Índice general

Capítulo 1

Intro R

1.1 Clase 1

Presentación del curso. Entrega de programas. Conformar grupos (si hay muchos estudiantes). Conocer el nivel de los estudiantes en la programación de R.

1.2 Clase 2

Programación básica en R.

- Creación de objetos

```
x<-y<-5
x=y=5
a<-c(3,5,10)
b<-c(2:20)
```

- Operaciones básicas

```
3+5
```

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

```
2-10
```

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

```
2*3
```

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

```
2/5
```

```
## [1] 0.4
```

```
3^-1
```

```
## [1] 0.3333333
```

- Operaciones lógicas: `>`, `>=`, `<`, `<=`, `==`, `!=`, `&`, `|`. Ejemplo:

```
3>5
```

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

```
3<4
```

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

```
2==2.01
```

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

```
3==3&2<2
```

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

```
3==3|2<2
```

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

- Funciones matemáticas básicas Función logarítmica: `log()`, `log10()`, `log2()`. Ejemplos:

```
log(2)
```

```
## [1] 0.6931472
```

```
log(2, base=10)
```

```
## [1] 0.30103
```

```
log(2,10)
```

```
## [1] 0.30103
```

```
log10(2)
```

```
## [1] 0.30103
```

```
log(5,base=2)
```

```
## [1] 2.321928
```

```
log2(5)
```

```
## [1] 2.321928
```

Función exponencial: `exp()`.

```r
exp(1)
```

```
## [1] 2.718282
```

```r
exp(-2)
```

```
## [1] 0.1353353
```

Raíz cuadrada: `sqrt()`
Valor absoluto: `abs()`

- Funciones estadísticas básicas
  Para un conjunto de datos:
  Media: `mean()`, mediana: `median()`, varianza: `var()`, desviación estándar: `sd()`, percentiles: `quantile()`, algunas medidas descriptivas: `summary()`
  Para dos conjunto de datos de la misma longitud:
  Covarianza: `cov(,)`, correlación: `cor(,)`

- Otraa funciones muy útiles: `rep()`, `seq()` para crear repeticiones y secuencias. `orden()` y `sort()` para cuestiones del órden

```r
rep(3,5)
```

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

```r
rep("A",5)
```

```
## [1] "A" "A" "A" "A" "A"
```

```r
rep(NA,10)
```

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

```r
c(rep("A",4),rep("B",6))
```

```
##  [1] "A" "A" "A" "A" "B" "B" "B" "B" "B" "B"
```

```r
a<-c(1:3)
rep(a,4)
```

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

```r
seq(1,10,by=1)
```

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

```r
seq(0,1,by=0.1)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```r
seq(0,1,by=0.11)

##  [1] 0.00 0.11 0.22 0.33 0.44 0.55 0.66 0.77 0.88 0.99

seq(0, 1, length.out = 13)

##  [1] 0.00000000 0.08333333 0.16666667 0.25000000 0.33333333 0.41666667
##  [7] 0.50000000 0.58333333 0.66666667 0.75000000 0.83333333 0.91666667
## [13] 1.00000000

d<-c(3,-1,0,4)
sort(d)

## [1] -1  0  3  4

order(d)

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

- Extraer componentes de un vector Se utiliza los corchetes cuadradas `[ ]`, por ejemplo

```r
a<-c(-3:11)
a

##  [1] -3 -2 -1  0  1  2  3  4  5  6  7  8  9 10 11

a[4]

## [1] 0

a[-2]

##  [1] -3 -1  0  1  2  3  4  5  6  7  8  9 10 11

a[c(1,6,7)]

## [1] -3  2  3

a[a>0]

##  [1]  1  2  3  4  5  6  7  8  9 10 11
```

Paquetes en R

- Descarga de un paquete: `install.packages("name")`
- Uso de un paquete: `library(name)`
- Ayuda sobre un paquete

Ver los atributos de un objeto en R: `names(), $`

```
library(rpart)
USArrests # a veces es necesario usar data(USArrests)
```

```
##                Murder Assault UrbanPop Rape
## Alabama          13.2     236       58 21.2
## Alaska           10.0     263       48 44.5
## Arizona           8.1     294       80 31.0
## Arkansas          8.8     190       50 19.5
## California        9.0     276       91 40.6
## Colorado          7.9     204       78 38.7
## Connecticut       3.3     110       77 11.1
## Delaware          5.9     238       72 15.8
## Florida          15.4     335       80 31.9
## Georgia          17.4     211       60 25.8
## Hawaii            5.3      46       83 20.2
## Idaho             2.6     120       54 14.2
## Illinois         10.4     249       83 24.0
## Indiana           7.2     113       65 21.0
## Iowa              2.2      56       57 11.3
## Kansas            6.0     115       66 18.0
## Kentucky          9.7     109       52 16.3
## Louisiana        15.4     249       66 22.2
## Maine             2.1      83       51  7.8
## Maryland         11.3     300       67 27.8
## Massachusetts     4.4     149       85 16.3
## Michigan         12.1     255       74 35.1
## Minnesota         2.7      72       66 14.9
## Mississippi      16.1     259       44 17.1
## Missouri          9.0     178       70 28.2
## Montana           6.0     109       53 16.4
## Nebraska          4.3     102       62 16.5
## Nevada           12.2     252       81 46.0
## New Hampshire     2.1      57       56  9.5
## New Jersey        7.4     159       89 18.8
## New Mexico       11.4     285       70 32.1
## New York         11.1     254       86 26.1
## North Carolina   13.0     337       45 16.1
## North Dakota      0.8      45       44  7.3
## Ohio              7.3     120       75 21.4
## Oklahoma          6.6     151       68 20.0
## Oregon            4.9     159       67 29.3
## Pennsylvania      6.3     106       72 14.9
## Rhode Island      3.4     174       87  8.3
## South Carolina   14.4     279       48 22.5
## South Dakota      3.8      86       45 12.8
## Tennessee        13.2     188       59 26.9
## Texas            12.7     201       80 25.5
## Utah              3.2     120       80 22.9
## Vermont           2.2      48       32 11.2
## Virginia          8.5     156       63 20.7
```

```
## Washington        4.0     145       73 26.2
## West Virginia     5.7      81       39  9.3
## Wisconsin         2.6      53       66 10.8
## Wyoming           6.8     161       60 15.6

names(USArrests)

## [1] "Murder"   "Assault"  "UrbanPop" "Rape"

USArrests$Murder

##  [1] 13.2 10.0  8.1  8.8  9.0  7.9  3.3  5.9 15.4 17.4  5.3  2.6 10.4  7.2
## [15]  2.2  6.0  9.7 15.4  2.1 11.3  4.4 12.1  2.7 16.1  9.0  6.0  4.3 12.2
## [29]  2.1  7.4 11.4 11.1 13.0  0.8  7.3  6.6  4.9  6.3  3.4 14.4  3.8 13.2
## [43] 12.7  3.2  2.2  8.5  4.0  5.7  2.6  6.8

USArrests$Mur

##  [1] 13.2 10.0  8.1  8.8  9.0  7.9  3.3  5.9 15.4 17.4  5.3  2.6 10.4  7.2
## [15]  2.2  6.0  9.7 15.4  2.1 11.3  4.4 12.1  2.7 16.1  9.0  6.0  4.3 12.2
## [29]  2.1  7.4 11.4 11.1 13.0  0.8  7.3  6.6  4.9  6.3  3.4 14.4  3.8 13.2
## [43] 12.7  3.2  2.2  8.5  4.0  5.7  2.6  6.8
```

Ayuda en R

- Usar `?` para comandos existentes en R
- Usar `??` para averiguar sobre algún tema o si no está seguro de la forma de escribir el comando

```
?array
??kalman
help(array)
help(kalman)

## No documentation for 'kalman' in specified packages and libraries:
## you could try '??kalman'
```

Ejercicio

1. Calcular los valores de las siguientes expresiones
   - $$\frac{1}{\sqrt{10\pi}} \exp\left\{-\frac{(4-3)^2}{5}\right\}$$
   - $x = -2$, $y = \sqrt{|x|+1}$,
     $$z = \text{máx}\{\cos y, \log(x^2-1)\} + [y]$$
2. Descargar el paquete `TeachingSamplings` y para la base de datos `Lucy` y calcular
   - Las medidas descriptivas como la media, mediana, desviación estándar, cuartiles de la variable Impuesto.

- Las medidas descriptivas como la media, mediana, desviación estándar, cuartiles de la variable Taxes segregado según si envía correo SPAM o no.
- Las medidas descriptivas como la media, mediana, desviación estándar, cuartiles de la variable Income segregado según si la variable Taxes toma valores mayores o menores de 3.5.

1.3 Clase 3, Matrices

1.3.1 Creación de un vector, matriz, arreglo

Para crea una matriz utilizamos la función `matrix`.

Por ejemplo

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

crea una matriz de 2 filas y 3 columnas, las entradas se rellenan por columna.

```
A<-matrix(c(1,2,5,2,4,5),2,3,byrow=T)
```

crea una matriz de 2 filas y 3 columnas, las entradas se rellenan por fila.

Si el número de entradas de la matriz es mayor al número de valores provistos, R rellena toda la matriz repitiendo los valores provistos. Por ejemplo, vea la salida de

```
A<-matrix(c(1,2,5,2,4,5),2,5)

## Warning in matrix(c(1, 2, 5, 2, 4, 5), 2, 5): data length [6] is not a sub-multiple
or multiple of the number of columns [5]
```

Si el número de entradas de la matriz es menor al número de valores provistos, R solo toma los primeros valores necesarios para llenar la matriz. Por ejemplo, vea la salida de

```
A<-matrix(c(1,2,5,2,4,5),2,2)
```

Si no especifica ni el número de filas ni el número de columnas en el comando `matrix`, R crea automáticamente un vector columna. Vea la salida de

```
A<-matrix(c(1,2,5,2,4,5))
```

R toma los vectores creados con `c()` como vectores filas.

Un arreglo es como la generalización de una matriz que permite más de dos dimensiones (fila y columna), la creación de un arreglo se lleva a cabo usando`array` especificando los detalles de la dimensión. Vea la salida de

```
B<-array(c(1:12),c(2,3,2))
```

Ojo: a diferencia de `matrix`, en `array` se debe especificar los órdenes dentro de `c()`.

Algunas veces, necesitamos crear matrices a partir de matrices ya existentes, es decir, necesitamos unir matrices. Podemos unir matrices por filas o por columnas.

```
A<-matrix(c(1,2,5,2,4,5),2,3)
B<-matrix(c(10,20),2,1)
C<-matrix(c(100,200),1,3)

## Warning in matrix(c(100, 200), 1, 3): data length [2] is not a sub-multiple or multiple
of the number of columns [3]

cbind(A,B)

##      [,1] [,2] [,3] [,4]
## [1,]    1    5    4   10
## [2,]    2    2    5   20

rbind(A,C)

##      [,1] [,2] [,3]
## [1,]    1    5    4
## [2,]    2    2    5
## [3,]  100  200  100
```

Para crear una matriz diagonal, utilizamos `diag()`, por ejemplo `diag(c(2,3,0))` crea una matriz de tamaño 3×3 donde los elementos en la diagonal son 2, 3, y 0. (Si aplicamos `diag()` a una matriz, qué calcula?)

1.3.2 Operaciones básicas

La suma y resta entre matrices sólo se puede efectuar si las matrices son del mismo tamaño (para ver el tamaño de una matriz, usa `dim()`).

Una matriz puede sumar, restar, multiplicar, dividir, elevar potencia con un número escalar, y la operación se efectúa para todos las entradas de la matriz. Vea la salida de

```
A<-matrix(c(1,2,5,2,4,5),2,3)
A

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

A+1

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

A-3

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

```
A*0.5

##      [,1] [,2] [,3]
## [1,]  0.5  2.5  2.0
## [2,]  1.0  1.0  2.5

A/4

##      [,1] [,2] [,3]
## [1,] 0.25 1.25 1.00
## [2,] 0.50 0.50 1.25

A^3

##      [,1] [,2] [,3]
## [1,]    1  125   64
## [2,]    8    8  125
```

La multiplicación entre dos matrices A y B es válida si el número de columnas de A es igual al número de filas de B, en este caso la operación se lleva a cabo usando **%*%**, vea la salida de

```
A<-matrix(c(1,2,5,2,4,5),2,3)
B<-matrix(c(0,1,2))
A%*%B

##      [,1]
## [1,]   13
## [2,]   12

C<-matrix(c(0,1,2),1,3)
A%*%C

## Error in A%*% C: non-conformable arguments
```

Cuando utilizamos * para la multiplicación de matrices, éstas deben tener la misma dimensión y el producto es de componente a componente. Vea la salida de

```
A<-matrix(c(1,2,5,5),2,2)
B<-matrix(c(0,1,2,1),2,2)
C<-matrix(c(0,1,2),1,3)
A*B

##      [,1] [,2]
## [1,]    0   10
## [2,]    2    5

A*C

## Error in A * C: non-conformable arrays
```

Cuando aplicamos las funciones de `sum()` y `mean()` a una matriz, obtenemos la suma y la media de todos los valores que contiene la matriz. Otras funciones útiles son `colSums()` y `rowSums()` que arrojan las sumas por columnas y por filas, respectivamente. Vea la salida de

```
A<-matrix(c(1,2,5,5),2,2)
sum(A)

## [1] 13

mean(A)

## [1] 3.25

colSums(A)

## [1]  3 10

rowSums(A)

## [1] 6 7
```

1.3.3 Otras operaciones

- Calcular la inversa de una matriz es muy importante, y lo hacemos usando `solve()` (tenga en cuenta que solo algunas matrices cuadradas tienen inversa).

```
solve(A)

##      [,1] [,2]
## [1,] -1.0  1.0
## [2,]  0.4 -0.2

solve(C)

## Error in solve.default(C): á'(1 x 3) must be square
```

- El determinante de una matriz se calcula con `det()`
- La transpuesta de una matriz se calcula con `t()`
- Los valores y vectores propios se calcula con `eigen()`
- El producto kronecker entre dos matrices se lleva a cabo con `kronecker(,)`

```
A

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

```
B

##      [,1] [,2]
## [1,]    0    2
## [2,]    1    1

kronecker(A,B)

##      [,1] [,2] [,3] [,4]
## [1,]    0    2    0   10
## [2,]    1    1    5    5
## [3,]    0    4    0   10
## [4,]    2    2    5    5
```

- La descomposición en valores singulares se lleva a cabo con `svd()`
- Para extraer submatrices, utilizamos `[ ]`, teniendo en cuenta las posiciones de las filas y columnas

```
kronecker(A,B)[1,2]

## [1] 2

kronecker(A,B)[1:3,2]

## [1] 2 1 4

kronecker(A,B)[c(1,4),2]

## [1] 2 2

kronecker(A,B)[c(1,4),c(2,4)]

##      [,1] [,2]
## [1,]    2   10
## [2,]    2    5

kronecker(A,B)[c(1,4),]

##      [,1] [,2] [,3] [,4]
## [1,]    0    2    0   10
## [2,]    2    2    5    5
```

1.3.4 Ejercicios

1. Crea la matriz de identidad de tamaño 20.

2. Para la matriz

$$A = \begin{pmatrix} 1 & 0 & 2 & -1 \\ 2 & -1 & 3 & 0 \\ 3 & -1 & 5 & -1 \\ 1 & 5 & 2 & 1 \end{pmatrix}$$

- Diga si A es invertible
- Encuentra la forma de calcular el rango de A
- Verifique en R que la propiedad que afirma que i) el determinante de una matriz es el producto de sus valores propios, ii) la traza de una matriz es la suma de sus valores propios.
- Encuentre los valores y vectores propios de A.
- A partir de A, construya la siguiente matriz diagonal por bloques

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & -1 & 0 & 0 \\ 0 & 0 & 5 & -1 \\ 0 & 0 & 2 & 1 \end{pmatrix}$$

3. Para los datos de Lucy del paquete TeachingSampling,

- crea una matriz de datos que contienen las variables de `Income`, `Employees` y `Taxes` donde cada fila de la matriz contiene los valores de estas tres variables para una empresa.
- Crea un vector que contiene los promedios de las tres variables, $\bar{x}$
- Calcula la matriz de varianzas y covarianzas muestrales de las tres variables S
- Sea $\mu = (450, 65, 14)'$, calcular

$$(\bar{x} - \mu)' S^{-1} (\bar{x} - \mu)$$

4. Encuentre una solución para cada uno de los sistemas de ecuaciones lineales dados a continuación

(a)

$$\begin{aligned} 3x_1 - 2x_2 - x_3 &= 1 \\ -x_1 + 2x_2 + 2x_3 &= 2 \\ x_1 + 2x_2 + 3x_3 &= 4 \end{aligned}$$

(b)

$$\begin{aligned} x_1 - 2x_2 + x_3 + x_4 &= 2 \\ 3x_1 + 2x_2 - 2x_4 &= -8 \\ 4x_2 - x_3 - x_4 &= 1 \\ 5x_1 + 3x_3 - x_4 &= -3 \end{aligned}$$

5. Para las siguientes matrices,

(a)

$$\begin{pmatrix} -1 & 1 & 0 \\ 2 & -3 & 1 \\ 1 & -2 & 1 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 2 & 1 & -2 \end{pmatrix}$$

(c)

$$\begin{pmatrix} 5 & -2 \\ 3 & -1 \\ 0 & 1 \end{pmatrix}$$

(d)

$$\begin{pmatrix} 1 \\ 0 \\ 3 \end{pmatrix}$$

calcule la inversa generalizada de las siguientes matrices y verifique que cumpla las 4 condiciones

- $AA^{-}A = A$
- $A^{-}AA^{-} = A^{-}$
- AA^{-} es simétrica
- $A^{-}A$ es simétrica

donde A^{-} denota la inversa generalizada de la matriz A.

6. Para ajustar una línea de regresión $y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik}$ entre una variable de interés Y y k variables explicativas $X_1, \cdots, X_k$, los estimadores de mínimos cuadrados de los coeficientes $\beta = (\beta_0, \beta_1, \cdots, \beta_k$ están dados por $(X'X)^{-1}X'Y$ donde Y es el vector que contiene las n observaciones de la variable Y y X está dado por

$$X = \begin{pmatrix} 1 & X_{11} & \cdots & X_{1k} \\ 1 & X_{21} & \cdots & X_{2k} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & X_{n1} & \cdots & X_{nk} \end{pmatrix}$$

Para la base de datos `Lucy`, utilice la variable `Taxes` como la variable de interés y las demás variables CUANTITATIVAS como variables explicativas y obtenga la ecuación de la línea de regresión múltiple.

7.
- Crea un arreglo de números de tamaño $2 \times 3 \times 4$ donde la matriz en las entradas $[1, ,]$ sea $\begin{pmatrix} 2 & 0 & -2 & 0 \\ 1 & 2 & -1 & -10 \\ 0 & 0 & 2 & 1 \end{pmatrix}$ y la entrada $[2, ,]$ sea $\begin{pmatrix} -0.1 & 0.5 & -2 & 10 \\ 10 & 0 & -5 & 10 \\ 1 & -1 & 20 & 1 \end{pmatrix}$
- Para el anterior arreglo, reemplace todas entradas negativas por -100.

1.4 Clase 5. Ciclos

Los ciclos son herramientas fundamentales para realizar programaciones estadísticas. Los principales comandos son:

1.4.1 for

El comando `for` puede ser utilizado para repetir un número grande de iteraciones, como por ejemplo, llenar un vector o una matriz, o también sirve para crear una secuencia de valores de forma recursiva.

El uso del comando es

```
for(contador){
accion
}
```

Ejemplo 1.4.1. Crear una sucesión de 100 números comenzando en 1 y el incremento entre dos números consecutivos sea 3. Es decir, 1, 4, 7, 10,...

```
x<-rep(NA,100)
x[1]<-1
for(i in 2:100){
x[i]<-x[i-1]+3
}
x
```

```
##   [1]   1   4   7  10  13  16  19  22  25  28  31  34  37  40  43  46  49
##  [18]  52  55  58  61  64  67  70  73  76  79  82  85  88  91  94  97 100
##  [35] 103 106 109 112 115 118 121 124 127 130 133 136 139 142 145 148 151
##  [52] 154 157 160 163 166 169 172 175 178 181 184 187 190 193 196 199 202
##  [69] 205 208 211 214 217 220 223 226 229 232 235 238 241 244 247 250 253
##  [86] 256 259 262 265 268 271 274 277 280 283 286 289 292 295 298
```

Los mismos resultados se pueden obtener con lo siguiente

```
x<-cumsum(c(1,rep(3,99)))
```

¿Por qué?

Un `for` puede estar dentro de otro `for`

Ejemplo 1.4.2. Crear una matriz de tamaño 10×10 donde el elemento i, j de la matriz sea $|i - j|$

```
A<-matrix(NA,10,10)
for(i in 1:10){
  for(j in 1:10){
    A[i,j]<-abs(i-j)
  }
}
A
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    1    2    3    4    5    6    7    8     9
##  [2,]    1    0    1    2    3    4    5    6    7     8
##  [3,]    2    1    0    1    2    3    4    5    6     7
##  [4,]    3    2    1    0    1    2    3    4    5     6
##  [5,]    4    3    2    1    0    1    2    3    4     5
##  [6,]    5    4    3    2    1    0    1    2    3     4
##  [7,]    6    5    4    3    2    1    0    1    2     3
##  [8,]    7    6    5    4    3    2    1    0    1     2
##  [9,]    8    7    6    5    4    3    2    1    0     1
## [10,]    9    8    7    6    5    4    3    2    1     0
```

Recomendación:

- Poner las instrucciones que no involucre al contador del `for` o no dependa de las iteraciones anteriores fuera del `for`. Por ejemplo, no se recomienda escribir:

```
# Simular 20 valores de las distribuciones N(1,1), N(2,1), ..., N(20,1)
x<-c()
for(i in 1:20){
sigma<-1
x[i]<-rnorm(1,i,sigma)
}
```

  Es mejor usar:

```
# Simular 20 valores de las distribuciones N(1,1), N(2,1), ..., N(20,1)
x<-c()
sigma<-1
for(i in 1:20){
x[i]<-rnorm(1,i,sigma)
}
```

- No usar el nombre del contador para otros objetos.

1.4.2 Ejercicios

1. Crear una sucesión de 500 números comenzando en 0 y $a_n = a_{n-1} - 4$.
2. Crear una sucesión de 100 números comenzando en 1 y $a_n = 2a_{n-1}$.
3. Crea la siguiente sucesión (de longitud 150) de números cuadrados: 1, 4, 9, 16, 25, 36, 49, 64, ... (hágalo de dos formas, con for y sin for)
4. Crea la siguiente sucesión (de longitud 150) de números (números triangulares): 1, 3, 6, 10, 15, 21, 28, 36, 45, ...
5. Crea una secuencia de 50 números donde $a_1 = 5$, $a_n = a_1 + 2(n-1)$.
6. Crea 120 números de Fibonacci.
7. Crea 100 números de la sucesión de Padovan ($a_1 = a_2 = a_3 = 1$, $a_n = a_{n-2} + a_{n-3}$).
8. Crea 100 números de Perrin ($a_1 = 3$, $a_2 = 0$, $a_3 = 2$, $a_n = a_{n-2} + a_{n-3}$).
9. Para las variables cuantitativas de los datos de `Lucy`, usando `for` crea una matriz de dos filas y tantas columnas como el número de variables, donde las dos entradas de cada columna corresponda a la media y la desviación estándar de la variable.
10. Para una secuencia de datos y_t con $t = 1, \cdots, T$, la autocorrelación de rezago k se define como

$$\rho_k = \frac{\sum_{t=1}^{T-k}(y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})^2}$$

  Para los datos `AirPassengers` en `R` (no se necesita ningún paquete específico), calcular $\rho_0, \cdots, \rho_{10}$, y calcular el determinante de la siguiente matriz

$$\begin{pmatrix} 1 & \rho_1 & \rho_2 & \cdots & \rho_9 & \rho_{10} \\ \rho_1 & 1 & \rho_1 & \cdots & \rho_8 & \rho_9 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \rho_{10} & \rho_9 & \rho_8 & \cdots & \rho_1 & 1 \end{pmatrix}$$

1.5 Clase 6. Ciclos II

1.5.1 if

El comando `if` permite ejecutar determinadas acciones según si cumplen o no condiciones lógicas. El uso del comando es

```
if(condicion1){
  accion1
}
if(condicion2){
  accion2
}
```

Ejemplo 1.5.1. Dado $x = (-1, 0, -2, 1, 5)$, reemplazar los valores negativos de x por el valor absoluto.

```
x<-c(-1,0,-2,1,5)
for(i in 1:length(x)){
  if(x[i]<0){x[i]<-abs(x[i])}
}
x
```

```
## [1] 1 0 2 1 5
```

Ejemplo 1.5.2. Dado $x = (-1, 0, -2, 1, 5)$, reemplazar los valores negativos de x por el valor absoluto y reemplazar los valores positivos por su cuadrado.

```
x<-c(-1,0,-2,1,5)
for(i in 1:length(x)){
  if(x[i]<0){x[i]<-abs(x[i])}
  if(x[i]>0){x[i]<-x[i]^2}
}
x
```

```
## [1]  1  0  4  1 25
```

Lo anterior NO es correcto. Lo correcto puede ser

```
x<-y<-c(-1,0,-2,1,5)
for(i in 1:length(x)){
  if(x[i]<0){y[i]<-abs(x[i])}
  if(x[i]>0){y[i]<-x[i]^2}
}
y
```

```
## [1]  1  0  2  1 25
```

Ejemplo 1.5.3. Dado $x = (-1, 0, -2, 1, 5)$, reemplazar los valores negativos de x por el valor absoluto y reemplazar los otros valores por -10.

```
x<-y<-c(-1,0,-2,1,5)
for(i in 1:length(x)){
  if(x[i]<0){y[i]<-abs(x[i])}
  else{y[i]<-100}
}
y

## [1]   1 100   2 100 100
```

lo cual es equivalente a

```
x<-y<-c(-1,0,-2,1,5)
for(i in 1:length(x)){
  if(x[i]<0){y[i]<-abs(x[i])}
  if(x[i]>=0){y[i]<-100}
}
y

## [1]   1 100   2 100 100
```

Otro ejemplo.

Ejemplo 1.5.4. Dado $x = (-1, 0, -2, 1, 5)$, reemplazar los valores negativos de x por el valor absoluto, reemplazar los valores positivos por su cuadrado y reemplazar los valores nulos por 100.

```
x<-y<-c(-1,0,-2,1,5)
for(i in 1:length(x)){
  if(x[i]<0){y[i]<-abs(x[i])}
  if(x[i]>0){y[i]<-x[i]^2}
  else{y[i]<-100}
}
y

## [1] 100 100 100   1  25
```

Lo anterior no sive, es decir, `else` sólo puede ser usado con un único `if`. Tendríamos que usar

```
x<-y<-c(-1,0,-2,1,5)
for(i in 1:length(x)){
  if(x[i]<0){y[i]<-abs(x[i])}
  if(x[i]>0){y[i]<-x[i]^2}
  if(x[i]==0){y[i]<-100}
}
y

## [1]   1 100   2   1  25
```

En `R` discretizar una variabl cuantitativa es muy sencillo. Por ejemplo, suponga que $a = (1.4, 2, 2.5, 0.6, 0.8, 1.6, 5.2, 10.1)$ corresponden a ingreso mensual (medida en millón de pesos) de 8 personas y suponga que queremos

volver a cualtitativa clasificando los individuos en grupos de ingreso alto y bajo según si su ingreso es mayor a 2 millones o no. En principio, tendríamos que usar los siguientes códigos

```
a<-categ<-c(1.4,2,2.5,0.6,0.8,1.6,5.2,10.1)
for(i in 1:length(a)){
  if(a[i]<2){categ[i]<-0}
  if(a[i]>=2){categ[i]<-1}
}
categ

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

Sin embargo, los siguientes comandos son más simples

```
as.double(a>=2)

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

Si necesitamos reemplzar todos los ingresos mayores a 5 millones por 0, podemos usar

```
a[a>5]<-0
a

## [1] 1.4 2.0 2.5 0.6 0.8 1.6 0.0 0.0
```

Si necesitmamos eliminar de `a` todos los valores menores a 2 millones,

```
a<-c(1.4,2,2.5,0.6,0.8,1.6,5.2,10.1)
a<-a[which((a<2)==FALSE)]
a

## [1]  2.0  2.5  5.2 10.1
```

Otro comando útil para reemplazar valores según si cumple o no cierta condición es `ifelse`, el uso de la función es

```
ifelse(condicion, valor si cumple la condicion, valor si no la cumple)
```

Por ejemplo, queremos que los valores de `a` mayores a 2 sea reemplazado por Alto y los otros reemplazods por Medio, podemos usar

```
a<-c(1.4,2,2.5,0.6,0.8,1.6,5.2,10.1)
ifelse(a>2,"Alto", "Medio")

## [1] "Medio" "Medio" "Alto"  "Medio" "Medio" "Medio" "Alto"  "Alto"
```

También podemos exigir que los valores mayores a 2 en `a` sean cambiados por 1000, y mantener sin cambiar los valores no mayores a 2

```r
a<-c(1.4,2,2.5,0.6,0.8,1.6,5.2,10.1)
ifelse(a>2,1000,a)

## [1]    1.4    2.0 1000.0    0.6    0.8    1.6 1000.0 1000.0
```

A veces, los datos pueden contener datos faltantes, que usualmente se denota con NA, queremos identificar cuáles entradas son NA, y tal vez eliminarlos

```r
a<-c(1.4,2,NA,0.6,NA,1.6,5.2,10.1)
is.finite(a)

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

which(is.finite(a)==F)

## [1] 3 5

is.na(a)

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

which(is.na(a)==T)

## [1] 3 5

a[is.finite(a)]

## [1]  1.4  2.0  0.6  1.6  5.2 10.1
```

1.5.2 while

`while` se utiliza cuando queremos que se ejecute algo siempre y cuando se cumple alguna condición lógica. El uso de la función es

```r
while(condicion){
  accion
}
```

Ejemplo 1.5.5. Queremos crear una sucesión de valores de forma $1/(x^2)$. como la sucesión converge a 0, entonces queremos crear todos valores desde 1 hasta 0.001 (prácticamente 0).

```r
a<-1
i<-1
while(min(a)>0.001){
  i<-i+1
  a<-c(a,1/(i^2))
}
a
```

```
##  [1] 1.0000000000 0.2500000000 0.1111111111 0.0625000000 0.0400000000
##  [6] 0.0277777778 0.0204081633 0.0156250000 0.0123456790 0.0100000000
## [11] 0.0082644628 0.0069444444 0.0059171598 0.0051020408 0.0044444444
## [16] 0.0039062500 0.0034602076 0.0030864198 0.0027700831 0.0025000000
## [21] 0.0022675737 0.0020661157 0.0018903592 0.0017361111 0.0016000000
## [26] 0.0014792899 0.0013717421 0.0012755102 0.0011890606 0.0011111111
## [31] 0.0010405827 0.0009765625
```

1.5.3 Ejercicios

1. Calcular el valor de la función $f(x)$ para los valores dados de x (en lo posible, escriba dos comandos para cada función, uno utilizando `if`, y otro utilizando `ifelse`)

   - $$f(x) = \begin{cases} 2x+5 & \text{si } x > -2 \\ 1 & \text{si no} \end{cases}$$

     valores de x: -5, -3, -2, 0, 1, 2, 6.

   - $$f(x) = \begin{cases} x^2 & \text{si } x \text{ es entero} \\ [x] & \text{si no} \end{cases}$$

     valores de x: 1, 1.2, 1.4, 1.6, 1.8, 2, 2.2, 2.4, 2.6, 2.8, 3, 3.2, 3.4, 3.6, 3.8, 4.

   - $$f(x,y) = \begin{cases} x+y & \text{si } x > 0 \text{ y } y > 0 \\ 0 & \text{si no} \end{cases}$$

     valores de (x, y): (-2,-1), (-2,0), (-2,1), (1,2), (1,-2), (3,2)

   - $$f(x,y) = \begin{cases} x+y & \text{si } x > 0 \text{ y } y > 0 \\ x-y & \text{si } x > 0 \text{ y } y < 0 \\ 0 & \text{si no} \end{cases}$$

     valores de (x, y): (-2,-1), (-2,0), (-2,1), (1,2), (1,-2), (3,2)

   - $$f(x,y) = \begin{cases} x+y & \text{si } x \text{ y } y \text{ tiene el mismo signo} \\ x-y & \text{si no} \end{cases}$$

     valores de (x, y): (-2,-1), (-2,0), (-2,1), (1,2), (1,-2), (3,2)

2. Crear una sucesión de números de Fibonacci menores a 5000.

1.6 Clase 7. Funciones

Una función en `R` realiza operaciones sobre los argumentos que el usuario provee y arroja uno o más resultados. La sintáxis para escribir una función es

```
function(argumentos){
 computos
}
```

Ejemplo 1.6.1. Queremos esribir una función que calcula lo siguiente:

$$f(x) = 3x - 2$$

Podemos utilizar el siguiente código

```
MF1<-function(x){
  3*x-2
}
MF1(3)

## [1] 7

MF1(c(0:3))

## [1] -2  1  4  7
```

Ahora, calculamos una función un poco más compleja:

Ejemplo 1.6.2. Queremos esribir una función que calcula lo siguiente:

$$f(x) = \begin{cases} 2x + 5 & \text{si } x > -2 \\ 1 & \text{si no} \end{cases}$$

Podemos utilizar el siguiente código

```
MF2<-function(x){
  if(x>-2){2*x+5}
  else{1}
}
MF2(3)

## [1] 11

MF2(-5)

## [1] 1

MF2(c(0:3))

## Warning in if (x >-2) {: the condition has length >1 and only the first element will
be used

## [1]  5  7  9 11

MF2(c(-3:0))

## Warning in if (x >-2) {: the condition has length >1 and only the first element will
be used

## [1] 1
```

Una función con más de un argumento

Ejemplo 1.6.3. Queremos esribir una función que calcula lo siguiente:

$$f(x) = x^2 + y^2$$

Podemos utilizar el siguiente código

```
MF3<-function(x,y){
  x^2+y^2
}
MF3(1,2)

## [1] 5

MF3(1)

## Error in MF3(1): argument "y" is missing, with no default
```

Tenga en cuenta que si dentro de la función no especificas los valores que quieres que muestre, la función no muestra nada, en el siguiente código no muestra el resultado del cálculo

```
MF3<-function(x,y){
  z<-x^2+y^2
}
MF3(1,2)
```

Para que muestre lo calculado, tendría que usar

```
MF3<-function(x,y){
  z<-x^2+y^2
  z
}
MF3(1,2)

## [1] 5
```

También una función puede arrojar resultados de varios cálculos, pero no sirve así:

```
MF3<-function(x,y){
  x^2+y^2
  x+y
}
MF3(1,2)

## [1] 3
```

y tampoco sirve

```
MF3<-function(x,y){
  z<-x^2+y^2
  w<-x+y
  z
  w
}
MF3(1,2)

## [1] 3
```

Para que una función arroje varios resultados, necesitamos usar `list`, veamos el siguiente ejemplo

Ejemplo 1.6.4. Queremos esribir una función que al proveer un vector de datos calcule la suma, el promedio y la desviación estándar. Podemos utilizar el siguiente código

```
MF4<-function(x){
  a1<-sum(x)
  a2<-mean(x)
  a3<-sd(x)
  list(suma=a1,promedio=a2,desviacion=a3)
}
a<-c(1,5,2,1,4,5,2)
MF4(a)

## $suma
## [1] 20
##
## $promedio
## [1] 2.857143
##
## $desviacion
## [1] 1.772811
```

Para extraer solo alguno de los resultados de la función `MF4`, por ejemplo el promedio, utilizamos

```
MF4(a)$promedio

## [1] 2.857143
```

También podemos guardar los resultados de `MF4` en un objeto, y llamarlo cada vez que lo necesitemos.

```
bb<-MF4(a)
bb$prom

## [1] 2.857143

bb$des

## [1] 1.772811
```

Función con mensaje de error

Ejemplo 1.6.5. Queremos esribir una función que calcula lo siguiente:

$$f(x) = \sqrt{x}$$

que solo está definido para valores enteros. Podemos utilizar el siguiente código

```
MF5<-function(x){
  if(x<0){stop("No se puede calcular la raiz cuadrada de un valor negativo")}
  else{sqrt(x)}
}
MF5(-1)

## Error in MF5(-1): No se puede calcular la raiz cuadrada de un valor negativo

MF5(5)

## [1] 2.236068
```

Muchas funciones en R trae valores por defecto, si queremos crear una función asì, debemos especificar los valores defecto en los argumentos de la función. Veamos el siguiente ejemplo

Ejemplo 1.6.6.
```
MF6<-function(x,order=2){
x^{1/order}
}
MF6(5)

## [1] 2.236068

MF6(5,2)

## [1] 2.236068

MF6(5,3)

## [1] 1.709976
```

1.6.1 Ejercicios

1. Escriba una función que al ingresar un conjunto de datos, verifica si los datos contienen datos faltantes o no. La función debe arrojar cuáles son los componentes que son datos faltantes, el conjunto de datos eliminándolos, y si los datos ingresados no contienen datos faltantes, la función debe arrojar un mensaje que indica eso. Comprueba que la función esté correcta aplicándola a datos que: 1) no contienen datos faltantes, 2) contienen 1 dato faltante, 3) contiene dos datos faltantes.

2. Escriba una función que al ingresar un conjunto de datos, calcule los cuartiles, mínimo y máximo. Si los datos ingresados contienen datos faltantes, la función debe arrojar un mensaje de error

indicando eso y calcular las anteriores medidas descriptivas eliminando los datos faltantes. Comprueba que la función esté correcta aplicándola a datos que: 1) no contienen datos faltantes, 2) contienen 1 dato faltante, 3) contiene dos datos faltantes.

3. Escriba una función que al ingresar una matriz o una base de datos (donde una fila representa un individuo y una columna representa una variable), arroje una matriz o un `data.frame` que para cada variable calcule la media, desviación estándar y coeficiente de variación. Verifique que funcione bien con los datos numéricos de `Lucy`.

4. Escriba una función que calcule las funciones del primer ejecicio de la sección 6.3.

5. Escriba una función que al ingresar una matriz cuadrada invertible, calcule la matriz inversa y el determinante. La función debe arrojar error si el usuario ingresa una matriz rectangular o una matriz cuadrada no invertible.Verifica que funcione bien con algunas matrices.

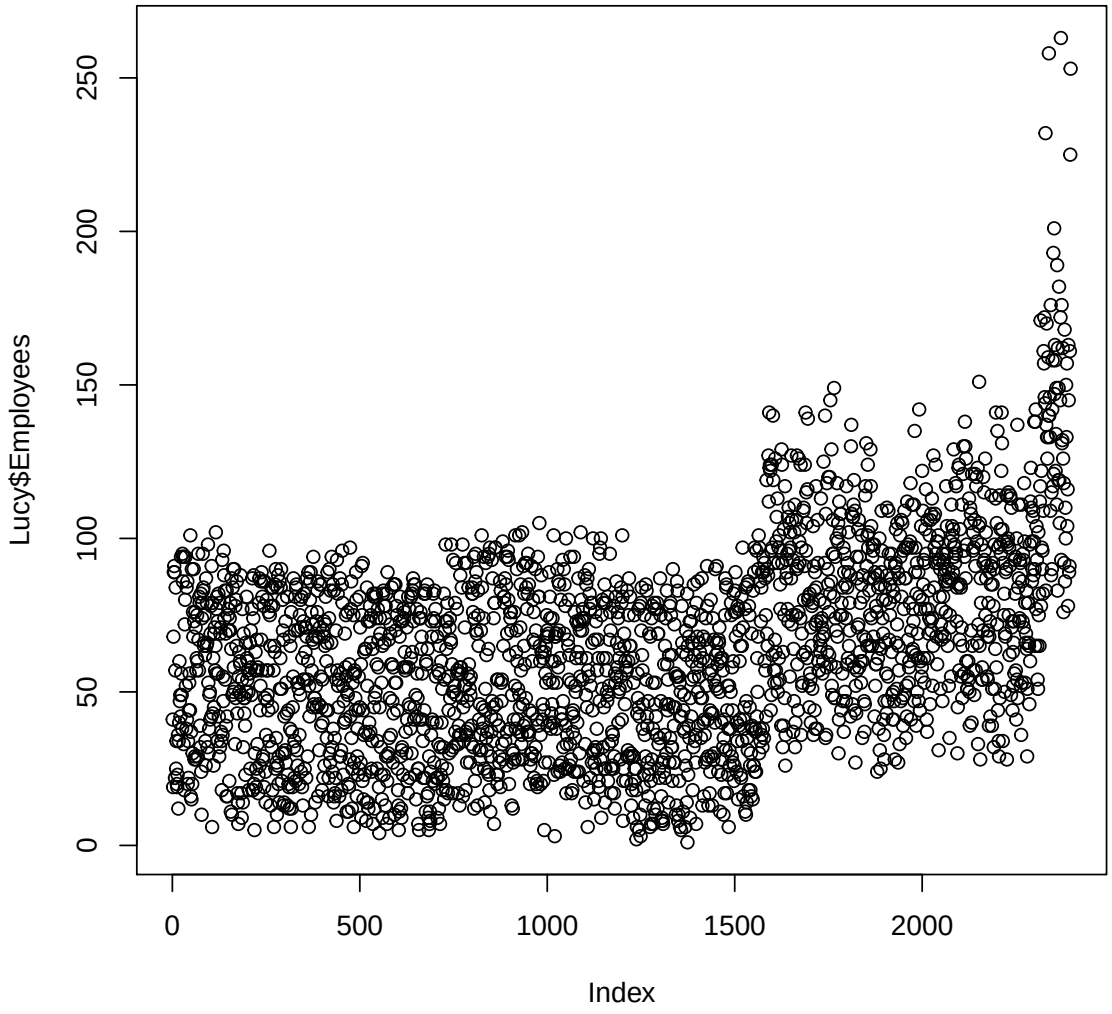
1.7 Clase 9. Gráficas

1.7.1 Graficar un conjunto de datos

Las gráficas estándares de `R` son planas en el sentido de que los colores, estilos de líneas y puntos, títulos entre otras cosas deben ser especificados por los usuarios.

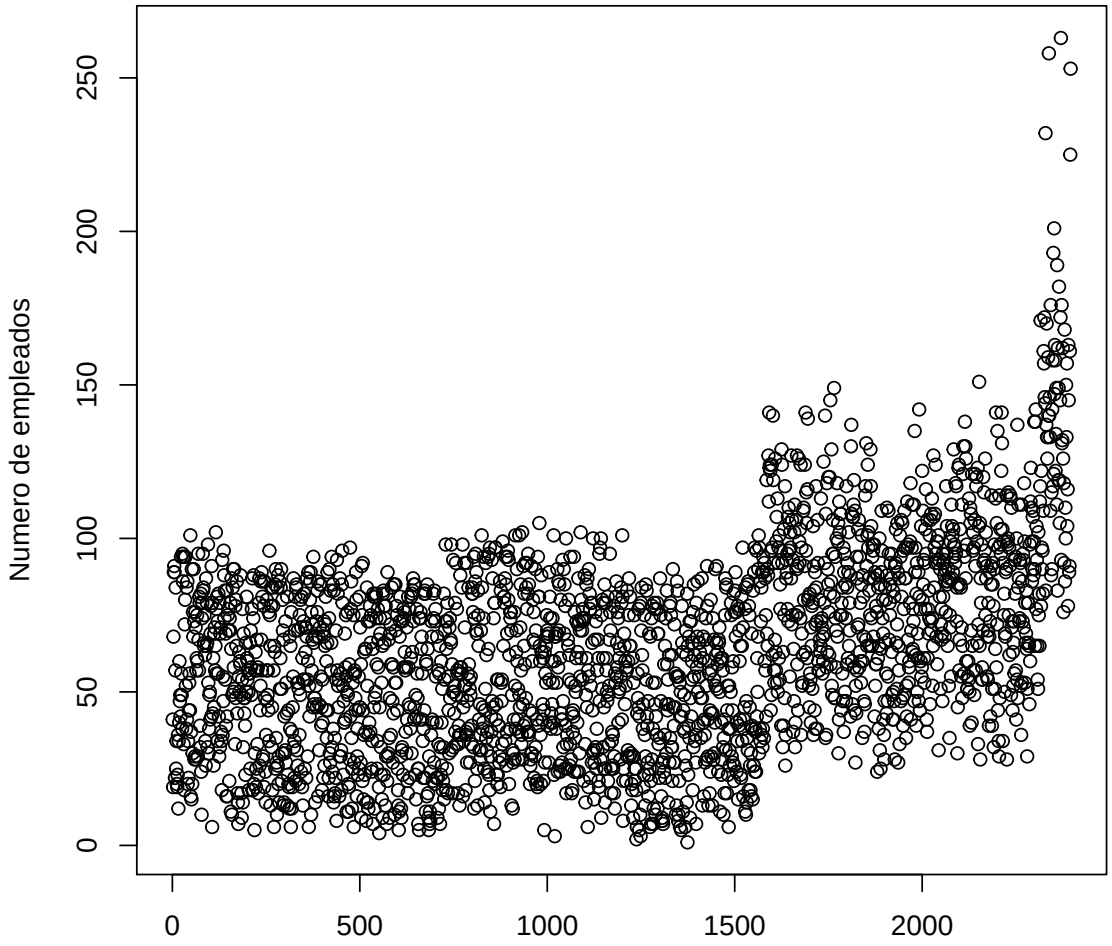
Una de las gráficas más usuales de la estadística es la gráfica de dispersión que se logra con el comando `plot` cuando se aplica a datos numéricos.

```
library(TeachingSampling)
data(Lucy)
plot(Lucy$Employees)
```



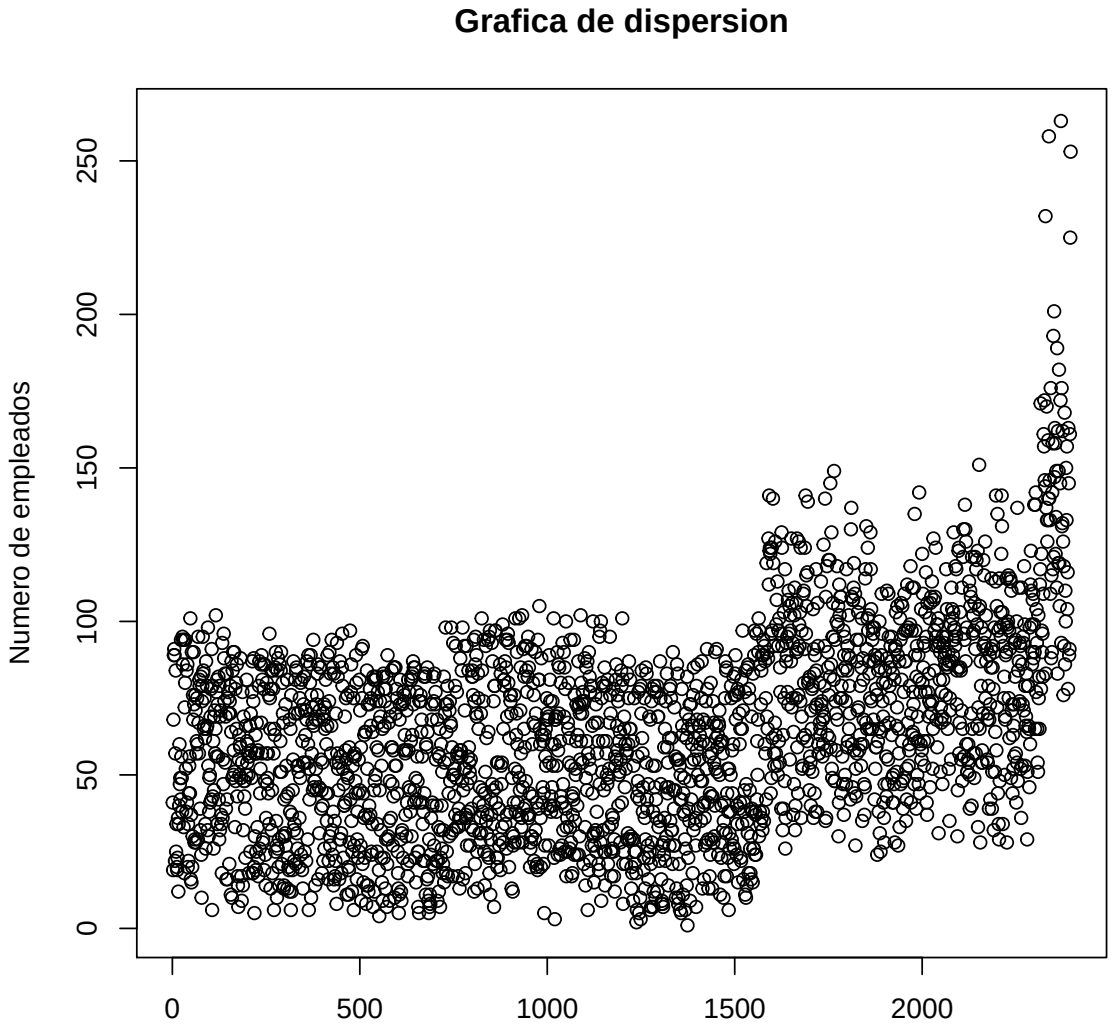
Para cambiar el nombre de los ejes, usamos las opciones de `xlabs` y `ylabs` dentro de `plot`, por ejemplo:

```
plot(Lucy$Employees,xlab="",ylab="Numero de empleados")
```



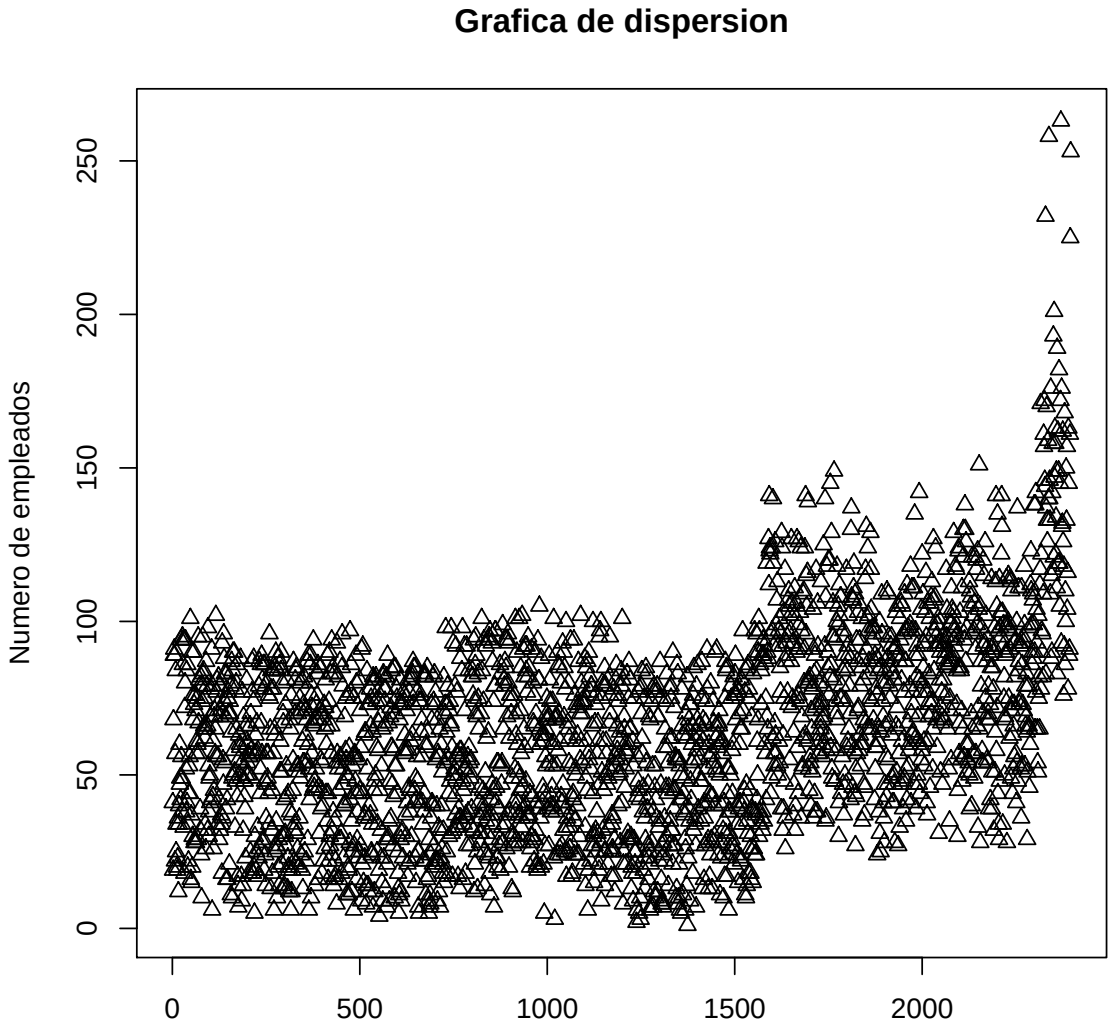
Si queremos un título para la gráfica, usamos `main` dentro de plot, por ejemplo:

```
plot(Lucy$Employees,xlab="",ylab="Numero de empleados",main="Grafica de dispersion")
```



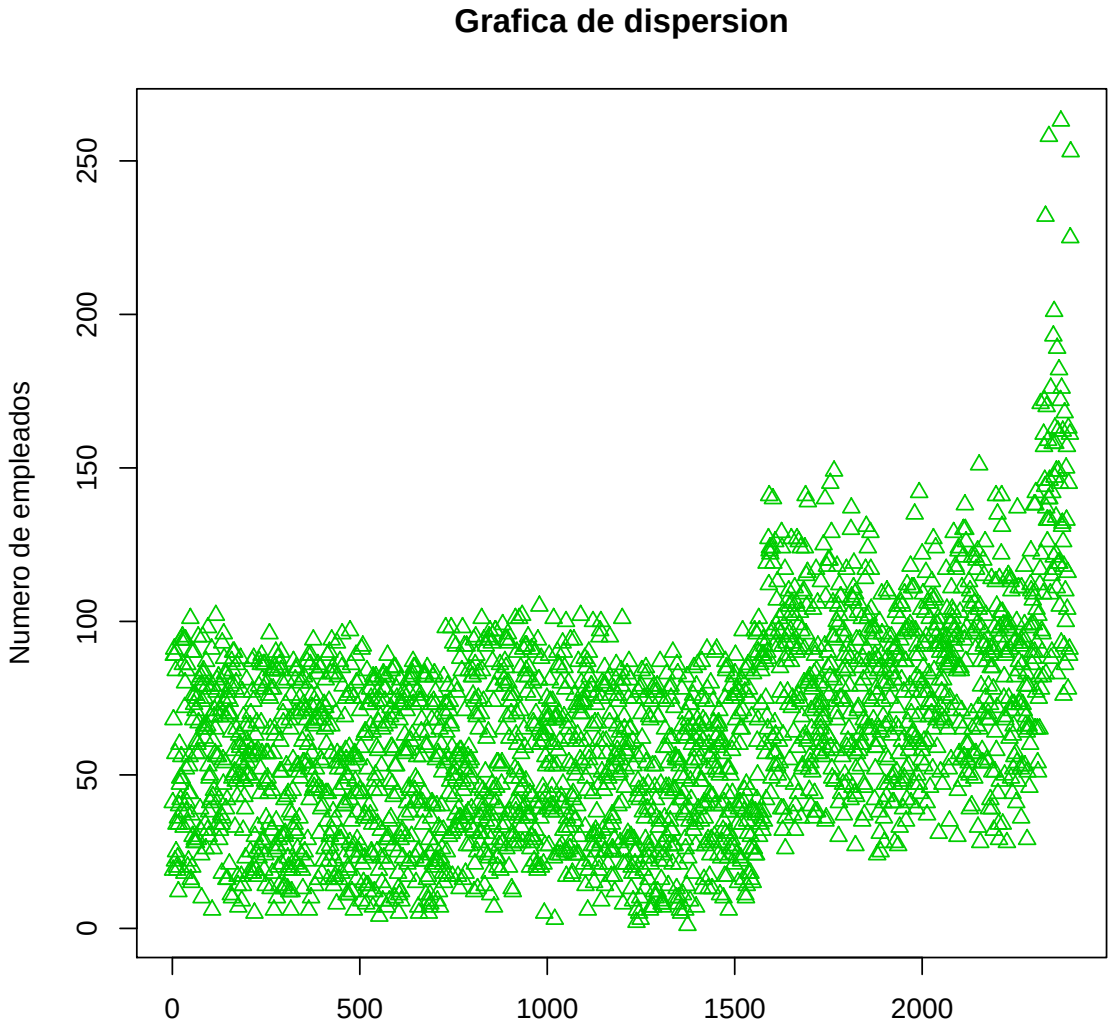
También podemos cambiar el estilo de puntos por triángulos, cuadrados u otros. Esto se logra con la opción `pch`, por ejemplo:

```
plot(Lucy$Employees,xlab="",ylab="Numero de empleados",main="Grafica de dispersion",pch=2)
```



Los colores se pueden cambiar con la opción `col`, por ejemplo:

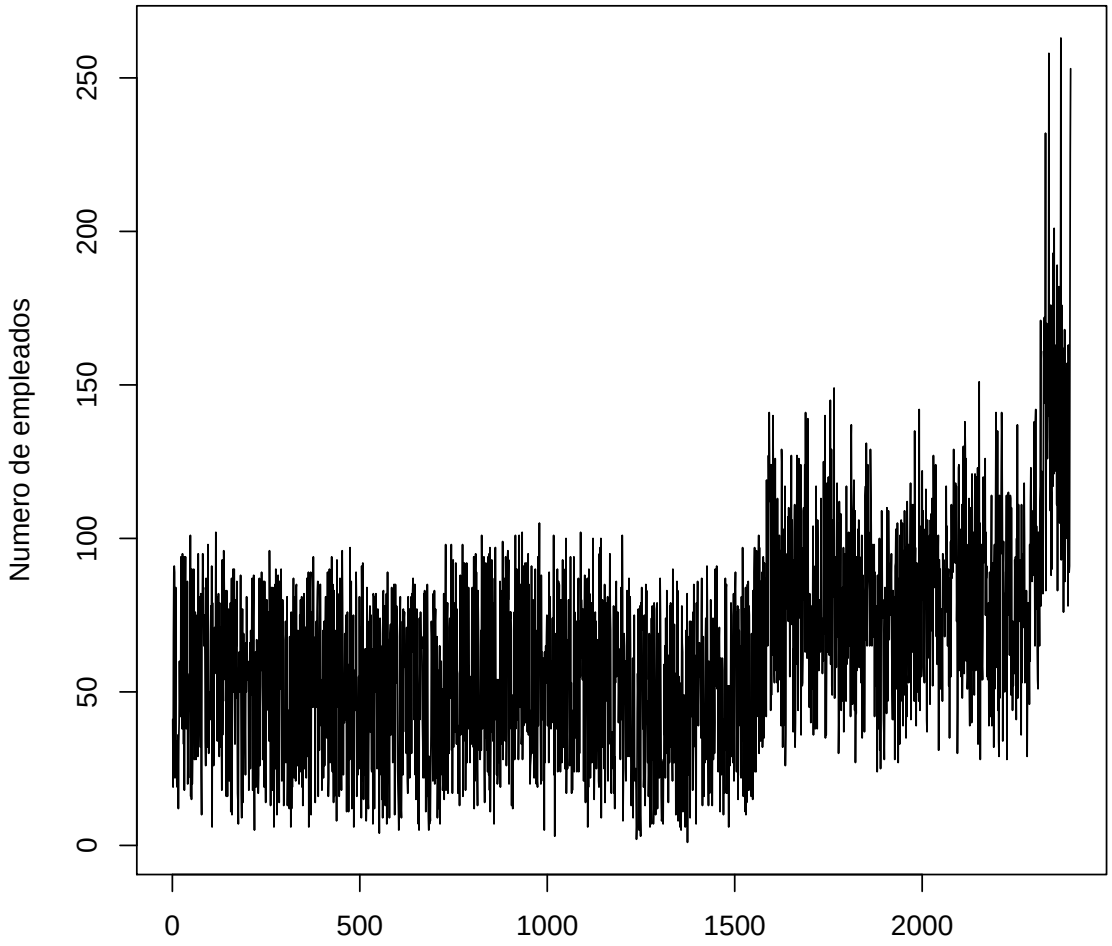
```
plot(Lucy$Employees,xlab="",ylab="Numero de empleados",main="Grafica de dispersion",pch=2,col="3")
```



En la anterior expresión, `col=3` es equivalente a `col="green"`. En la página http://www.stat.columbia.edu/ tzheng/files/Rcolor.pdf, se puede encontrar una gran variedad de colores.

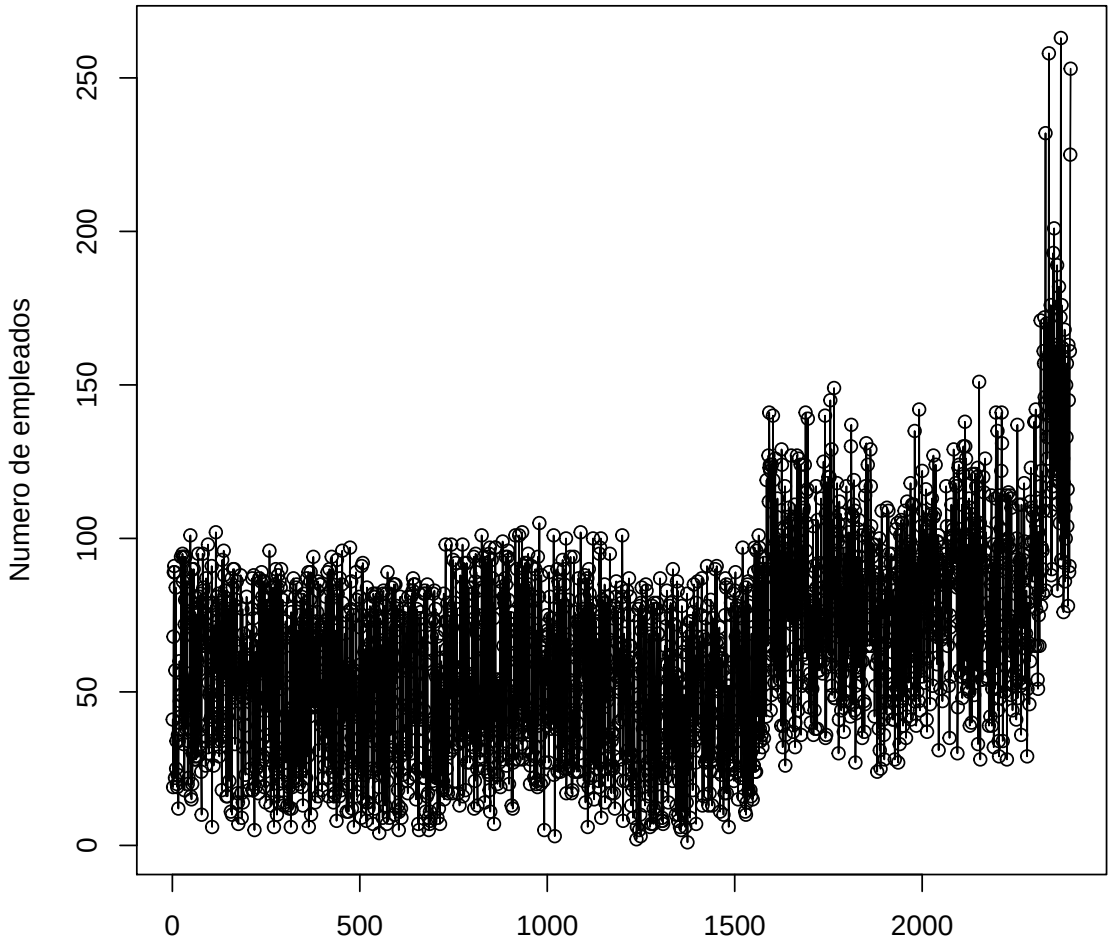
Si queremos usar línea u ot en vez de puntos de la gráfica, podemos cambiar la opción `type` por `"l"`

```
plot(Lucy$Employees,xlab="",ylab="Numero de empleados",type="l")
```



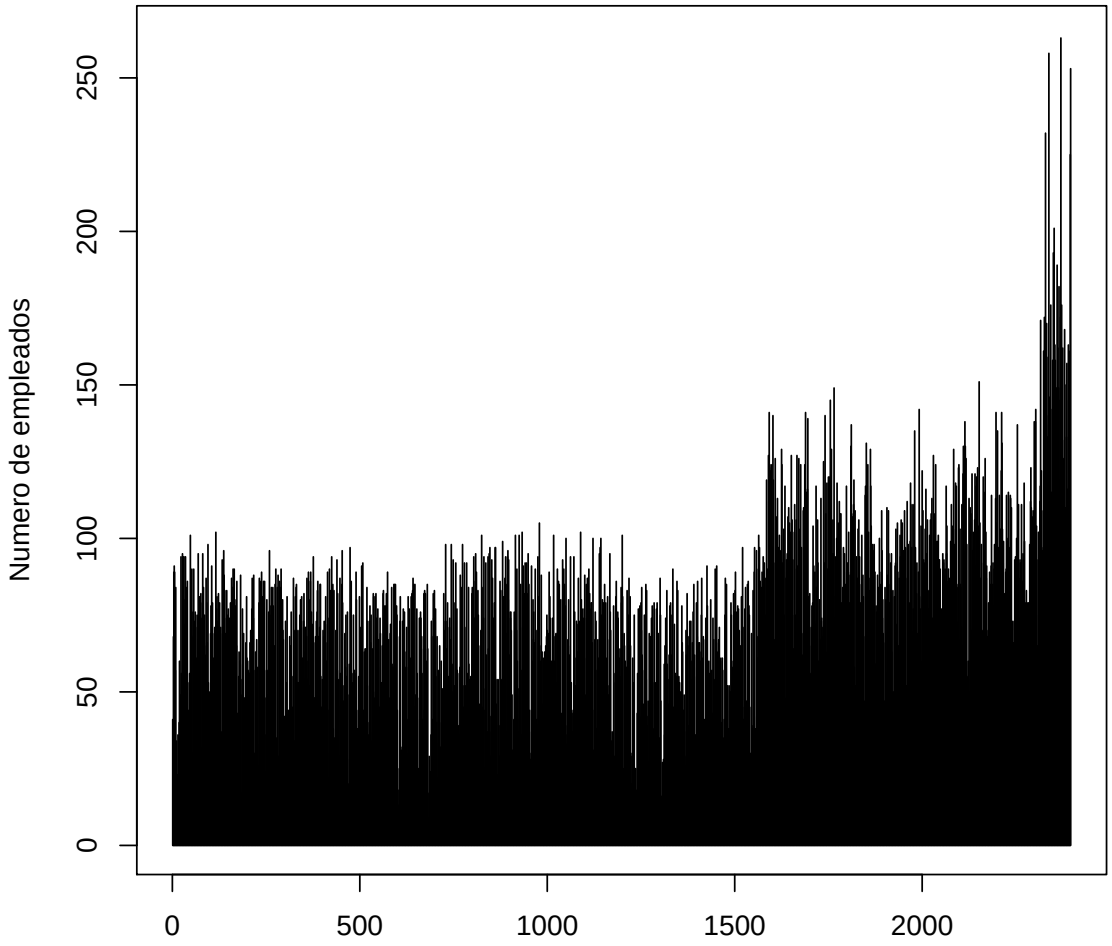
Veamos qué pasa si usamos `o`

```
plot(Lucy$Employees,xlab="",ylab="Numero de empleados",type="o")
```



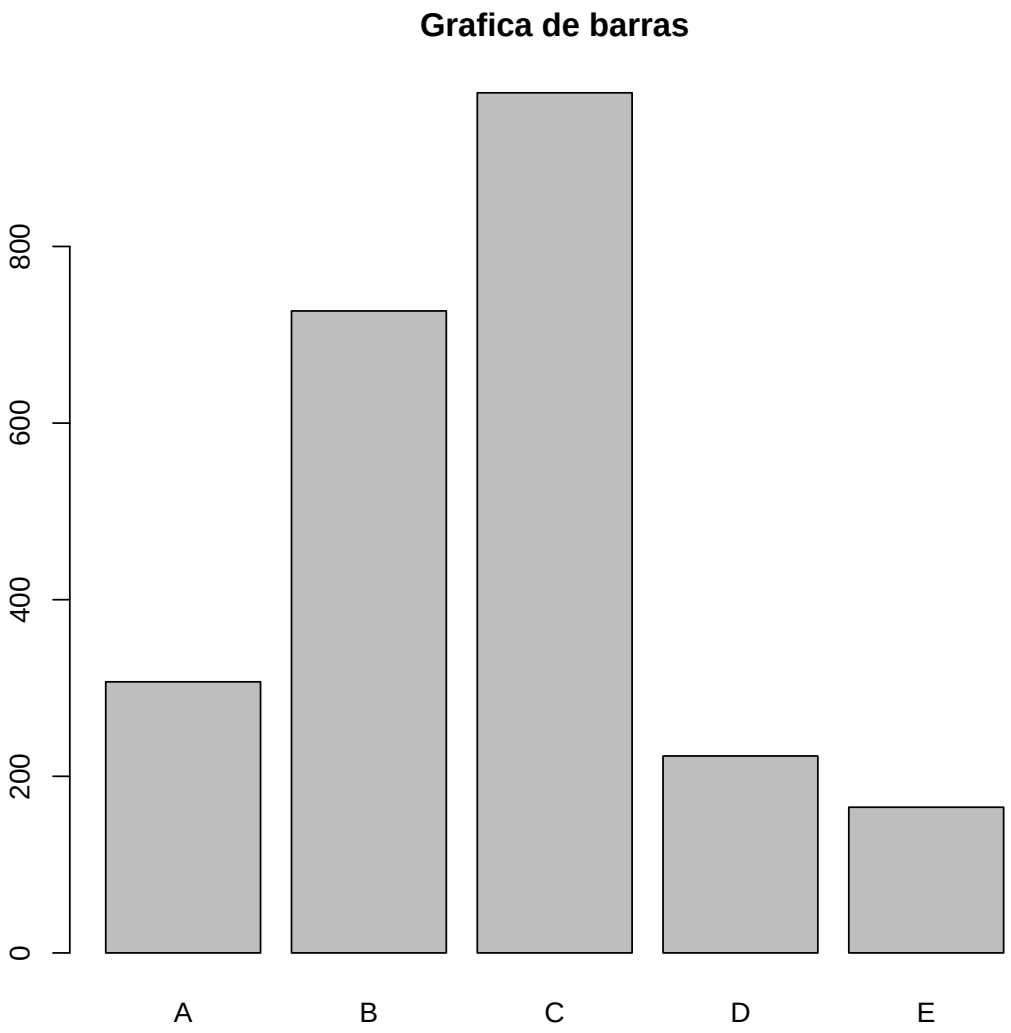
Veamos qué pasa si usamos `h`

```
plot(Lucy$Employees,xlab="",ylab="Numero de empleados",type="h")
```



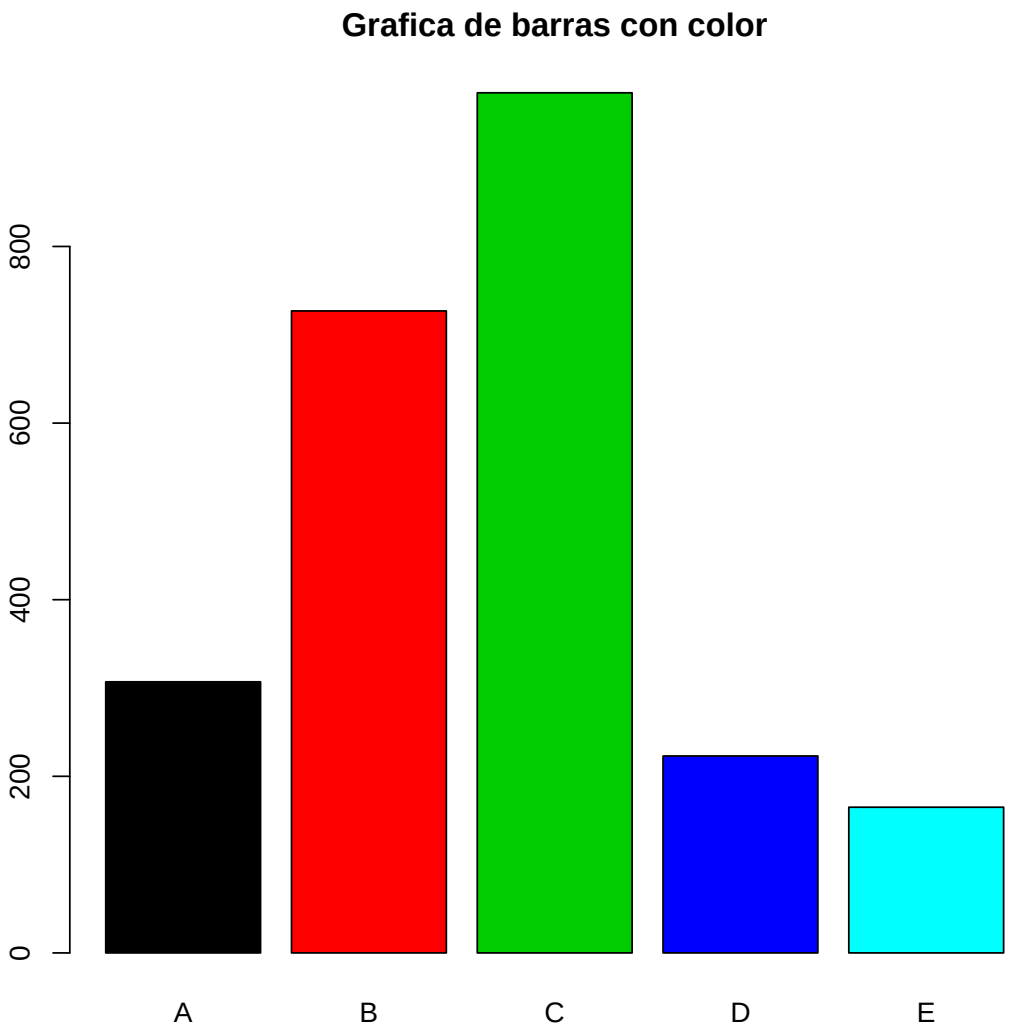
Al usar `plot` para un conjunto de datos categóricos, se genera una gráfica de barras, de color gris, por defecto.

```
plot(Lucy$Zone,main="Grafica de barras")
```



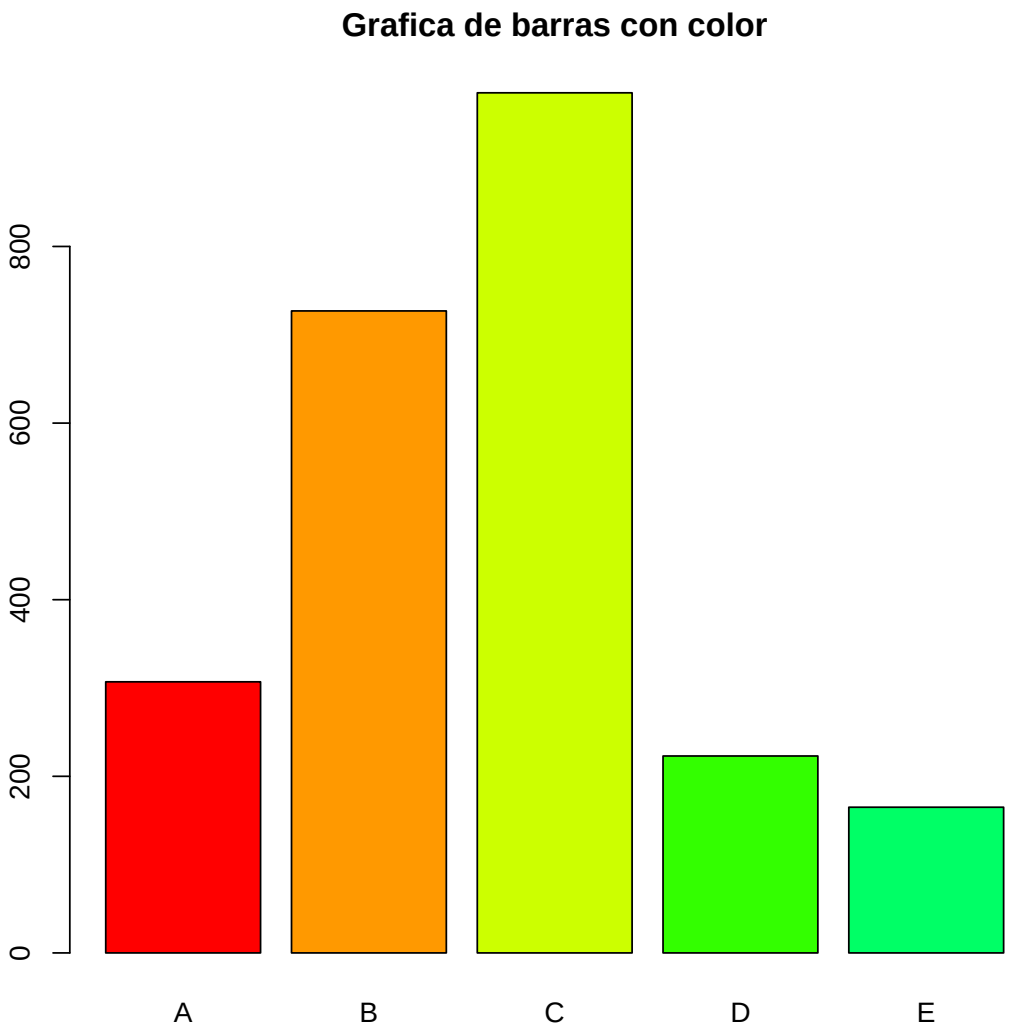
Pongamos un poco de color a la gráfica anterior.

```
plot(Lucy$Zone,main="Grafica de barras con color",col=c(1,2,3,4,5))
```



R también pone colores de forma automática.

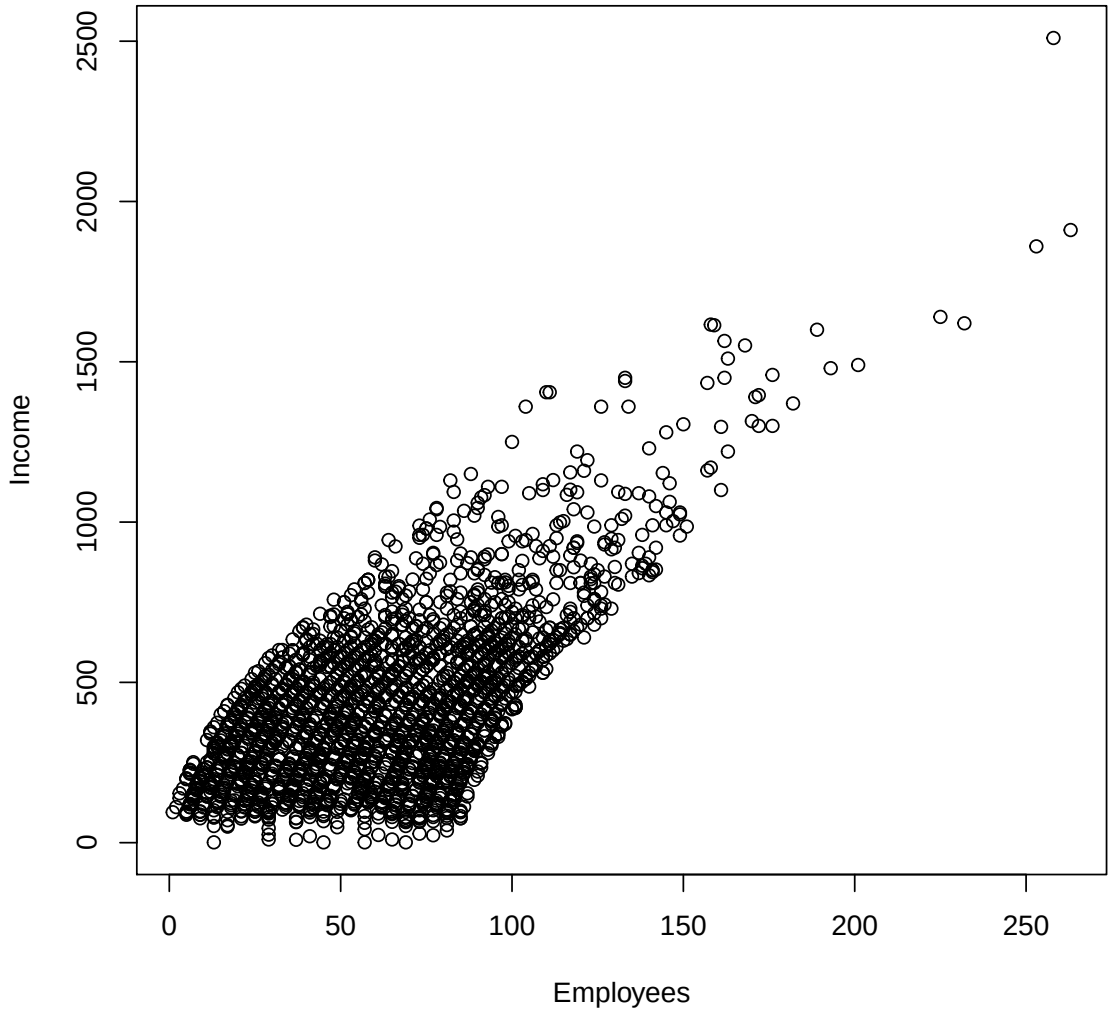
```
plot(Lucy$Zone,main="Grafica de barras con color",col=rainbow(10))
```



1.7.2 Graficar dos conjuntos de datos

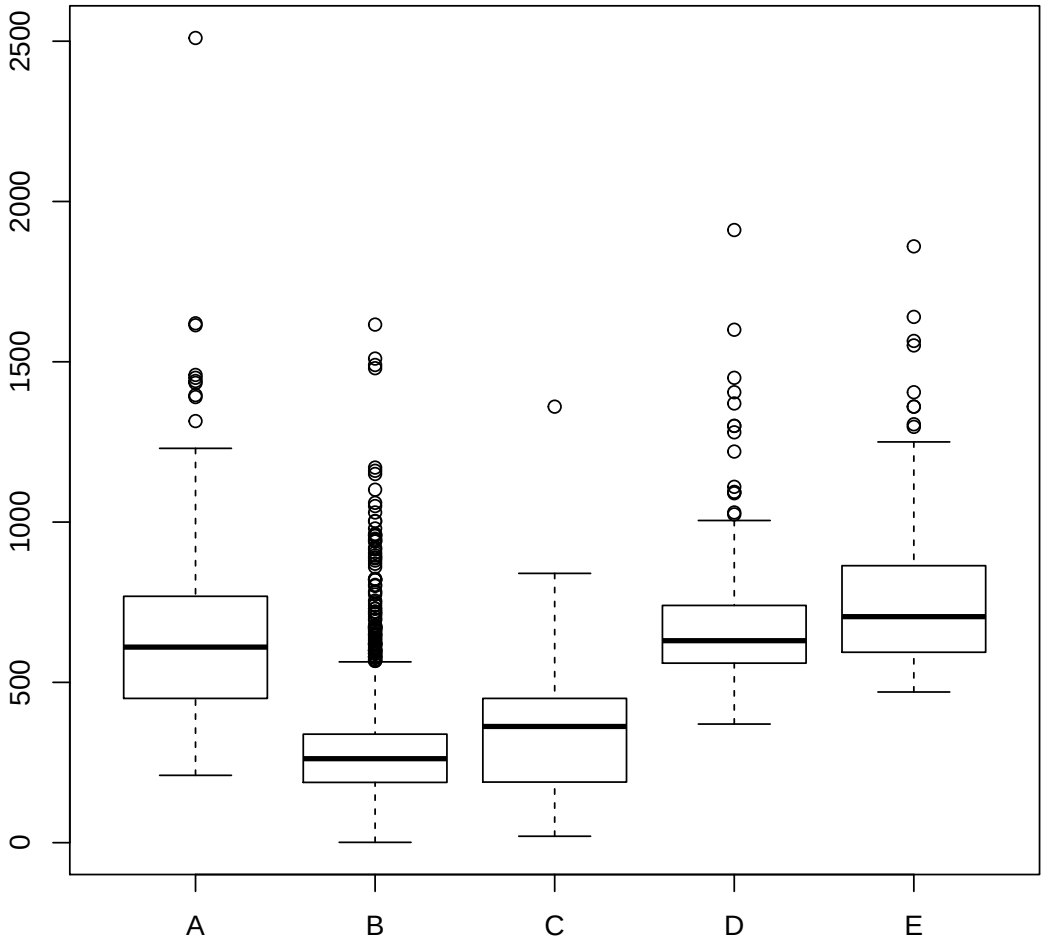
Al tener dos conjunto de datos numéricos, una gráfica de dispersion puede mostrar la relación que haya entre las dos variables. Por ejemplo

```
attach(Lucy)
plot(Employees, Income)
```



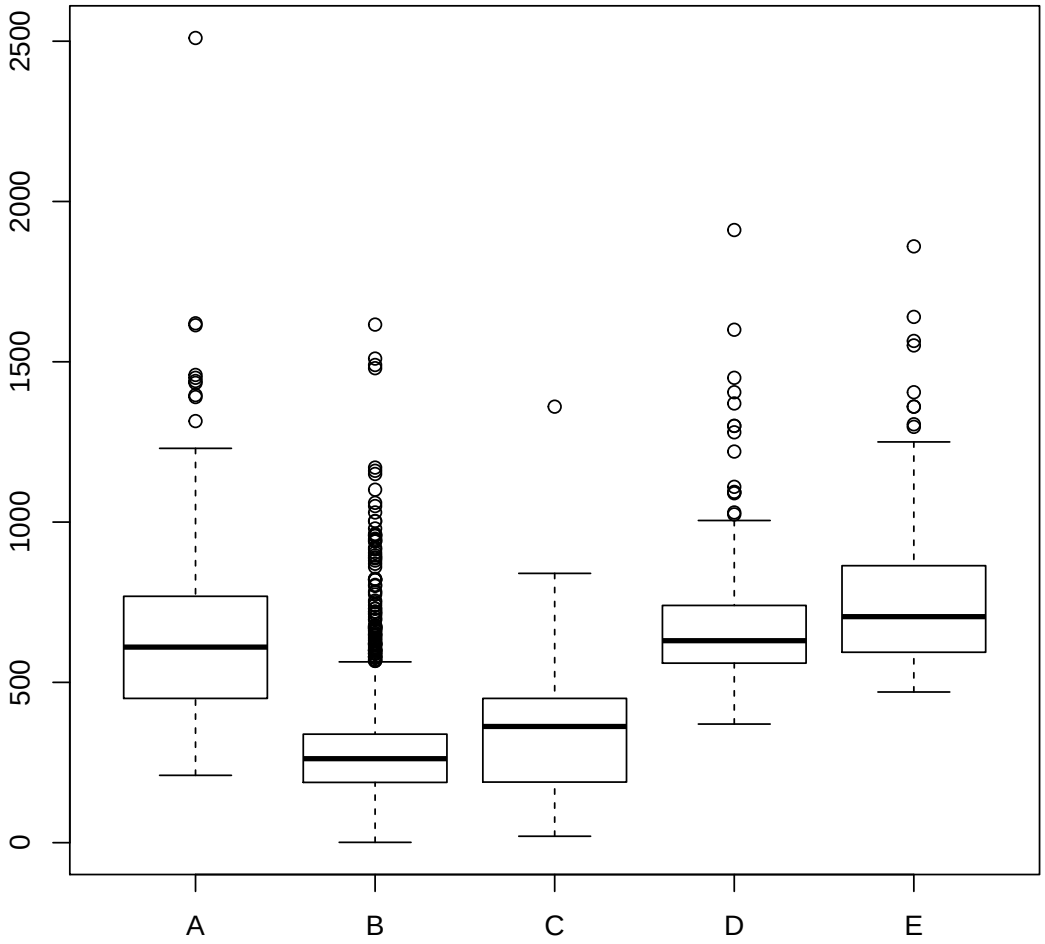
Si una de las dos variables es categórica, la función `plot` arroja diagramas de caja para la variable numérica en las diferentes categorías.

```
plot(Zone, Income)
```



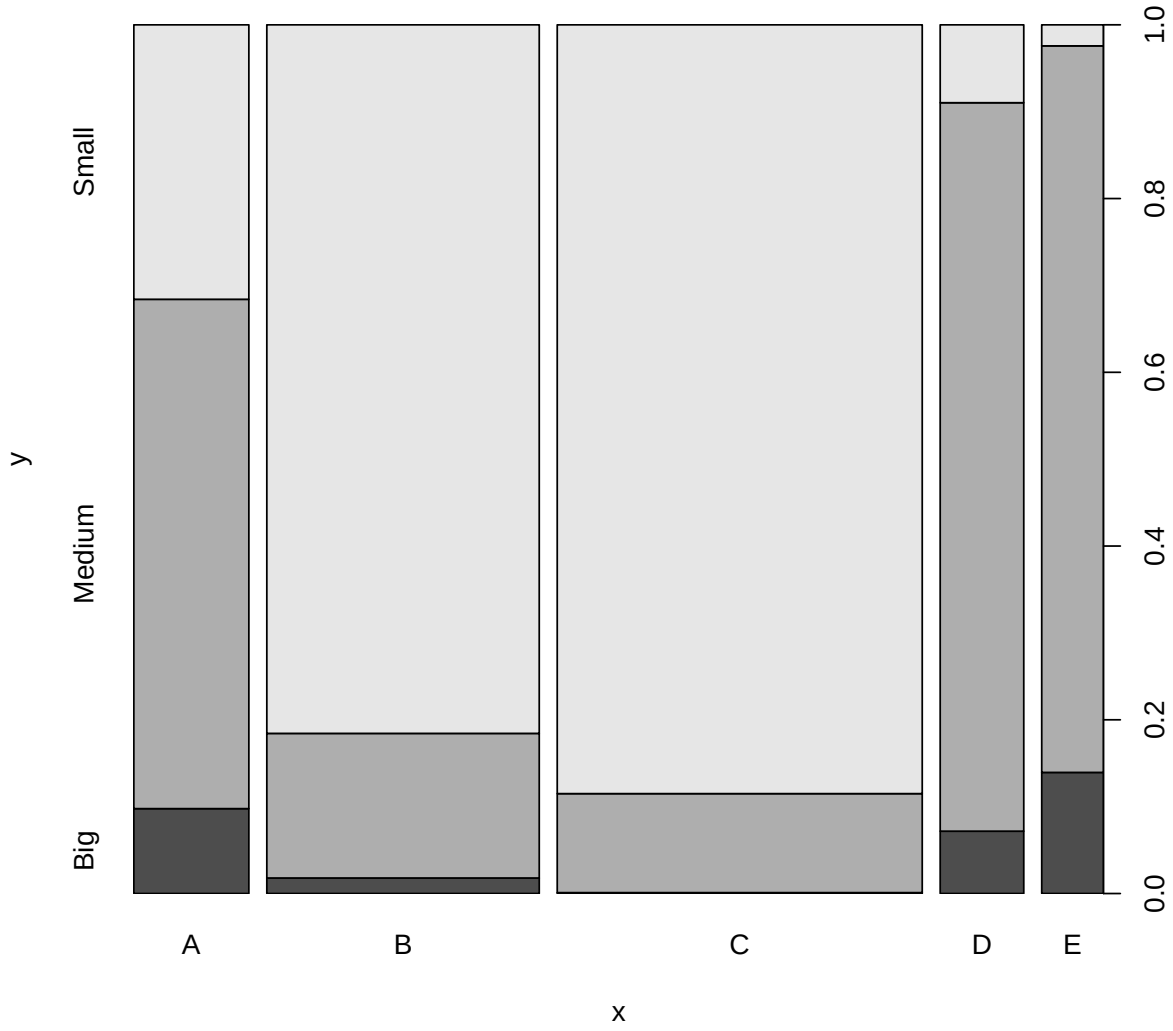
Lo cual es equivalente a

```
boxplot(Income~Zone)
```



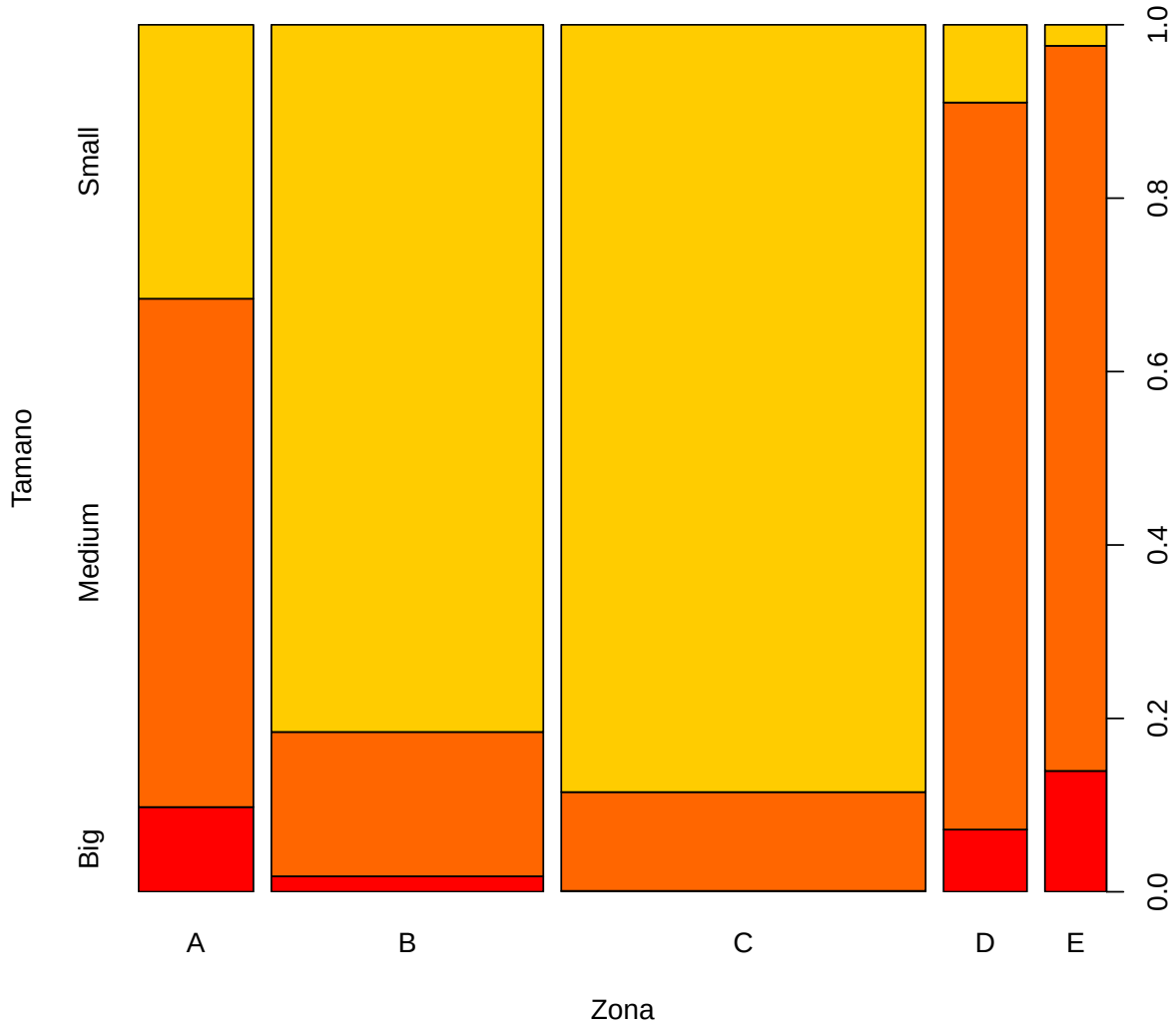
Cuando tenemos dos conjuntos de datos categóricos.

```
plot(Zone, Level)
```



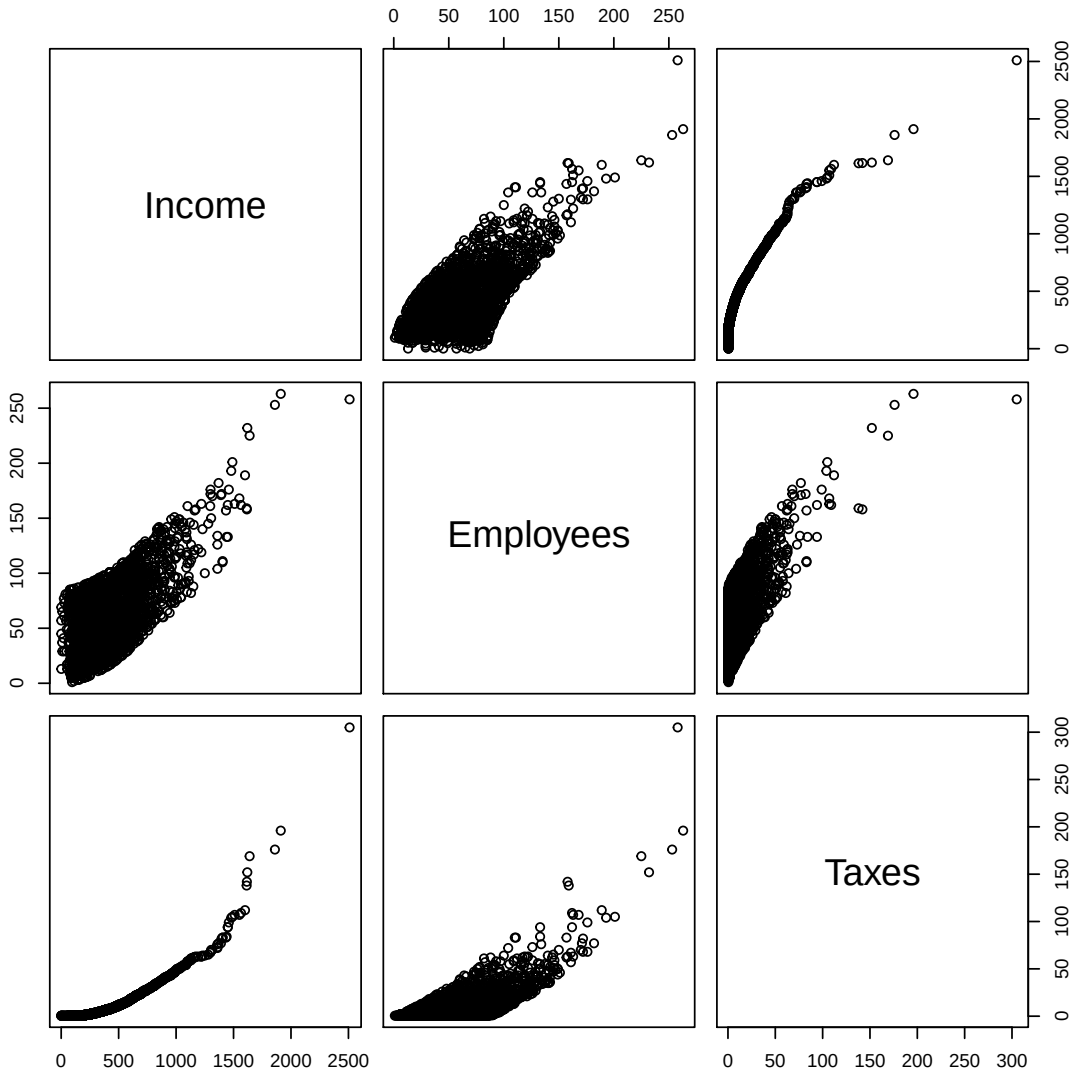
Un poco de color, y cambiemos las leyendas de los ejes.

```
plot(Zone, Level,col=rainbow(15),xlab="Zona", ylab="Tamano")
```



1.7.3 Graficar más de dos conjuntos de datos

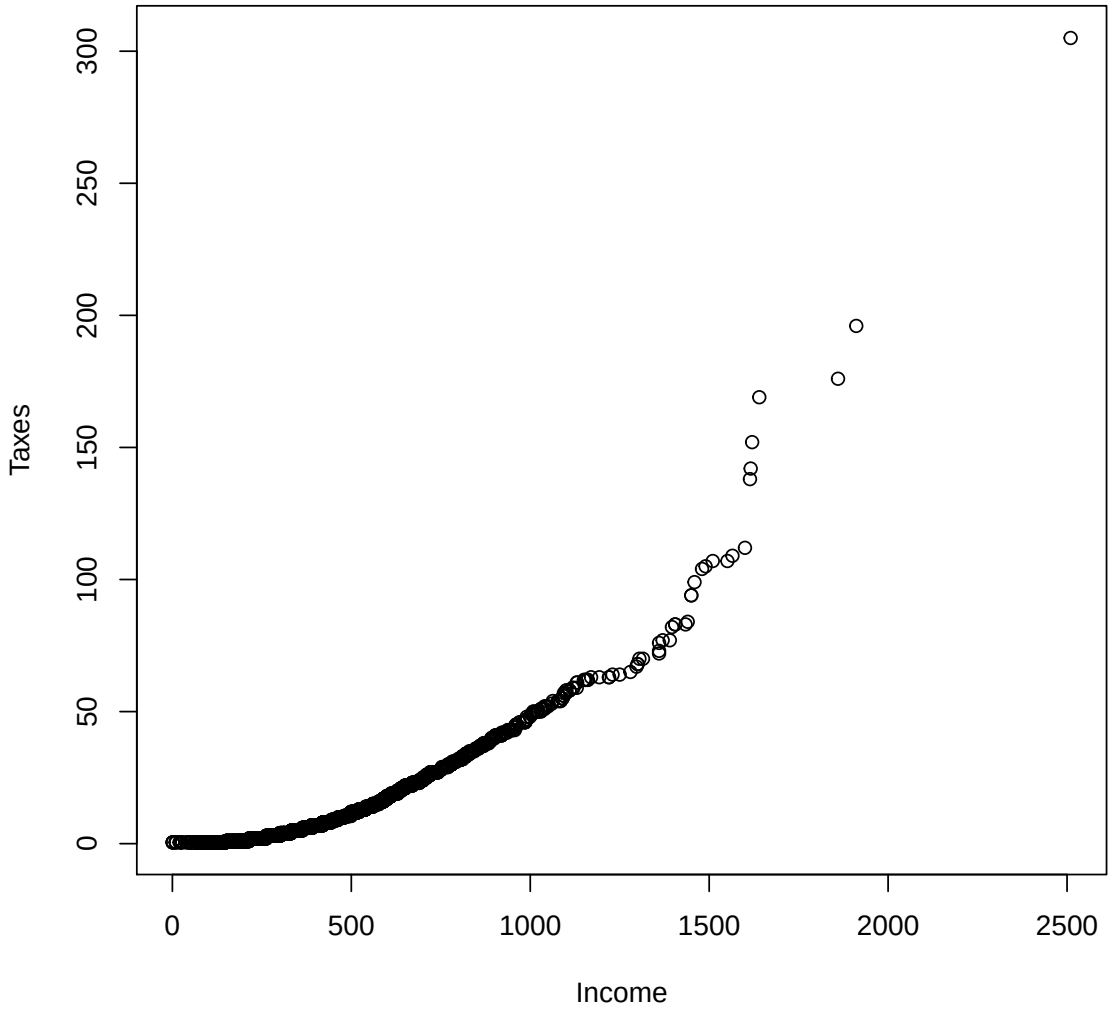
```
X<-cbind(Income, Employees,Taxes)
pairs(X)
```



1.7.4 Agregar puntos o líneas

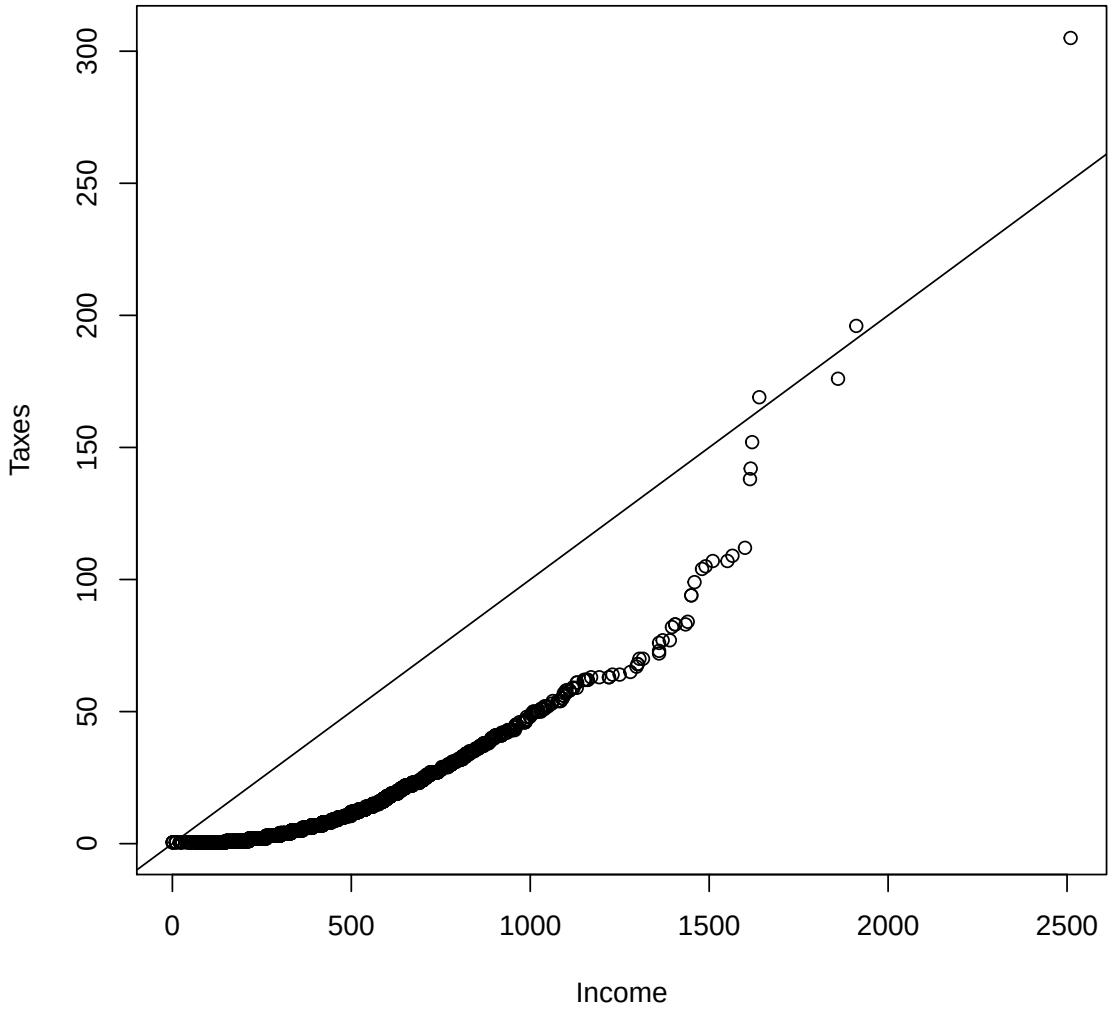
Para agregar una línea horizontal, vertical o una ecuación de la forma $y = a + bx$, usamos la función `abline`. Por ejemplo, a la siguiente gráfica

```
plot(Income,Taxes)
```



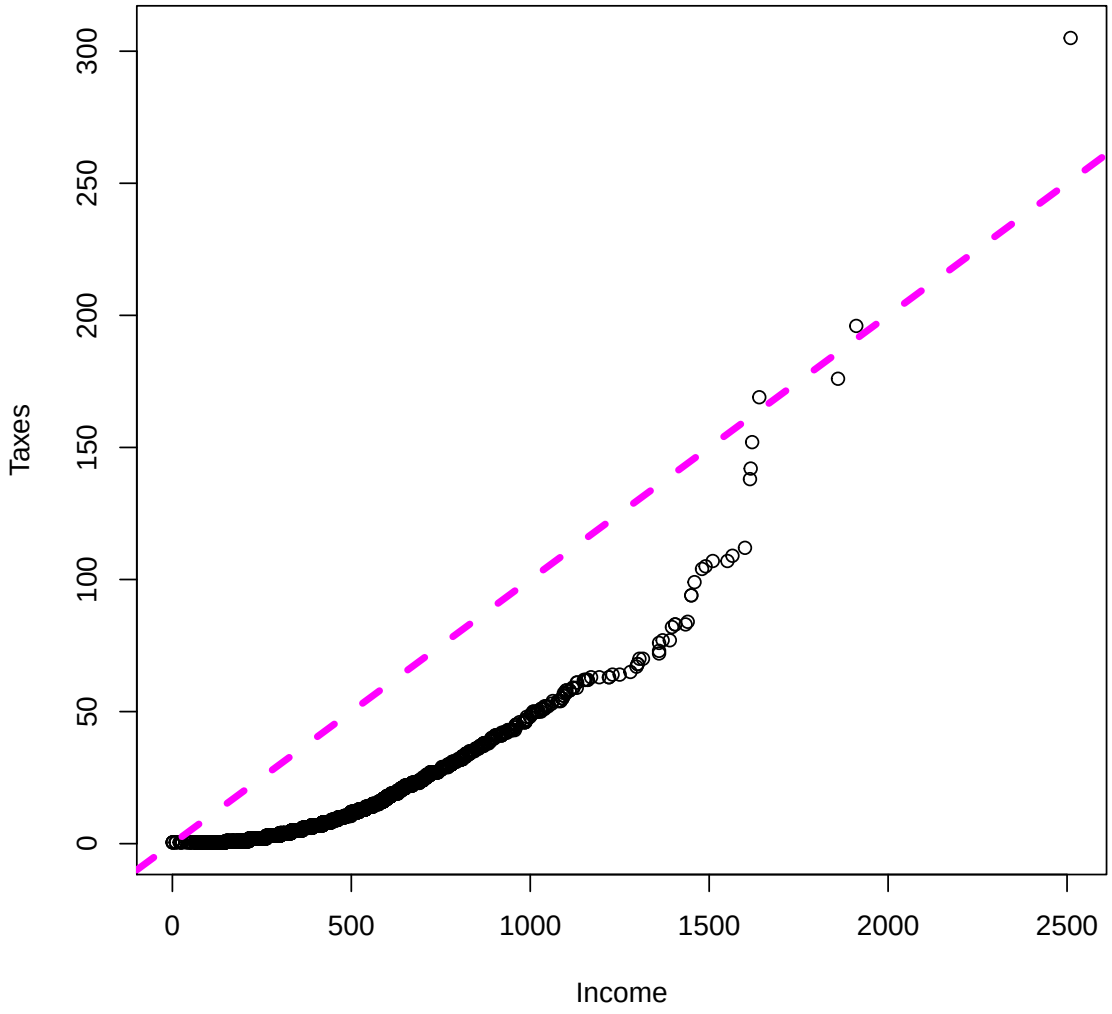
queremos agregar una línea que pasa por el origen de pendiente 0.1,

```
plot(Income,Taxes)
abline(a=0,b=0.1)
```



Podemos cambiar el estilo, grosor y el color de la línea como en el siguiente comando

```
plot(Income,Taxes)
abline(a=0,b=0.1,lty=2,lwd=4, col=6)
```

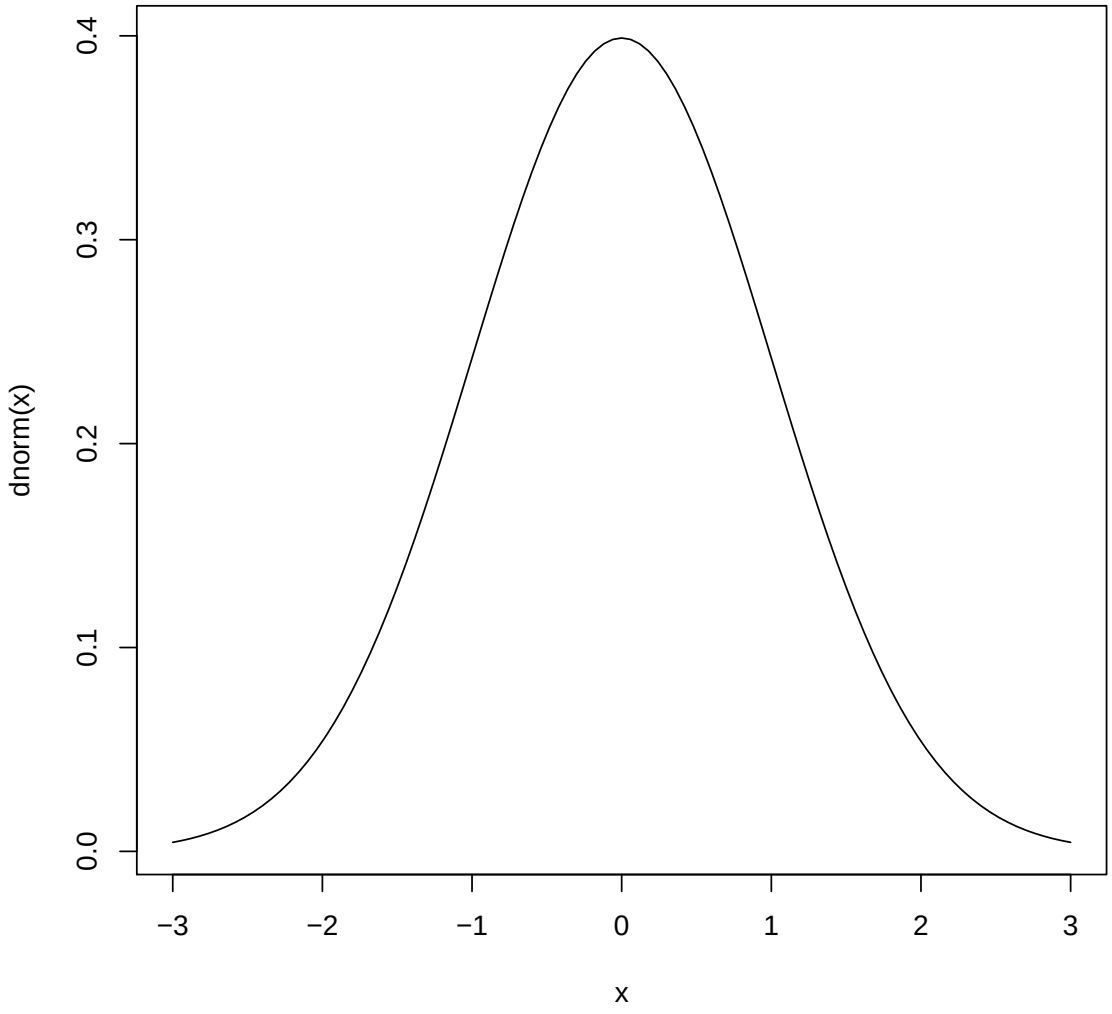


Otras opciones para agregar líneas y puntos son `lines` y `points`.

1.7.5 Graficar una función

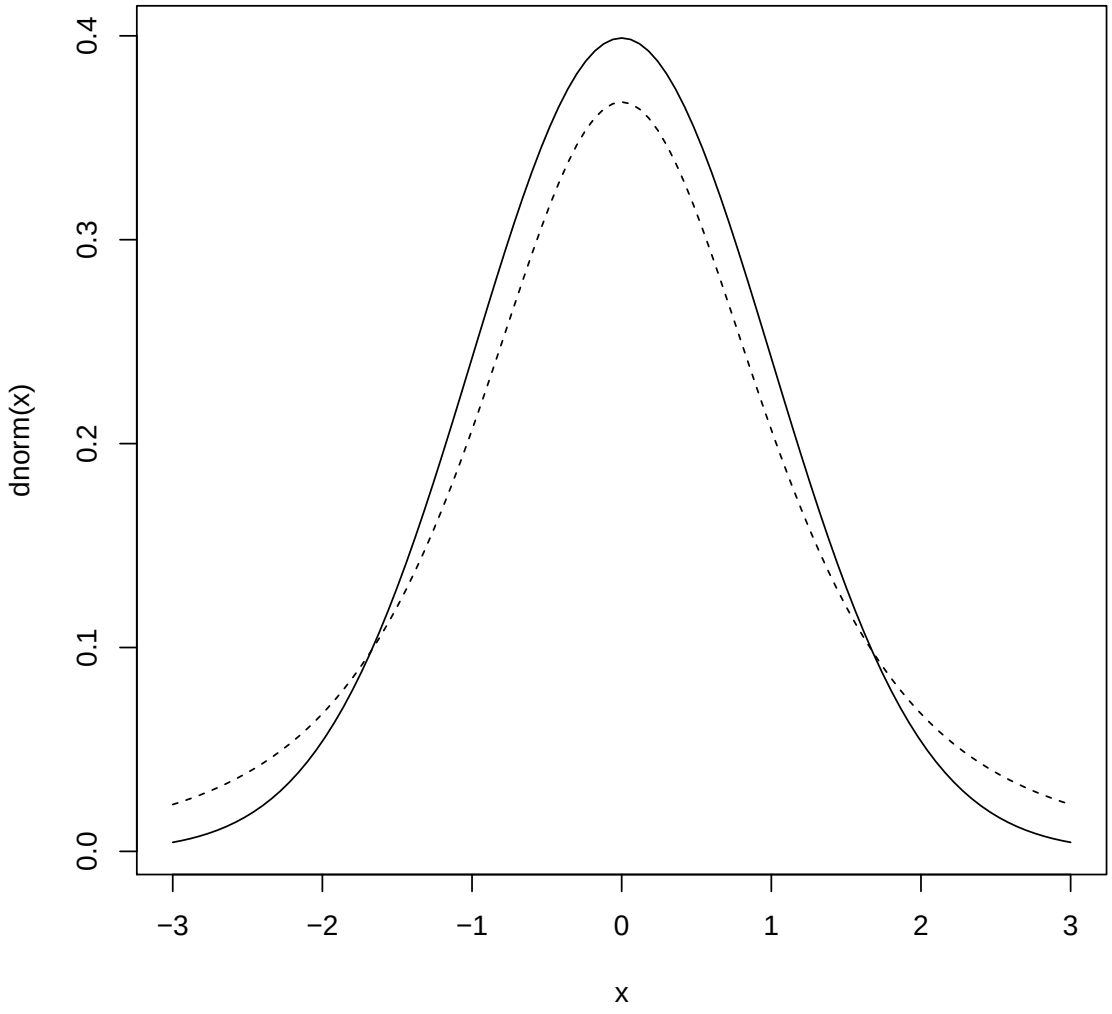
La forma de graficar una función es usando `curve`. Por ejemplo, queremos visualizar la función de densidad de la distribución normal estándar desde -3 hasta 3. Podemos utilizar lo siguiente:

```
curve(dnorm(x),-3,3)
```



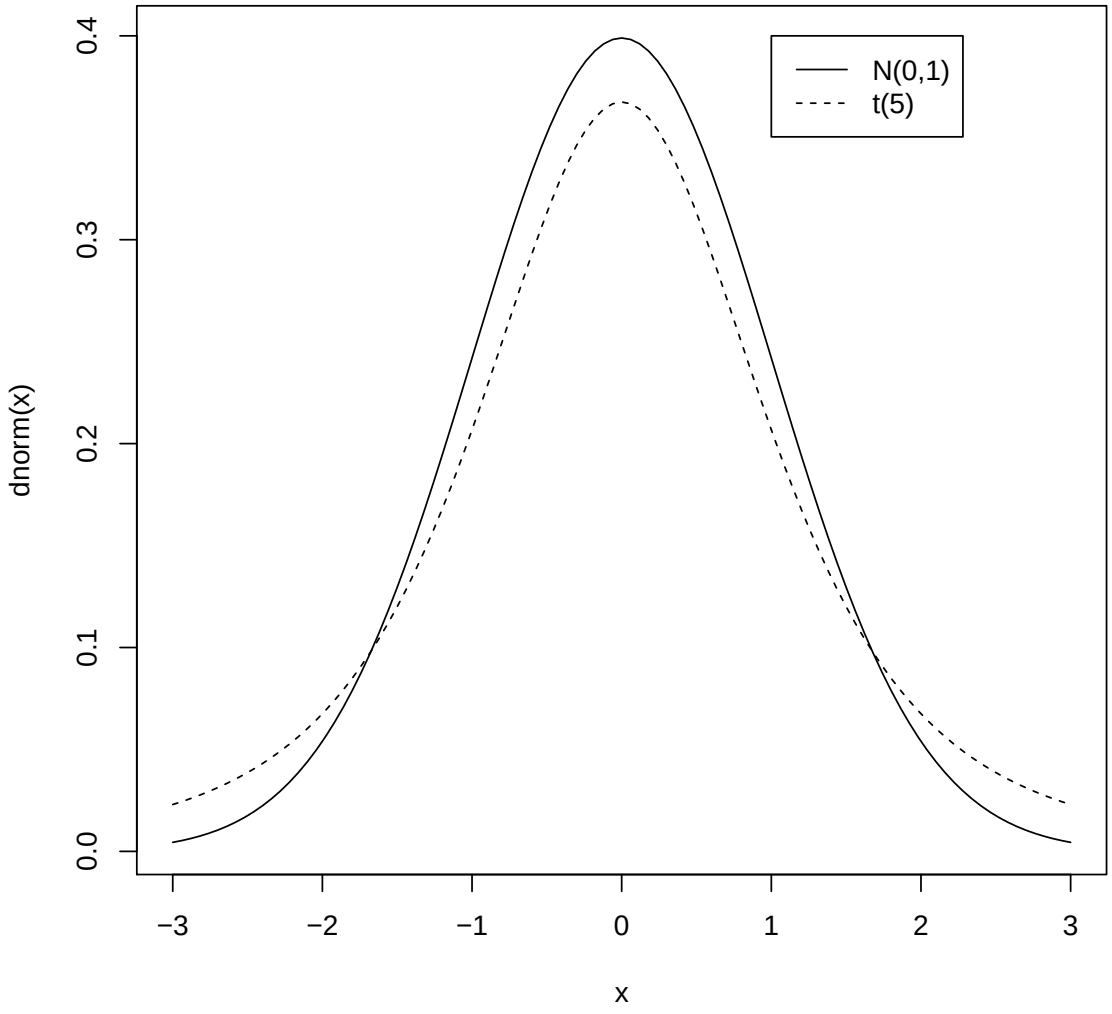
Podemos agregar la función de densidad de la distribución t con 5 grados de libertad

```
curve(dnorm(x),-3,3)
curve(dt(x,df=3),-3,3,add=T,lty=2)
```



Para mejorar la lectura de la gráfica, le podemos agregar unas leyendas

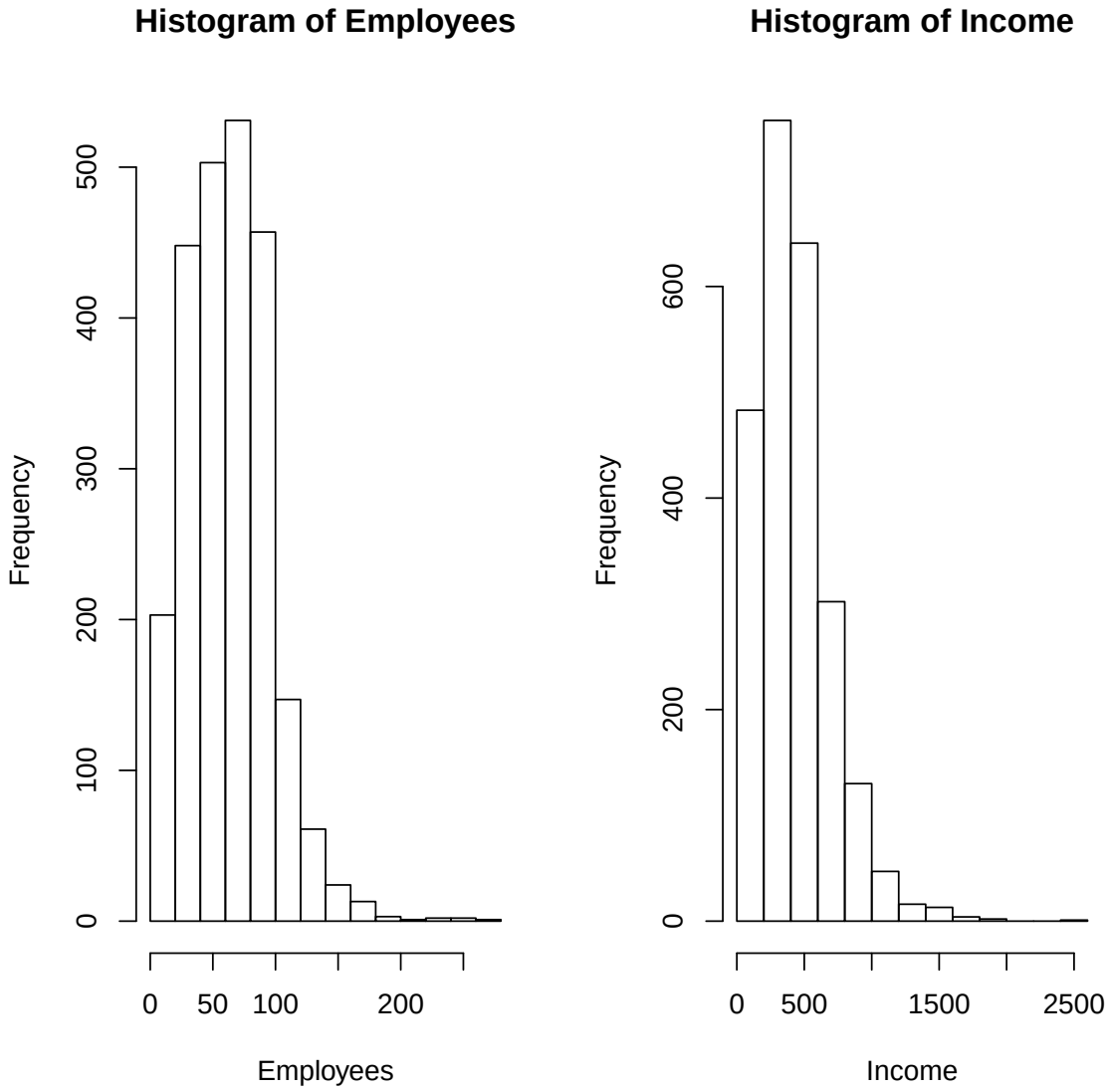
```
curve(dnorm(x),-3,3)
curve(dt(x,df=3),-3,3,add=T,lty=2)
legend(1,0.4,c("N(0,1)","t(5)"),lty=c(1,2))
```



1.7.6 Varias gráficas en una sola

Queremos crear una matriz de gráficas, para eso, primero abrimos los espacios con la función `par`. Por ejemplo

```
par(mfrow=c(1,2))
hist(Employees)
hist(Income)
```



1.7.7 Ejercicios

1. Elaboara una gráfica de diagramas de caja para ilustar el nivel de ingreso para empresas de diferentes tamaños (variable `Level`). Poner diferentes colores a las cajas, el eje y debe tener leyenda, y debe haber un título principal.

Índice de figuras

Índice de cuadros