

# Introducción Rápida a R

*Irán Apolinar Peredo Cortes*

*18 de julio de 2019*

## Introducción

A continuación se mostrarán los comando básicos en R necesarios para poder tener un conocimiento básico del uso de las librerías básicas en cursos introductorios de estadística. Estas notas se desarrollarán mostrando el conjunto de código que se introduce en R así como su respectiva salida, por tal motivo no es necesario para su revisión la instalación y el uso de R, sin embargo en el momento de correr programas si será necesario. Se recomienda tenerlo instalado y practicar para que la lectura subsecuente sea fluida. Estas notas así como las bases de datos utilizadas estarán disponibles en el siguiente enlace: <https://github.com/IranNash/>.

En R podemos llamar a librerías con el comando `library()`, así como instalar paquetes con `install.packages()`. Para solicitar información de un comando podemos utilizar `help()`. Podemos introducir comentarios dentro del código anteponiendo un `#` al comentario sin que afecte la ejecución.

**Operaciones Básicas:** definimos  $a = 10$ ,  $b = 20$  y  $c = 30$ .

```
a<-10
b<-20
c<-15
```

La asignación de valores en R se da mediante el simbolo `<-`, tambien puede ser utilizado el simbolo `=`. La suma, resta multiplicación y división utilizan `+`, `-`, `*` y `/` respectivamente. La jerarquía de operaciones puede ser definitida utilizando parentesis.

```
a+b
```

```
## [1] 30
```

```
a-b
```

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

```
a*b
```

```
## [1] 200
```

```
a/b
```

```
## [1] 0.5
```

```
a+((b/c)*(c-a))*a
```

```
## [1] 76.66667
```

Para **imprimir texto** en pantalla utilizamos el comando `print()`, tambien se puede utilizar para mostrar el valor guardado en variables. Podemos concatenar texto con variables con los comandos `paste()` o conatenar texto con varias variables u operaciones con el comando `sprintf()` y `%s` como sigue:

```
print("Hola")
```

```
## [1] "Hola"
```

```
print(a)
```

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

```
print(paste("el dato a vale:", a))
```

```
## [1] "el dato a vale: 10"
```

```
sprintf("la suma de a + b es %s,y la de a-b es %s",a+b, a-b)
```

```
## [1] "la suma de a + b es 30,y la de a-b es -10"
```

Para introducir **datos booleanos**, es decir, que tienen como salida verdadero o falso utilizamos los operadores `<`, `>`, `==`, `>=` o `<=`. También es posible guardar un dato booleano mediante `<-` y luego imprimirlo en pantalla con `print()` como sigue:

```
a == 10
```

```
## [1] TRUE
```

```
b >= 20
```

```
## [1] TRUE
```

```
c < a
```

```
## [1] FALSE
```

```
d <- a == 20
```

```
print(a)
```

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

El **módulo o residuo** de una división se obtiene con el comando `%%`. Para **truncar** a enteros un número se utiliza el comando `trunc()`:

```
a <- 41
```

```
a%%2
```

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

```
trunc(14.98765)
```

```
## [1] 14
```

Cuando se realizan programas es común solicitar la introducción de datos desde el teclado y que se almacene en alguna variable. Para poder introducir datos numéricos en R utilizamos el comando `scan` se debe especificar el número de datos a introducir. Para introducir datos de texto utilizamos junto con `scan` el comando `what` y especificamos que el tipo de datos es un `character` (este ejemplo se recomienda correrlo en su terminal de R).

```
d<-scan(n=1)
```

```
print(d)
```

```
## numeric(0)
```

```
ch<-scan(n=1, what = "character")
```

```
print(ch)
```

```
## character(0)
```

Las condicionales en R utiliza el comando `if(){} y else{}`, se establece la condición entre parentesis `()` y finalmente entre llaves `{}` se introduce lo que debe de realizarse si se cumple la condición. Por ejemplo, supongamos que `a` es igual a 10. Si esto se cumple entonces de imprime el texto `a es igual a 10`. Para este ejemplo utilizamos el operador `==` que se utiliza para *preguntar* si una variable es igual a otra, en este caso, si una variable es igual a un número:

```
a <- 10
```

```
if (a == 10){
```

```
print("a es igual a 10")
}
```

```
## [1] "a es igual a 10"
```

Hagamos ahora un ejemplo más complejo. Supongamos que  $a = 10$  y que  $b = 20$  y definamos dos variables auxiliares  $h = 0$  y  $g = 0$ . Preguntemos ahora si  $a < 5$  de ser verdad, entonces solicitamos que se realice la operación  $a * b$  y que el resultado se guarde en  $h$ . De ser falsa la afirmación entonces queremos que se realice una resta  $a - b$  y que se guarde en  $g$ . Finalmente queremos que muestre en pantalla el valor final de  $h$  y  $g$ . El código para este ejemplo es el siguiente:

```
a<-10
b<-20
h<-0
g<-0
if(a>5){
  h<-a*b
}else{
  g<-a-b
}
print(h)
```

```
## [1] 200
```

```
print(g)
```

```
## [1] 0
```

En R se pueden utilizar tres tipos de **ciclos(bucles o loops)**. Utilizamos el comando `while(){}`  para que un ciclo se repita mientras se cumple una condición. Normalmente este tipo de ciclos utiliza contadores, es decir, variables auxiliares donde especificamos donde empieza a contar el ciclo. Dentro del ciclo generalmente especificamos los incrementos del contador. Finalmente el ciclo se repetirá hasta que el contador cumpla la condición.

Por ejemplo, realicemos un ciclo que permita repetir 8 veces la frase `mi primer ciclo en R`. El código para este ejemplo es como sigue:

```
i <- 1
while(i <= 8){
  print("mi primer ciclo en R")
  i = i +1
}
```

```
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
```

Si leyeramos el código este diría “definimos un contador  $i = 1$ . Mientras el contador  $i$  sea menor o igual a 8” entonces imprime `mi primer ciclo en R`. Ahora suma al contador  $i$  una unidad para que en el siguiente ciclo valga 2” y así sucesivamente.

Realicemos el mismo ejemplo con un ciclo `for(){}` . en este caso definimos dentro de la condición el contador y le decimos el rango que va a recorrer. Finalmente le decimos en que hacer en cada ciclo. el código para realizar el mismo ejercicio es el siguiente:

```
for(i in 1:8){
  print("mi primer ciclo en R")
}
```

```
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
## [1] "mi primer ciclo en R"
```

Como podemos ver el código es más simple. Tenemos un contador `i` que va a recorrer de 1 a 8, es decir, ocho veces se repetirá la instrucción `print("mi primer ciclo en R")` se sabe que son 8 porque se a especificado con el código `in 1:8` para comprobarlo basta con imprimir `i` en pantalla

```
print(i)
```

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

## Vectores y Matrices

Podemos crear un vector con el comando `c()` y asignarlo a una variable. Podemos hacer una secuencia de numeros dentro de un vector utilizando la nomenclatura `c(j:k)` para que el vector tome los valores de `j` hasta `k`. Tambien es posible crear un vector de ceros con el comando `numeric()` especificando el número de ceros, es decir el tamaño del vector:

```
x<-c(1,2,3,4,5)
print(x)
```

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

```
x<-c(5:10)
print(x)
```

```
## [1] 5 6 7 8 9 10
```

```
x<- numeric(5)
print(x)
```

```
## [1] 0 0 0 0 0
```

Para buscar un dato específico dentro del vector utilizamos la nomenclatura `x[i]` donde `i` es la posición del elemento que buscamos. Podemos extraer más de un número con el comando `c(i,j,k)` donde `i`, `j`, `k` nos muestran la posición de los datos a extraer:

```
x<-c(3,4,5,6,7,8,9,10,11)
print(x)
```

```
## [1] 3 4 5 6 7 8 9 10 11
```

```
x[2]
```

```
## [1] 4
```

```
x[c(2,4,6)]
```

```
## [1] 4 6 8
```

Podemos realiza operaciones aritméticas con los elementos de un vector como restarle o sumarle un número a cada elemento, multiplicar cada elemento por un número dado o elevar cada elemento a una cierta potencia:

```
x<-c(1,2,3,4,5,6,7,8,9)
print(x)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
x+5 #resta
```

```
## [1] 6 7 8 9 10 11 12 13 14
```

```
x-3 #suma
```

```
## [1] -2 -1 0 1 2 3 4 5 6
```

```
x*2 #multiplicación
```

```
## [1] 2 4 6 8 10 12 14 16 18
```

```
x^2 #potencia
```

```
## [1] 1 4 9 16 25 36 49 64 81
```

Estas mismas operaciones pueden realizarse entre vectores, por ejemplo:

```
x<-c(1,2,3,4,5,6,7,8,9)
y<-c(3,4,5,6,7,8,9,10,11)
x+y
```

```
## [1] 4 6 8 10 12 14 16 18 20
```

```
x-y
```

```
## [1] -2 -2 -2 -2 -2 -2 -2 -2 -2
```

```
x*y
```

```
## [1] 3 8 15 24 35 48 63 80 99
```

```
x^y
```

```
## [1] 1 16 243 4096 78125 1679616
```

```
## [7] 40353607 1073741824 31381059609
```

Podemos tambien crear vectores de caracteres, por ejemplo, supongamos que queremos introducir en un vector el genero de 9 personas, bastaría con introducir entre comillas " " el caracter correspondiente.

```
sexo <- c("hombre", "mujer", "mujer", "otro", "hombre", "hombre", "mujer", "mujer", "otro")
print(sexo)
```

```
## [1] "hombre" "mujer" "mujer" "otro" "hombre" "hombre" "mujer" "mujer"
```

```
## [9] "otro"
```

En la practica es común preguntarse si la variable es un factor, esto permite crear categorías dentro del vector. Para esto utilizamos el comando `is.factor()` para comprobar si es factor y `as.factor()` para transformar.

```
is.factor(sexo)
```

```
## [1] FALSE
```

```
sexo<-as.factor(sexo)
is.factor(sexo)
```

```
## [1] TRUE
```

Como vemos en la primera salida `is.factor(sexo)` muestra un **FALSE** lo que significa que no es un factor, es decir, el vector no tiene categorías. Posteriormente con el comando `as.factor(sexo)` se vuelve a guardar en `sexo` con lo cual remplazamos la variable original por la variable con factores. Ahora si mandamos a imprimir `sexo` tenemos:

```
print(sexo)
```

```
## [1] hombre mujer  mujer  otro   hombre hombre mujer  mujer  otro
## Levels: hombre mujer otro
```

nos muestra los tres niveles. Esto será importante cuando se trabaje con variables categóricas. Podemos generar una **matriz** a partir de combinación de varios vectores o contruirla directamente. Para el primer caso. Supongamos que creamos un vector con las estaturas de 9 personas, como en el caso anterior:

```
est<-c(1.85, 1.64, 1.56, 1.71, 1.79, 1.81, 1.56, 1.63, 1.69)
print(est)
```

```
## [1] 1.85 1.64 1.56 1.71 1.79 1.81 1.56 1.63 1.69
```

Con el comando `rbind()` podemos combinar dos vectores y ordenarlos como filas. Con el comando `cbind()` hacemos lo mismo pero lo ordenamos como columnas.

```
A<-cbind(sexo, est)
print(A)
```

```
##      sexo est
## [1,]    1 1.85
## [2,]    2 1.64
## [3,]    2 1.56
## [4,]    3 1.71
## [5,]    1 1.79
## [6,]    1 1.81
## [7,]    2 1.56
## [8,]    2 1.63
## [9,]    3 1.69
```

```
B<-rbind(sexo, est)
print(B)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## sexo 1.00 2.00 2.00 3.00 1.00 1.00 2.00 2.00 3.00
## est  1.85 1.64 1.56 1.71 1.79 1.81 1.56 1.63 1.69
```

Como puede observarse la variable `sexo` ha sido remplazada por numero del 1 al 3, esto significa que se ha asignado la categoría correspondiente. Esto es útil en el momento de hacer modelos o gráficos.

Para hacer una matriz directamente utilizamos el comando `matrix()` en el cual establecemos el número de filas `nrow` y el número de columnas `ncol` los datos de la matriz se introducen como si fuera un vector. La dimensión de la matriz se muestra con el comando `dim()`

```
A <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
print(A)
```

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

```
dim(A)
```

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

Las operaciones operaciones elementales con matrices se realizan de la siguiente forma:

```
A <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
B <- matrix(c(10,11,12,13), nrow = 2, ncol = 2)
t(A) #Transpuesta
```

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

```
A*3 #Multiplicación Escalar
```

```
##      [,1] [,2]
## [1,]    3    6
## [2,]    6   12
```

```
A+B #Suma de matrices
```

```
##      [,1] [,2]
## [1,]   11   15
## [2,]   13   17
```

```
A-B #Resta de matrices
```

```
##      [,1] [,2]
## [1,]   -9   -9
## [2,]   -9   -9
```

```
A%*%B #Producto Matricial
```

```
##      [,1] [,2]
## [1,]   43   51
## [2,]   64   76
```

```
solve(A) #Matriz Inversa
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

La matriz identidad se logra con el comando `diag()` donde especificamos el tamaño de la matriz. Por ejemplo una matriz identidad de 5x5 se realiza como:

```
diag(5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
## [5,]    0    0    0    0    1
```

## Dataframe

Un data frame es un arreglo rectangular que permite almacenar distintos tipos de datos, tanto numéricos como de carácter. Una forma sencilla de hacerlo es a partir de vectores, por ejemplo:

```
sexo <- c("hombre", "mujer", "mujer", "otro", "hombre", "hombre", "mujer", "mujer", "otro")
est<-c(1.85, 1.64, 1.56, 1.71, 1.79, 1.81, 1.56, 1.63, 1.69)
edad <- c(18, 20, 23, 25, 22, 18, 17, 19, 20)
```

```
data <- data.frame(sexo, est, edad)
print(data)
```

```
##      sexo  est edad
## 1 hombre 1.85   18
## 2  mujer 1.64   20
## 3  mujer 1.56   23
## 4   otro 1.71   25
## 5 hombre 1.79   22
## 6 hombre 1.81   18
## 7  mujer 1.56   17
## 8  mujer 1.63   19
## 9   otro 1.69   20
```

La estructura de los datos se obtiene con el comando `str()`:

```
str(data)
```

```
## 'data.frame':   9 obs. of  3 variables:
## $ sexo: Factor w/ 3 levels "hombre","mujer",...: 1 2 2 3 1 1 2 2 3
## $ est : num  1.85 1.64 1.56 1.71 1.79 1.81 1.56 1.63 1.69
## $ edad: num  18 20 23 25 22 18 17 19 20
```

Las estadísticas básicas del dataframe se obtiene con el comando `summary()`:

```
summary(data)
```

```
##      sexo      est      edad
## hombre:3  Min.   :1.560  Min.   :17.00
## mujer :4   1st Qu.:1.630  1st Qu.:18.00
## otro  :2   Median :1.690  Median :20.00
##          Mean    :1.693  Mean    :20.22
##          3rd Qu.:1.790  3rd Qu.:22.00
##          Max.    :1.850  Max.    :25.00
```

Podemos guardar los datos en formato `.txt` con el comando `write.table`. Se debe especificar el nombre del archivo así como el tipo de separación de los datos y si tiene o no encabezado entre otras definiciones, como sigue:

```
write.table(data, file = "data.txt", sep = ",", col.names = TRUE)
```

Para poder llamar una base de datos podemos utilizar `read.table()` si es en formato `.txt`. Podemos llamar un `.csv` con `read.csv()`. Recordemos que para tener todas las especificaciones del comando basta con teclear `help("read.csv")` para tener todas las funciones.

```
data<-read.table("data.txt", header = TRUE, sep = ",")
```

```
help("read.csv")
```

Para llamar una variable específica dentro del `data.frame` utilizamos el simbolo `$` después del nombre del `data.frame`.

```
data$sexo
```

```
## [1] hombre mujer  mujer  otro   hombre hombre mujer  mujer  otro
## Levels: hombre mujer otro
```

```
summary(data$est)
```

```
##      Min. 1st Qu.  Median      Mean 3rd Qu.      Max.
```



```
##    1.560    1.630    1.690    1.693    1.790    1.850
```

Finalmente, cuando se trabajan con bases de datos con muchas variables, puede llegar a ser complicado utilizar el simbolo \$ anteponiendolo al nombre de la base para llamar variables. En su lugar podemos utilizar el comando `attach()`

```
attach(data)
```

```
## The following objects are masked _by_ .GlobalEnv:
```

```
##
```

```
##    edad, est, sexo
```

```
summary(est)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
##    1.560    1.630    1.690    1.693    1.790    1.850
```

Para regresar a la forma de llamar los datos originalmente con \$ utilizamos `detach()`

```
detach(data)
```

## Gráficos básicos

En R existen muchas posibilidades de hacer gráficos, podemos utilizar los gráficos nativos de R o utilizar librerías como `ggplot2`. Para esta introducción rápida vamos a mostrar los gráficos nativos de R.

El gráfico de dispersión puede realizarse simplemente dando dos variables con la función `plot()`, por ejemplo:

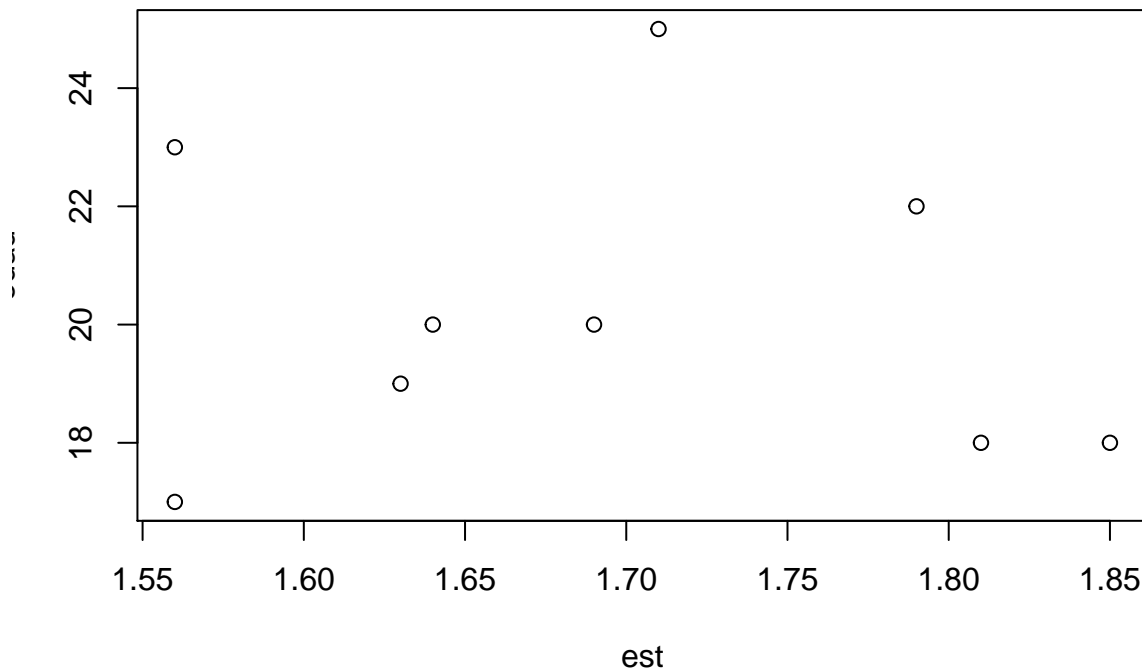
```
attach(data)
```

```
## The following objects are masked _by_ .GlobalEnv:
```

```
##
```

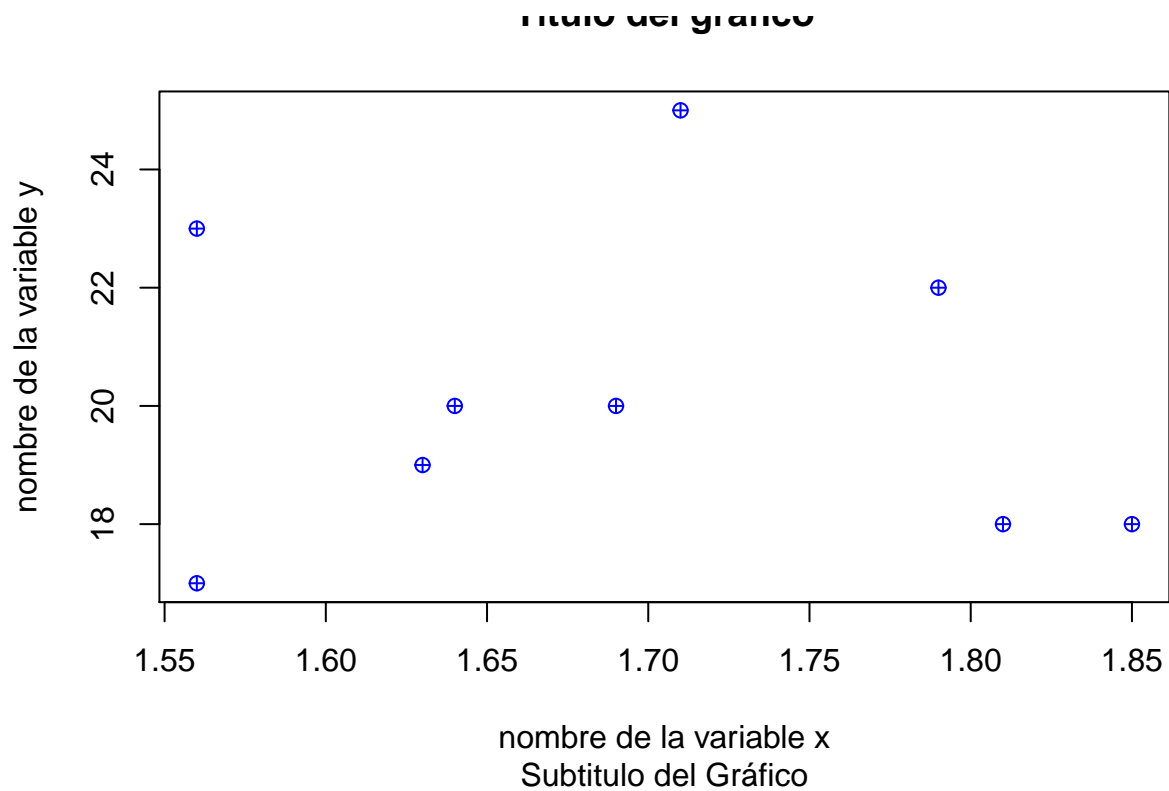
```
##    edad, est, sexo
```

```
plot(est, edad)
```

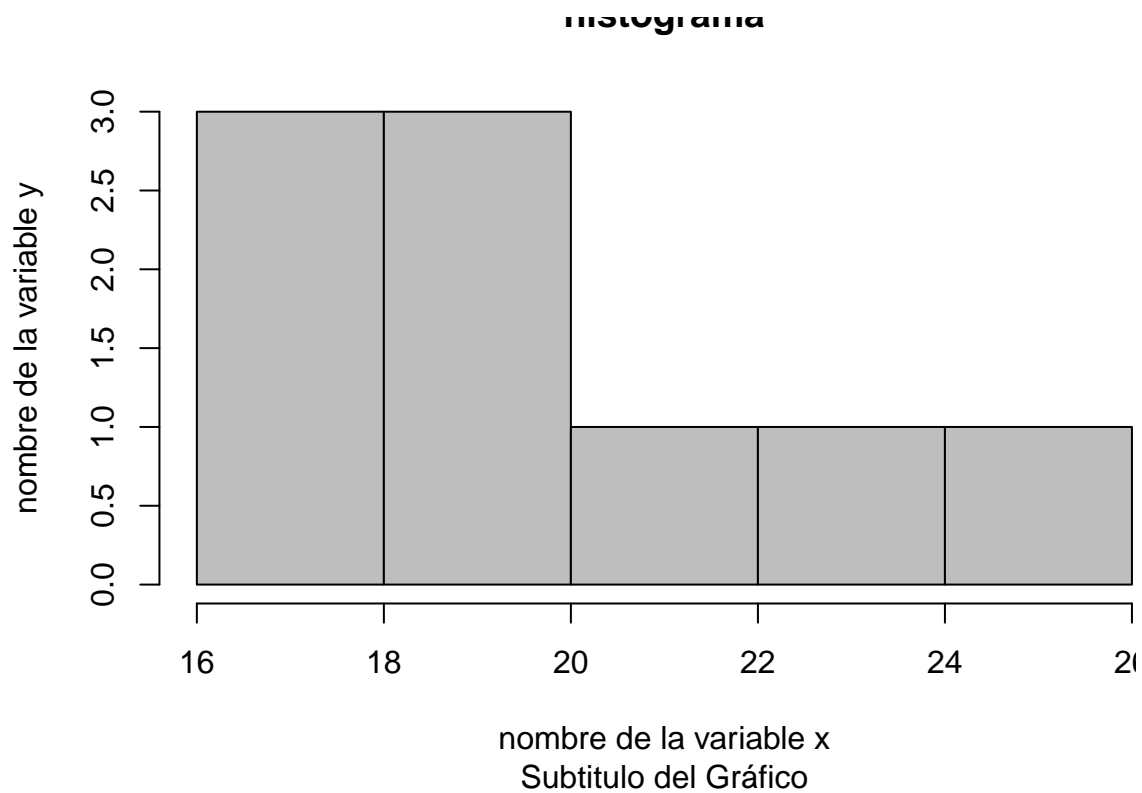


Podemos darle formato a los gráficos nativos en R de la siguiente forma:

```
plot(est, edad,
     type = "p",
     pch = 10,
     col = "blue",
     main = "Título del gráfico",
     sub= "Subtitulo del Gráfico",
     xlab = "nombre de la variable x",
     ylab = "nombre de la variable y")
```

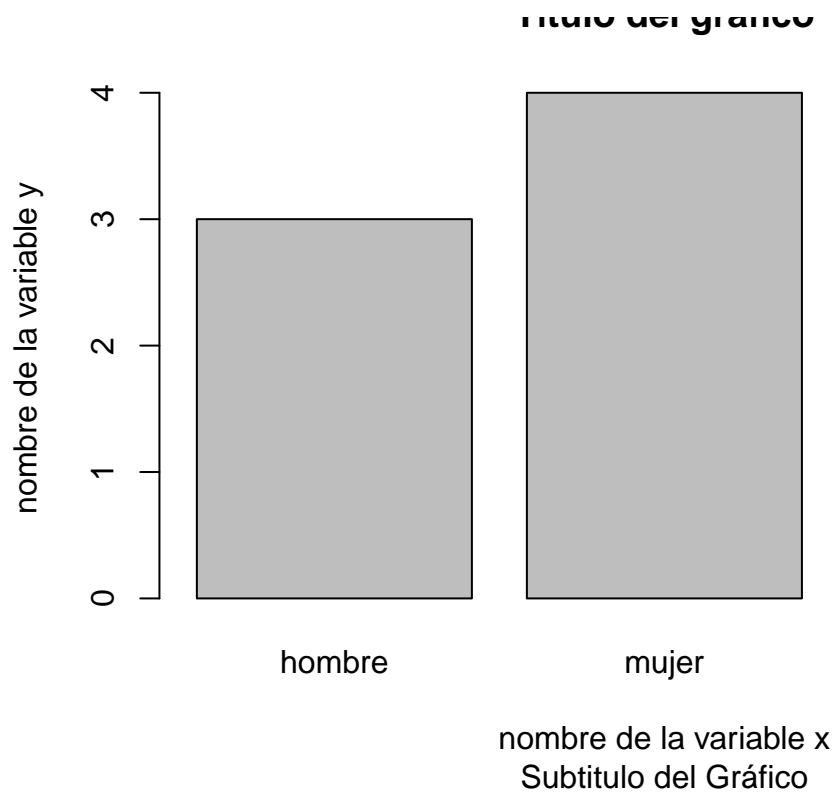


```
# histograma
hist(edad,
     col = "gray",
     main = "histograma",
     sub= "Subtitulo del Gráfico",
     xlab = "nombre de la variable x",
     ylab = "nombre de la variable y")
```



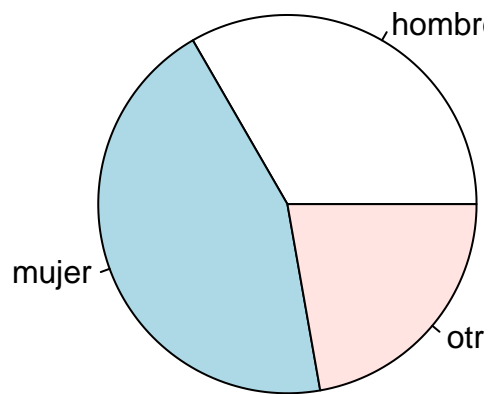
Para utilizar el gráfico de barras o el gráfico de pastel se realiza una tabla con el comando `table()` ya que se trata de datos categóricos.

```
#Gráfico de barras
barplot(table(sexo),
  main = "Título del gráfico",
  sub= "Subtitulo del Gráfico",
  xlab = "nombre de la variable x",
  ylab = "nombre de la variable y")
```



```
#Gráfico de pastel  
pie(table(sexo),  
  main = "Título del gráfico",  
  sub= "Subtitulo del Gráfico",  
  xlab = "nombre de la variable x",  
  ylab = "nombre de la variable y")
```

nombre de la variable y

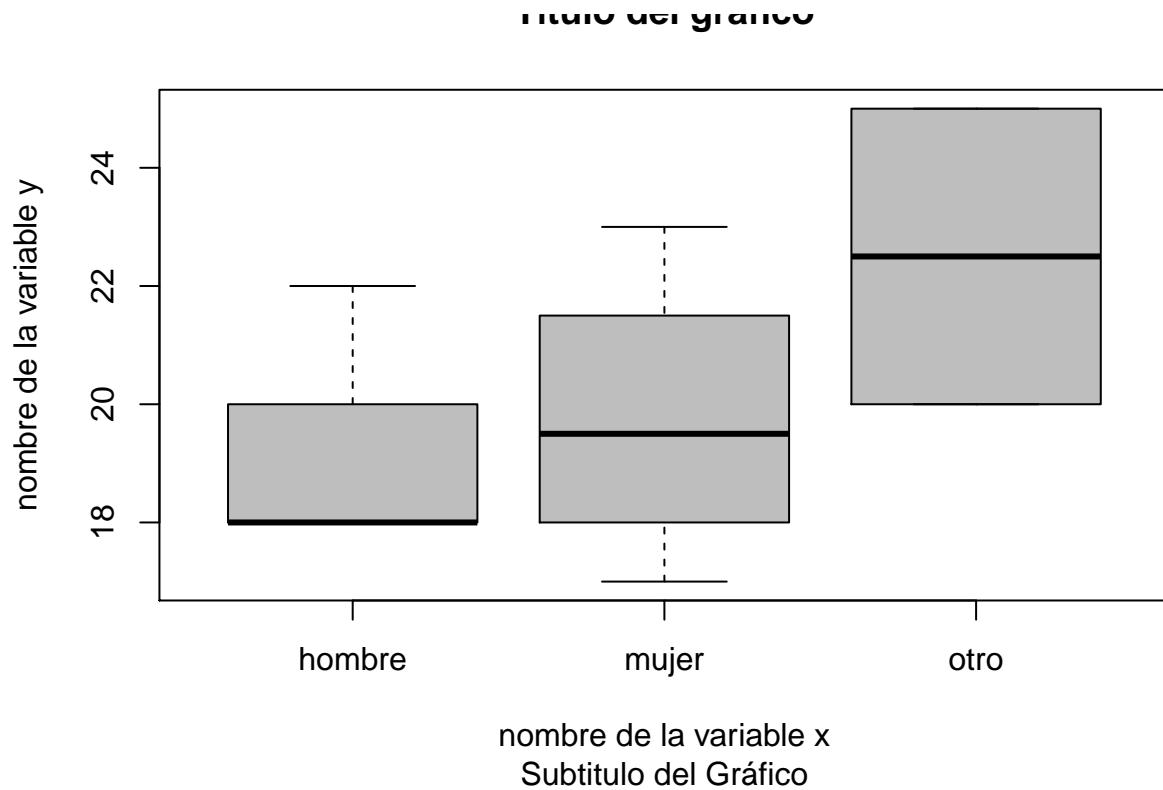


nombre de la variable x  
Subtitulo del Gráfico

Para realizar un el gráfico de cajas, enfrentamos una variable categórica contra una variable continua, es por eso que se expresa como un modelo, es decir, utilizamos el simbolo ~.

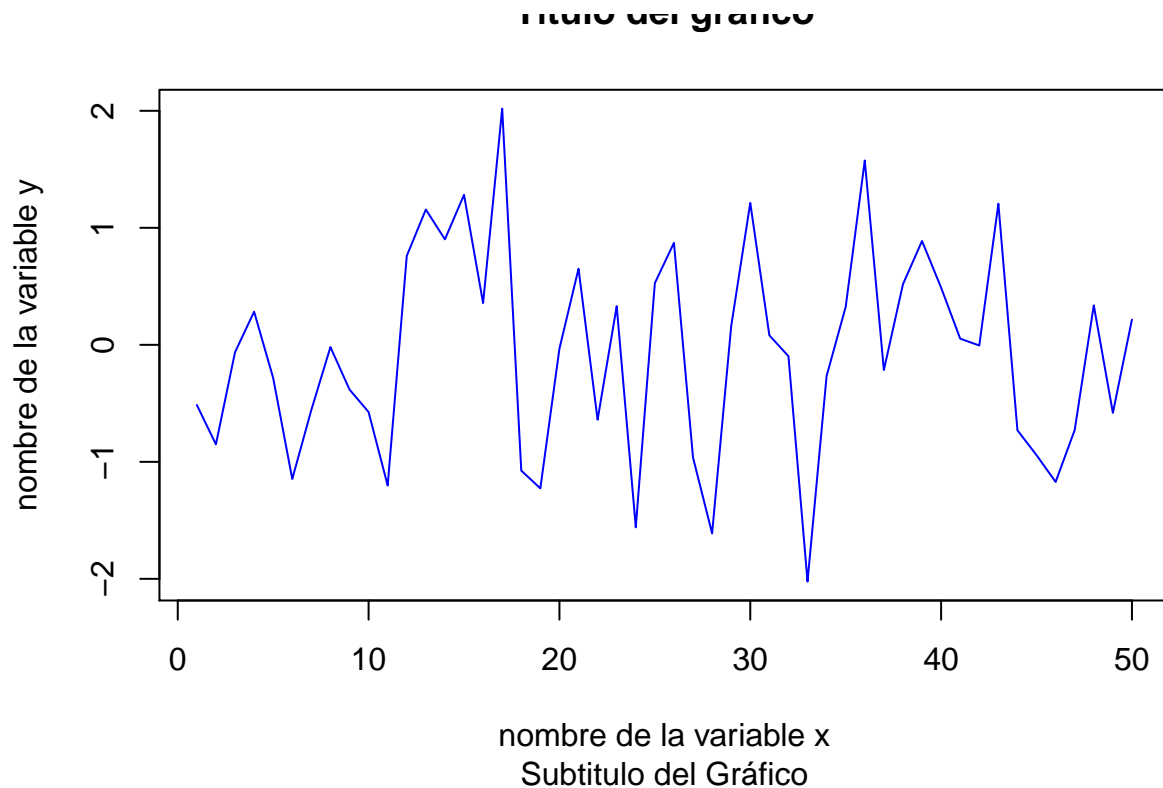
*#Gráfico de Cajas*

```
boxplot(edad~sexo,  
        col = "gray",  
        main = "Título del gráfico",  
        sub= "Subtitulo del Gráfico",  
        xlab = "nombre de la variable x",  
        ylab = "nombre de la variable y")
```



Para hacer gráficos de líneas podemos utilizar también la función `plot()`. Con el comando `rnorm()` podemos generar números aleatorios basados en la distribución normal.

```
x<-rnorm(50, 0, 1) #media 0, varianza 1
plot(x,
      type = "l",
      col = "blue",
      main = "Título del gráfico",
      sub= "Subtitulo del Gráfico",
      xlab = "nombre de la variable x",
      ylab = "nombre de la variable y")
```



### Aplicación: Calculo de mínimos cuadrados ordinarios

Apliquemos parte de lo aprendido, realicemos el calculo de regresión lineal mediante un modelo por aproximación de mínimos cuadrados ordinarios  $\mathbf{y} = \theta\mathbf{x} + \epsilon$ . Este modelo se basa en aproximar una variable minimizando el error cuadrático. primero definamos la matriz de coeficientes:

```
x <-c(600, 600, 700, 700, 700, 900, 950, 950) #variable independiente
i <-c(1,1,1,1,1,1,1,1)
X <-cbind(i,x) #Matriz de coeficientes
print(X)
```

```
##      i    x
## [1,] 1 600
## [2,] 1 600
## [3,] 1 700
## [4,] 1 700
## [5,] 1 700
## [6,] 1 900
## [7,] 1 950
## [8,] 1 950
```

```
y <-c(40, 44, 48, 46, 50, 48, 46, 45) #variable de respuesta
```

El vector  $\theta$  que minimizan el error cuadrático medio se calcula evaluando  $\theta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$  se calculo así:

```
#Transponer
Xt<-t(X)
#Producto Matricial
B<-Xt%*%X
#Invertir la matriz
```

```

C<-solve(B)
#Producto matricial
D<-C%*%Xt
u<-D%*%y
print(u)

```

```

##           [,1]
## i 40.853658537
## x  0.006585366

```

En la práctica, la solución del modelo de mínimos cuadrados se hace mediante el comando `lm()` de la siguiente manera:

```

m <- lm(y~x)
summary(m)

```

```

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4.8049 -1.3598 -0.1341  1.5488  4.5366
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 40.853659   6.156123   6.636 0.000565 ***
## x           0.006585   0.007943   0.829 0.438812
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.115 on 6 degrees of freedom
## Multiple R-squared:  0.1028, Adjusted R-squared:  -0.04676
## F-statistic: 0.6873 on 1 and 6 DF,  p-value: 0.4388

```

Para realizar el gráfico utilizamos `plot()`

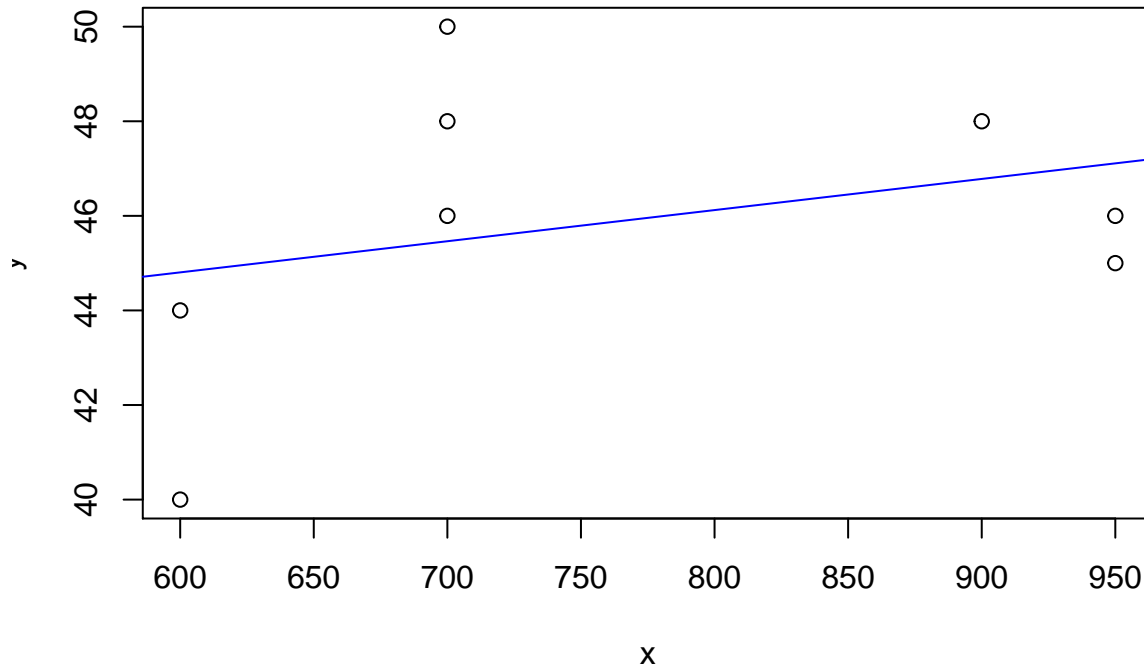
```

plot(x,y, type = "p",
      xlab = "x",
      ylab = "y",
      main="Regresión Lineal Simple")
abline(m, col="blue")

```



## Regresión Lineal Simple



Realicemos ahora el mismo ejercicio matricial pero agreguemos un ajuste cuadrático:

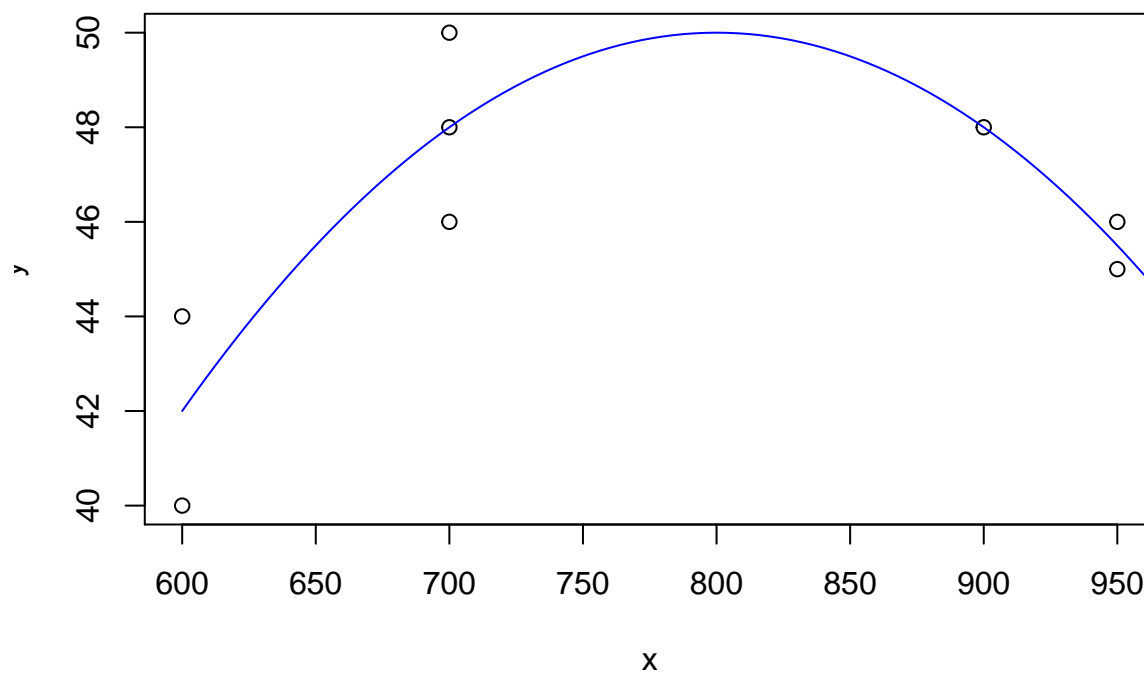
```
x <-c(600, 600, 700, 700, 700, 900, 950, 950) #variable independiente
i <-c(1,1,1,1,1,1,1,1)
x2 <- x^2
X <-cbind(i,x, x2) #Matriz de coeficientes
y <-c(40, 44, 48, 46, 50, 48, 46, 45) #variable de respuesta
#Transponer
Xt<-t(X)
#Producto Matricial
B<-Xt%*%X
#Invertir la matriz
C<-solve(B)
#Producto matricial
D<-C%*%Xt
u<-D%*%y
print(u)
```

```
##      [,1]
## i  -78.0000
## x   0.3200
## x2 -0.0002
```

La grafica correspondiente se puede hacer mediante el comando `curve()`. Se especifica con `from = y to =` los límites de la curva en el eje de las x. Con `add =` especificamos si se quiere superponer a la gráfica anterior.:

```
plot(x,y, type = "p", xlab = "x", ylab = "y",
      main="Regresión con ajuste cuadrático")
b<-curve(u[1]+u[2]*x + u[3]*x^2, from = 600, to=1000, add = TRUE, col="blue")
```

### regresión con ajuste cuadrático



Este ajuste cuadrático para este caso es mejor que el ajuste lineal.