Aprendiendo R

Introducción a R

Aprendiendo R

1. Instalar R:www.r-project.org/
2. Instalar R-Studio: https://www.rstudio.com/

Si se quiere se puede usar la consola directamente para hacer calculos. Pero lo mejor es escribir en source para poder guardar el código (script). En la consola de R también se corren los comandos que se ejecutan en la parte de source y se muestran los resultados.

R trabaja con objetos (por eso se lo clasifica como un lenguaje orientado a objetos), y todos los datos ingresados, variables creadas, resultados obtenidos y funciones utilizadas son almacenados como objetos con un nombre específico que pueden ser inspeccionados y vueltos a usar una y otra vez. A su vez, estos objetos pueden manipularse con operadores (+, -, /, <, >, =, &, etc.) y funciones, que son las que permiten hacer todo tipo de análisis, visualización, manipulación de datos, etc. Las funciones vienen en paquetes, algunos de ellos ya incluidos en la versión de R descargada (librería de funciones), y muchos otros que se consiguen en Internet y que es necesario instalar para usar. Un código o script es un conjunto de comandos que uno puede generar y guardar como archivo de extensión ".R" para no tener la necesidad de volver a tipear todas las órdenes una y otra vez (en la próxima sección vamos a estar trabajando con nuestro primer script).

En R, para solicitar ayuda sobre un tema general podemos escribir, por ejemplo

```
help(mean)
```

Asignacion de valores a una variable (creación de objetos). Para realizar una asignación se utilizan los símbolos <- .Uso # para poder hacer comentarios y que no sea parte del código.

```
x<-1 #control-enter (o arriba presionar run)
y<-2
x+y #operaciones aritméticas
```

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

Puedo asignar listas de caracteres

```
y<-"caracteres"
```

OJO, distingue mayusculas de minusculas

```
#Y : Error: object 'Y' not found
```

Vectores (concatenación de caracteres o números)

```
v<-c(7,8,0,-1)
```

c() es una función que concatena elementos. Para usar funciones primero ponemos el nombre de la función y luego usamos paréntesis.

Ahora v es un vector (columna, si se quiere) Puedo llamar al vector o puedo querer extraer un elemento del vector. Para eso uso los corchetes [ ].

```
v
```

```
## [1]  7  8  0 -1
```

```
v[3] #Extrae el elemento de v en la posicion 3
```

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

Tambien armar vectores con nombres

```
nombres<-c("Francisco", "Pedro", "Jemina")

w<-1:10 #arma un vector con los numeros del 1 al 10

z<-seq(1,4,0.02) #se usa la función seq para generar los números

x<-rep(1,5)

y<-rep(c("a", "b"), 4)
```

Usamos la función seq para generar una secuencia de datos.Sirve para generar grillas.

Usamos la función 'rep()' para crear vectores con elementos repetidos, MUY UTIL!!!

Operaciones con vectores

```
w+5 # le suma 5 a cada elemento
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15
```

```
2*w #multiplica po2 a cada elemento del vector
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

Mas operaciones entre vectores

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

```
## [1]  6  8 10 12
```

```
x*y
```

```
## [1]  5 12 21 32
```

```
x^y
```

```
## [1]     1    64  2187 65536
```

Notemos que las operaciones con vectores se hacen elemento a elemento. Cuando los vectores tienen dimensiones diferentes, el de menor dimensión se extiende repitiendo los valores desde el principio, aunque da un mensaje de aviso.

Funciones con vectores

```
length(w)
```

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

```
sum(x)
```

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

```
max(y)
```

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

```
sort(w)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Vectores lógicos

```
x<-c(5,7,1)
y<-c(3,8,2)

z<-x<y #me dice si es verdadero o falso

z     # z pasa a tener un vector con valores lógicos.
```

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

```
sum(z)  # En R, el TRUE se puede pensar como 1 y el FALSE como 0.
```

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

```
x==y #analiza la igualdad y devuelve un vector de valores lógicos
```

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

```
x!=y #analiza si son distintos
```

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

Matrices Asi es como las armamos:

Para generar matrices

```
A<-matrix(c(1,5,2,-1,4,0,2,7,8),nrow=3, byrow=T)
dim(A)
```

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

Para operar con matrices:

Quiero hallar el producto de la matriz A con el vector x.

```
X<-c(2,8,0)
```

El R lo toma como si fuese una lista y no hace bien el producto, hay que ver a X como si fuera una matriz.

```
dim(X)
```

```
## NULL
```

Una forma de solucionarlo: le digo que quiero que sea una matriz

```
x<-as.matrix(X)
dim(x)
```

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

Otra forma: directamente cargarlo como matriz

```
X<-matrix(c(2,8,0),nrow=3, byrow=T)
dim(X)
```

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

Ahora para hacer A*X hago

```
b<-A%*%X
```

Dado b quiero resolver el sistema Ax=b

```
solve(A,b)
```

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

Juguemos un rato con el "azar": Experimentos con bolitas: Tengo una urna con 5 bolitas y quiero sacar 3 al azar. Uso de la función 'sample()'.

```
set.seed(27)#para que cada vez que comienza el experimento el R haga lo mismo

sample(1:5,3,replace = TRUE)
```

```
## [1] 5 1 5
```

```
sample(1:5,3,replace = FALSE)
```

```
## [1] 2 1 5
```

Si ahora tengo un mazo de 40 cartas, y las quiero mezclar.

```
Cartas<-1:40
#las mezclo
M<-sample(Cartas,length(Cartas),replace = FALSE)
```

Funciones.

Para R, una funcion es una variable. La forma de crear una función es con la siguiente instrucción: Nombre_de_la_función<- function(x, y, ...) {expresión de la función} siendo x, y, ... los argumentos de la función. Luego la función se ejecuta como Nombre_de_la_función(x,y,...)

```
mezclar<-function(v)
{
  v2<-sample(v,length(v),replace = FALSE)
  return(v2)
}

mezclar(Cartas)
```

```
##  [1] 19  7  1 24  4 30  6 20 21 27 28  9 18 36 12 26  5 14 22 35  2 11 39
## [24] 40 23  3 17 32 10 37 16 15 13  8 38 31 33 29 25 34
```

Ahora quiero sumar todos los numeros del 1 al 100 (o de 1 a n). Usamos la sentencia 'for' que permite repetir una cierta operación un número de veces.

```
suma<-function(n)
{
  acum<-0
  for(i in 1:n)
  {
    acum<-acum+i
  }
  return(acum)
}

suma(3)
```

```
## [1] 6
```

```
suma(45)
```

```
## [1] 1035
```

¿Lo hice bien?

```
(45*46)/2
```

```
## [1] 1035
```

Creamos una función para sumar elementos de un vector:

```
sumar_vector<-function(v)
{
  i<-0
  for(elem in v)
  {
    i<-i+elem
  }
  return(i)
}

w<-c(4,8,-7,45,-20)

sumar_vector(w)
```

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

Ahora solo quiero sumar los negativos. Usamos la sentencia 'if':

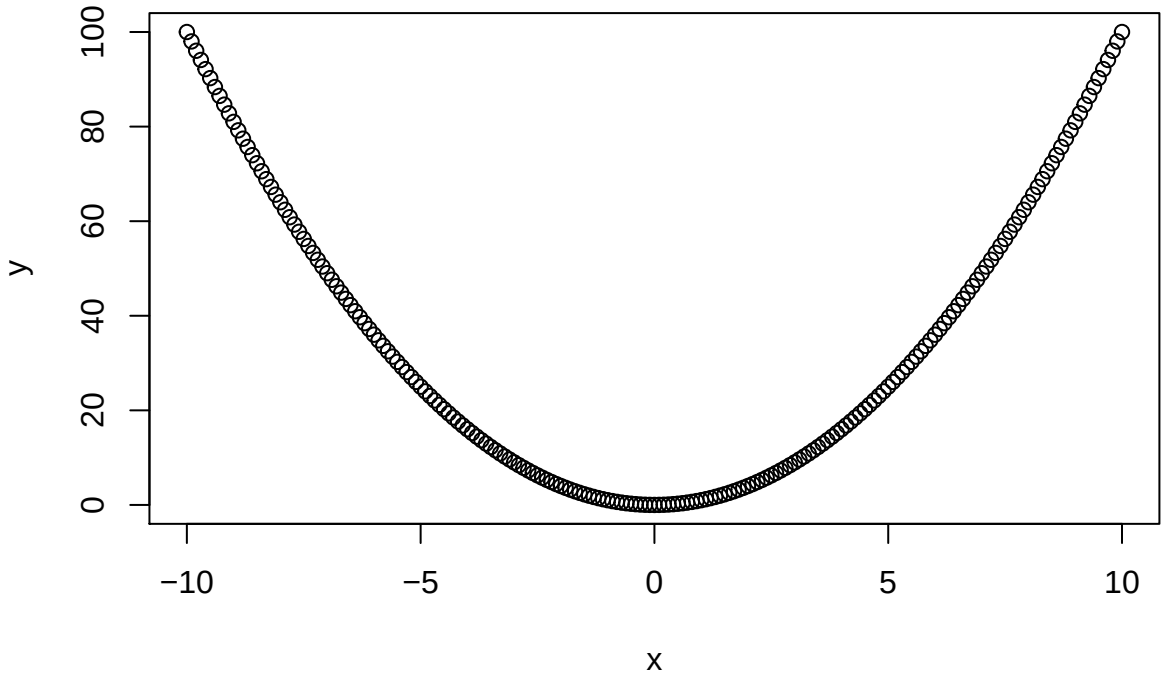
```
sumar_negativos<-function(v)
{
  i<-0
  for(elem in v)
  {
    if(elem<0)
    {
      i<-i+elem
    }
  }
  return(i)
}

sumar_negativos(w)
```

```
## [1] -27
```

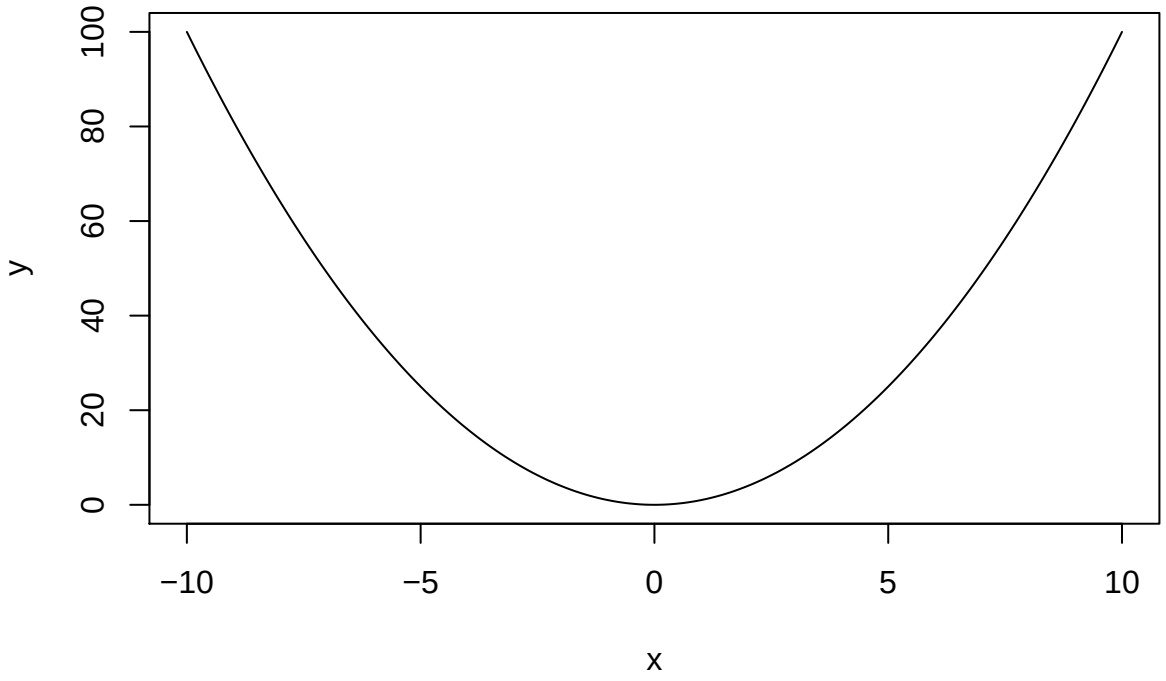
GRAFICOS: Función Plot

La función 'Plot()'devuelve un grafico de dispersion, tambien a veces llamado scatter plot

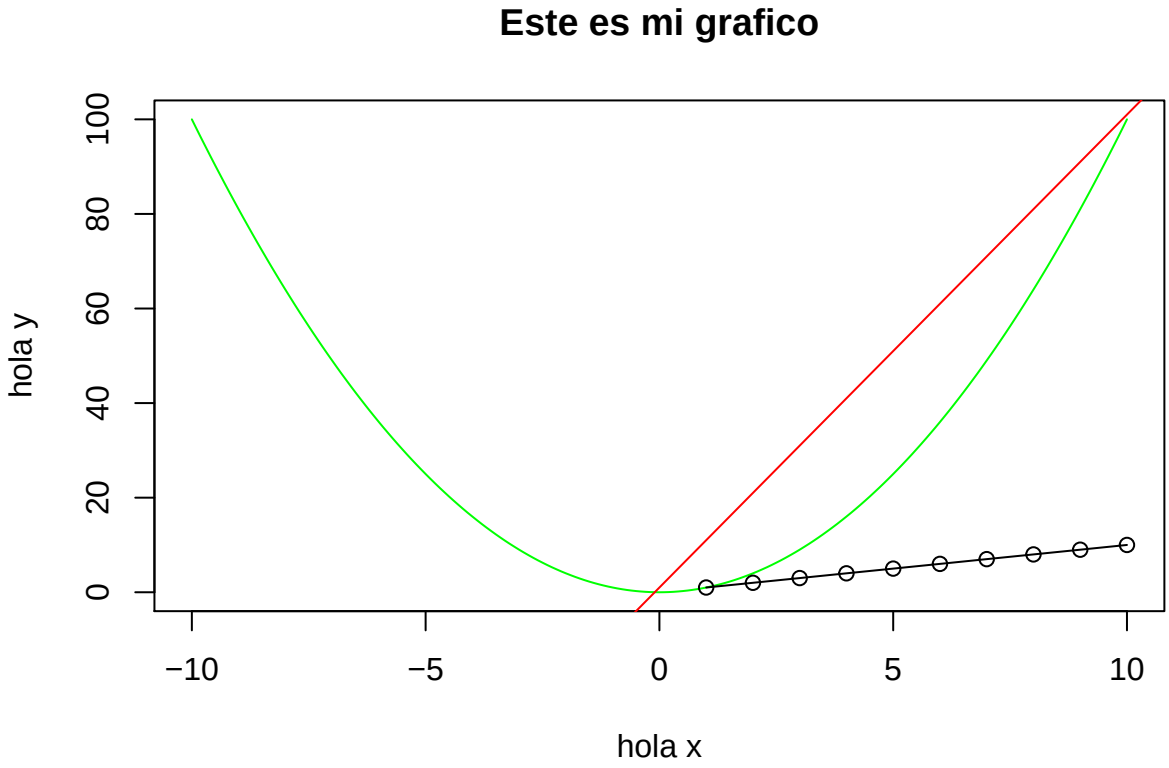
```
x<-seq(-10,10,0.1)
y<-x^2
plot(x,y)
```




```
#si me molesta que sean puntos
plot(x,y,type = 'l')
```




```
#ahora lo decoramos

plot(x,y,main = "Este es mi grafico", xlab = "hola x",
     ylab = "hola y", col="green", type = 'l')
abline(1,10,col="red")
points(1:10)
lines(1:10)
```



Si sobre ese grafico quiero otro, no puedo usar la función plot, cuando pones plot esa es la base y lo redibuja cada vez, hay que usar otros comandos:

Si me olvido como funciona o que debo poner como argumento

```
help(lines)
```

Importar archivos.

En general .txt pero muchas veces vienen en xls no olvidar avisarle al R donde estan los archivos, si no no sabe donde buscar. Yo en general pongo todo en la misma carpenta entonces le digo que busque en el lugar en el que esta guardado en este archivo: session,set working directory, to source file location

```
tabla<-read.table("abalone.txt", sep = ",")
```

Por defecto interpreta el espacio como separacion, si no avisarle que es con una coma para que lo lea bien Es una tabla super grande con muchos datos, mas bonito verla si hago doble click en la ventana de al lado –>

Super importante leer tablas!!! es lo que vamos a usar tooodo el tiempo. Ojo que no lo toma como matriz, es un “data frame”, vamos a enseñarle asi nos sirve Si la informacion me la dan en excel, tengo que primero instalar un paquete para poder abrirlos. Si no pasarlo primero a .txt o .csv (lo cual es recomendable)

```
library(readxl)
```

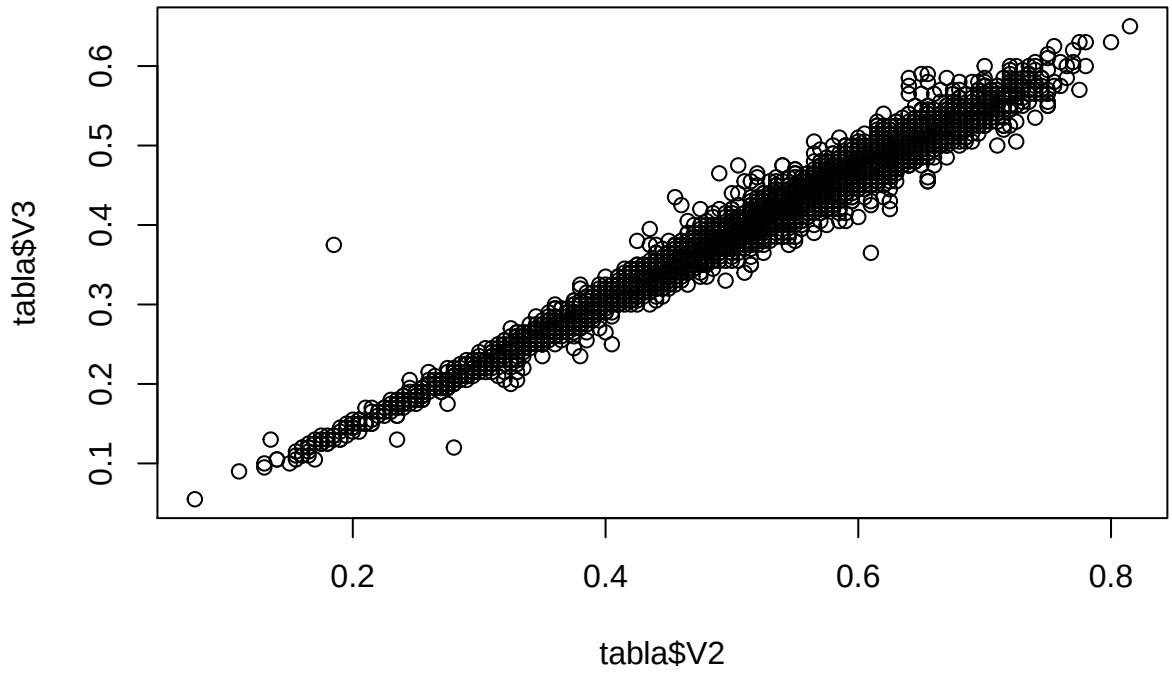
```
## Warning: package 'readxl' was built under R version 3.5.2
```

```
gener<-read_excel("Gener1.xls")

#para acceder a las columnas

gener$y
```

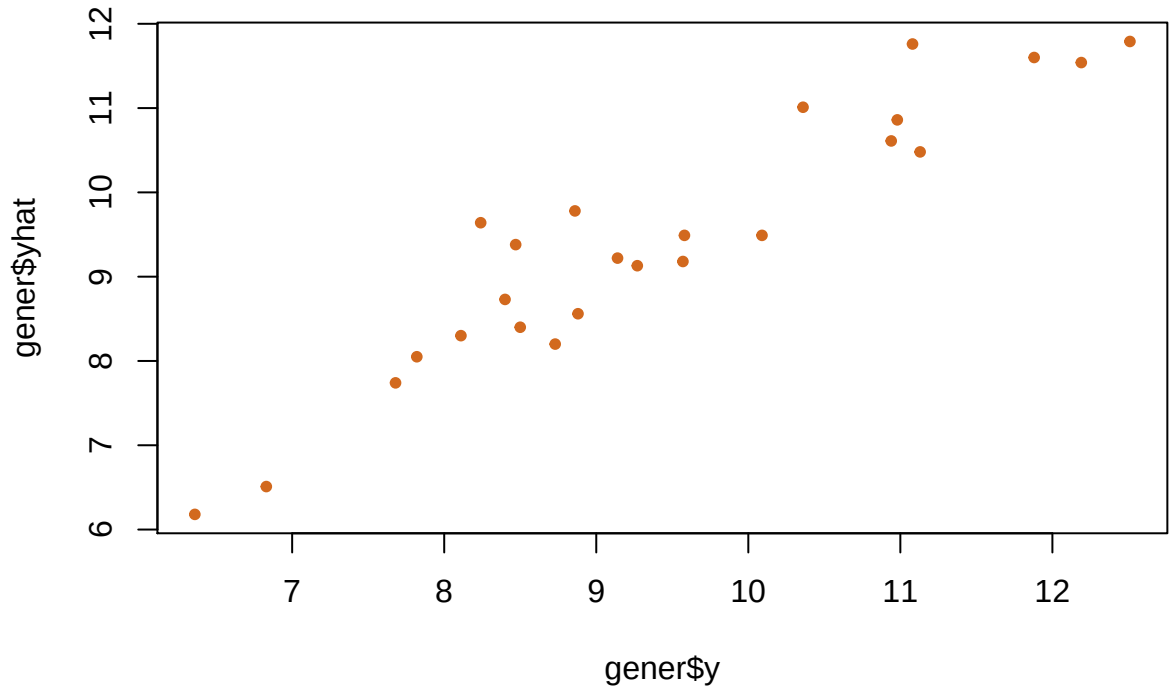
```
##  [1] 10.98 11.13 12.51  8.40  9.27  8.73  6.36  8.50  7.82  9.14  8.24
## [12] 12.19 11.88  9.57 10.94  9.58 10.09  8.11  6.83  8.88  7.68  8.47
```

```
## [23]  8.86 10.36 11.08
```

```
#puedo dibujar

plot(tabla$V2,tabla$V3) # eso si es un grafico de dispersion!
```

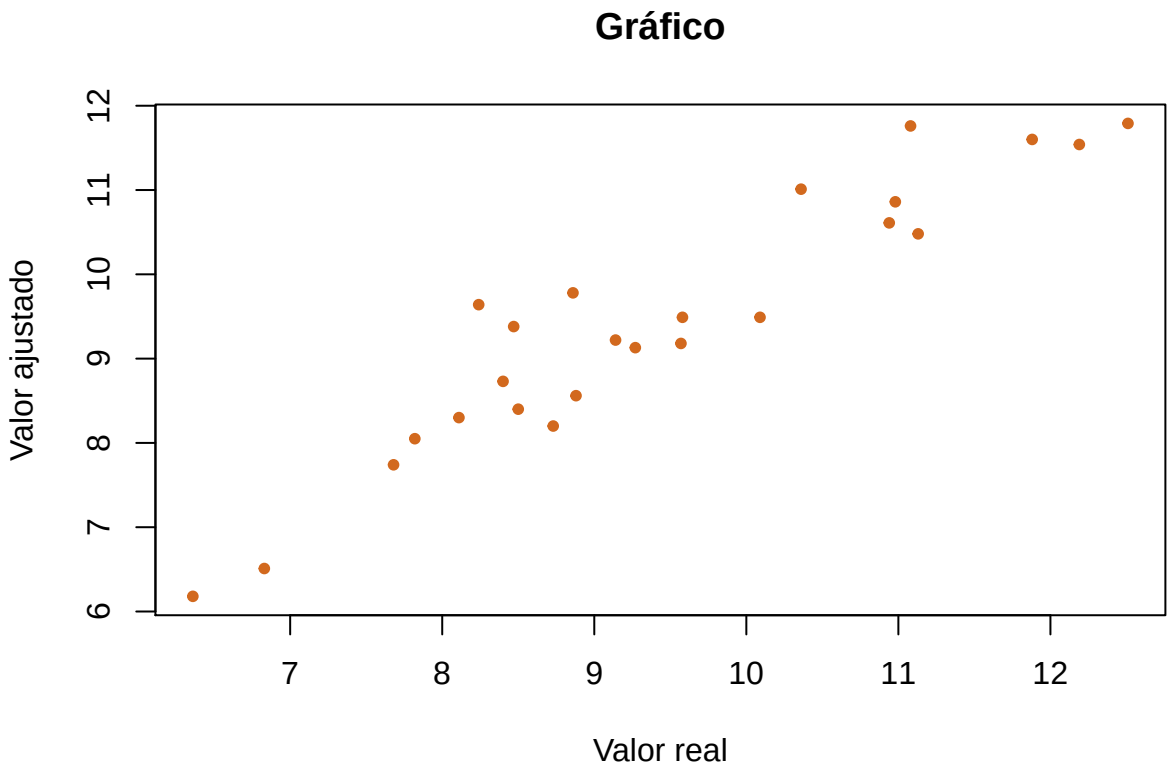



```
plot(gener$y,gener$yhat, pch=20, col= "chocolate")
```



Para que quede mejor no olvidar ponerle los nombres correspondientes asi entendemos que es lo que estamos mirando

```r
plot(gener$y,gener$yhat, pch=20, col= "chocolate",
     main= "Gráfico", xlab = "Valor real", ylab = "Valor ajustado")
```



Y para hacer varios graficos en paralelo:

```r
par(mfrow=c(2,2))
```

Bueno, hay miles de cosas para aprender, las iremos aprendiendo sobre la marcha, y hay muchas cosas para leer en internet :) Lo mas importante que hay que aprender es como buscar en los foros!

Para la próxima... Conjuntos

Tiremos monedas

```r
set.seed(27)
A<-rbinom(1000000,1,0.5)
B<-rbinom(1000000,1,0.5)
mean(A & B) #interseccion - estoy estimando la P(A int B)
```

```
## [1] 0.249794
```

```r
mean(A|B) #unión - estimación de P(A U B).
```

```
## [1] 0.749954
```

```r
#En ambos casos suponiendo que los eventos son independientes.

#reviso
mean(A)*mean(B)
```

```
## [1] 0.2498739
```

```r
mean(A)+mean(B)-mean(A)*mean(B)
```

```
## [1] 0.7498741
```

```
#para 3: mean(A|B|C)
```