Sesión 1

Björn Reu, Yovanny Duran, Zarith Villamizar, Silvia Ardila y Sergio Bolivar

2021-11-05

Contents

Preface

This is the very first part of the book.

Chapter 1

Introducción

El curso Introducción al análisis de datos utilizando R está dirigido a estudiantes de Biología y de Ingeniería Forestal en sus primeros semestres para familiarizarlos con el uso del lenguaje de programación R.

R es un lenguaje de programación de entorno de software libre para computación estadística y gráfica (R Development Core Team, 2021). En años recientes, R se ha vuelto una herramienta indispensable para el análisis de datos (o ciencia de datos) en muchas áreas de las ciencias básicas y aplicadas (Tippmann, 2015) y en la industria (Gandrud, 2013). En lugar de aprender múltiples herramientas, los estudiantes pueden utilizar un entorno único para una gran diversidad de métodos de análisis de datos. Más que un lenguaje de programación, R ofrece nuevas vías para la enseñanza de la estadística y análisis de datos, debido a que los problemas estadísticos pueden ser desarrollados y analizados de manera transparente y reproducible (Incerti et al., 2019). Esto permite un aprendizaje interactivo y orientado a resolver problemas, combinando la teoría y la práctica. De esta manera, los estudiantes pueden lograr una comprensión más profunda de análisis de datos de forma más sencilla, lo cual les permitirá practicar análisis de datos durante sus estudios utilizando todas las herramientas y posibilidades que les ofrece el lenguaje de programación R.

Este libro y el diseño del curso Introducción al análisis de datos utilizando R son el resultado del proyecto Análisis de datos utilizando el lenguaje de programación R: una experiencia de aprendizaje virtual en el aula invertida, de la convocatoría Iniciativas de innovación didáctica mediante el uso de tecnologías, INNOVA-TIC 2021 de la Vicerrectoría Académica de la Universidad Industrial de Santander, UIS.

Chapter 2

Introducción a la programación con R

Comenzaremos con las operaciones básicas ejecutadas en la línea de comandos en R o terminal. Este será nuestro espacio de trabajo en el cual ejecutaremos las ordenes cuyos resultados se mostrarán en la consola.

Todo lo que siga después de un numeral (#) será considerado como un comentario que al ejecutar será omitido. Para ejecutar el código en R podemos utilizar el botón "run" que disponemos arriba a la derecha, o utilizar un atajo:

- **Command + Enter (Mac)**
- **Control + Enter (Windows, Linux)**

2.0.1 La línea de comandos (Terminal)

Línea de comandos como calculadora:

```
2+5
- [1] 7
7-3
- [1] 4
(2+3)*5
- [1] 25
(2+3)/((2+3)*5)
- [1] 0.2
```

Se pueden ejecutar varias operaciones utilizando punto y coma (;):

```
2+5; 4+6
- [1] 7
- [1] 10
```

Dispone de diferentes funciones como raíz cuadrada o logaritmo:

```
?Arithmetic
- starting httpd help server ... done
sqrt(9) # *raíz cuadrada*
- [1] 3
log(1)  # logaritmo
- [1] 0
log10(10) # logaritmo base 10
- [1] 1
3^2 # potenciación (exponente 2)
- [1] 9
```

Las letras y variables más importantes estan disponibles en R:

```
?Constants
pi
- [1] 3.141593
letters
-  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
- [20] "t" "u" "v" "w" "x" "y" "z"
LETTERS
-  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
- [20] "T" "U" "V" "W" "X" "Y" "Z"
```

2.0.2 Función ayuda

Existen tres formas de obtener ayuda:

- Área miscelánea -> “Help”
- Símbolo “?” antes de la función
- Tecla Tabulador (Autocompleta el nombre)

```
x <- c(1,2,3,4,5,6,7,8) # <- símbolo de asignación, permite agrupar los elementos dent
```

```
x = c(1,2,3,4,5,6,7,8) # "<-" o "=" significa lo mismo
```

```
mean(x)
- [1] 4.5
?sum
?rep
?sort
?order
```

Estructura de la función:

- Función: “mean()”
- Objeto: “x”

- Argumento ('arg'): "na.rm=FALSE"

Ejercicio 1:

- *Explicar brevemente la función "order". ¿Qué hace y qué significan los argumentos?*

2.0.3 Creando nuevas variables:

Para crear una nueva variable usted puede utilizar "<-" (símbolo de asignación) o "=". Es aconsejable el uso de "<-" ya que se refiere a la indexación dentro de un vector.

Cada variable necesita un nombre que puede estar compuesto de letras, números, "." o "_" y nunca por números. Ejemplo: "my_data".

R diferencia entre minúsculas y mayúsculas; "a" es diferente de "A". Las variables pueden ser reescritas. Una vez escrita una variable podemos llamar al objeto:

```
x <- 3
x
- [1] 3
x <- 9
x
- [1] 9
resultado <- 3+9
resultado
- [1] 12
mode(resultado)
- [1] "numeric"
```

Además de las variables ('numeric'), R permite la definición de texto ('character') y de variables lógicas ('logic', ej: TRUE y FALSE). Los caracteres deben ser puestos entre comillas dobles ("…") para que R los identifique como tal.

Estos tipos de vectores se definen como vectores atómicos:

```
y <- "test"
y
- [1] "test"
mode(y)
- [1] "character"
a <- FALSE
a
- [1] FALSE
mode(a)
- [1] "logical"
```

```
mode(T) # T es igual a TRUE, y F es igual a FALSE
- [1] "logical"
```

¿Qué ocurre aquí?

```
s <- as.numeric(c(T,F,F))
mode(s)
- [1] "numeric"
```

2.0.4 El área de trabajo o memoria - Ventana arriba a la derecha 'Environment'

Todos los objetos creados durante una sesión son guardados en el área de trabajo. Para ver el área de trabajo ejecute ls(), para remover algún objeto de esta ejecute rm().

También puede utilizar los botones en el área de trabajo

```
ls()
- [1] "a"         "resultado" "s"         "x"         "y"
rm(s)
ls()
- [1] "a"         "resultado" "x"         "y"
```

2.0.5 Vamos a practicar

Puedes hacer cálculos simples con variables númericas

```
a <- 5
b <- 7
a+b+3
- [1] 15
```

¿Qué ocurre a continuación? Por favor explique.

```
b = 2
b == 2
- [1] TRUE
b == 3
- [1] FALSE
c = 7
d = -3
?Syntax
```

Ejercicio 2:

- *Comprobar las siguientes afirmaciones lógicas:*
  - "c" mayor que "d"
  - "c" menor o igual que "d"
  - "c" igual a "d"

2.0.6 R como calculadora - Operadores y funciones.

```
?Syntax
```

```
* <-               Asignar
* + - * / % ^      Aritméticas
* > >= < <= == !=  Relación (orden y comparación)
* ! & && | ||      lógicas
* $                lista indexada
* :                Crear una secuencia
```

Operadores lógicos para:

```
* !      NO
* &      Y
* |      O
* <     Menor que
* <=    Menor o igual que
* >     Mayor que
* >=    Mayor o igual que
* ==    IGUAL
* !=    NO IGUAL (diferente de)
* &&    AND with IF (y con si <condicional>)
* ||    OR with IF (o con si <condicional>)
```

2.0.7 Tipos de Datos y Objetos

- **"Logical"** :(FALSO/VERDADERO) / (FALSE/TRUE)
- **"Number"** : (Entero, Decimal, Complejo (e.g. 3i))
- **"Character"** : Letras y palabras (" ", o ' ')

Otros tipos de datos son **"list"**, **"expression"**, **"name"**, **"symbol"** and **"function"**

Para operaciones más complejas necesitamos estructuras de datos más complejas. R ofrece más que solo objetos que contienen un elemento, como los vectores o las matrices.

2.0.8 Vectores I: Vectores sencillos

Para hacer un vector utilice la función concatenar "c()" . Los elementos sencillos de un vector están separados por ",".

```
x <- c (2, 5, 10, 14, 3, 1, 18, 24, 17)
x
- [1]  2  5 10 14  3  1 18 24 17
mode(x)
- [1] "numeric"
```

```
a <- c ("text", 2, 6, TRUE)
mode(a) ## ¿Qué ocurre aquí? Por qué "character"?
- [1] "character"
a <- c ( 2, 6, F)
mode(a) #y ahora?
- [1] "numeric"
```

Crear un vector vacío:

```
b <- numeric(20)
b
-  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Ejercicio 3:

- *Crear un vector con nombre "vec" que contenga los números de 1 a 10.*

Un vector también puede contener diferentes tipos de variables. Usted puede realizar cálculos con vectores que contengan solo elementos "numeric"

- **Suma**

```
sort(x)
- [1]  1  2  3  5 10 14 17 18 24
sum(x)
- [1] 94
```

- **Promedio**

```
mean(x)
- [1] 10.44444
```

- **Cuartil**

```
quantile(x)
-   0%  25%  50%  75% 100%
-    1    3   10   17   24
```

- **Longitud del vector**

```
length (x)
- [1] 9
length(letters)
- [1] 26
```

- **Ordenar**

```
x
- [1]  2  5 10 14  3  1 18 24 17
?sort(x)
sort(x, decreasing = F)
- [1]  1  2  3  5 10 14 17 18 24
sort(x, decreasing = FALSE)
- [1]  1  2  3  5 10 14 17 18 24
sort(x, decreasing = T)
- [1] 24 18 17 14 10  5  3  2  1
sort(x, decreasing = TRUE)
- [1] 24 18 17 14 10  5  3  2  1
```

¿Qué ocurre si colocamos "decreasing = TRUE"? Compruebe con "?sort"

"sort" y **"order"** realizan la misma función; sin embargo **"order"** puede ser aplicado a otro tipo de objetos diferentes a vectores, como los data frames.

- **Cálculo con vectores**

```
x
- [1]  2  5 10 14  3  1 18 24 17
length(x)
- [1] 9
#1
x + 10
- [1] 12 15 20 24 13 11 28 34 27
x
- [1]  2  5 10 14  3  1 18 24 17
#2
y <- c(1, 3, 5)
x + y
- [1]  3  8 15 15  6  6 19 27 22
x
- [1]  2  5 10 14  3  1 18 24 17
#3
y <- c (4, 2, 8, 5, 3, 9, 3, 10, 1)
xy <- x + y
xy
- [1]  6  7 18 19  6 10 21 34 18
x
- [1]  2  5 10 14  3  1 18 24 17
```

```
#4
y <- c(1, 3)
x + y
- Warning in x + y: longitud de objeto mayor no es múltiplo de la longitud de uno
- menor
- [1]  3  8 11 17  4  4 19 27 18
x
- [1]  2  5 10 14  3  1 18 24 17
#5
y <- c (4, 2, 8, 5, 3, 9, 3, 10, 1)
sum(x)
- [1] 94
sum(x,y)
- [1] 139
sum(x + y)
- [1] 139
```

- **Etiquetas**

```
x <- c (2, 5, 10, 14, 3, 1, 18, 24, 17)
a <- c ("E1","E2","E3","E4","E5","E6","E7","E8","E9")
names (x) <- a
x
- E1 E2 E3 E4 E5 E6 E7 E8 E9
-  2  5 10 14  3  1 18 24 17
names(x) # Vector de los nombres del vector numérico x
- [1] "E1" "E2" "E3" "E4" "E5" "E6" "E7" "E8" "E9"
str (x) # sobre la estructura del vector x (muy útil!)
-  Named num [1:9] 2 5 10 14 3 1 18 24 17
-  - attr(*, "names")= chr [1:9] "E1" "E2" "E3" "E4" ...
summary (x) # indica los valores mínimos, máximos, etc.
-    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-    1.00    3.00   10.00   10.44   17.00   24.00
head (x)
- E1 E2 E3 E4 E5 E6
-  2  5 10 14  3  1
tail (x)
- E4 E5 E6 E7 E8 E9
- 14  3  1 18 24 17
```

Ejercicio 4:

- *Sumar los números enteros de 1 hasta 5.*
- *Crear un variable v1 que contenga una letra.*
- *Copiar v1 a v2.*

- *Comparar los valores de v1 y v2*
- *Crear un vector de longitud 20 con el tipo de datos que usted quiera y muestre las primeras nueve entradas.*
- *Crear un vector que contenga los seis primeras letras del abecedario.*

2.0.9 Vectores II: Vectores más complejos

- **Secuencias**

```
z <- 3:10
z
- [1]  3  4  5  6  7  8  9 10
```

```
?seq
seq(from = 1, to = 6, by = 0.2) # El argumento "by" genera frecuencias #
-  [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6
- [20] 4.8 5.0 5.2 5.4 5.6 5.8 6.0
seq(from = 1, to = 6, length.out = 20)  # ¿Cuál es la diferencia con respecto al anterior?
-  [1] 1.000000 1.263158 1.526316 1.789474 2.052632 2.315789 2.578947 2.842105
-  [9] 3.105263 3.368421 3.631579 3.894737 4.157895 4.421053 4.684211 4.947368
- [17] 5.210526 5.473684 5.736842 6.000000
```

La secuncia puede iniciar desde cualquier punto

```
seq(from = 530, to = 620, by = 30)
- [1] 530 560 590 620
```

```
z <- c(1, 3, 9)
z
- [1] 1 3 9
?rep
rep (z, times = 10) # ¿Qué ocurre aqui?
-  [1] 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9 1 3 9
```

La secuencia de elementos del vector z se repite 10 veces

```
rep (z, each = 10) # ¿Cuál es la diferencia con la línea anterior?
-  [1] 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 9 9 9 9 9 9 9 9 9 9
```

Aquí cada uno de los elementos del vector z se repite 10 veces.

2.0.10 Indexar

Para extraer uno o varios elementos de un vector utilice "[]". Por favor observe y explique que ocurre en las siguientes líneas:

```
x
- E1 E2 E3 E4 E5 E6 E7 E8 E9
-  2  5 10 14  3  1 18 24 17
x[5]
- E5
-  3
x[4]
- E4
- 14
x[3:5]
- E3 E4 E5
- 10 14  3
x[c(3,5)]
- E3 E5
- 10  3
id<-c(1, 9, 2)
x[id]
- E1 E9 E2
-  2 17  5
x[c (1, 9, 2)]
- E1 E9 E2
-  2 17  5
x[-c (1, 9, 2)]
- E3 E4 E5 E6 E7 E8
- 10 14  3  1 18 24
```

Ejercicio 5:

- *Crear un vector "x" con los números de 1 a 100.*
- *Extraer del vector el elemento numero 87.*
- *Extrer todos los elementos excepto el 87 para crear un nuevo vector "z".*
- *Verificar la longitud del nuevo vector.*

PLUS: Extraer cada segundo elemento del vector

- *Crear un vector vacío de longitud 10 y asignar a su tercer elemento el valor que tiene la suma de x.*

Ejercicio 5:

- *Crear un vector vacío de cuatro elementos*
- *En el vector vacío indexe para cada posición la primera inicial de sus nombres y apellidos. (Si tiene un solo nombre indexe NA).*

2.0.11 NA´s

```
?NA
e <- c(122, 324, 34, NA, 234)
mean(e) # Por qué da este resultado? Con NA no se puede calcular!
- [1] NA
?mean
mean(e, na.rm = T) # Ahora sin tener en cuenta el NA
- [1] 178.5
is.na(e)        # Me indica cuales son NA
- [1] FALSE FALSE FALSE  TRUE FALSE
which(is.na(e)) # Me indica la posición del NA
- [1] 4
```

2.0.12 Matrices

La matriz es un conjunto de elementos que son del mismo tipo, ya sea numéricos, de carácter o lógico, distribuidos en dos dimensiones, filas y columnas.

- [1,] indica la primera fila
- [,1] indica la primera columna

```
mat <- matrix(data = 1:12, nrow = 3, ncol = 4, byrow = T)
mat
-      [,1] [,2] [,3] [,4]
- [1,]    1    2    3    4
- [2,]    5    6    7    8
- [3,]    9   10   11   12
mat[,1]
- [1] 1 5 9
mat[2,2]
- [1] 6
mat[1:3,1:3]
-      [,1] [,2] [,3]
- [1,]    1    2    3
- [2,]    5    6    7
- [3,]    9   10   11
mat[,1]
- [1] 1 5 9
mat[1,]
- [1] 1 2 3 4
```

- **Transponer**

```
mat2 <- t(mat)
mat2
```

```
-      [,1] [,2] [,3]
- [1,]    1    5    9
- [2,]    2    6   10
- [3,]    3    7   11
- [4,]    4    8   12
mat
-      [,1] [,2] [,3] [,4]
- [1,]    1    2    3    4
- [2,]    5    6    7    8
- [3,]    9   10   11   12
colnames(mat2) <- c("A","B","C")
mat2
-      A B  C
- [1,] 1 5  9
- [2,] 2 6 10
- [3,] 3 7 11
- [4,] 4 8 12
rownames(mat2) <-c ("sp1","sp2","sp3","sp4")
mat2
-     A B  C
- sp1 1 5  9
- sp2 2 6 10
- sp3 3 7 11
- sp4 4 8 12
class(mat2[,1])
- [1] "integer"
class(mat2[2,1])
- [1] "integer"
```

Ejercicio 6:

- *Realice una matriz con 11 columnas y 11 filas.*
- *Incluya en la matriz el número 8 en cada esquina y en la mitad de la matriz.*

- **cbind y rbind**

```
mat
-      [,1] [,2] [,3] [,4]
- [1,]    1    2    3    4
- [2,]    5    6    7    8
- [3,]    9   10   11   12
c5 <- c(5,5,5)
cbind(mat, c5)
-                 c5
```

```
- [1,] 1  2  3  4  5
- [2,] 5  6  7  8  5
- [3,] 9 10 11 12  5
r4 <- c(4,4,4,4)
rbind(mat, r4)
-    [,1] [,2] [,3] [,4]
-       1    2    3    4
-       5    6    7    8
-       9   10   11   12
- r4    4    4    4    4
```

- **rownames y colnames**

```
rownames(mat) <- c("a", "b", "c")
mat
-   [,1] [,2] [,3] [,4]
- a    1    2    3    4
- b    5    6    7    8
- c    9   10   11   12
colnames(mat) <- c("E1", "E2","E3","E4")
mat
-   E1 E2 E3 E4
- a  1  2  3  4
- b  5  6  7  8
- c  9 10 11 12
```

Chapter 3

Gráficas en R

En esta sesión vamos a explorar las herramientas en R para realizar gráficas a partir de datos, y los diferentes elementos que permiten modificar una gráfica para personalizarla

3.1 Gráficas y diagrama de dispersión simple

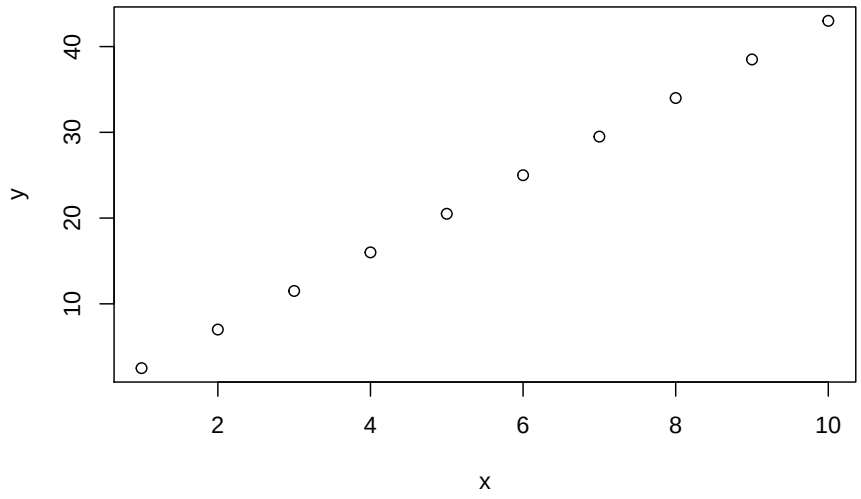
3.1.1 Diagrama de dispersión: función plot()

Comenzamos creando algunos datos a graficar:

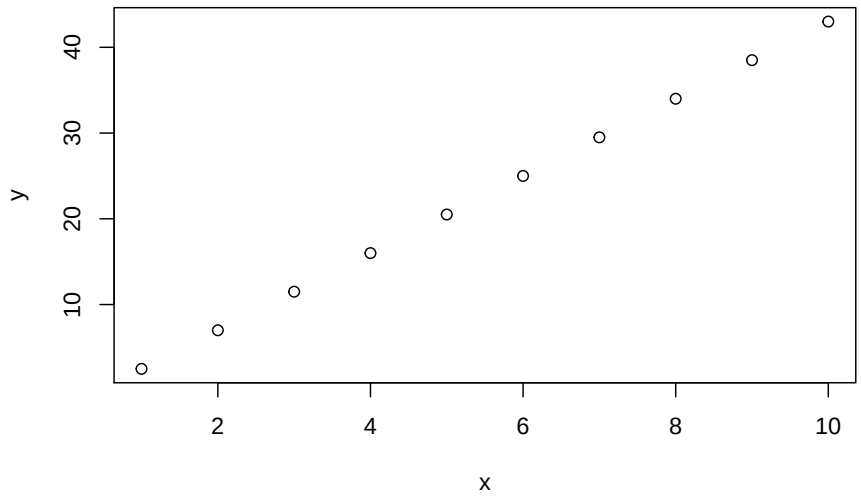
```
x <- 1:10
y <- 1.5 * x + seq(from = 1, to = 30,  by = 3) #
```

El diagrama de dispersión puede ser escrito de dos formas:

```
plot(x, y)
```

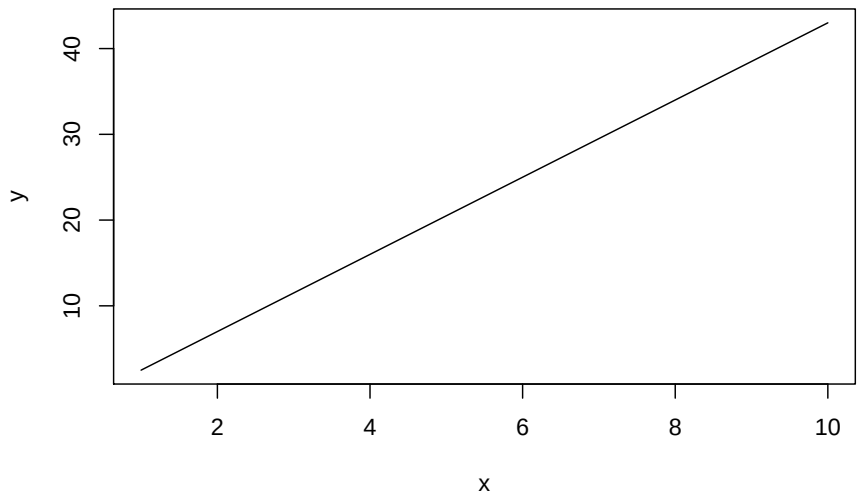



```
plot(y~x)
```

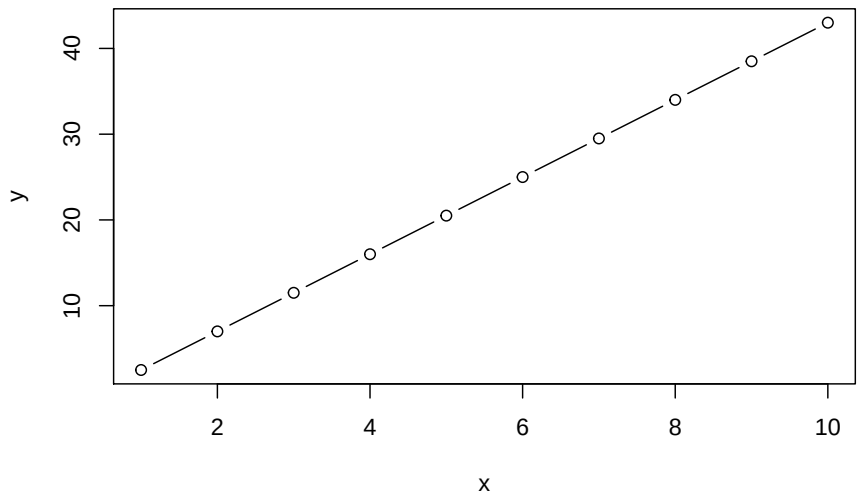


Con type es posible cambiar la disposición del diagrama de dispersión:

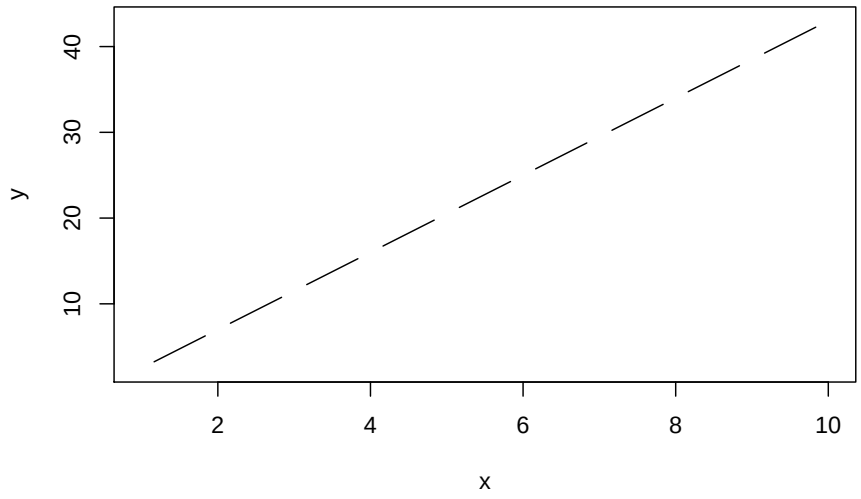
```
plot(x, y, type="l")
```



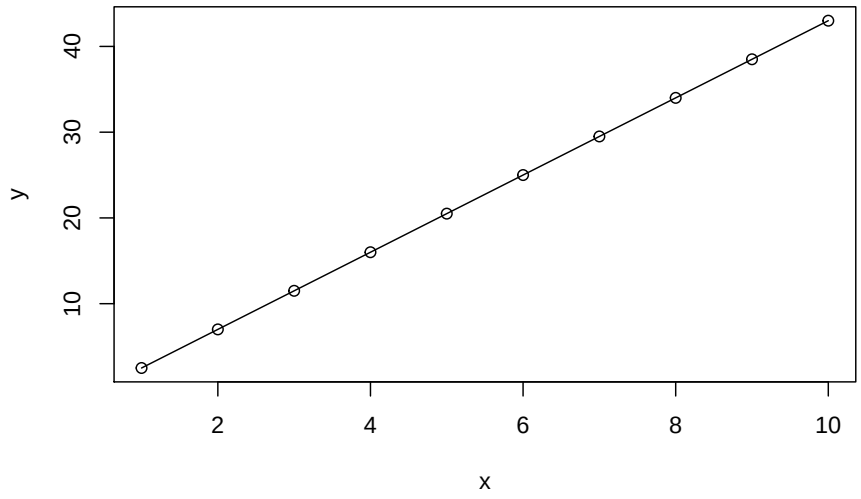

```
plot(x, y, type="b")
```



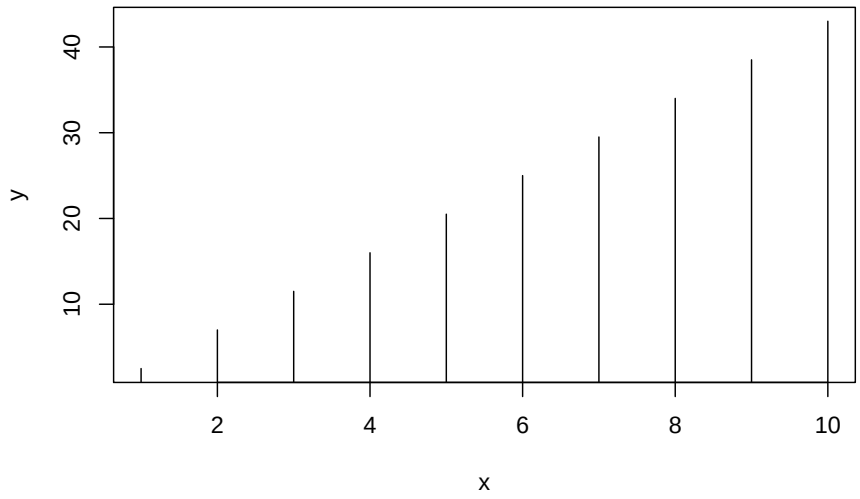
```
plot(x, y, type="c")
```



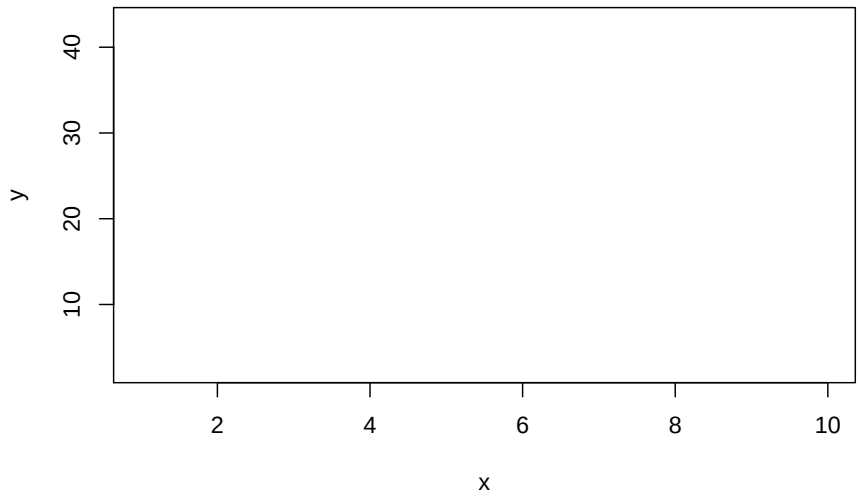

```
plot(x, y, type="o")
```



```
plot(x, y, type="h")
```

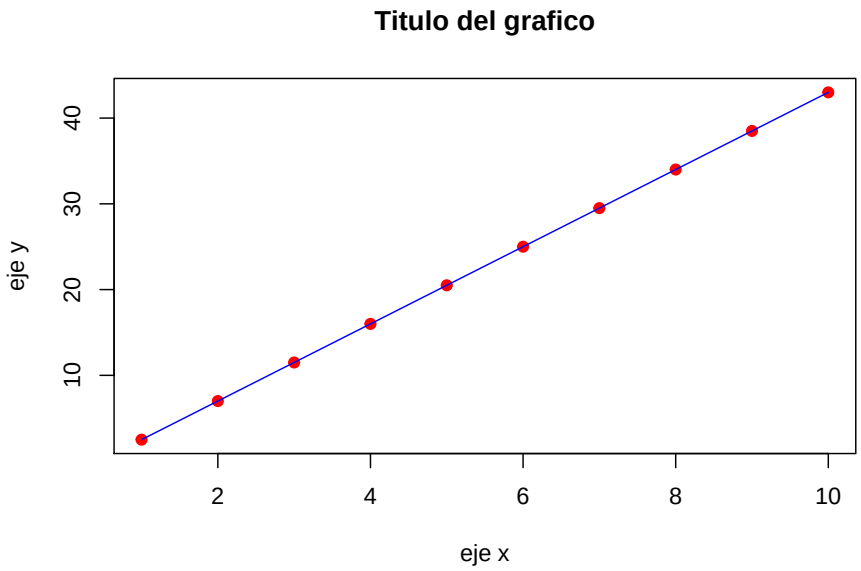



```
plot(x, y, type="n")
```



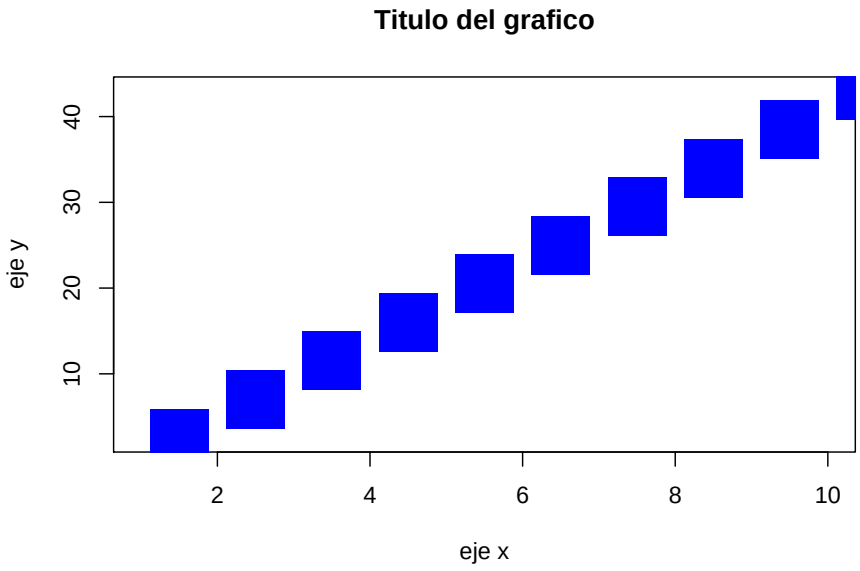
```
plot(x, y, xlab = "eje x", ylab = "eje y", main = "Titulo del grafico", type = "n")
points(x , y, col = "red", pch = 19)
lines(x, y, col = "blue")
```

3.1.2 Etiquetas de la gráfica: xlab, ylab, main



¿Qué ocurre aquí? Por favor explique:

```
plot(x, y, xlab = "eje x", ylab = "eje y", main = "Titulo del grafico", type = "n")
points(x + 0.5, y, col = "blue", pch = 15, cex = 5.5)
```



¿Para que son los argumentos “pch” y “cex”?

?points

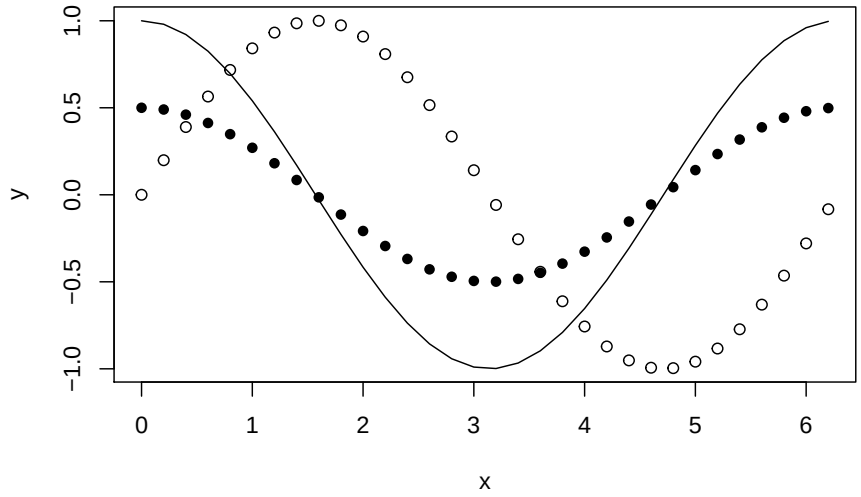
Ejercicios:

1. Haga una linea en zigzag con puntos de colores en los puntos de giro

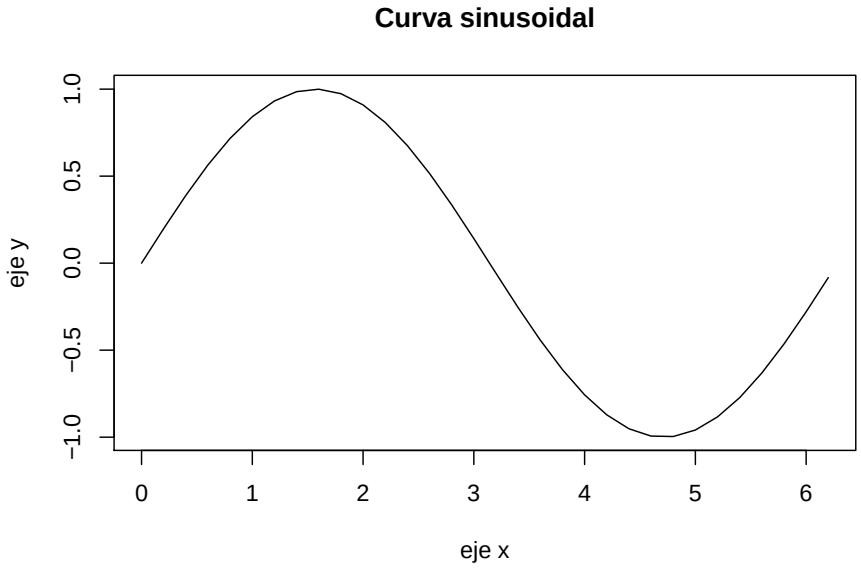
2. Haga una carita (:|)

Más gráficas: {-}

```
x <- seq(from = 0, to = 2 * pi, by = 0.2)
y <- sin(x) # Función seno
plot(x, y)

z <- cos(x) #Función coseno
lines(x, z)  # Agregamos líneas al grafico anterior
points(x, z * 0.5, pch = 16) # Agregamos puntos al gráfico anterior
```

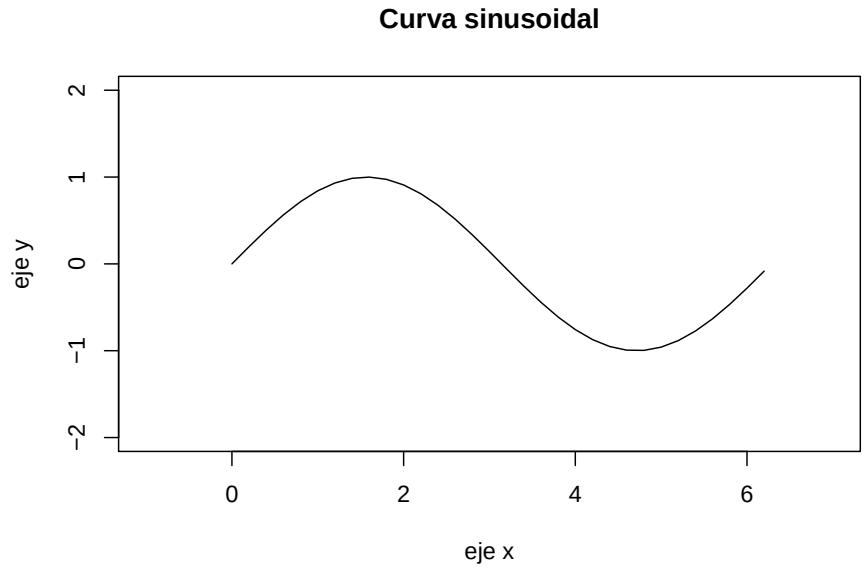



```
#De nuevo, eje, etiquetas y título
plot(x, y, xlab = "eje x", type = "l", ylab = "eje y", main = "Curva sinusoidal")
```

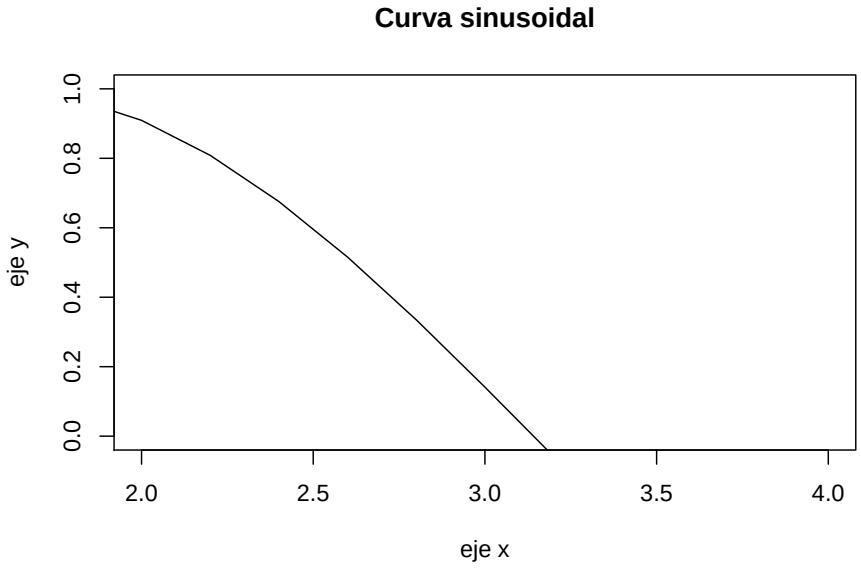



```
#Cambiando la extension de los ejes "x" y "y" usando "xlim" y "ylim"
```

```
plot(x, y, xlab = "eje x", type = "l", ylab = "eje y", main = "Curva sinusoidal",
     xlim = c(-1, 7), ylim = c(-2, 2))
```




```
plot(x, y, xlab = "eje x", type = "l", ylab = "eje y", main = "Curva sinusoidal",
     xlim = c(2, 4), ylim = c(0, 1))
```

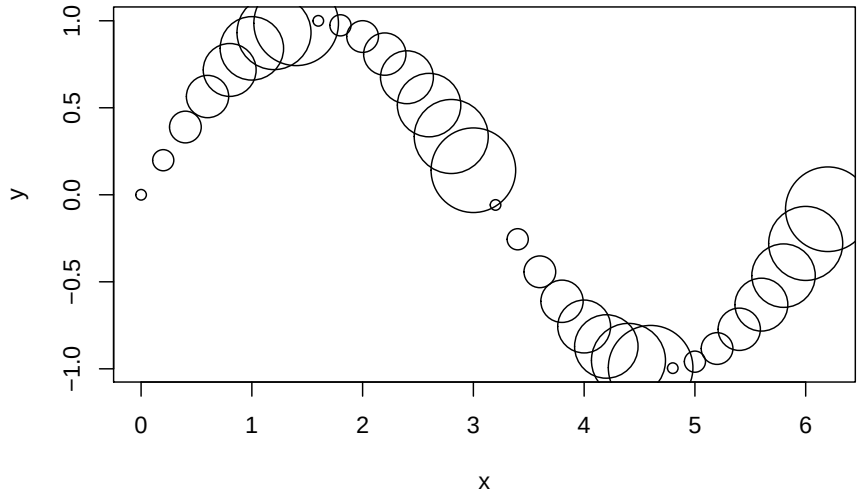


3.2 Demostración de la versatilidad de la gráfica

La representación gráfica en R es muy flexible. Vamos a cambiar mas parametros gráficos:

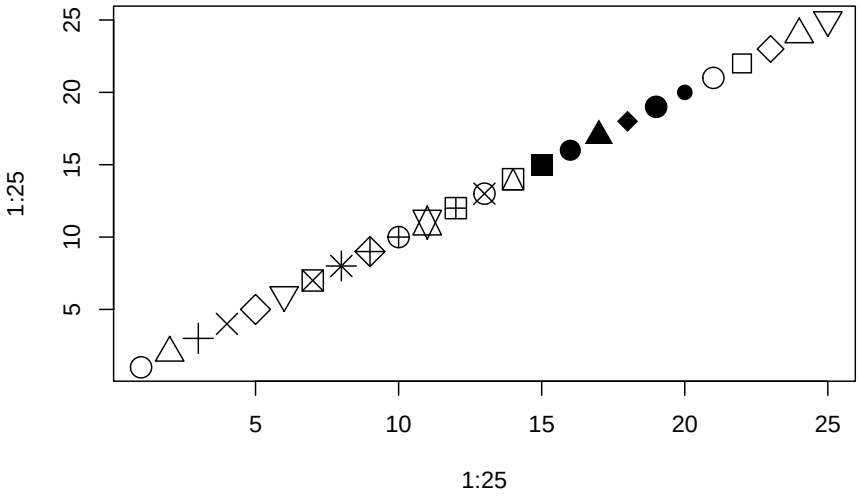
- 'cex': escala de simbolos (tamano)

```
plot(x, y, cex = rep(1:8, times = 4))
```



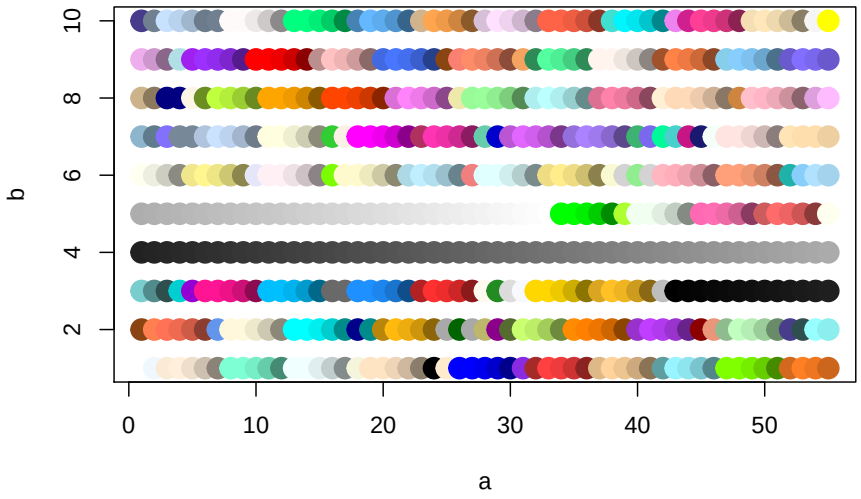
- 'pch': cambia el simbolo de los puntos

```
plot(1:25, 1:25, cex = 2, pch = 1:25)
```



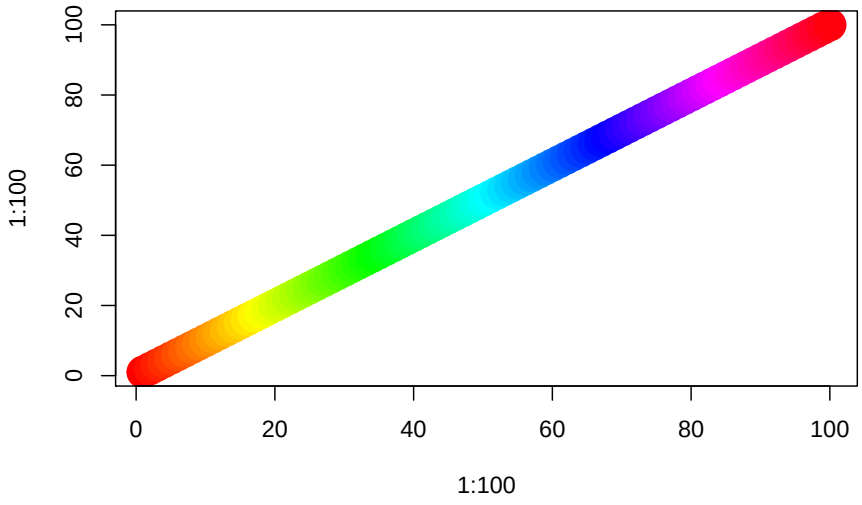
- 'col': colores de los puntos. En R estan disponibles mas de 600 colores

```
a <- rep(1:55, times = 10); b <- rep(1:10, each = 55)
plot(a, b, pch = 19, cex = 2, col = colors()[-c(260:361, 653:657)])
```



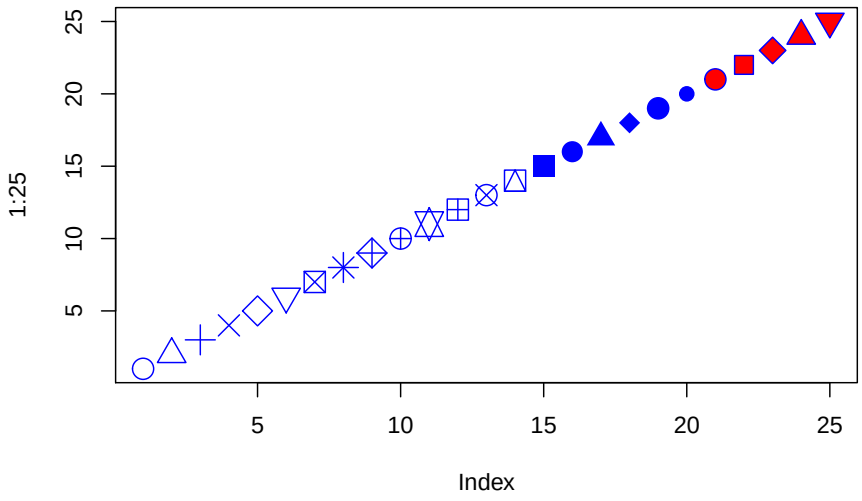
Los gradientes de colores tambien son posibles: ?rainbow

```
plot(1:100, 1:100, pch = 19, cex = 3, col = rainbow(100))
```



Para los simbolos 21 a 25 de los puntos (ver ?points) puede tener diferentes colores de relleno

```
plot (1:25, cex=2, pch=1:25, col="blue", bg="red")
```



Ejercicios:

1. Consulte la pagina web de Quick-R para ver ejemplos de diagramas de dispersion (http

3.3 Barras, cajas e histogramas

Algunas de las gráficas más populares para reportar y explorar datos es mediante las graficas de barras (conteo de observaciones), cajas (estadisticos de nuestros datos) e histogramas (frecuencia de valores)

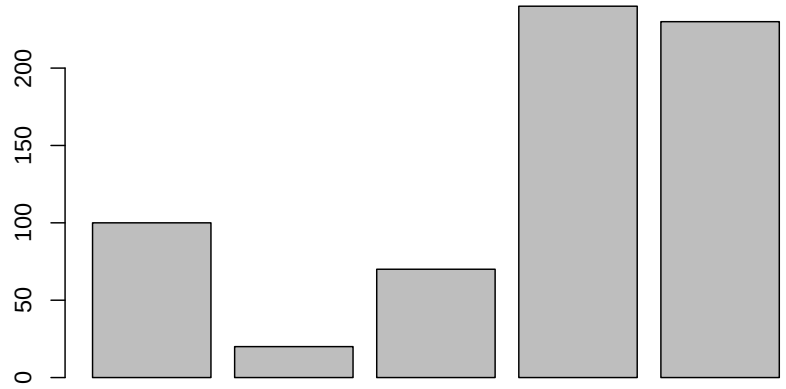
En R, estas gráficas se realizan meidante las siguientes funciones:

?barplot() ?boxplot() ?hist()

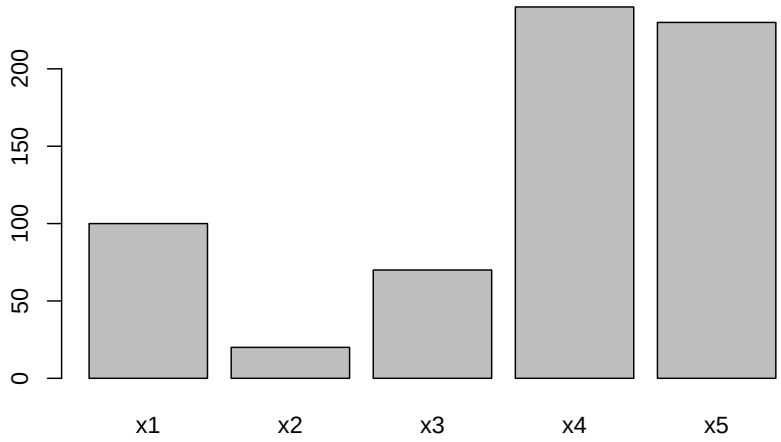
Vamos a explorar estas funciones.

Si queremos ver el número de registros de nuestras variables, podemos hacerlo mediante una gráfica de barras. Para poder graficarlo, necesitamos un vector o matriz que conenga valores numéricos.

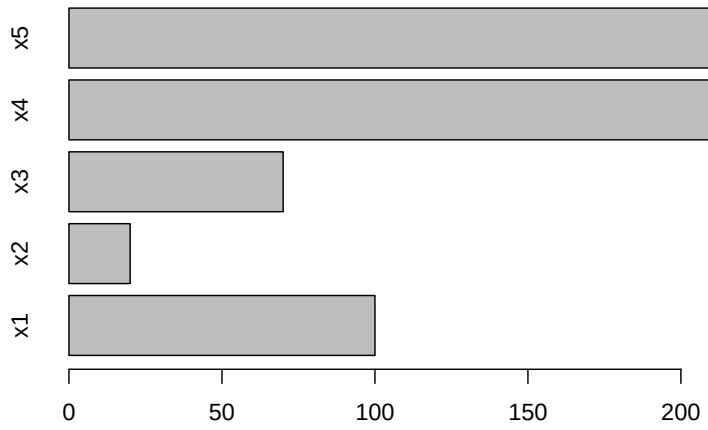
```
x <- c(100,20,70,240,230)

barplot(x)
```



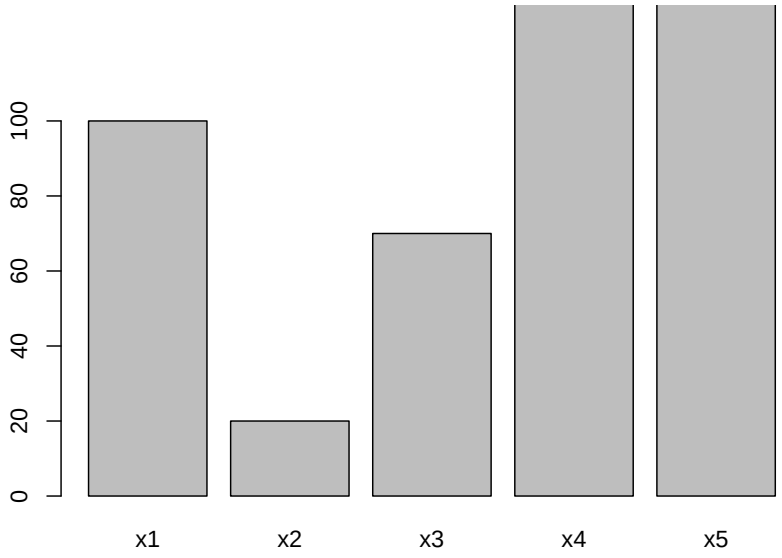

```
barplot(x, names.arg = c("x1", "x2", "x3", "x4", "x5")) #Asignamos nombres a cada barr
```




```
barplot(x, names.arg = c("x1", "x2", "x3", "x4", "x5"), horiz = TRUE) #Podemos invertir los ejes
```




```
barplot(x, names.arg = c("x1", "x2", "x3", "x4", "x5"), ylim = c(0,100)) #Podemos limitar el valo
```



Realice la misma gráfica anterior pero con colores, nombres en los ejes y título.

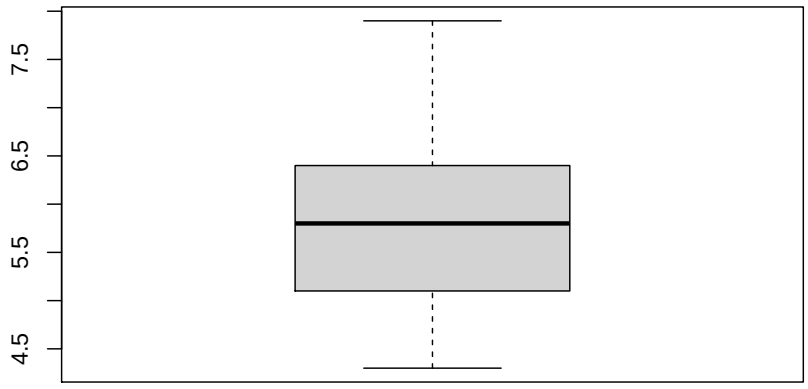
Mediante boxplot podemos hacer gráficos sencillos que nos muestren datos estadísticos de nuestros datos

```
data(iris)
```

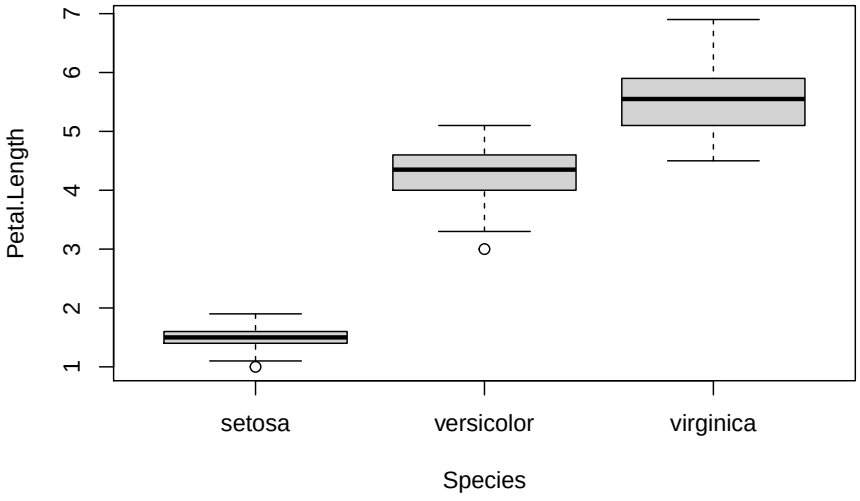
Cargamos un set de datos predeterminado en R y observamos su contenido.

El set de datos iris se trata de medidas del sépalo y pétalo de distintas especies de plantas.

```
boxplot(iris$Sepal.Length) # En este boxplot podemos ver la mediana (linea negra centr
```




```
boxplot(Petal.Length ~ Species, data = iris) #Podemos graficar los estadisticos para cada grupo
```

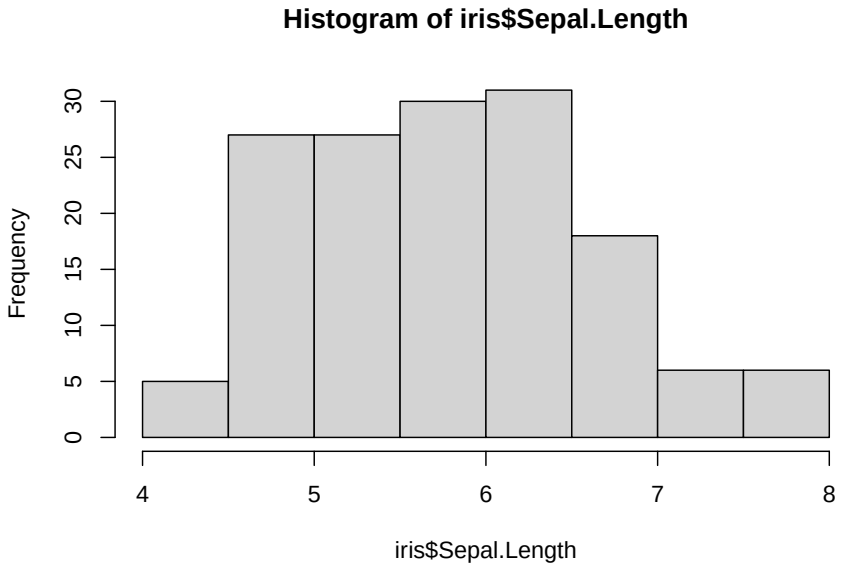


¿Qué pasa si invertimos las variables?

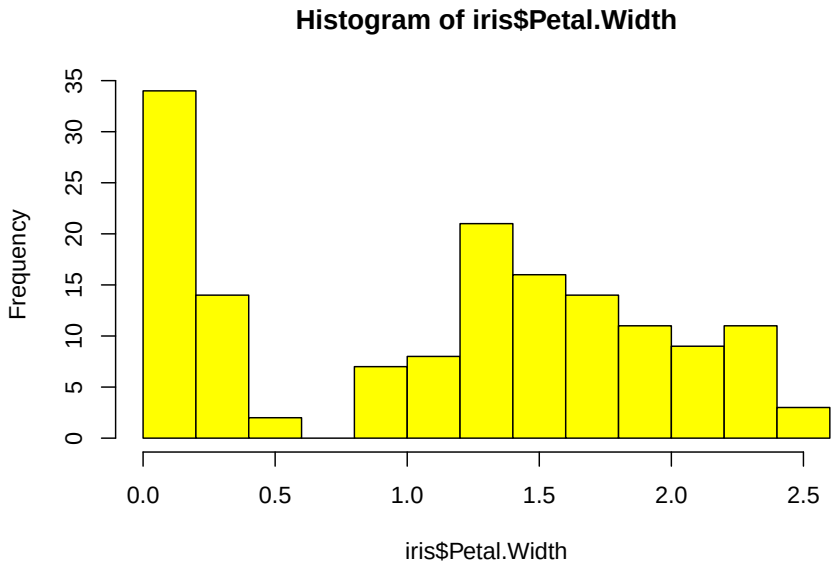
```
boxplot(Species ~ Petal.Length, data = iris)
```

Finalmente, mediante el gráfico de histograma podemos explorar frecuencias de valores.

```
hist(iris$Sepal.Length) # En el grafico, el observamos la frecuencia (eje y) y el rang
```




```
hist(iris$Petal.Width, col = "yellow")
```



Explore los diferentes argumentos dentro de cada función.

3.4 Función par()

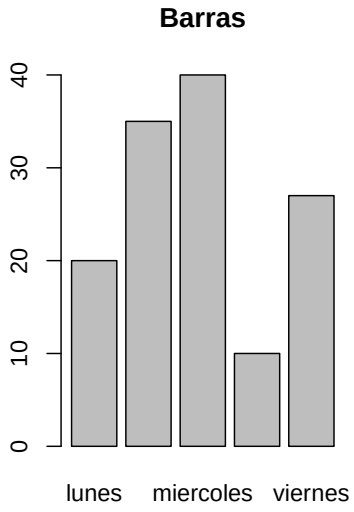
Mediante la función par() podemos ajustar distintos parametros de nuestra gráfica, asi como mostrar mas de una gráfica al tiempo, colores, texto y muchas otras características

?par

```
opar <- par() #Antes de empezar a utilizar par, guardamos la configuración por defecto en un vec

dev.off() #O podemos cerrar los plots creados anteriormente para reestablecer la configuración
- null device
-           1
dat_dias <- data.frame(
  dias = c("lunes", "martes", "miercoles", "jueves", "viernes"),
  ganancias = c(20,35,40,10,27))
```

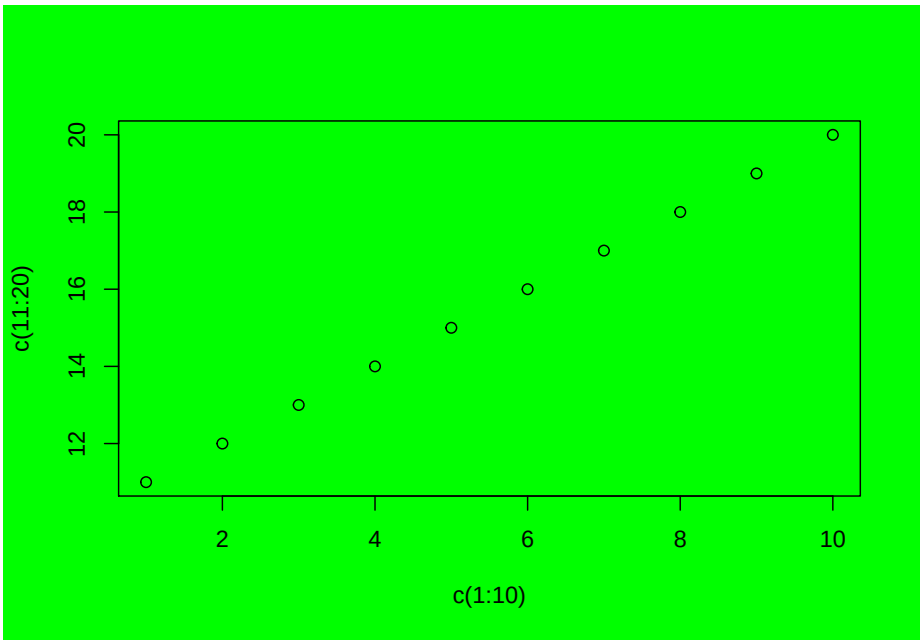
Realizaremos dos gráficas al tiempo mediante el argumento mfrow

```
par(mfrow = c(1,2))
barplot(dat_dias$ganancias, main = "Barras", names.arg = dat_dias$dias)
pie(dat_dias$ganancias, labels = dat_dias$dias, main = "Torta", clockwise = T)
```



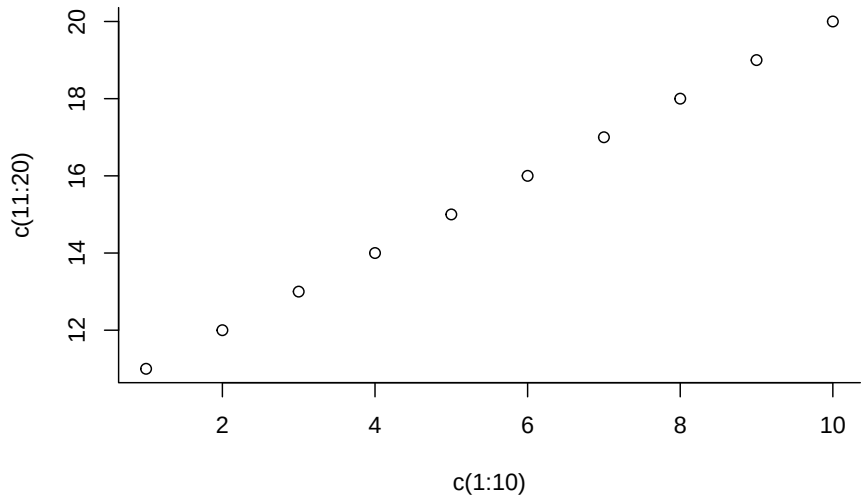
Utilice el argumento bg para cambiar el color de fondo del siguiente plot

```
plot(x = c(1:10), y = c(11:20))

par(mfrow = c(1,1), bg = "green")
plot(x = c(1:10), y = c(11:20))
```



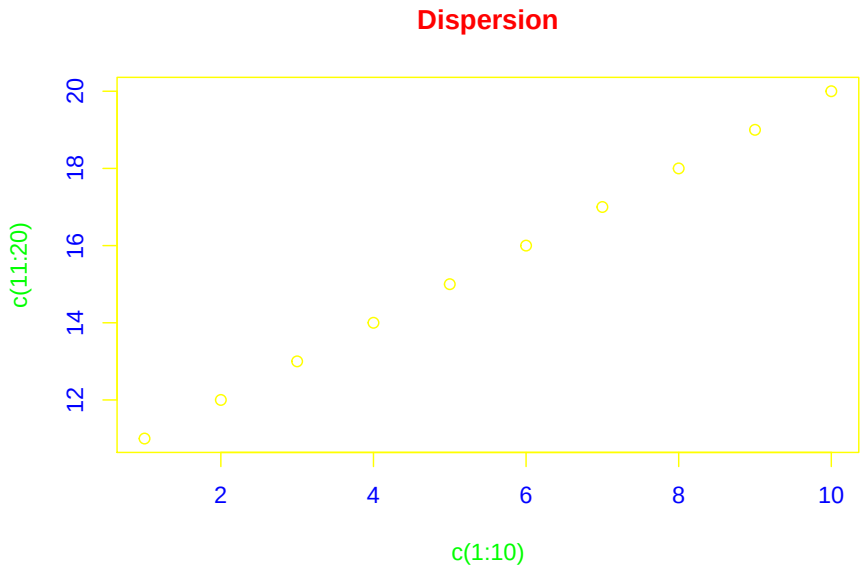
Mediante bty podemos cambiar la forma de la caja del plot:

```
par(mfrow = c(1,1), bg = "white", bty = "l") #Pruebe cambiando "l" por o,7,c,u,],n
plot(x = c(1:10), y = c(11:20))
```



Podemos agregar colores a los diferentes elementos del gráfico

```
par(mfrow = c(1,1), bg = "white", col.axis = "blue", col.lab = "green", col.main = "red
plot(x = c(1:10), y = c(11:20), main = "Dispersion")
```



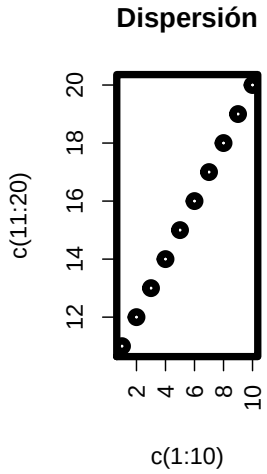
Para cambiar la fuente del texto:

- font.axis = valor numérico
- font.lab
- font.main
- font.sub

Explore los siguientes argumentos disponibles en par(), explique su función y utilícelas en una gráfica

- las
- lty
- lwd
- fig
- mai
- pin
- new

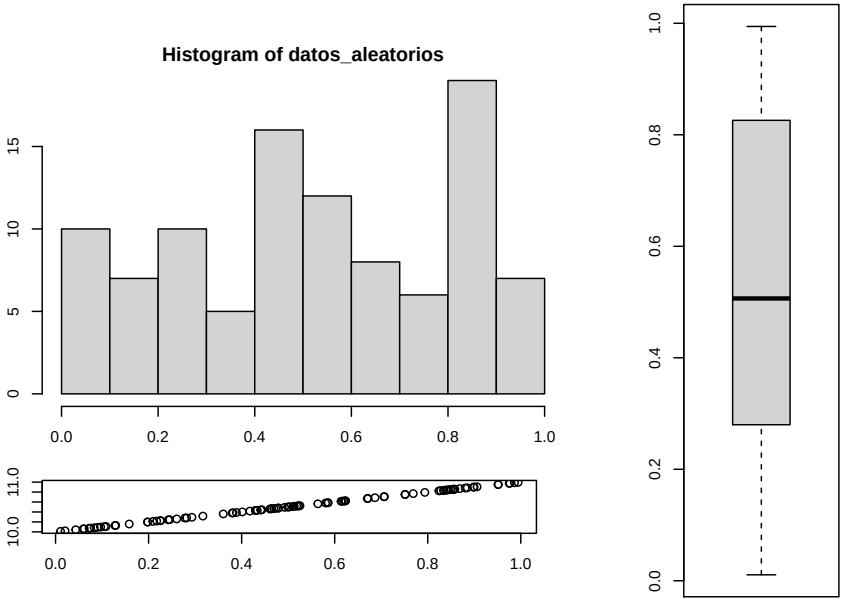
```
par(mfrow = c(1,1), bg = "white", las = 3, lty = 3, lwd = 5, pin = c(1,2))
plot(x = c(1:10), y = c(11:20), main = "Dispersión")
```



3.4.1 Gráficos compuestos

Usando algunas de las funciones anteriores, podemos ajustar varias figuras en un plot

```
datos_aleatorios <- runif(100)

par(cex=0.7, mai=c(0.1,0.1,0.2,0.1), fig=c(0.1,0.7,0.3,0.9))
hist(datos_aleatorios) # Configuramos el tamaño de nuestro histograma y lo graficamos

par(fig=c(0.8,1,0,1), new=TRUE) # Configuramos la posición de un boxplot y agregamos new para que
boxplot(datos_aleatorios)

par(fig=c(0.1,0.67,0.1,0.25), new=TRUE)
plot(x = datos_aleatorios, y = datos_aleatorios + 10) # Y añadimos un gráfico de dispersion de ot
```


Histogram of datos_aleatorios
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
10.0
11.0
0.0
0.2
0.4
0.6
0.8
1.0