Machine Learning para Economistas

Estructuras de control y Funciones en R

2020–UTDT

Loops (ciclos) "for" y "while"

En varias ocasiones vamos a necesitar replicar una tarea o conjunto de instrucciones pre-determinadas, y para ello vamos a utilizar estos *loops* o *bucles*. Al bucle `for` lo utilizamos cuando queremos realizar operaciones una cantidad determinada de veces y esto lo convierte en una herramienta *ideal* para tareas de *simulación*. Tiene la siguiente estructura general:

```
for(i in 1:n){
    # Aquí dentro colocas las tareas que quieres que R ejecute para cada valor
    # que tomará el índice "i" hasta llegar finalmente a 'i=n'
    }
```

Por ejemplo (un tanto absurdo):

```
for(i in 1:5){print(i)}
```

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

No tiene porque ser una secuencia continua de valores las que tome el índice i, por ejemplo:

```
for (i in c(2010,2011,2012,2013,2014,2015)){
  print(paste("Este año es ", i))
}
```

```
## [1] "Este año es  2010"
## [1] "Este año es  2011"
## [1] "Este año es  2012"
## [1] "Este año es  2013"
## [1] "Este año es  2014"
## [1] "Este año es  2015"
```

```
for(i in c(1,3,7,10)) {
   cat("i = ",i, " --> 2^i = ", 2^i, "\n", sep = "")
}
```

```
## i = 1 --> 2^i = 2
## i = 3 --> 2^i = 8
## i = 7 --> 2^i = 128
## i = 10 --> 2^i = 1024
```

Ilustración y tarea en clase: Vamos a ver como utilizar el comando "for" para obtener los primeros 48 números de la siguiente sucesión: $1, 1, 2, 3, 5, 8, ...$ (sucesión de Fibonacci):

```
Fibonacci = c() # Creamos un vector 'vacío' de dimensión arbitraria.
Fibonacci[1] = 1; # Asignamos al primer elemento del vector el valor 1.
Fibonacci[2] = 1; # Asignamos al segundo elemento del vector el valor 1.

for(i in 3:48){Fibonacci[i]= Fibonacci[i-1]+Fibonacci[i-2]}
# Para i desde 3 hasta 48, el elemento que ocupa la posición i es igual
# a  la suma de los DOS elementos inmediatamente anteriores en dicho vector.
print(Fibonacci)
```

```
##  [1]          1          1          2          3          5          8
##  [7]         13         21         34         55         89        144
## [13]        233        377        610        987       1597       2584
## [19]       4181       6765      10946      17711      28657      46368
## [25]      75025     121393     196418     317811     514229     832040
## [31]    1346269    2178309    3524578    5702887    9227465   14930352
## [37]   24157817   39088169   63245986  102334155  165580141  267914296
## [43]  433494437  701408733 1134903170 1836311903 2971215073 4807526976
```

A continuación vas a leer los datos que aparecen en este tutorial, donde tienes 3 covariables (llamadas x_1, x_2 y x_3) y una variable de respuesta (y), y ejecutando las siguiente línea de comandos reflexiones sobre el output de algunas funciones desconocidas:

```
load('....coloca aquí la ruta hasta el archivo\datos.RData')
dim(datos)

str(datos)     # ¿para qué sirve este comando?

head(datos, 3) # ¿para qué sirve este comando?

tail(datos, 3) # ¿para qué sirve este comando?

pairs(datos)   # ¿para qué sirve este comando?


reg = lm(datos[,1] ~ datos[,2], data = datos)
summary(reg)  # ¿para qué sirve la función "lm"?

reg$coefficients # ¿que información extraemos del modelo?
```

Tarea en clase: Utiliza el bucle for para que R se encargue de correr 3 regresiones, utilizando para ello cada uno de los regresores a la vez, y guardá en un vector las pendientes de cada uno de los modelos. ¿Cuál de los 3 modelos tiene la mayor pendiente? (utiliza el comando `which()` para responder a esta pregunta).

Los bucles *for()* se pueden utilizar también dentro de otros bucles *for()* (incluso dentro de funciones creadas por vos). El bucle *for()* también se puede utilizar con todos los objetos de R (vectores, matrices, data-frames o listas). Así por ejemplo lo podemos emplear para crear una matriz que contiene en la posición de la fila i y columna j el número $i + j$:

```
M = matrix(0,ncol=3,nrow=3)
for(i in 1:3){
  for(j in 1:3){  M[i,j]= i+j }    }
print(M)
```

```
##      [,1] [,2] [,3]
```

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

Ejercicios:

1- Utiliza el loop *for()* para crear un vector que contenga el producto acumulado de los números del 1 al 10.

2- Utiliza el loop *for()* para crear una matriz de dimensión 5×5 que contenga en la posición de la fila i columna j el número $2^{(i+j)}$. Es esta matriz simétrica?

3- Estudia numéricamente el comportamiento de la serie $S(n) = \sum_{i=1}^{n} \sin\left(\frac{1}{i}\right)$ utilizando el bucle `for` para distintos valores de n. ¿La serie diverge o converge?

El loop while():

En este curso no vamos a hacer uso de este bucle; sin embargo lo mencionamos por ser bastante parecido al *for()*, la única diferencia reside en que las tareas se repetirán hasta que se cumple una determinada condición lógica (por eso su nombre). Su estructura general se escribe como:

```
while('condición lógica'){
    #Lista de las tareas que se ejecutan hasta que se deje de verificar la condición
    }
```

Veamos un ejemplo simple:

```
i = 1
while(i<5){
             print(i)
             i = i + 1
          }
```

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

IMPORTANTE: Ojo cuando ejecutes estos loops ya que puedes dejar al R corriendo un programa hasta el infinito! (por ejemplo si no actualizas el valor de i en el loop del ejemplo anterior). Si esto te llega a pasar alguna vez no desesperes: utiliza la combinación de teclas *'Ctrl + C'* (presiona las dos al mismo tiempo) y podrás parar la ejecución.

Condicionales: if else y ifelse

Usamos el `if` para pedirle a R que verifique una condición lógica antes de realizar alguna tarea. IMPORTANTE: Para evitar errores en el código es importante que te acostumbres a escribir correctamente esta instrucción. Para ello debes respetar la estructura de R que es formalmente la siguiente:

```
if(...) {
   # Instrucción en caso de cumplirse la condición
  } else {
   #Instrucción en caso de NO cumplirse la condición
}
```

es de importancia que en la misma línea de código coloques *} else {*, si no lo haces así R no comprenderá la instrucción y dará error. Prueba lo siguiente en casa:

```
p = 2; q = 3
if(p < q) {
    print('p es menor que q')
          } else {
        print('p es mayor que q')
          }
```

```
## [1] "p es menor que q"
```

Los condicionales se pueden también aplicar sobre vectores o matrices, para ello hay que usar un comando ligeramente diferente:

ifelse('condiciones para cada elemento', 'que hacer si se cumple', 'que hacer si no se cumple')

Supongamos que queremos hacer 0 todos los elementos de un vector x o matriz M que sean negativos o que sean mayores que 10:

```
x = c(-1,2,8,21,-2,11,9,-1,7)

# ifelse tiene que evaluar cada elemento del vector x y si el
# elemento es negativo o mayor que 10 lo transforma
# en cero, caso contrario lo mantiene como aparece en x:

ifelse (x < 0 | x > 10,0,x)
```

```
## [1] 0 2 8 0 0 0 9 0 7
```

```
# Observa que podrías haber programado el condicional de otra manera:

ifelse (x > 0 & x <= 10,x,0)
```

```
## [1] 0 2 8 0 0 0 9 0 7
```

```
# ... obviamente ambos resultado coinciden.

# Veamos el mismo ejemplo, pero ahora con una matriz:
M = matrix(c(-1,2,8,21,-2,11,9,-1,7),ncol=3)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]   -1   21    9
## [2,]    2   -2   -1
## [3,]    8   11    7
```

```
ifelse (M < 0 | M > 10,0,M)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    9
## [2,]    2    0    0
## [3,]    8    0    7
```

```
ifelse (M > 0 & M <= 10,M,0)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    9
## [2,]    2    0    0
## [3,]    8    0    7
```

Programando nuestras propias Funciones en R

Una *función* en R es simplemente un pequeño programa en el que una vez introducido sus argumentos (imputs) genera un conjunto de outputs. La estructura general que utilizamos para definir una función en R es la siguiente:

```
nombre_fun <- function(arg1, arg2, ...) {
    #Conjunto de instrucciones que pueden involucran cálculos, loops y/o condicionales
  return('objeto que prentes que devuelva la función')
                                        }
```

Una vez escribiste tu función, R necesita leerla **UNA VEZ** para guardarla en su memoria y hasta que no cierres el programa la puedes utilizar/llamar cuantas veces quieras sin que la tengas que volver a leer nuevamente. Veamos por ejemplo como programar la función *cubo()* que tiene por argumento un número y devuelve el cubo de dicho número:

```
cubo <- function(x) { return(x^3) } # El argumento es 'x' y simplemente computa x^3
# Una vez que pasamos la función por la consola (una vez que esta en la memoria de R)
# la podemos utilizar cuantas veces queramos sin necesidad de volver a ejecutar la linea anterior.

cubo(2)
```

```
## [1] 8
```

```
cubo(15)
```

```
## [1] 3375
```

```
cubo(pi)
```

```
## [1] 31.00628
```

Las funciones de R pueden contener dentro otras funciones (siempre que estén bien definidas). Por ejemplo al intentar crear una función para calcular la **media-geométrica** definida como:

$$\bar{x}_{\text{geom}} = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

```
# Antes de ejecutar estas líneas de código mira en la ayuda de R
# para que sirve la función prod()
media.geom <- function(x) { return( prod(x)^(1/length(x)) ) }
# Nuevamente el vector x es el único argumento de esta función y
# con el mismo realiza las operaciones que se expresan dentro de los corchetes {}.
# Así que para el vector x = goles del ZZZ tendremos
x = c(1,2,5,10,7,25)
media.geom(x)
```

```
## [1] 5.095338
```

Es posible utilizar R como herramienta para evaluar y graficar funciones matemáticas. Para ello necesitamos primero definir la función en cuestión. Imaginemos que nos interesa estudiar el polinomio $p(x) = 2x^2 - 0.9x - 1$ en el intervalo $[-1, 2]$:

```
p <- function(x){return(2*x^2 -0.9*x -1) } # Declaramos el objeto.

# Podemos simplemente evaluar la función:
p(0)    # Imput un nro -> output un nro
```

```
## [1] -1
```

```r
p(1.5)  # Imput un nro -> output un nro
```

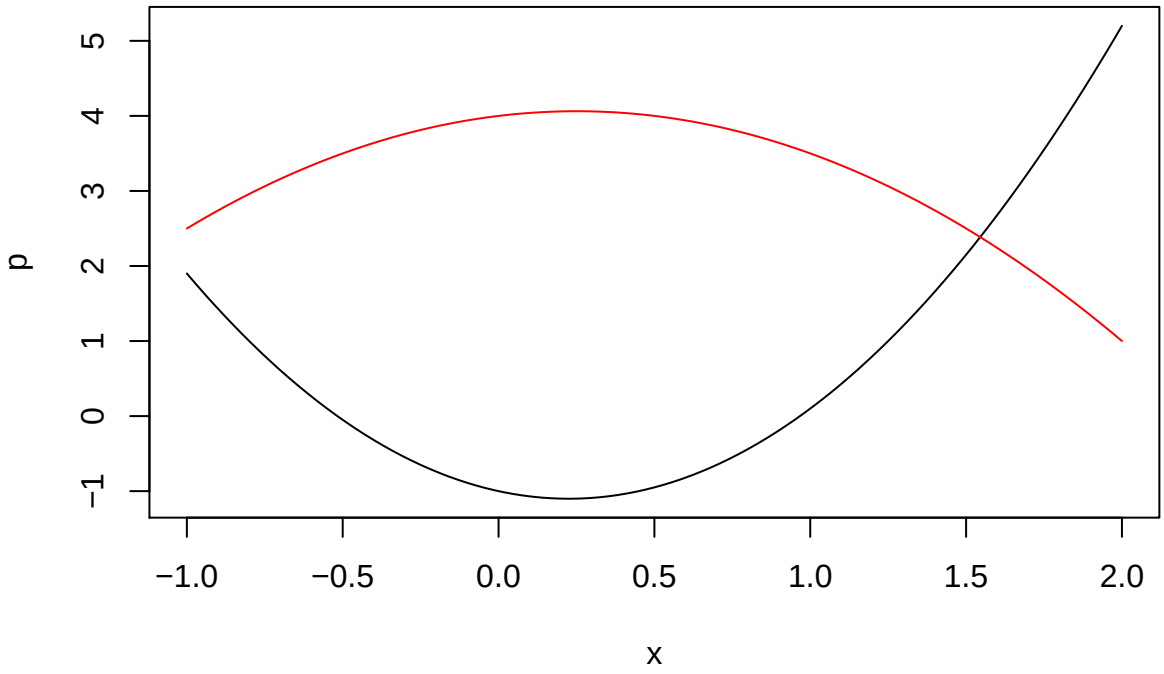
```
## [1] 2.15
```

```r
p(c(-1,0,1.5,2)) # Imput un vector -> output un vector
```

```
## [1]  1.90 -1.00  2.15  5.20
```

```r
# Podemos graficar la función en el intervalo deseado:
plot(p, from=-1, to=2)

# Podemos graficar más de una función en mismo gráfico:
q <- function(x){return(-x^2 + 0.5*x +4) } # Declaramos el objeto.
plot(q, from=-1, to=2, col='red', add=T)
```



Anexo: Estadística descriptiva básica con R

R tiene una gran interfaz gráfica, puedes mirar una breve reseña tipeando *demo('graphics')* en tu consola. El objetivo de este apéndice es que te familiarices con los comandos básicos para resumir y graficar la información.

Tablas de frecuencia:

```
# Imagina que tiramos 25 veces un dado, y estos fueron los resultados:
dados<-c(1,2,5,3,6,4,2,1,2,4,1,5,3,2,4,1,6,2,3,1,6,2,4,2,1)


Fa = table(dados)
Fa
```

```
## dados
## 1 2 3 4 5 6
## 6 7 3 4 2 3
```

```
# Calculamos la frecuencia absoluta ACUMULADA de cada valor de la variable
SFa = cumsum(table(dados))
SFa
```

```
##  1  2  3  4  5  6
##  6 13 16 20 22 25
```

```
# Calculamos la frecuencia relativa de cada valor de la variable
fr = table(dados)/length(dados)
fr
```

```
## dados
##    1    2    3    4    5    6
## 0.24 0.28 0.12 0.16 0.08 0.12
```

```
# Calculamos la frecuencia absoluta ACUMULADA de cada valor de la variable
Sfr = cumsum(table(dados)/length(dados))
Sfr
```

```
##    1    2    3    4    5    6
## 0.24 0.52 0.64 0.80 0.88 1.00
```

```
# Ahora todo junto en una sola matriz/tabla:
Tabla.Freq = cbind(Fa,SFa,fr,Sfr)
print(Tabla.Freq)
```

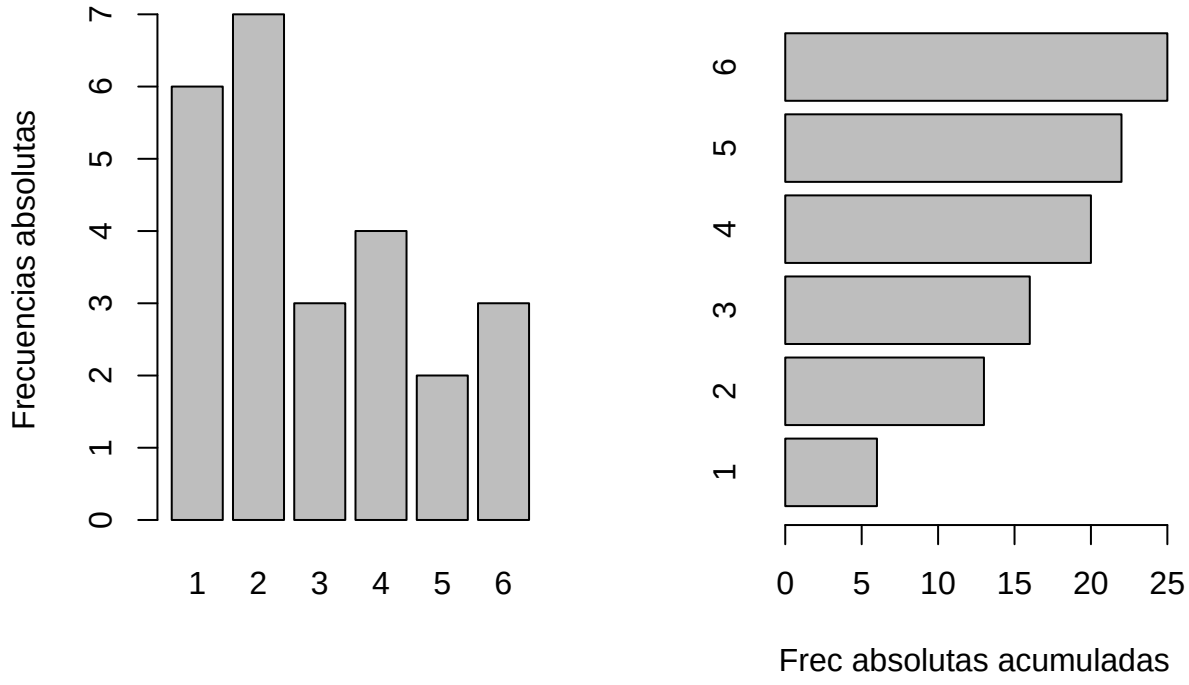
```
##   Fa SFa   fr  Sfr
## 1  6   6 0.24 0.24
## 2  7  13 0.28 0.52
## 3  3  16 0.12 0.64
## 4  4  20 0.16 0.80
## 5  2  22 0.08 0.88
## 6  3  25 0.12 1.00
```

Distribuciones empíricas

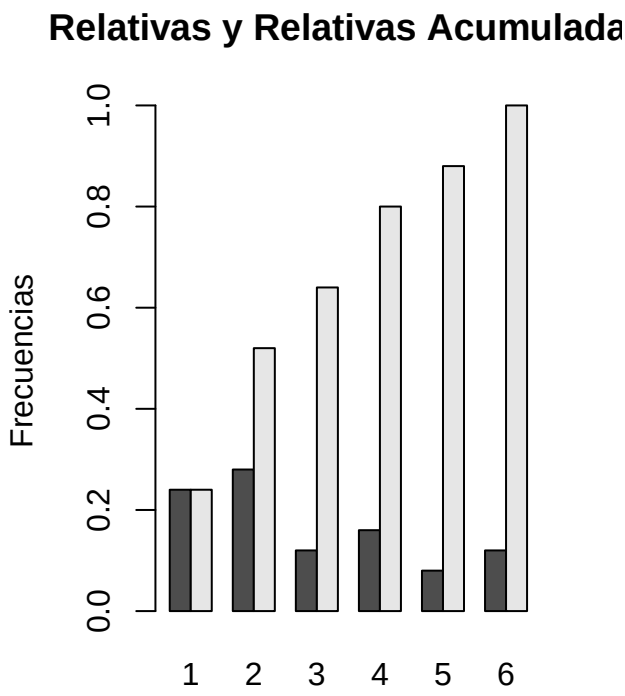
Cuando disponemos de la información de una muestra resumida en una tabla de frecuencia, podemos hacer resúmenes gráficos de la misma para facilitar la lectura y la comprensión de los datos. Dependiendo de que tipo de variables estemos analizando podríamos utilizar un tipo u otro de gráficas. Cuando se trata de una

variable cuantitativa (discreta), lo habitual es utilizar un diagrama de barras para describir la *distribución empírica* de los datos en la muestra. Veamos como hacer esto con los datos del ejemplo anterior:

```
par(mfrow=c(1,2))
barplot(Fa,ylab="Frecuencias absolutas",main=" ")
barplot(SFa,xlab="Frec absolutas acumuladas",main="", horiz = T)
```




```
# También podemos graficar ambas frecuencias en una sola gráfica:
barplot(rbind(fr,Sfr),ylab="Frecuencias",main="Relativas y Relativas Acumuladas",beside = T)
```

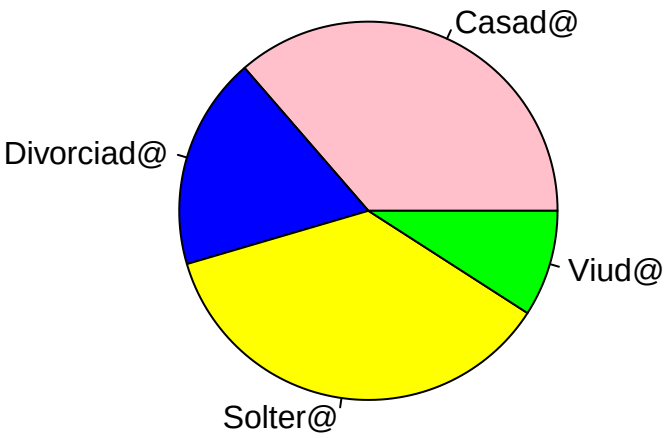


Cuando se trata de variables cualitativas (ejemplo: genero, estado civil, lugar de nacimiento, etc) también podemos computar tablas de frecuencias y representar gráficamente la información de las misma a través de diagramas de barras y/o tarta (pie-charts).

```
sexo = c('H','H','M','H','H','M','H','M','H','M','H')
ecivil = c('Solt','Solt','Solt','Div','Cas','Cas','Viud','Cas','Solt','Div','Cas')

# Gráfico de barras para el sexo
barplot(table(sexo), ylab = "Frecuencia", main = 'Género de las personas en la muestra')
```


```
# pie chart
pie(table(ecivil),labels=c('Casad@','Divorciad@','Solter@','Viud@'),
col = c('pink','blue','yellow','green'),
main ="Distribución de las personas en la muestra por Estado Civil" )
```

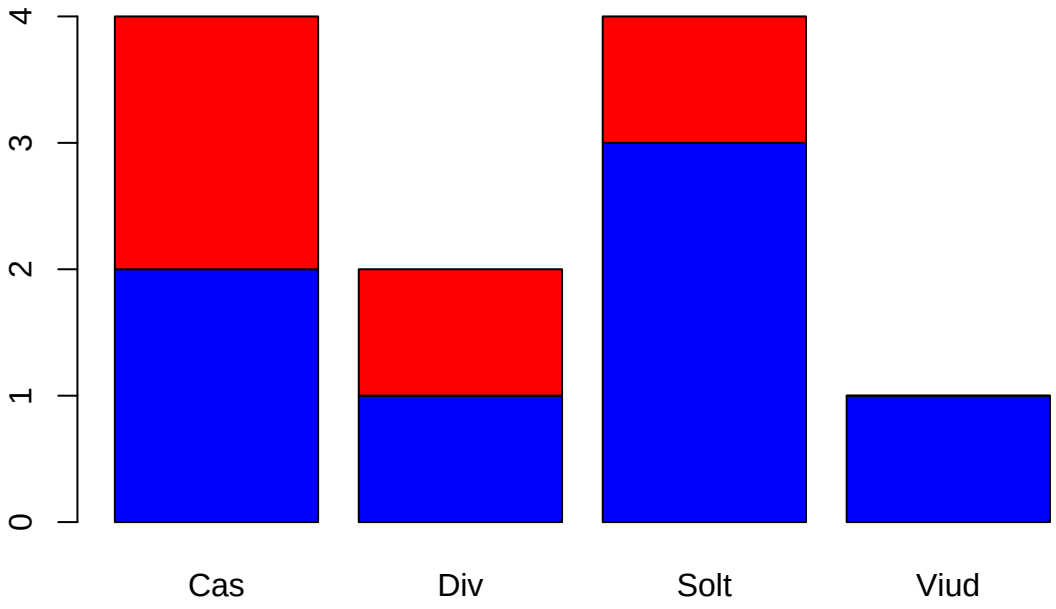


Cuando trabajamos con variables cualitativas y/o discretas de a pares, también resulta interesante mirar las distribuciones conjuntas (si queremos combinar o analizar variables continuas las solemos discretizar por intervalos para poder meterlas en una tabla de frecuencias). Para ello podemos utilizar tablas de *doble-entrada* como sigue en R:

```
table(sexo,ecivil)
```

```
##     ecivil
## sexo Cas Div Solt Viud
##    H   2   1    3    1
##    M   2   1    1    0
```

```
barplot(table(sexo,ecivil),col=c('blue','red'))
```



Cuando trabajamos con una variable continua, por ejemplo el tiempo trabajado en la última semana a los individuos encuestados en el ejercicio 1, necesitaremos discretizamos la/s variable/s haciendo intervalos. En R usamos el comando *cuts()*, que nos permite hacer dichos intervalos de manera rápida. Veamos un ejemplo:

```
# Cargamos los datos (en este ejemplo inventados)
horas_trabajadas = c(40, 35.5, 46, 44.5 , 45, 26, 47.5, 35, 36.5, 46, 47.5)

# Calculamos la frecuencia absoluta de cada valor de la variable
Fa = table(cut(horas_trabajadas,breaks=3)) # Breaks controla el número de cortes.
Fa
```

```
##
##   (26,33.2] (33.2,40.3] (40.3,47.5]
##           1           4           6
```

```
# Calculamos la frecuencia absoluta ACUMULADA de cada valor de la variable
SFa = cumsum(Fa)
SFa
```

```
##   (26,33.2] (33.2,40.3] (40.3,47.5]
##           1           5          11
```

```
# Calculamos la frecuencia relativa de cada valor de la variable
fr = table(cut(precip,breaks=3))/length(precip)
fr
```

```
##
## (6.94,27]   (27,47] (47,67.1]
## 0.2428571 0.6000000 0.1571429
```

```r
# Calculamos la frecuencia absoluta ACUMULADA de cada valor de la variable
Sfr = cumsum(fr)
Sfr
```

```
## (6.94,27]   (27,47] (47,67.1]
## 0.2428571 0.8428571 1.0000000
```

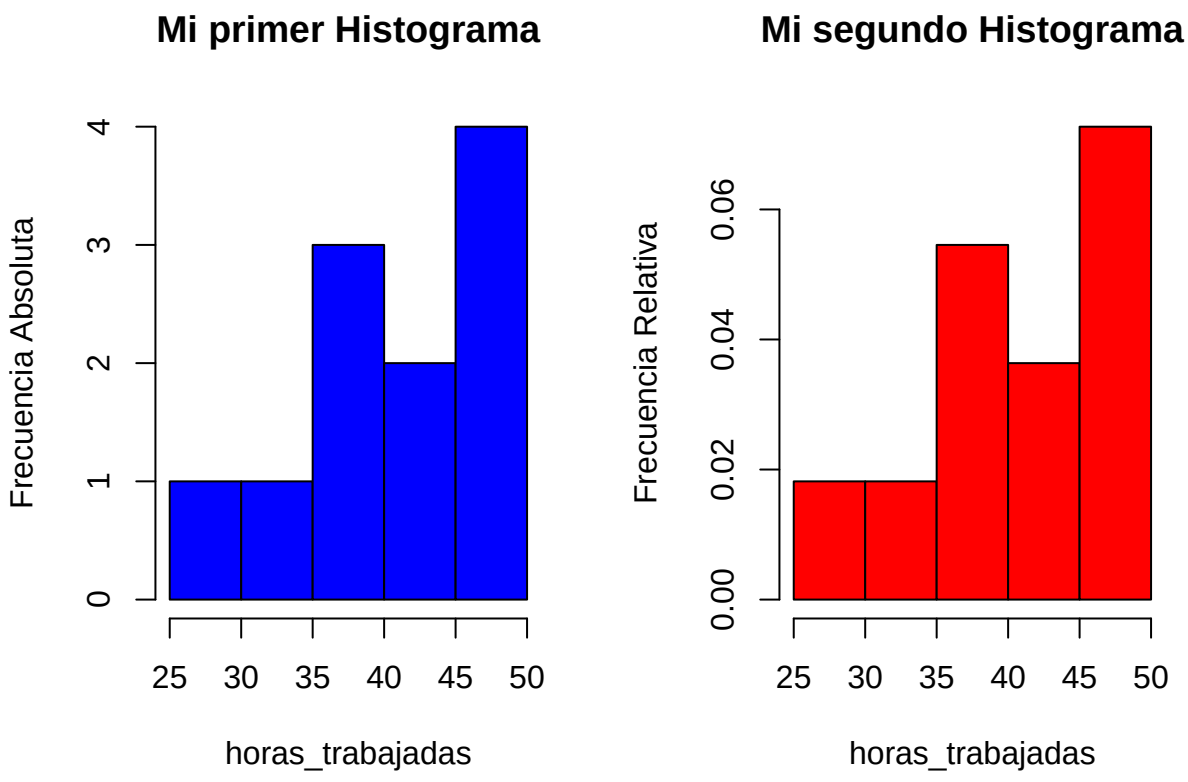
```r
# Ahora todo junto en una sola matriz/tabla:
Tabla.Freq = cbind(Fa,SFa,fr,Sfr)
print(Tabla.Freq)
```

```
##             Fa SFa        fr       Sfr
## (26,33.2]    1   1 0.2428571 0.2428571
## (33.2,40.3]  4   5 0.6000000 0.8428571
## (40.3,47.5]  6  11 0.1571429 1.0000000
```

La manera habitual de representar la información de una tabla con una variable continua es a través de un histograma.

```r
par(mfrow=c(1,2))
hist(horas_trabajadas,breaks = 5, col= 'blue', main='Mi primer Histograma',
     probability = F,ylab = 'Frecuencia Absoluta')

hist(horas_trabajadas,breaks = 5, col= 'red', main='Mi segundo Histograma',
     probability = T,ylab = 'Frecuencia Relativa')
```



Por último, en muchos software estadísticos se empleó durante mucho tiempo el gráfico de *tallo y hoja* (hoy en día este sería un dispositivo gráfico *vintage*). En R podemos generar una gráfica de este tipo con el comando

stem:

```
stem(horas_trabajadas)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   2 | 6
##   3 |
##   3 | 567
##   4 | 0
##   4 | 556688
```

Medidas de posición, dispersión y simetría (variables cuantitativas)

Con frecuencia resulta conveniente resumir la información de una muestra con pocos números. Una medida de *posición* es un número que caracteriza el centro de la distribución (empírica) de los datos (también se denomina medida o parámetro de tendencia central o de centralización). Entre las medidas de tendencia central más utilizadas en la práctica tenemos: Media (aritmética, ponderada, geométrica, armónica), la mediana (que resulta más robusta que la anterior) y la moda (es la única medida de posición que podríamos calcular para datos cualitativos).

R es un programa estadístico y por ello resulta relativamente simple computar estas medidas sobre un conjunto de datos. Veamos algunos ejemplos utilizando los datos de horas trabajadas en la encuesta de 11 personas.

```
# Medidas centrales:
mean(horas_trabajadas) # Media aritmética (o media a secas)
```

```
## [1] 40.86364
```

```
mean(horas_trabajadas,weights = rep(1/11,11) ) # Media ponderada (pesos =)
```

```
## [1] 40.86364
```

```
median(horas_trabajadas) # Mediana
```

```
## [1] 44.5
```

```
# R no tiene una función que nos permita calcular la moda.
# Tendremos que calcular manualmente la moda:

Fa = table(cut(horas_trabajadas,breaks=5))
Fa # Entonces, cual es la moda?
```

```
##
##   (26,30.3] (30.3,34.6] (34.6,38.9] (38.9,43.2] (43.2,47.5]
##           1           0           3           1           6
```

Cuando los datos que analizamos sean variables categóricas, entonces solo podremos computar la *moda* (ya que no se pueden sacar promedios ni ordenar los datos no numéricos).

Las medidas de *dispersión* muestran la variabilidad de una distribución (empírica en este caso), indicando por medio de un número, si las diferentes puntuaciones de una variable están muy alejadas del centro de la misma. Cuanto mayor sea ese valor, mayor será la variabilidad (heterogeneidad de los datos), cuanto menor sea, más homogéneos resultarán los elementos de la muestra. Entre las medidas de dispersión más utilizadas en la práctica tenemos: Varianza (y desviación standard), el rango (rango inter-cuartílico) y los cuantiles empíricos de la distribución. Veamos como calcularlos en R:

```r
# Varianza y desviación standard:
var(horas_trabajadas)
```

```
## [1] 47.50455
```

```r
sd(horas_trabajadas) # raíz cuadrada de la varianza
```

```
## [1] 6.892354
```

```r
# Rango
range(horas_trabajadas) # = max(horas_trabajadas) - min(horas_trabajadas)
```

```
## [1] 26.0 47.5
```

```r
# Cuantiles empiricos:
quantile(horas_trabajadas, 0.10)
```

```
## 10%
##  35
```

```r
quantile(horas_trabajadas, 0.20)
```

```
##  20%
## 35.5
```

```r
quantile(horas_trabajadas, 0.90)
```

```
##  90%
## 47.5
```

```r
quantile(horas_trabajadas, 0.99)
```

```
##  99%
## 47.5
```

```r
# Rango intercuartílico:
IQR(horas_trabajadas) # quantile(horas_trabajadas, 3/4) - quantile(horas_trabajadas, 1/4)
```

```
## [1] 10
```

Como estas medidas (las de centralidad y de dispersión) las vas a computar muchas veces cuando trabajes en R, te va a ser útil el comando:

```r
# Ejecuta y organiza varias de las instrucciones habituales
summary(horas_trabajadas)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   26.00   36.00   44.50   40.86   46.00   47.50
```

Por último, las medidas de simetría son indicadores que permiten establecer el grado de simetría (o asimetría) que presenta una la distribución de los datos de una muestra. Entre las medidas de asimetría más utilizadas en la práctica tenemos:

`Coeficiente de asimetría de Fisher:`

$$A_F = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{s^3}$$

donde $\bar{x}$ es la media muestral de una muestra de tamaño n.

`Coeficiente de asimetría de Pearson`

$$A_P = \frac{\bar{x} - moda}{s}$$

`Coeficiente de asimetría de Bowley`

$$A_B = \frac{q_3 - q_1 - 2Mediana}{q_3 - q_1}$$

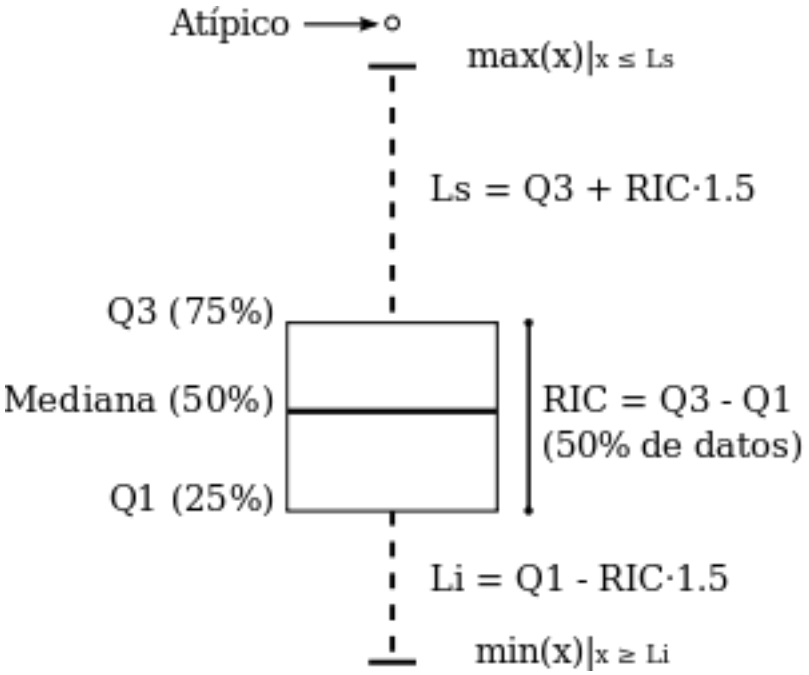
Cuando los coeficientes son positivos, la distribución es asimétrica positiva (o asimétrica hacia la derecha) y al revés en caso de ser negativo. En la medida que los coeficientes se aproximan a cero, entonces decimos que la distribución es simétrica. En R existen algunas librerías que tienen disponibles cada una de estas medidas. Sin embargo resultará más conveniente programar nuestras propias funciones para computarlas. Veamos como ejemplo el coeficiente de Fisher:

```
Af = function(datos){return(sum( (datos - mean(datos))^3 ) / sd(datos)^3 ) }
Af(horas_trabajadas)
```

```
## [1] -8.396023
```

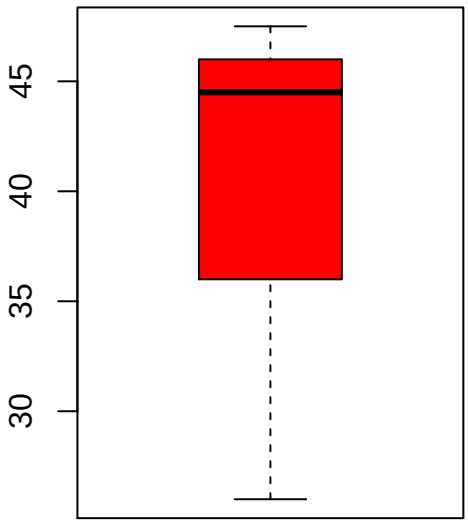
```
# Es simétrica la distribución de las horas trabajadas?
```

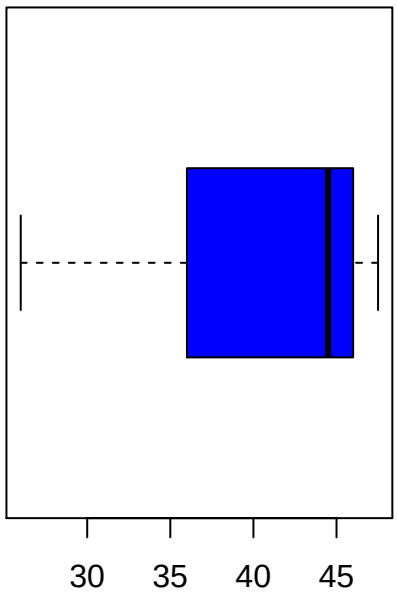
Un gráfico de mucha utilidad para resumir las medidas de posición y la dispersión y simetría en un conjunto de datos es el *box-plot*. Este gráfico se debe interpretar de la siguiente manera:



Veamos que pinta tiene el box-plot de los datos en la muestra de 11 individuos:

```
par(mfrow=c(1,2))
boxplot(horas_trabajadas,col='red',main='Mi primer box-plot en R :)')
boxplot(horas_trabajadas,col='blue',main='Mi segundo box-plot en R :)',horizontal = T)
```


Mi primer box−plot en R :)
45
40
35
30


Mi segundo box−plot en R :)
30
35
40
45