10){
+   cara=0
+   cruz=0
+   canto=0
+   for(i in 1:n){
+     if(canto()==TRUE)
+       canto=canto+1
+     else{
+       random=sample(c(0,1), size=1)
+       if(random==0)
+         cara=cara+1
+       else
+         cruz=cruz+1
+     }
+   }
+   return(c(cara, cruz, canto))
+ }
> help(data.frame)
> nombres<-c("Cara", "Cruz", "Canto")
> hoja_experimento <- data.frame(row.names=nombres, n_50=Lanz_moneda(50),
+ n_500=Lanz_moneda(500), n_5000=Lanz_moneda(5000), n_15000=Lanz_moneda(15000))
> hoja_experimento
      n_50 n_500 n_5000 n_15000
Cara    28   248   2508   7532
Cruz    22   252   2492   7463
Canto    0    0      0        5

```

Hemos asignado los nombres de las filas en la llamada a la función y, el resto de parámetros, son llamadas a la función `Lanz_moneda` para así crear una columna por cada llamada. Vemos que para $n=50$ la moneda no ha caído de canto ninguna vez, mientras que para $n=15000$ sí. Juguemos un poco más con este experimento...

Probemos ahora el gráfico *barplot*:

```

barplot(Lanz_moneda(5), names.arg = c("cara","cruz","canto"),
        main="Lanzamiendo de 5 monedas",
        ylab = "Numero de apariciones",
        col = c("royalblue", "red","seagreen"))

```

Realizaremos 4 llamadas a la función anterior, aumentando el valor de las tiradas y comprobando que a mayor valor, la proporciones cara-cruz se igualarán.

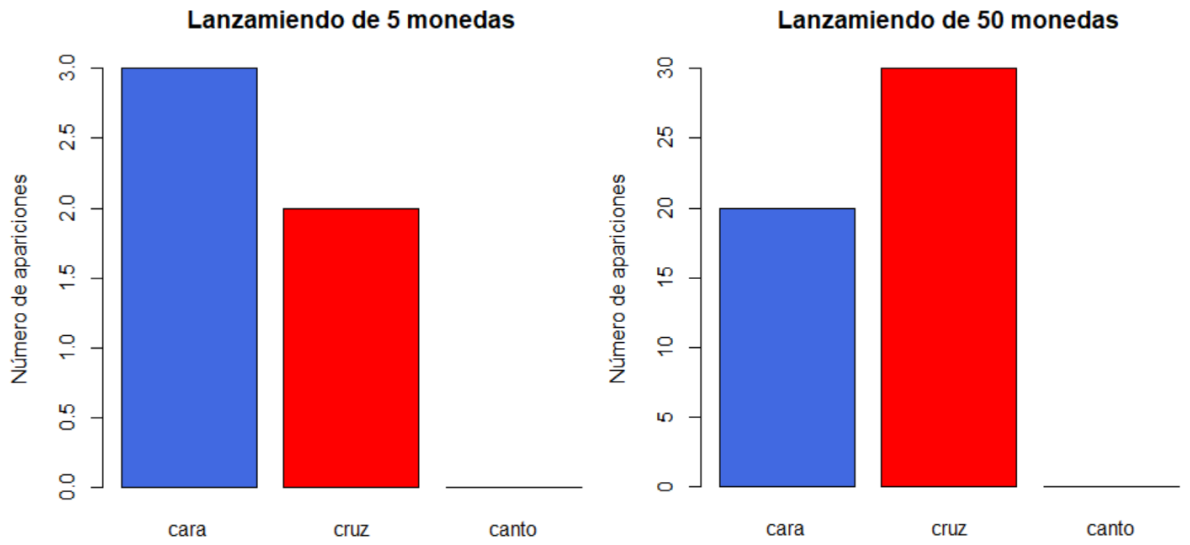


Figura 3.1: Lanzamiento de 5 y 50 monedas

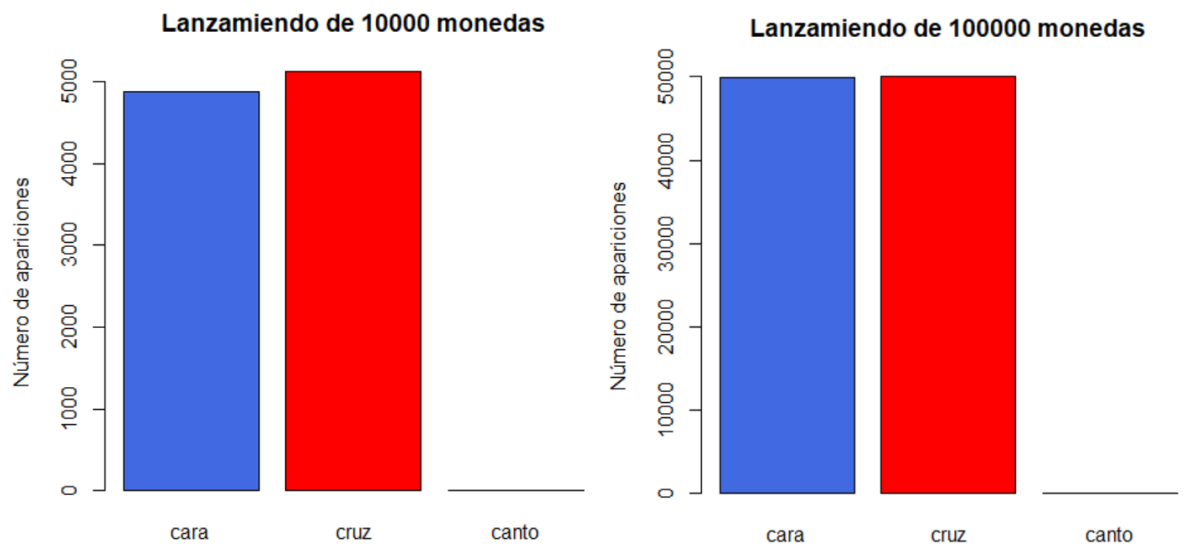


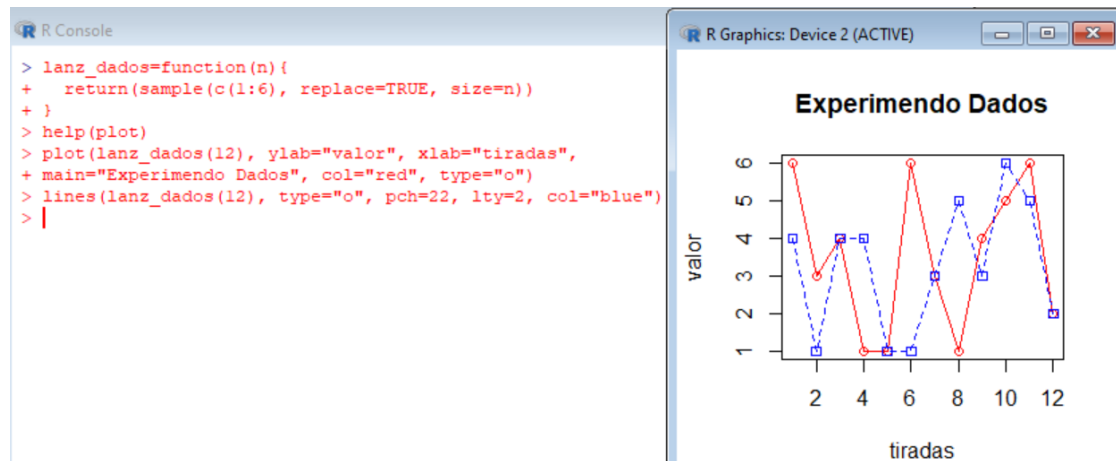
Figura 3.2: Lanzamiento de 10.000 y 100.000 monedas

4. Simulación del lanzamiento de dos dados

Definimos a continuación una sencilla función para simular el lanzamiento de n dados. Claramente los dados funcionan independientemente, es decir, con reemplazamiento. Invocamos la función `help(sample)` y nos damos cuenta que tiene un parámetro indicando si se desea realizar la extracción con reemplazamiento (por defecto estaba en FALSE).

```
lanz_dados=function(n){  
  return(sample(c(1:6), replace=TRUE, size=6))  
}
```

Ahora bien, probaremos el gráfico que nos proporciona `plot`



Con la ayuda de este gráfico podemos comparar diferentes distribuciones e incluso estudiar si tienen correlación ya que es muy visual. Mostramos en el gráfico anterior los resultados tras repetir el experimento de tirar 12 dados. En la línea roja continua se representa la primera tirada mientras que en la discontinua azul, la segunda.

Añadimos una función que nos muestra el número de experimentos que se han realizado para obtener el valor m tras n tiradas, donde m es la suma de los valores obtenidos en las n tiradas.

```
exp_lanz=function(n,m){  
  contador=0  
  repeat{  
    contador=contador+1  
    if (sum(lanz_dados(n))==m)  
      break  
  }  
  cat("Necesaria ",contador," tiradas para obtener ",m," con ",n," dados")  
  return(contador)  
}
```

Llamamos a la función anterior un total de 6 veces y calculamos cuántas tiradas se necesitan para obtener la suma máxima, es decir, que se obtenga un 6 en cada dado tirado.

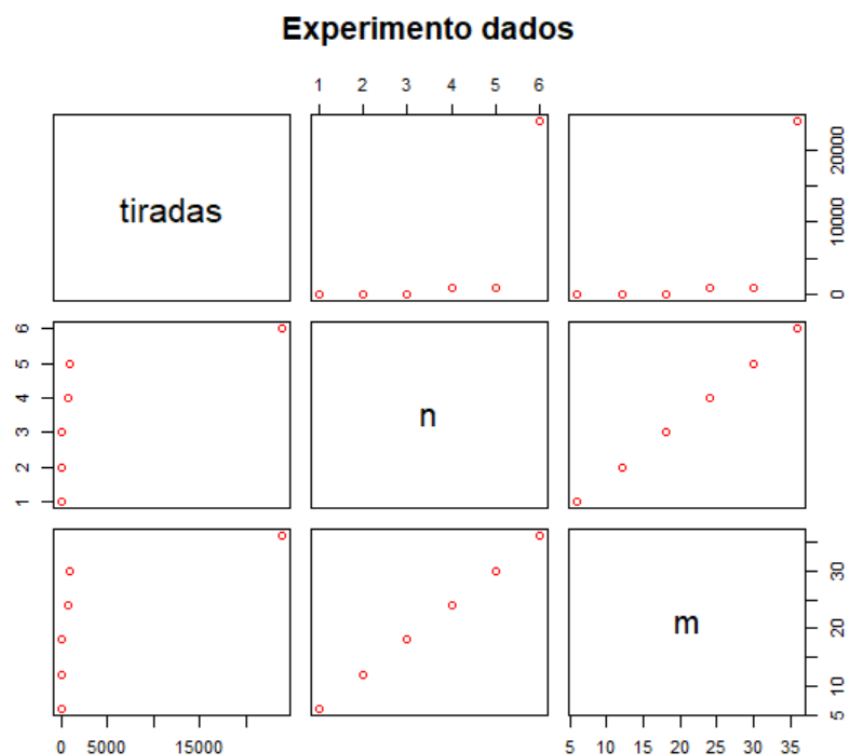
```

R Console
> tiradas<-vector(length=6)
> for(i in 1:6){
+   tiradas[i]<-exp_lanz(i,i*6)
+ }
Se necesitan 1 tiradas para obtener 6 con 1 dados
Se necesitan 40 tiradas para obtener 12 con 2 dados
Se necesitan 56 tiradas para obtener 18 con 3 dados
Se necesitan 796 tiradas para obtener 24 con 4 dados
Se necesitan 913 tiradas para obtener 30 con 5 dados
Se necesitan 23896 tiradas para obtener 36 con 6 dados
> n<-c(1:6)
> m<-n*6
> hoja<-data.frame(tiradas,n,m)
>
> hoja
  tiradas n  m
1      1 1  6
2     40 2 12
3     56 3 18
4    796 4 24
5     913 5 30
6   23896 6 36

```

Creamos un data.frame con los resultados obtenidos y, apartir de este, llamamos a la función *plot*.

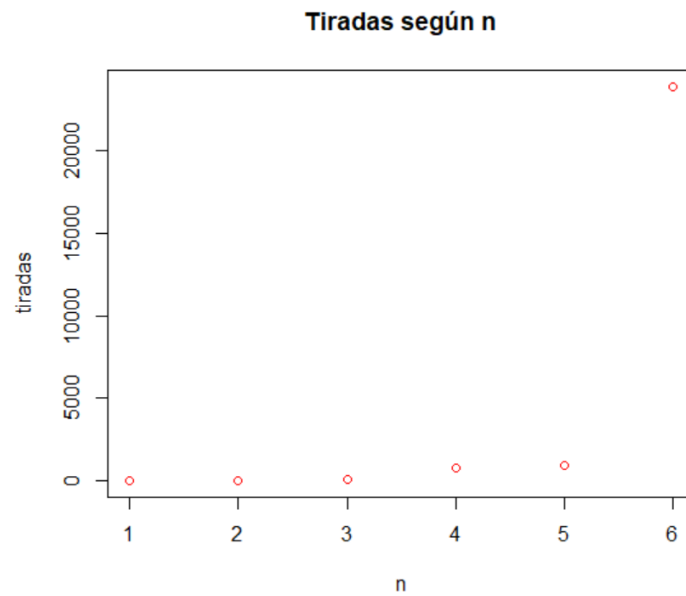
```
plot(hoja , main="Experimento dados", col="red")
```



Los resultados pueden parecer algo liosos, pero podemos llamar a la función *plot* de la siguiente forma para comprender mejor la representación:

```
plot(hoja[2:1], main="Tiradas segun n", col="red")
```

En el eje X se representa el valor **n** (número de dados) mientras que en el eje Y se representa el número de tiradas necesarias para obtener el máximo valor en esos **n** dados.

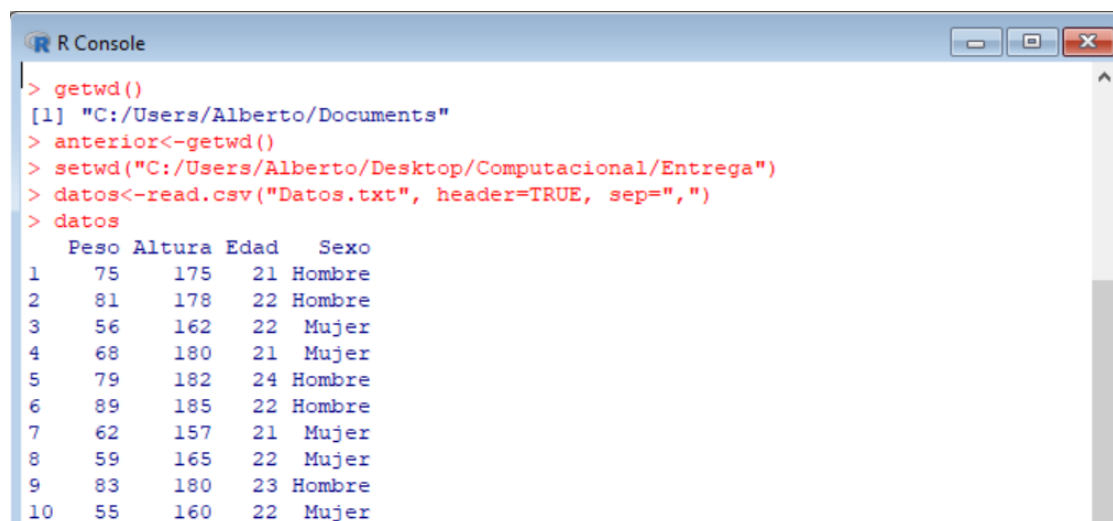


5. Estudio de Datos.txt

Abrimos el fichero de datos *Datos.txt*, teniendo en cuenta que dicho archivo tiene que estar en el directorio de trabajo actual. Si no es así, usamos el comando *setwd* para cambiarlo. Una vez hecho esto, usamos la función *read.csv* para abrirlo, pasándole como parámetro *header=TRUE* (cabeceras) y como separador una coma.

```
setwd("C:/Users/Alberto/Desktop/Computacional/Entrega")
datos<-read.csv("Datos.txt", header=TRUE, sep=",")
```

Guardamos los datos en una variable llamada *datos* que podemos llamar para mostrar su contenido:



```
> getwd()
[1] "C:/Users/Alberto/Documents"
> anterior<-getwd()
> setwd("C:/Users/Alberto/Desktop/Computacional/Entrega")
> datos<-read.csv("Datos.txt", header=TRUE, sep=",")
> datos
  Peso Altura Edad  Sexo
1   75    175   21 Hombre
2   81    178   22 Hombre
3   56    162   22  Mujer
4   68    180   21  Mujer
5   79    182   24 Hombre
6   89    185   22 Hombre
7   62    157   21  Mujer
8   59    165   22  Mujer
9   83    180   23 Hombre
10  55    160   22  Mujer
```

Aunque en el apartado anterior ya se introdujo la función *plot*, destacamos su potencial. Podemos consultar sus parámetros con la ayuda *help*. Podemos realizar distintas consultas a nuestro dataframe:

- Realizar una selección de filas: Seleccionamos las tres primeras filas de nuestro data.frame, que como vemos a continuación, la forma de hacerlo no es única.

```
> datos<-read.csv("Datos.txt", header=TRUE, sep=",")
> head(datos, n=3)
  Peso Altura Edad  Sexo
1   75    175   21 Hombre
2   81    178   22 Hombre
3   56    162   22  Mujer
> datos[1:3,]
  Peso Altura Edad  Sexo
1   75    175   21 Hombre
2   81    178   22 Hombre
3   56    162   22  Mujer
> |
```

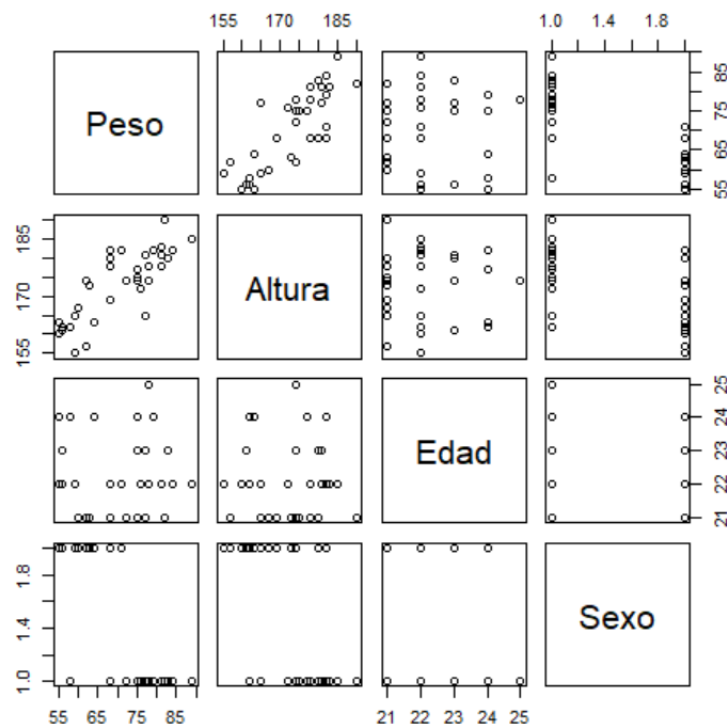
- Condición lógica con 1 variable. Pedimos que se muestren los datos que se correspondan con Sexo=Mujer:

```
> datos[datos$Sexo=="Mujer",]
  Peso Altura Edad  Sexo
3    56    162   22 Mujer
4    68    180   21 Mujer
7    62    157   21 Mujer
8    59    165   22 Mujer
10   55    160   22 Mujer
12   56    161   23 Mujer
14   64    163   24 Mujer
16   68    169   21 Mujer
19   62    157   21 Mujer
21   71    182   22 Mujer
22   55    163   24 Mujer
27   60    167   21 Mujer
29   68    182   22 Mujer
32   59    155   22 Mujer
33   63    173   21 Mujer
35   62    174   21 Mujer
```

- Condición lógica con 2 variables. Intersecamos la condición anterior con aquellas mujeres cuya altura sea mayor a 175 cm:

```
> datos[datos$Sexo=="Mujer"&datos$Altura>175,]
  Peso Altura Edad  Sexo
4    68    180   21 Mujer
21   71    182   22 Mujer
29   68    182   22 Mujer
```

Volviendo a nuestro fichero datos original, invocamos la función *plot()* para y visualizamos el gráfico resultante:

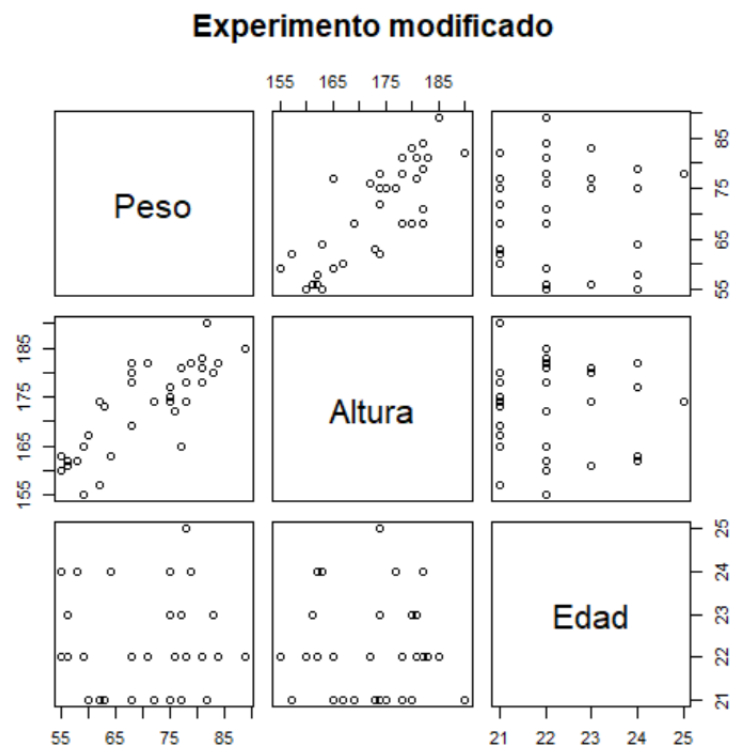


Si nos fijamos, vemos que la variable 'Sexo' no es numérica y no tiene mucho sentido su representación, luego decidimos borrarla y mostrar de nuevo su diagrama modificado.

Para ello, añadimos el libro *dplyr* y usamos su función *select*. Cabe destacar que esta forma no es única pero elegimos esta forma para practicar el uso de nuevos libros.

```
library(dplyr)
datos_mod<-select(datos, Peso, Altura, Edad)
```

```
> datos<-read.csv("Datos.txt", header=TRUE, sep=",")
> plot(datos)
> library(dplyr)
> datos_mod <-select(datos, Peso, Altura, Edad)
> plot(datos_mod, main="Experimento modificado")
```

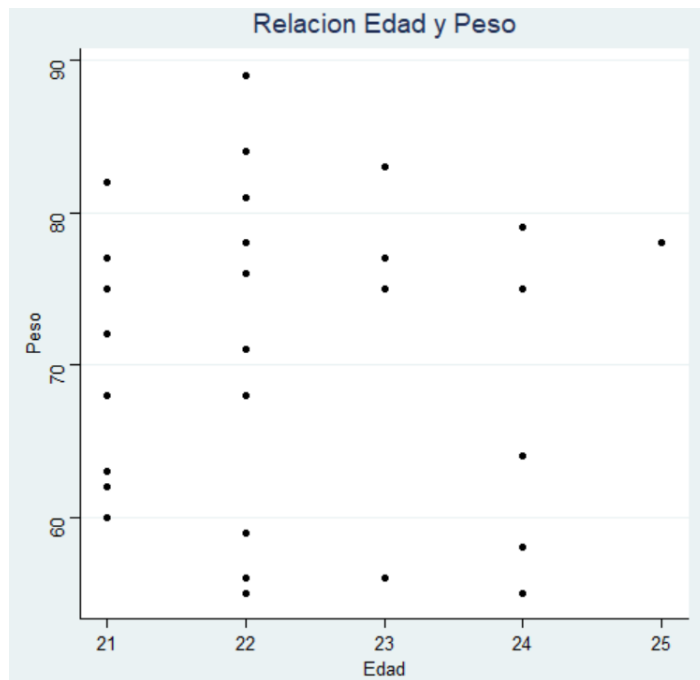


Estudiamos ahora la correlación de las variables Peso y Edad del archivo en cuestión. Para ello, necesitamos añadir el libro *ggplot2* y *ggthemes*. Este último no es obligatorio ya que lo usamos para poner un tema de fondo. En lo que respecta a *ggplot2*, usamos la función *ggplot*.

```
install.packages("ggthemes")
library(ggthemes)
library(ggplot2)
```

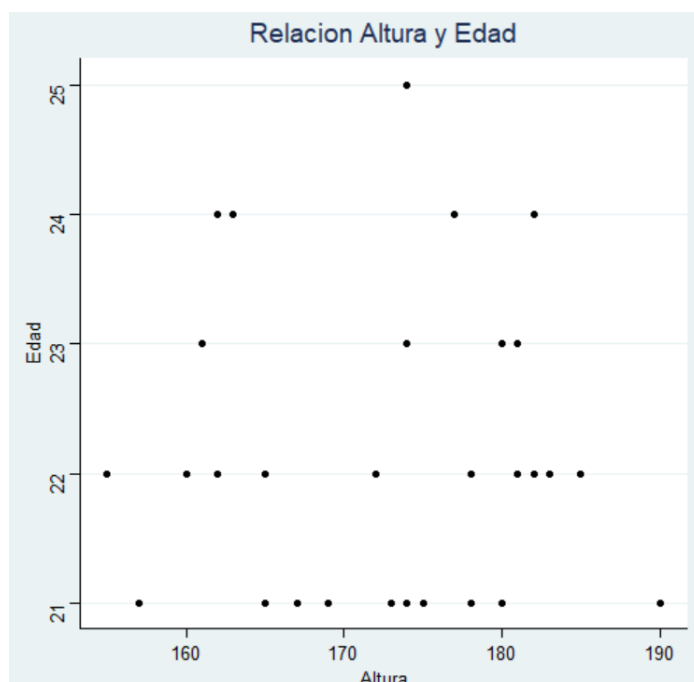
Mostramos primero la relación entre Edad y Peso de nuestro dataset que, como vemos, no tienen relación. Nuestro dataset corresponde a un grupo de gente joven (entre 21 y 25 años de edad) y un peso entre 50 y 90 Kg.

```
ggplot(data=datos , aes(x=Edad, y=Peso))
+geom_point()+theme_stata()+ggtitle("Relacion Edad y Peso")
```



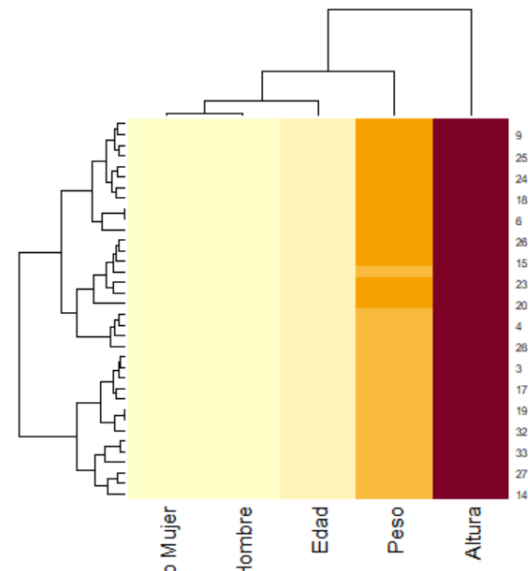
Mostramos ahora la relación entre Edad y Altura (la altura de los datos se encuentra entre 150 y 190 cm)

```
ggplot(data=datos , aes(x=Altura , y=Edad))+
geom_point()+theme_stata()+ggtitle("Relacion Altura y Edad")
```



Por último, nos interesa mostrar el mapa de calor o *heatmap* de nuestros datos. Tenemos el problema de que nuestros datos deben ser todos numérico, sin embargo, tenemos la columna 'Sexo' cuyos valores (Hombre, Mujer) son nominales. Usamos por ello la función *dummies*.

```
install.packages("dummies")
library(dummies)
datos.new<-dummy.data.frame(datos, sep=" ")
heatmap(as.matrix(datos.new[,1:5]))
```

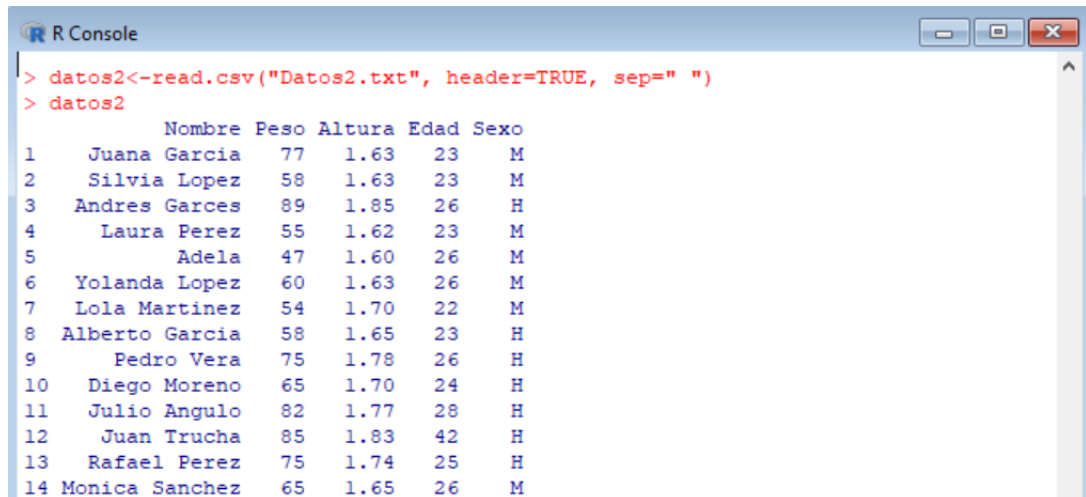


El mapa quizás no nos aporte mucha información. Realmente habría que normalizar los datos ya que las diferentes variables no se mueven en la misma distancia. Por ejemplo los valores de Altura se mueven en torno a 150-190 mientras que la edad en torno a 21-25.

6. Estudio de Datos2.txt

Estudiamos el archivo *Datos2.txt*. Lo abrimos con la función *read.csv*.

```
datos2<-read.csv("Datos2.txt", header=TRUE, sep=" ")
```



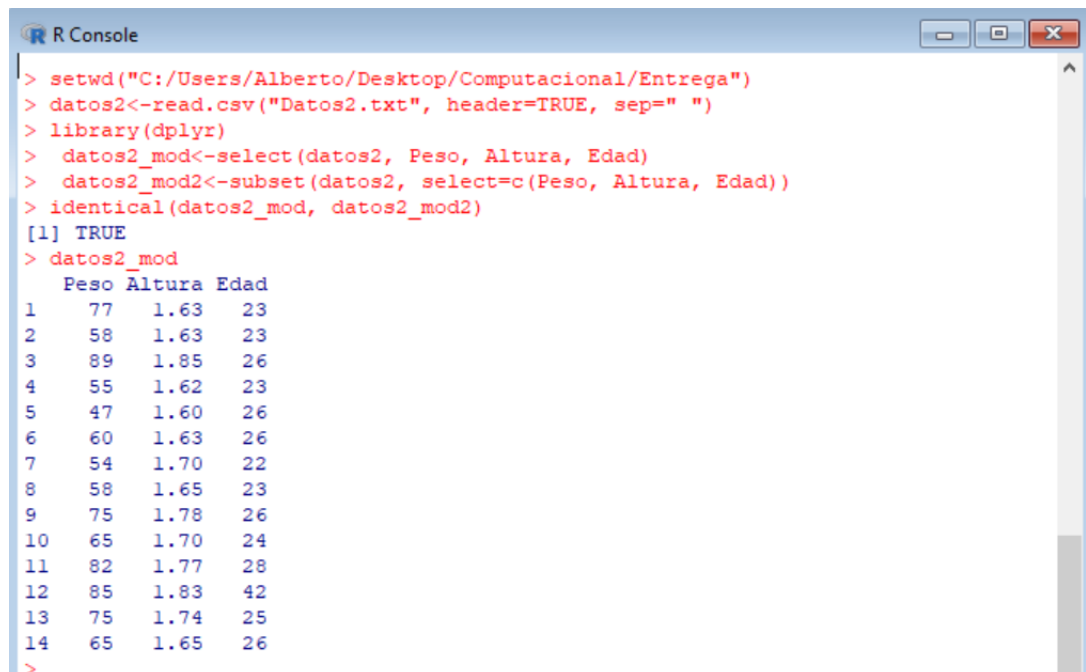
The screenshot shows an R Console window with the following code and output:

```
> datos2<-read.csv("Datos2.txt", header=TRUE, sep=" ")
> datos2
```

	Nombre	Peso	Altura	Edad	Sexo
1	Juana Garcia	77	1.63	23	M
2	Silvia Lopez	58	1.63	23	M
3	Andres Garces	89	1.85	26	H
4	Laura Perez	55	1.62	23	M
5	Adela	47	1.60	26	M
6	Yolanda Lopez	60	1.63	26	M
7	Lola Martinez	54	1.70	22	M
8	Alberto Garcia	58	1.65	23	H
9	Pedro Vera	75	1.78	26	H
10	Diego Moreno	65	1.70	24	H
11	Julio Angulo	82	1.77	28	H
12	Juan Trucha	85	1.83	42	H
13	Rafael Perez	75	1.74	25	H
14	Monica Sanchez	65	1.65	26	M

Imaginemos que queremos estudiar las variables numéricas. Tenemos el problema de que esta vez tenemos una columna con los nombres y otra con el Sexo, luego decidimos quitar estas variables.

En el apartado anterior vimos la opción de usar la función *select*, del paquete *dplyr*, sin embargo, podemos usar *subset* para obtener el mismo resultado.



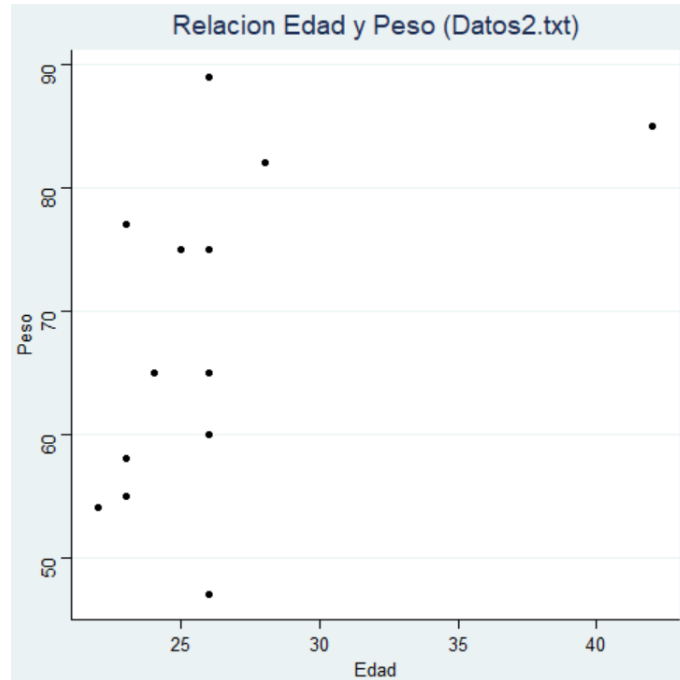
The screenshot shows an R Console window with the following code and output:

```
> setwd("C:/Users/Alberto/Desktop/Computacional/Entrega")
> datos2<-read.csv("Datos2.txt", header=TRUE, sep=" ")
> library(dplyr)
> datos2_mod<-select(datos2, Peso, Altura, Edad)
> datos2_mod2<-subset(datos2, select=c(Peso, Altura, Edad))
> identical(datos2_mod, datos2_mod2)
[1] TRUE
> datos2_mod
```

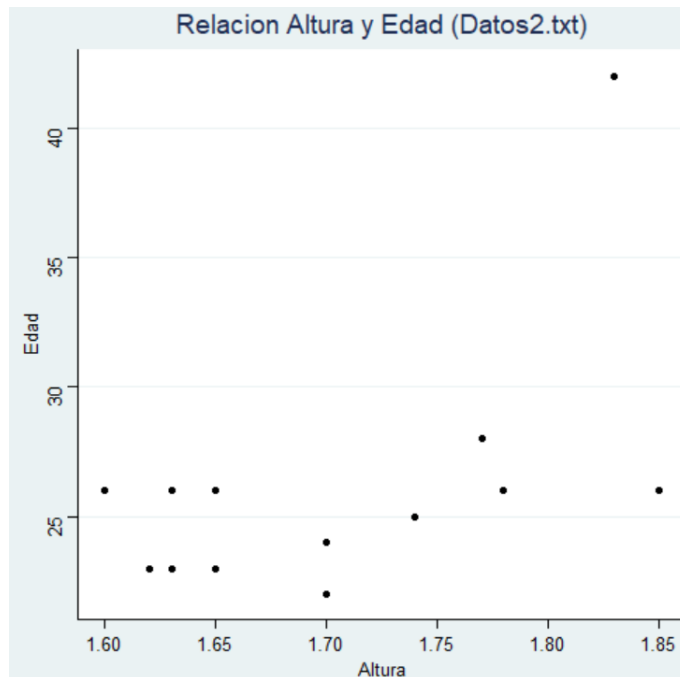
	Peso	Altura	Edad
1	77	1.63	23
2	58	1.63	23
3	89	1.85	26
4	55	1.62	23
5	47	1.60	26
6	60	1.63	26
7	54	1.70	22
8	58	1.65	23
9	75	1.78	26
10	65	1.70	24
11	82	1.77	28
12	85	1.83	42
13	75	1.74	25
14	65	1.65	26

Volvemos a llamar a estas funciones pero para el caso de *Datos2.txt*, teniendo en cuenta que no hay que invocar a la llamada del libro *ggplot2* ya que se había realizado antes.

```
ggplot(data=datos2, aes(x=Edad, y=Peso))+geom_point()+
theme_stata()+ggtitle("Relacion Edad y Peso (Datos2.txt)")
```



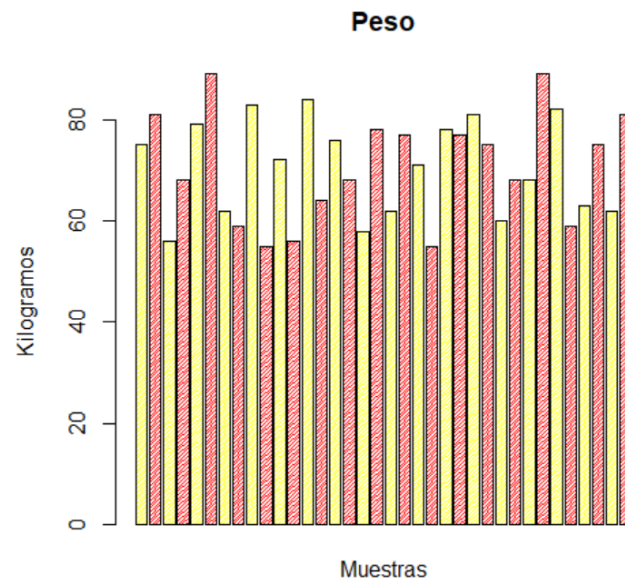
```
ggplot(data=datos2, aes(x=Altura, y=Edad))+geom_point()+
theme_stata()+ggtitle("Relacion Altura y Edad (Datos2.txt)")
```



Vemos que se trata de un dataset de personas de entre 50 y 90 Kg, en torno a 20-30 años de edad (a excepción de un outlier de unos 45 años) y altura variadas, desde 1,60 a 1,85 metros.

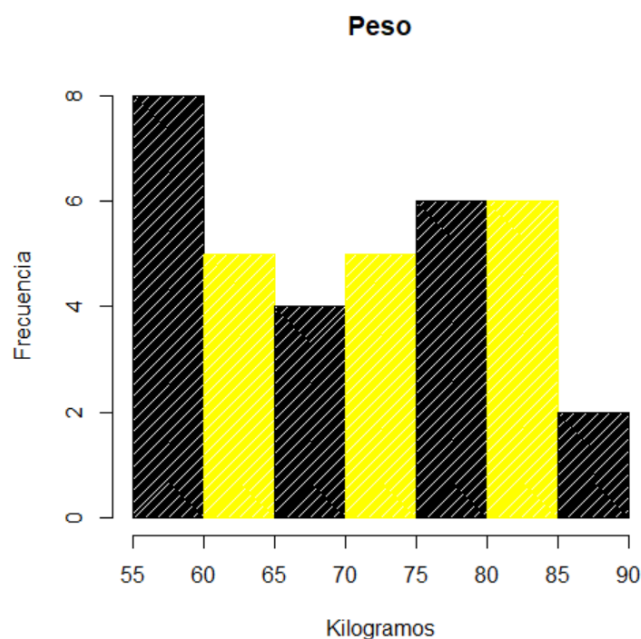
Realizamos otros tipo de gráficos. Mostramos a continuación un diagrama de barras del archivo. Recordamos que tanto para este gráfico como para el histograma de abajo se ha hecho uso de la función *help*, donde hemos podido informarnos acerca de todos sus parámetros y así hacer el gráfico más visual.

```
barplot(datos$Peso , main="Peso", col=c("yellow","red"),
names.arg=c("Muestras"),density=50, border="black", ylab="Kilogramos")
```



Presentamos un problema de estética en este primer diagrama de barras ya que debido al gran número de instancias, se crean demasiadas barras y quizás es conveniente usar otro tipo de gráfico. Por ello, introducimos los histogramas, que para este tipo de situaciones resulta bastante más visual:

```
hist(datos$Peso , main="Peso", col=c("black","yellow"),
xlab="Kilogramos", ylab="Frecuencia", density=120)
```



Referencias

- Guia de Programación en R
- Documentación oficial, función **Help()**