Estadistica descriptiva con R y Python

Hector Miguel Palomares Maldonado

20/Agosto/2021

Estructuras de datos en R

Vectores Y tipos de datos

Defininir vectores

```
library(reticulate)
c(1,2,3,4,5)
```

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

```
rep("Thor", 5)
```

```
## [1] "Thor" "Thor" "Thor" "Thor" "Thor"
```

Para definir un vector desde consola

```
scan() ### Dar los valores, y doble enter para terminar
```

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

Si haces un vector de diferentes tipos de variables,‘r“ los convierte a un solo tipo de variable, y es de forma jerarquica

```
c(2, TRUE, 3.5 )
```

```
## [1] 2.0 1.0 3.5
```

Progresiones y secuencias

```
seq(5,60, by=5)
```

```
##  [1]  5 10 15 20 25 30 35 40 45 50 55 60
```

```r
seq(5,60, by=3.5)
```

```
##  [1]  5.0  8.5 12.0 15.5 19.0 22.5 26.0 29.5 33.0 36.5 40.0 43.5 47.0 50.5 54.0
## [16] 57.5
```

```r
seq(100, 6, by=-9 )
```

```
##  [1] 100  91  82  73  64  55  46  37  28  19  10
```

```r
seq(4,35, length.out=7)
```

```
## [1]  4.000000  9.166667 14.333333 19.500000 24.666667 29.833333 35.000000
```

```r
seq(4, length.out=7, by=3)
```

```
## [1]  4  7 10 13 16 19 22
```

Un vector mas elaborado

```r
c(rep(pi,5), 5:10, -7)
```

```
##  [1]  3.141593  3.141593  3.141593  3.141593  3.141593  5.000000  6.000000
##  [8]  7.000000  8.000000  9.000000 10.000000 -7.000000
```

```r
x<-c(1,2,3,4,5)
c(10, x, 20, x, 30)
```

```
##  [1] 10  1  2  3  4  5 20  1  2  3  4  5 30
```

Funciones y Orden de vectores

aplicar una función a todoslos elementos de un vector

```r
y<-  c(1:10)
sapply(y, FUN=function(elem){sqrt(elem)})
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
##  [9] 3.000000 3.162278
```

```r
cuadrado=function(x){x^2}
v=c(1,2,3,4,5,6)
sapply(v, cuadrado)
```

```
## [1]  1  4  9 16 25 36
```

```r
mean(v) #función promedio
```

```
## [1] 3.5
```

```r
cumsum(v) #suma acumulada``
```

```
## [1]  1  3  6 10 15 21
```

Tarea 1

```r
vector=c(5,1,6,2,7,3)
Ord=sort(vector)
rev(Ord)
```

```
## [1] 7 6 5 3 2 1
```

Binomio de Newton

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} \cdot b^k$$

```r
BinNew=function(a,b,n){
  cumsum(choose(n,(0:n))*a^{n-(0:n)}*b^(0:n))

}

BinNew(2,1,2)
```

```
## [1] 4 8 9
```

Subvectores y Filtros

```r
v=c(14,5,6,19,32,1,8,32,1,32, 5,1,1,2,3,4,6,32,2,4,6,1)
v[2]
```

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

```r
v[-c(3,5)] # elimina estas posiciones
```

```
##  [1] 14  5 19  1  8 32  1 32  5  1  1  2  3  4  6 32  2  4  6  1
```

```r
v[v != 19 & v > 15]
```

```
## [1] 32 32 32 32
```

Condiciones con which

```r
which(v>6)# las posiciones de los numeros  que cumplen
```

```
## [1]  1  4  5  7  8 10 18
```

```r
v[which(v>6)] #los numeros  que cumplen
```

```
## [1] 14 19 32  8 32 32 32
```

```r
which.min(v)
```

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

```r
which(v == min(v)) ## Posiciones del minimo
```

```
## [1]  6  9 12 13 22
```

```r
which.max(v)
```

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

```r
which(v == max(v)) ## Posiciones del maximo
```

```
## [1]  5  8 10 18
```

Valores NA

```r
v
```

```
##  [1] 14  5  6 19 32  1  8 32  1 32  5  1  1  2  3  4  6 32  2  4  6  1
```

```r
length(v)
```

```
## [1] 22
```

```r
v[length(v) + 8]=5
sum(v) ## Veamos que no se puede realizar esta operación y otras si tienen valores NA
```

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

```r
sum(v, na.rm=TRUE) ## Hace la operación dejando fuera a NA
```

```
## [1] 222
```

```r
which(is.na(v)) # posiciones de los NA
```

```
## [1] 23 24 25 26 27 28 29
```

```r
y=v
y[is.na(y)]=mean(y, na.rm=TRUE )
y
```

```
##  [1] 14.000000  5.000000  6.000000 19.000000 32.000000  1.000000  8.000000
##  [8] 32.000000  1.000000 32.000000  5.000000  1.000000  1.000000  2.000000
## [15]  3.000000  4.000000  6.000000 32.000000  2.000000  4.000000  6.000000
## [22]  1.000000  9.652174  9.652174  9.652174  9.652174  9.652174  9.652174
## [29]  9.652174  5.000000
```

```r
cumsum(x[!is.na(y)]) ##en cambio esta función si se puede realizar por que excluye a los NA
```

```
##  [1]  1  3  6 10 15 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [26] NA NA NA NA NA
```

```r
na.omit(y) ## ELIMINA los NA solo al realizar la operación, NO del arreglo original
```

```
##  [1] 14.000000  5.000000  6.000000 19.000000 32.000000  1.000000  8.000000
##  [8] 32.000000  1.000000 32.000000  5.000000  1.000000  1.000000  2.000000
## [15]  3.000000  4.000000  6.000000 32.000000  2.000000  4.000000  6.000000
## [22]  1.000000  9.652174  9.652174  9.652174  9.652174  9.652174  9.652174
## [29]  9.652174  5.000000
```

Factores

es como un vector pero con estructura mas rica que permite usarlo para clasificar observaciones, para definirlo creamos un vector y lo transformamos por medio de una de las funciones *factor* o *as.factor()*

```r
paises = c("Mexico", "España", "Africa", "china", "Mexico", "España")
paises.factor = factor(paises)
paises.factor
```

```
## [1] Mexico España Africa china  Mexico España
## Levels: Africa china España Mexico
```

```r
sexo=c("H","M","H","M","M")
gFactor=as.factor(sexo)
gFactor
```

```
## [1] H M H M M
## Levels: H M
```

```r
gFactor2=factor(sexo, levels=c("H","M","B")) ##ojo es factor NO as factor
gFactor2
```

```
## [1] H M H M M
## Levels: H M B
```

```r
gFactor3=factor(sexo, levels=c("H","M","B"), labels=c("hombre", "mujer", "Hermafrodita"))
gFactor3
```

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

```r
levels(gFactor3)=c("Masculino","Femenino","Hibrido") #Cambia las etiquetas
gFactor3
```

```
## [1] Masculino Femenino  Masculino Femenino  Femenino
## Levels: Masculino Femenino Hibrido
```

Factor Ordenado, los niveles siguen un orden

```r
notas=c(5, 7,8,10)
ordered(notas, labels=c("reprobado", "Acreditado", "notable", "Excelente"))
```

```
## [1] reprobado  Acreditado notable    Excelente
## Levels: reprobado < Acreditado < notable < Excelente
```

Listas

```r
x=c(1,-5,4,-7,9,-3,2,-8,-6,0)
L=list(nombres="Temperaturas", datos=x, media=mean(x), sumas=cumsum(x))
L
```

```
## $nombres
## [1] "Temperaturas"
##
## $datos
##  [1]  1 -5  4 -7  9 -3  2 -8 -6  0
##
## $media
## [1] -1.3
##
## $sumas
##  [1]   1  -4   0  -7   2  -1   1  -7 -13 -13
```

Aceder a la lista

```
L$media
```

```
## [1] -1.3
```

```
L[[3]]
```

```
## [1] -1.3
```

```
L[[2]] ## Operar con los elementos de la lista
```

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

```
L[2] #Muestra como  una lista
```

```
## $datos
##  [1]  1 -5  4 -7  9 -3  2 -8 -6  0
```

```
2*L[[2]]
```

```
##  [1]   2 -10   8 -14  18  -6   4 -16 -12   0
```

```
str(L) ## Estructura de la lista
```

```
## List of 4
##  $ nombres: chr "Temperaturas"
##  $ datos  : num [1:10] 1 -5 4 -7 9 -3 2 -8 -6 0
##  $ media  : num -1.3
##  $ sumas  : num [1:10] 1 -4 0 -7 2 -1 1 -7 -13 -13
```

```
names(L)
```

```
## [1] "nombres" "datos"   "media"   "sumas"
```

Matrices

definición \emph{matrix(vector, nrow=n, byrow=valor_logico)}

```
M = matrix(1:12 , nrow=4)
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```r
MF = matrix(1:12 , nrow=4, byrow = TRUE)
MF
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

```r
M1 = matrix(1:12 , nrow=3)
M1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```r
unosM=matrix(1, nrow = 4, ncol = 5)
unosM
```

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

Otra forma para definir matrices *rbind(vector1, vector2,...)* construye la matriz por filas *cbind(vector1, vector2,...)* construye la matriz por columnas los vectores deben tener la misma longitud esta función tambien añade fila(s), o columna(s) a una matriz

```r
ranadir=rbind(MF, c(1,2,3), c(-11, -12, -13)) ## añade filas a una matriz
ranadir
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
## [5,]    1    2    3
## [6,]  -11  -12  -13
```

```r
cbind(ranadir, c(22,33,55, 27,28,40), c(-111, -112, -113, 27, -100, -60)) # añade columnas a la matriz
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3   22 -111
## [2,]    4    5    6   33 -112
## [3,]    7    8    9   55 -113
## [4,]   10   11   12   27   27
## [5,]    1    2    3   28 -100
## [6,]  -11  -12  -13   40  -60
```

```r
M4=rbind(c(1,2,3), c(-1,-2,-3))
M4
```

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

```r
M5=cbind(c(1,2,3), c(-1,-2,-3))
M5
```

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

```r
diag(c(5,10,15)) ## solo elementos en la  diagonal
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    0
## [2,]    0   10    0
## [3,]    0    0   15
```

```r
diag(10, nrow = 4)
```

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

formas de acceder a las matrices : *m[i,j]*, por fila *m[i,]*, por columna *m[,j]*

```r
ranadir
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
## [5,]    1    2    3
## [6,]  -11  -12  -13
```

```r
ranadir[2,3]
```

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

```r
ranadir[2,]
```

```
## [1] 4 5 6
```

```
ranadir[,3]
```

```
## [1]   3   6   9  12   3 -13
```

```
subMatriz=ranadir[c(4,5,6), 2:3]
subMatriz
```

```
##      [,1] [,2]
## [1,]   11   12
## [2,]    2    3
## [3,]  -12  -13
```

```
diag(ranadir)
```

```
## [1] 1 5 9
```

```
nrow(ranadir)
```

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

```
ncol(ranadir)
```

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

```
dim(ranadir)
```

```
## [1] 6 3
```

```
sum(unosM)
```

```
## [1] 20
```

```
prod(ranadir)
```

```
## [1] -4.9318e+12
```

```
mean(ranadir)
```

```
## [1] 2.666667
```

```
colSums(ranadir)
```

```
## [1] 12 16 20
```

```
rowSums(ranadir)
```

```
## [1]   6  15  24  33   6 -36
```

```
colMeans(ranadir)
```

```
## [1] 2.000000 2.666667 3.333333
```

```
rowMeans(ranadir)
```

```
## [1]   2   5   8  11   2 -12
```

Aplicar funciones a matrices

```
apply(ranadir, MARGIN = 1, FUN=function(x){sum(sqrt(x^2))}) #por filas = 1
```

```
## [1]  6 15 24 33  6 36
```

```
apply(ranadir, MARGIN = 2, FUN=function(x){sum(sqrt(x^2))}) #por columnas = 2
```

```
## [1] 34 40 46
```

Un poco de Algebra lineal

La transpusta de la matriz

```
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
t(M)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

suma de matrices

```
M+M
```

```
##      [,1] [,2] [,3]
## [1,]    2   10   18
## [2,]    4   12   20
## [3,]    6   14   22
## [4,]    8   16   24
```

multplicación de componentes de dos matrices (mismo numero de entradas)

```
M4
```

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

```
M4*M4
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    9
## [2,]    1    4    9
```

Mulplicación de matices

```
M %*% t(M)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  107  122  137  152
## [2,]  122  140  158  176
## [3,]  137  158  179  200
## [4,]  152  176  200  224
```

Determinante de la matriz

```
Matris1=cbind(c(2,1), c(1,2))
Matris1
```

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

```
det(Matris1)
```

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

```
Matris2=cbind(c(1,4,2), c(0,1,3), c(1,8,9))
Matris2T=solve(Matris2)
Matris2T
```

```
##      [,1] [,2] [,3]
## [1,]    3 -0.6  0.2
## [2,]    4 -1.4  0.8
## [3,]   -2  0.6 -0.2
```

```
Matris2T %*% Matris2 ## Identidad
```

```
##               [,1]          [,2]         [,3]
## [1,]  1.000000e+00  1.110223e-16 2.220446e-16
## [2,]  4.440892e-16  1.000000e+00 8.881784e-16
## [3,] -1.110223e-16 -1.110223e-16 1.000000e+00
```

la función solve, tambien resuelve $Ax = B$ donde A es una Matriz y B un vector

```
t(Matris2)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    2
## [2,]    0    1    3
## [3,]    1    8    9
```

```
solve(t(Matris2), c(1,2,3))
```

```
## [1]  5.0 -1.6  1.2
```

```
eigen((t(Matris2))) ##Valores y Vectores Propios
```

```
## eigen() decomposition
## $values
## [1] 11.5677644 -1.0000000  0.4322356
##
## $vectors
##            [,1]       [,2]        [,3]
## [1,] -0.2744671  0.7427814 -0.98750842
## [2,] -0.2626036 -0.5570860  0.15481805
## [3,] -0.9250444  0.3713907 -0.02930006
```

```
eigen(t(Matris2))$values
```

```
## [1] 11.5677644 -1.0000000  0.4322356
```

```
eigen(t(Matris2))$vectors
```

```
##            [,1]       [,2]        [,3]
## [1,] -0.2744671  0.7427814 -0.98750842
## [2,] -0.2626036 -0.5570860  0.15481805
## [3,] -0.9250444  0.3713907 -0.02930006
```

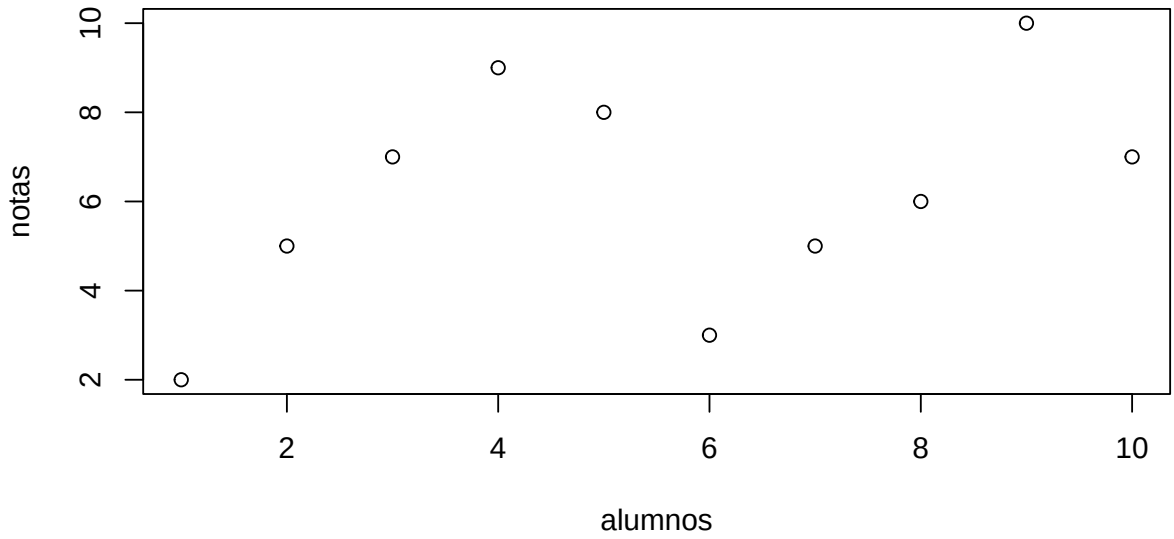
```
## Me equivoque en la matriz, en el ejemplo era la traspuesta, por eso pongo t(Matriz)
```

Introducción a la representacion grafica

Gráfico básico de puntos

- `plot(x,y)`: para dibujar un gráfico básico de puntos siendo x, y vectores numéricos
  - `plot(x)` = `plot(1:length(x),x)`
- `plot(x,función)`: para dibujar el gráfico de una función

```r
alumnos = c(1:10)
notas = c(2,5,7,9,8,3,5,6,10,7)
plot(alumnos,notas)
```

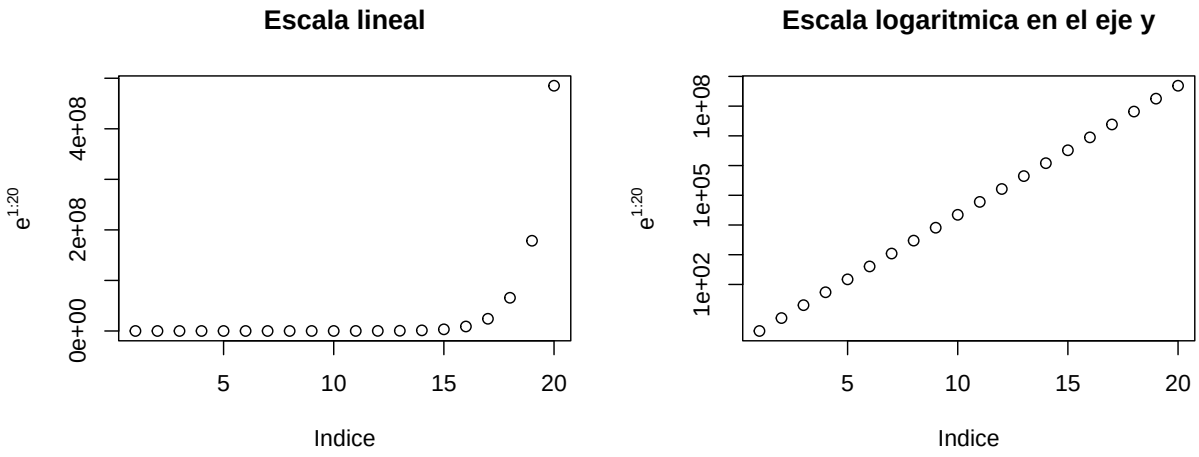


Parámetros de la función plot()

- `log`: para indicar que queremos el gráfico en escala logarítmica
- `main("título")`: para poner título al gráfico. Si en vez de un texto se puede poner una expresión matemática, utilizaremos la función `expression()`
- `xlab("etiqueta")`: para poner etiqueta al eje X
- `ylab("etiqueta")`: para poner etiqueta al eje Y
- `pch=n`: para elegir el símbolo de los puntos. $n = 0, 1, ..., 25$. El valor por defecto es `pch = 1`
- `cex`: para elegir el tamaño de los símbolos
- `col="color en inglés"`: para elegir el color de los símbolos. Gama de colores.

```r
par(mfrow = c(1,2))
plot = plot(exp(1:20), xlab = "Indice", ylab = expression(e^{1:20}),
            main = "Escala lineal")
plotLog = plot(exp(1:20), log = "y", xlab = "Indice", ylab = expression(e^{1:20}),
               main = "Escala logaritmica en el eje y")
```

Escala logarítmica



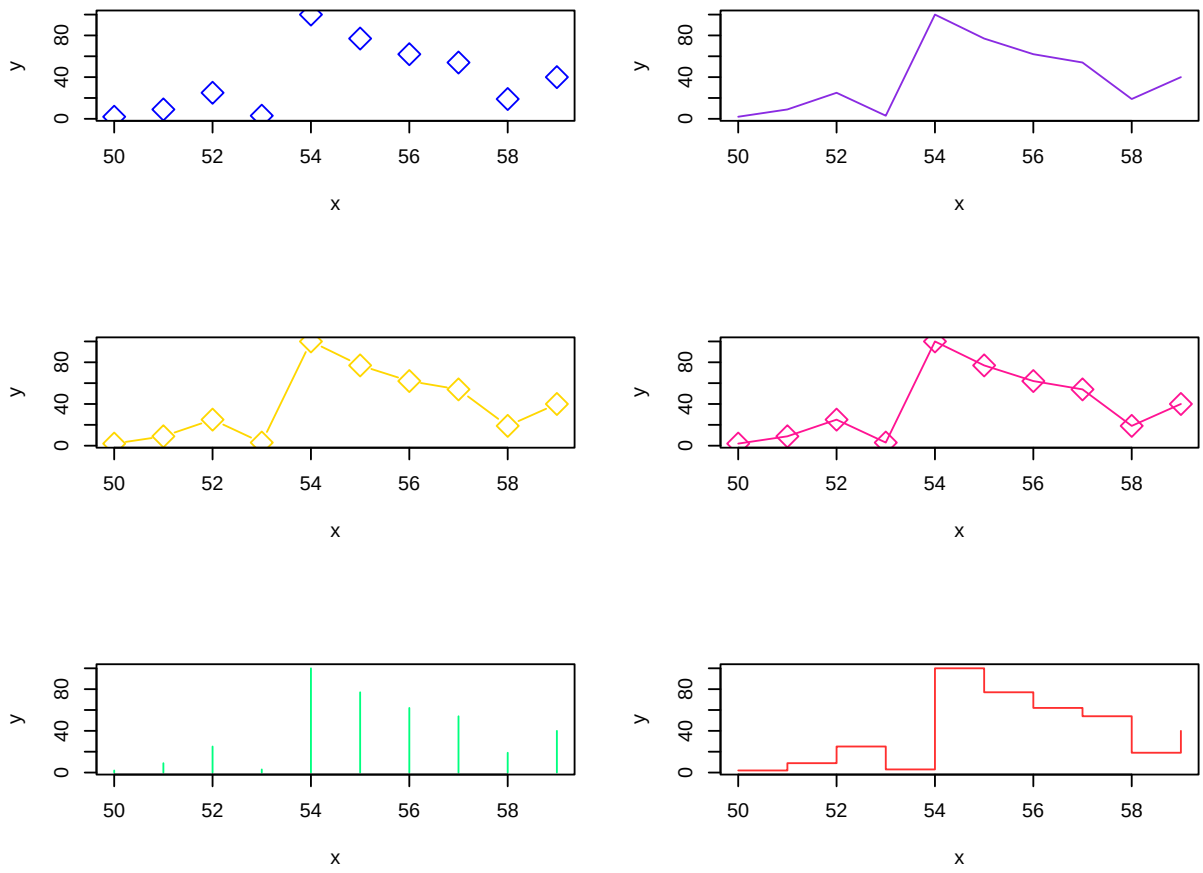

```
par(mfrow = c(1,1))
```

Más Parámetros de la función plot()

- `type`: para elegir el tipo de gráfico que queremos:
  - `p`: puntos (valor por defecto)
  - `l`: líneas rectas que unen los puntos (dichos puntos no tienen símbolo)
  - `b`: líneas rectas que unen los puntos (dichos puntos tienen símbolo). Las líneas no traspasan los puntos
  - `o`: como el anterior pero en este caso las líneas sí que traspasan los puntos
  - `h`: histograma de líneas
  - `s`: histograma de escalones
  - `n`: para no dibujar los puntos

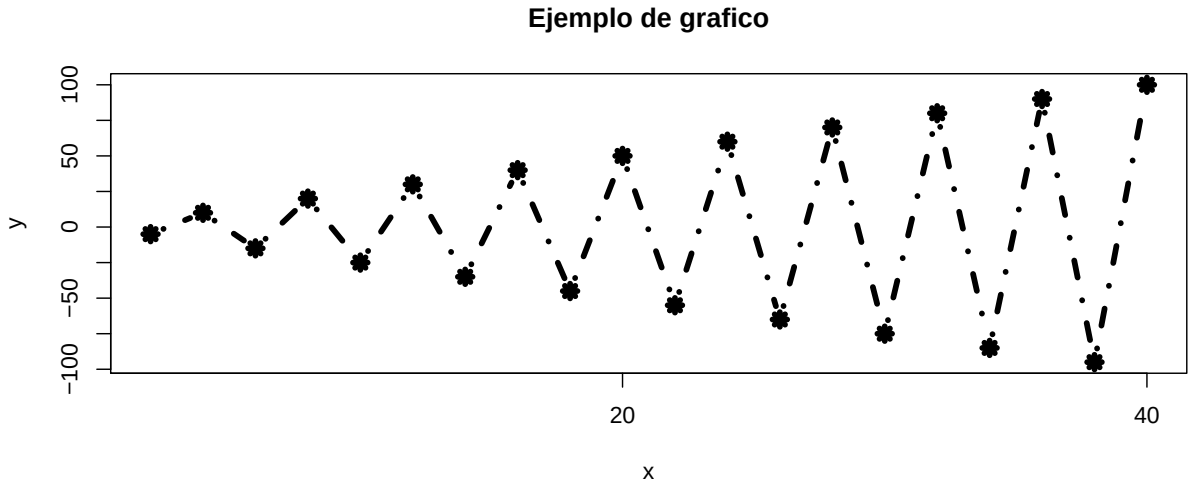
```
par(mfrow = c(3,2))
x = c(50:59)
y = c(2,9,25,3,100,77,62,54,19,40)
plot(x,y, pch = 23, cex = 2, col = "blue", type = "p")
plot(x,y, pch = 23, cex = 2, col = "blueviolet", type = "l")
plot(x,y, pch = 23, cex = 2, col = "gold", type = "b")
plot(x,y, pch = 23, cex = 2, col = "deeppink", type = "o")
plot(x,y, pch = 23, cex = 2, col = "springgreen", type = "h")
plot(x,y, pch = 23, cex = 2, col = "firebrick1", type = "s")
par(mfrow = c(1,1))
```

Tipos de gráfico



- `lty`: para especificar el tipo de línea
  - "solid" : 1: línea continua (valor por defecto)
  - "dashed" : 2: línea discontinua
  - "dotted" : 3: línea de puntos
  - "dotdashed" : 4: línea que alterna puntos y rayas
- `lwd`: para especificar el grosor de las líneas
- `xlim`: para modificar el rango del eje X
- `ylim`: para modificar el rango del eje Y
- `xaxp`: para modificar posiciones de las marcas en el eje X
- `yaxp`: para modificar posiciones de las marcas en el eje Y

```
x = (2*(1:20))
y = (-1)^(1:20)*5*(1:20)
plot(x,y, main = "Ejemplo de grafico", pch = 8, cex = 1, type = "b", lty = 4, lwd = 4,
     xaxp = c(0,40,2), yaxp = c(-100,100,8))
```

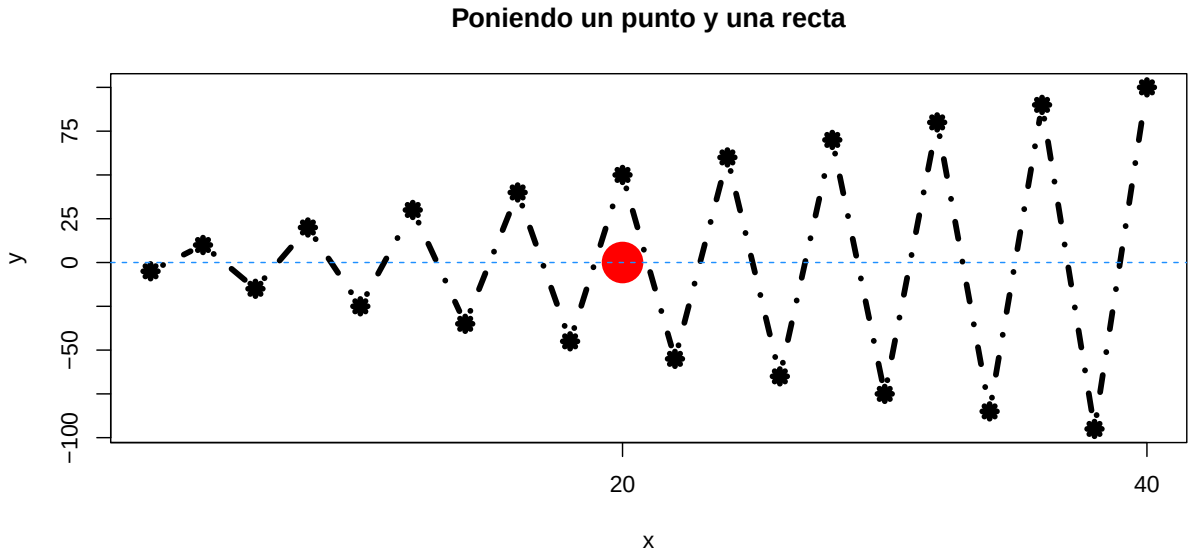


Añadir elementos al gráfico

- `points(x,y)`: añade un punto de coordenadas (x, y) a un gráfico ya existente
- `abline`: para añadir una recta a un gráfico ya existente
  - `abline(a,b)`: añade la recta $y = bx + a$
  - `abline(v = x0)`: añade la recta vertical $x = x_0$. v puede estar asignado a un vector
  - `abline(h = y0)`: añade la recta horizontal $y = y_0$. h puede estar asignado a un vector

Añadiendo punto y recta

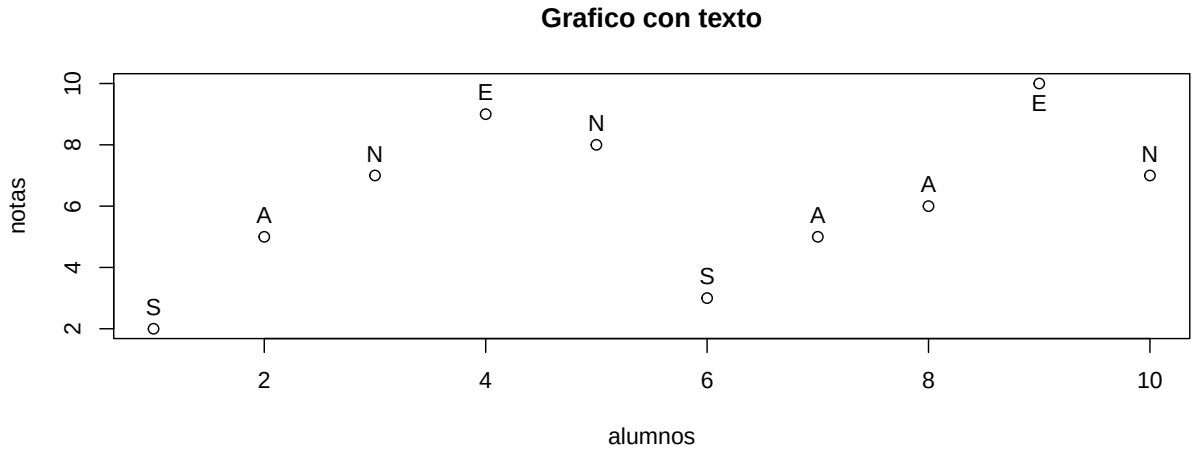
```
x = (2*(1:20))
y = (-1)^(1:20)*5*(1:20)
plot(x,y, main = "Poniendo un punto y una recta", pch = 8, cex = 1, type = "b", lty = 4,
     lwd = 4, xaxp = c(0,40,2), yaxp = c(-100,100,8))
points(20,0, col = "red", cex = 4, pch = 16)
abline (h = 0, lty = 2, col = "dodgerblue")
```



- `text(x,y,labels = "....")`: añade en el punto de coordenadas (x, y) el texto especificado como argumento de labels
  - `pos`: permite indicar la posición del texto alrededor de las coordenadas (x, y). Admite los siguientes valores:
    - 1: abajo
    - 2: izquierda
    - 3: arriba
    - 4: derecha
    - 5: sin especificar: el texto se sitúa centrado en el punto (x, y)

Añadiendo etiquetas

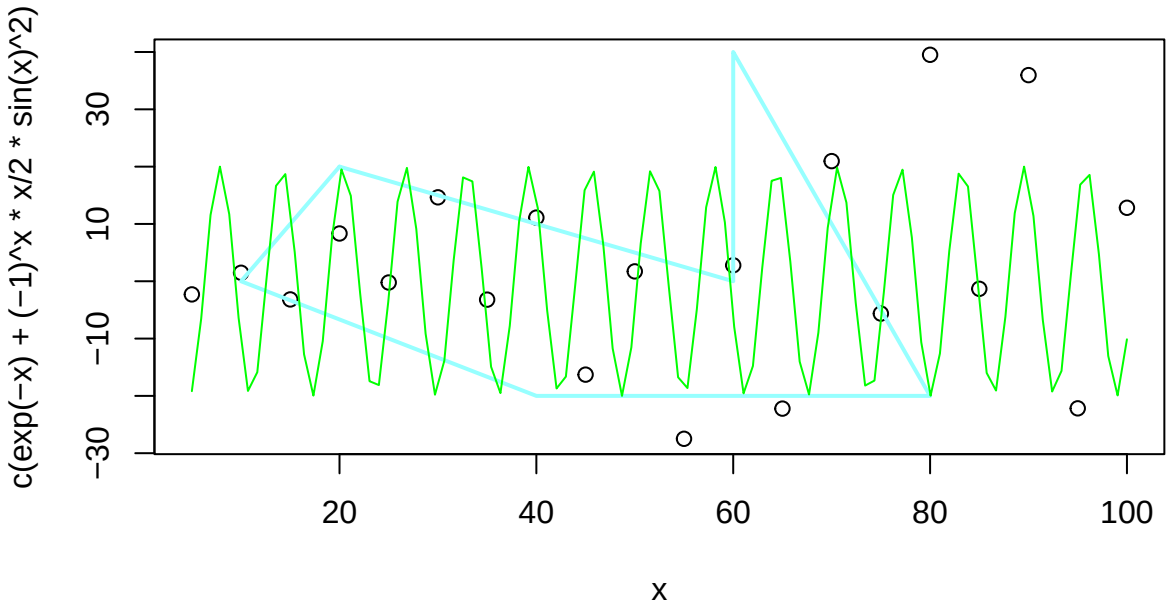
```
alumnos = c(1:10)
notas = c(2,5,7,9,8,3,5,6,10,7)
plot(alumnos,notas, main = "Grafico con texto")
text(alumnos,notas, labels = c("S","A","N","E","N","S","A","A","E","N"),
     pos = c(rep(3,times = 8),1,3))
```



- `lines(x, y)`:añade a un gráfico existente una línea poligonal que une los puntos (x_i, y_i) sucesivos. x, y son vectores numéricos
- `curve(curva)`: permite añadir la gráfica de una curva a un gráfico existente
  - `add=TRUE`: si no, la curva no se añade
  - La curva se puede especificar mediante una expresión algebraica con variable x, o mediante su nombre si la hemos definido antes

```
x = c(5*(1:20))
plot(x,c(exp(-x)+(-1)^x*x/2*sin(x)^2))
lines(c(20,10,40,80,60,60,20),c(20,0,-20,-20,40,0,20), lwd = 2, col = "darkslategray1")
curve(20*sin(x), add = TRUE, col = "green")
```

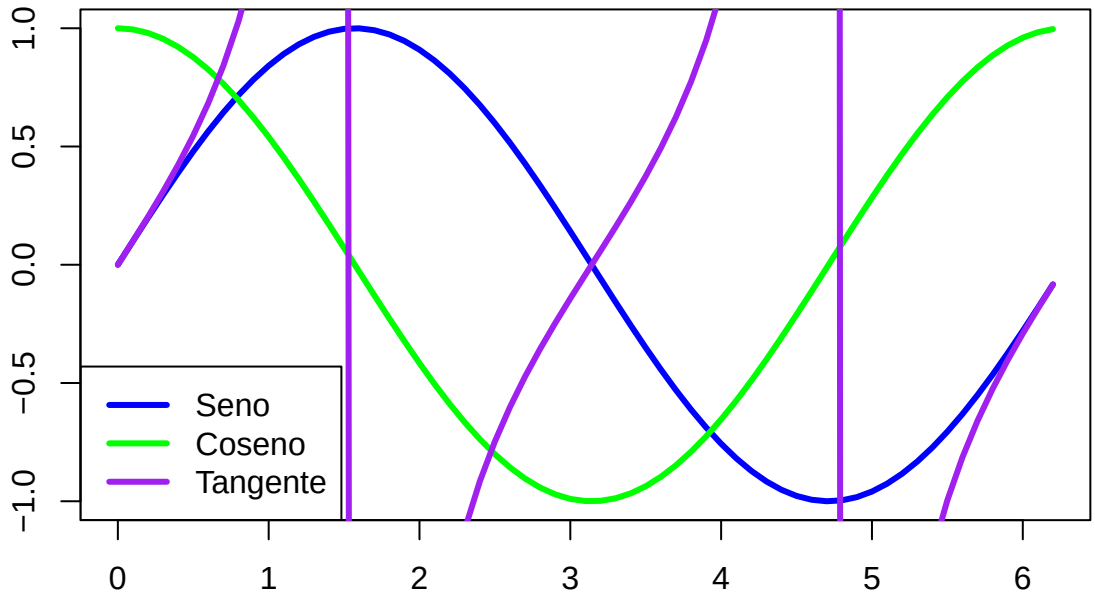
Añadiendo líneas y curvas



- `legend(posición, legend = ...)`: para añadir una leyenda
  - La posición indica donde queremos situar la leyenda. Puede ser o bien las coordenadas de la esquina superior izquierda de nuestra leyenda, o bien una de las palabras siguientes:
    - "bottom" / "bottomright" / "bottomleft"
    - "top" / "topright" / "topleft"
    - "center" / "right" / "left"
  - `legend`: contiene el vector de nombres entre comillas con los que queremos identificar a las curvas en la leyenda

Añadiendo leyenda

```
x = seq(0,2*pi,0.1)
plot(x,sin(x),type="l",col="blue",lwd=3, xlab="", ylab="")
lines(x,cos(x),col="green",lwd=3)
lines(x, tan(x), col="purple",lwd=3)
legend("bottomleft",col=c("blue","green","purple"), legend=c("Seno","Coseno", "Tangente"),
       lwd=3, bty="l")
```

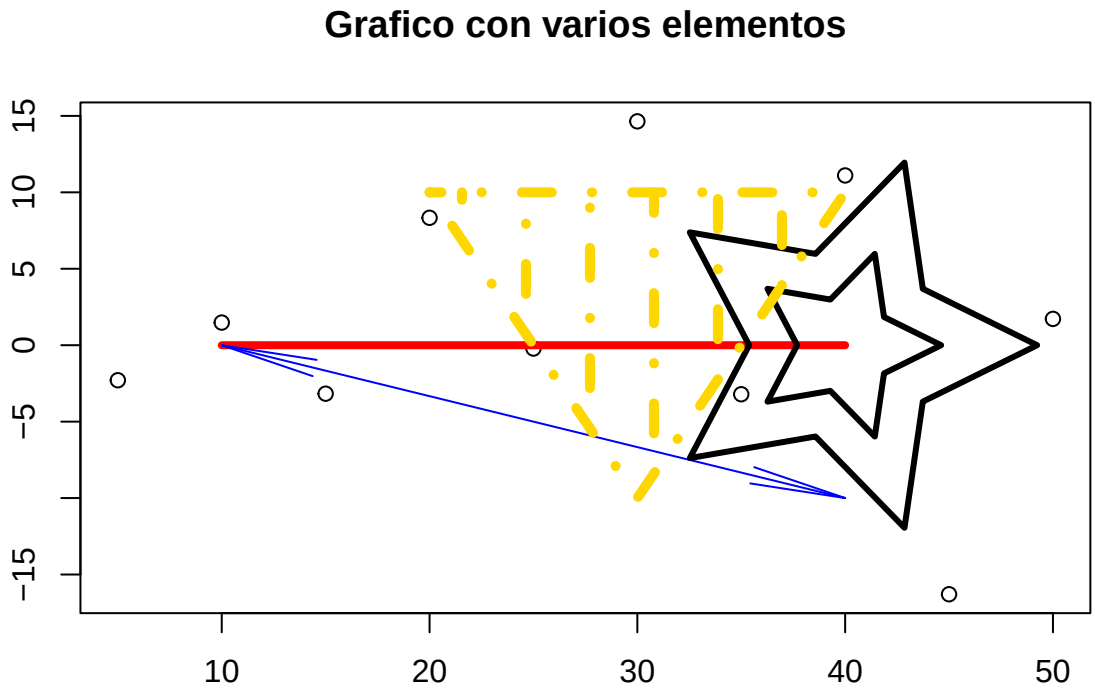


Segmentos, Fechas, Simbolos y Poligonos

Añadir elementos al gráfico

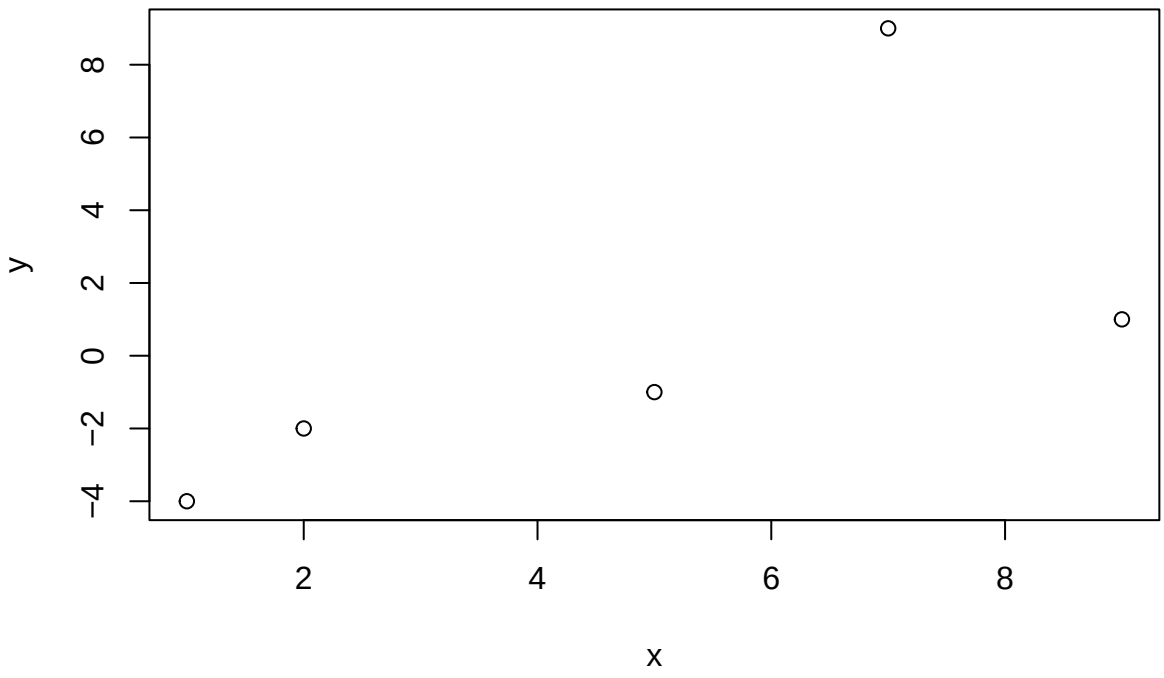
- `segments`: para añadir segmentos a un gráfico existente
- `arrows`: para añadir flechas a un gráfico existente
- `symbols`: para añadir símbolos a un gráfico existente
- `polygon`: para añadir polígonos cerrados especificando sus vértices a un gráfico existente

```
x = c(5*(1:10))
plot(x,c(exp(-x)+(-1)^x*x/2*sin(x)^2), xlab = "", ylab = "",
     main = "Grafico con varios elementos")
segments(10,0,40,0, col = "red", lwd = 4)
arrows(10,0,40,-10, col = " blue", length = 0.5, angle = 5, code = 3)
symbols(40,0,stars = cbind(1,.5,1,.5,1,.5,1,.5,1,.5), add = TRUE, lwd = 3, inches = 0.5)
symbols(40,0,stars = cbind(1,.5,1,.5,1,.5,1,.5,1,.5), add = TRUE, lwd = 3)
polygon(c(20,30,40),c(10,-10,10), col = "gold", density = 3, angle = 90, lty = 4,
        lwd = 5)
```



Extra

```
x=c(2,7,1,9,5)
y=c(-2,9,-4,1,-1)
plot(x,y)
```




```
f <- function(x){sqrt(x)}
plot(f)
```

```
##  [1]    1    1    2    3    5    8   13   21   34   55   89  144  233  377  610
## [16]  987 1597 2584 4181 6765
```

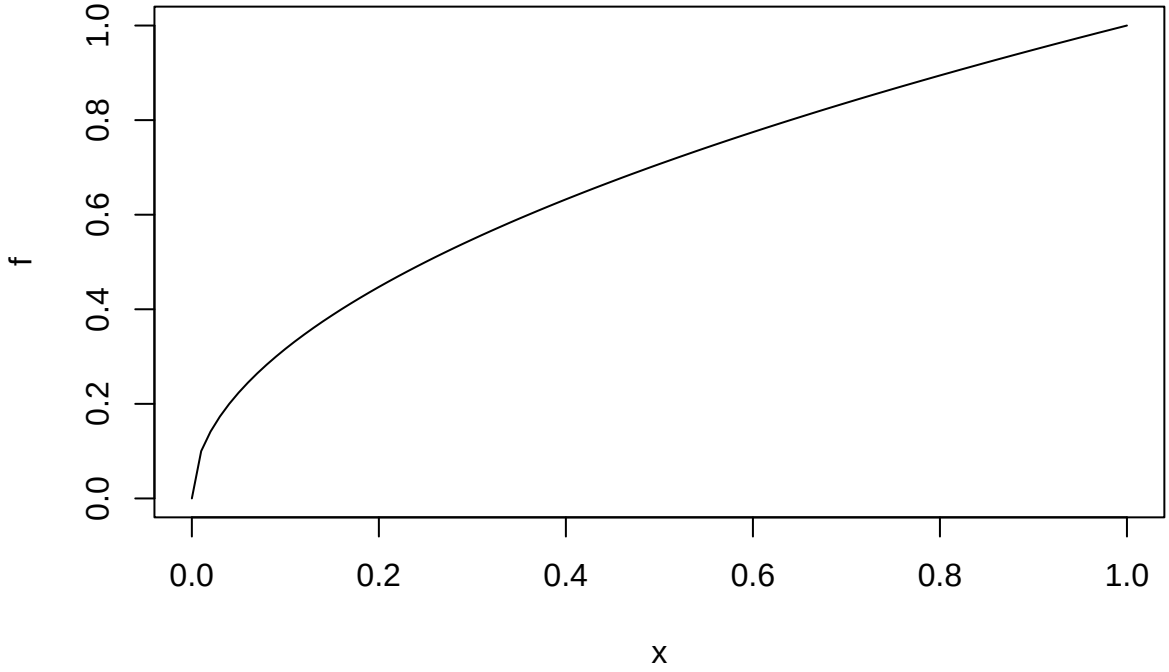
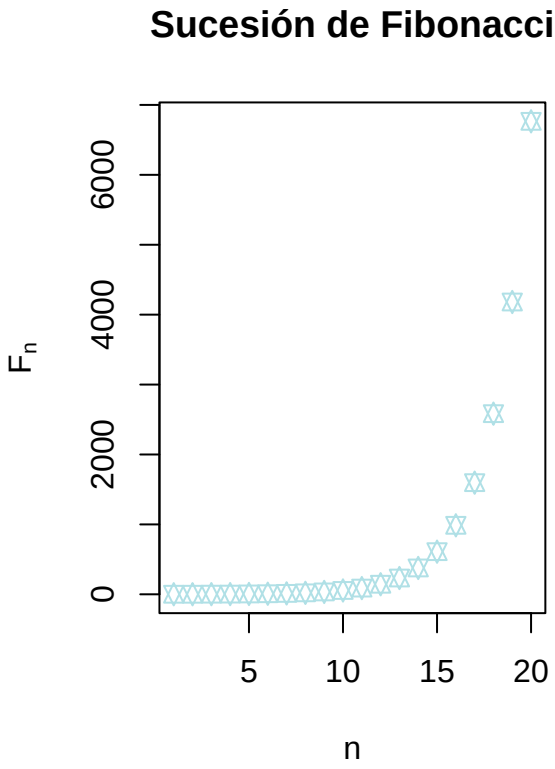
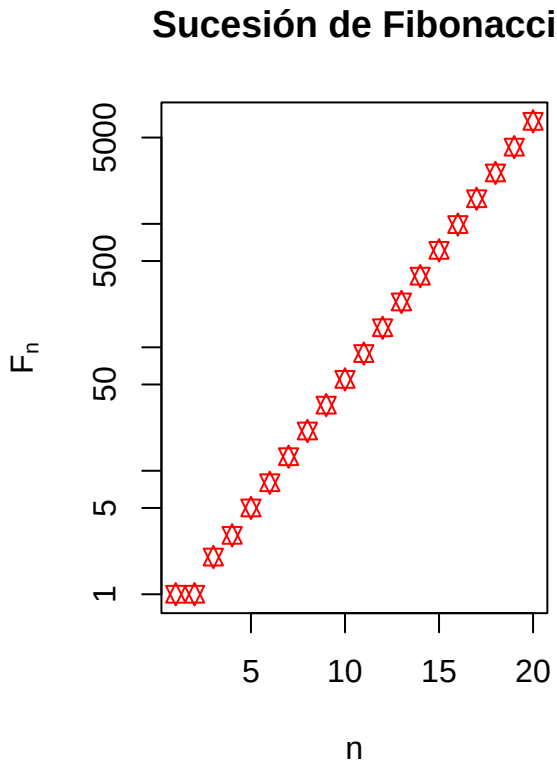
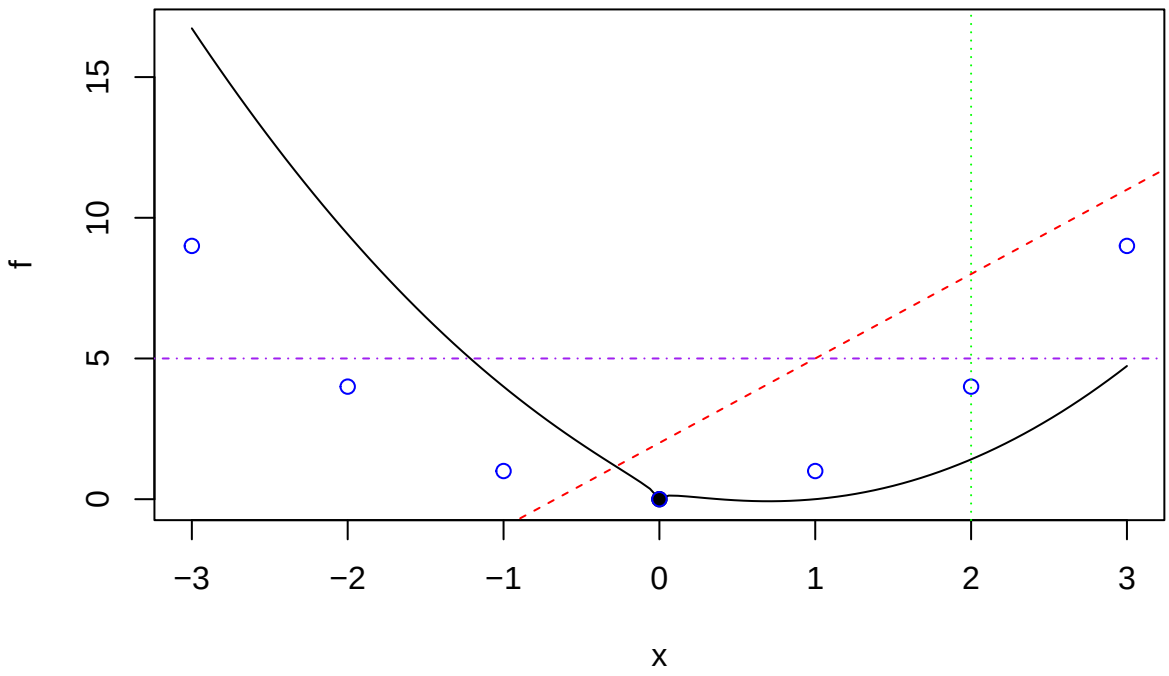


Figure 1: grafica de la función sqrt

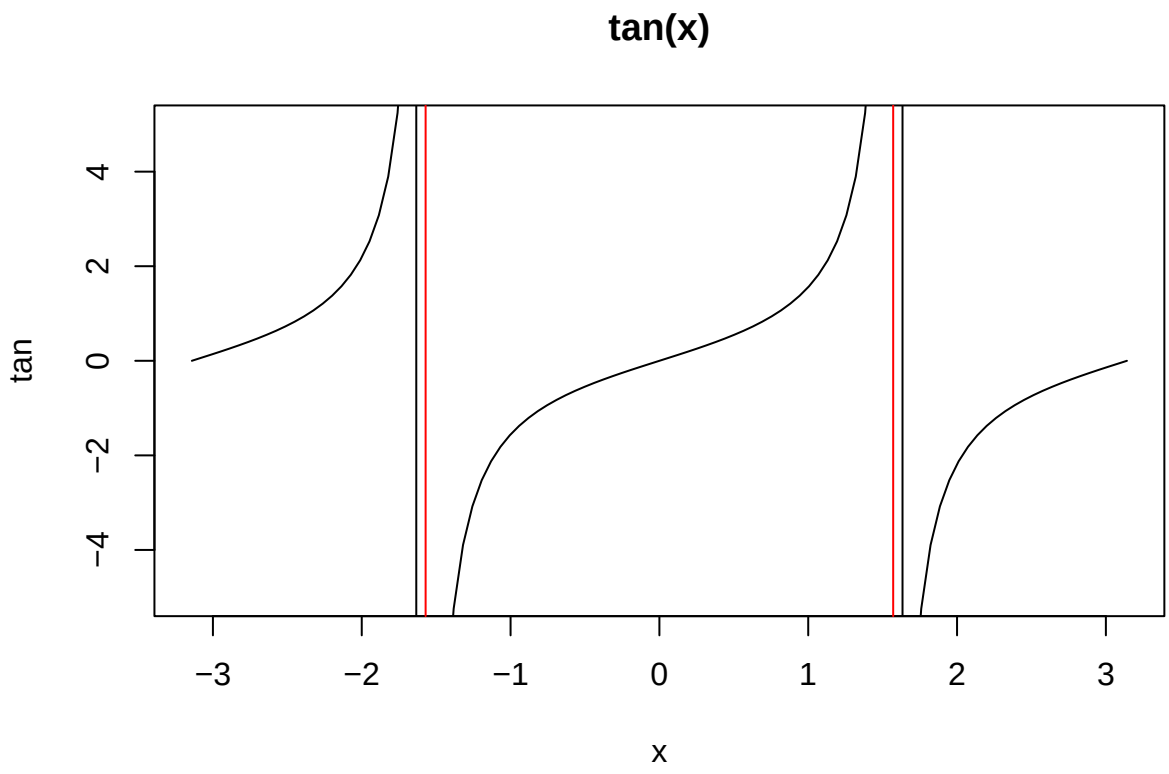




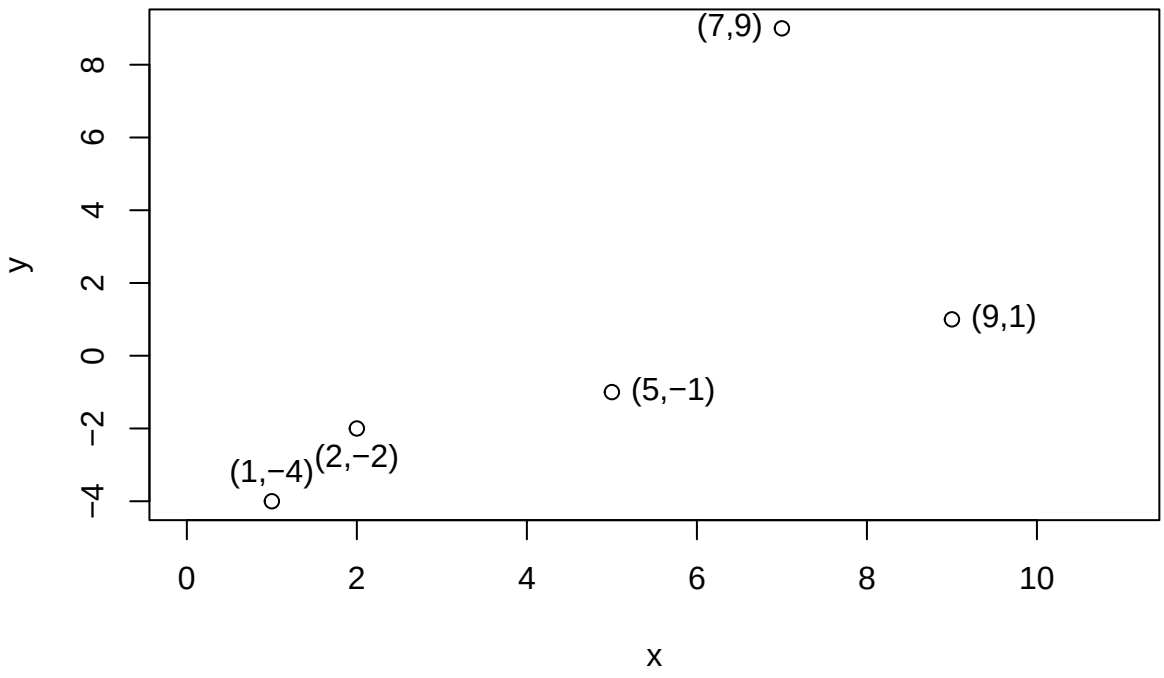

```r
f <- function(x){
   x^2-2*x + sqrt(abs(x))
}
plot(f, xlim = c(-3,3))
points(0,0, pch = 19) ## Coloca un punto en 0,0  de  tamaño 19
points(-3:3, (-3:3)^2, col = "blue") ##  puntos que forma una parabola
abline(2,3, lty = "dashed", col = "red") ## pendiente , ordenada
abline(v = 2, lty = "dotted", col = "green") ## v=vertical
abline(h = 5, lty = "dotdash", col = "purple") ## h=horizontal
```



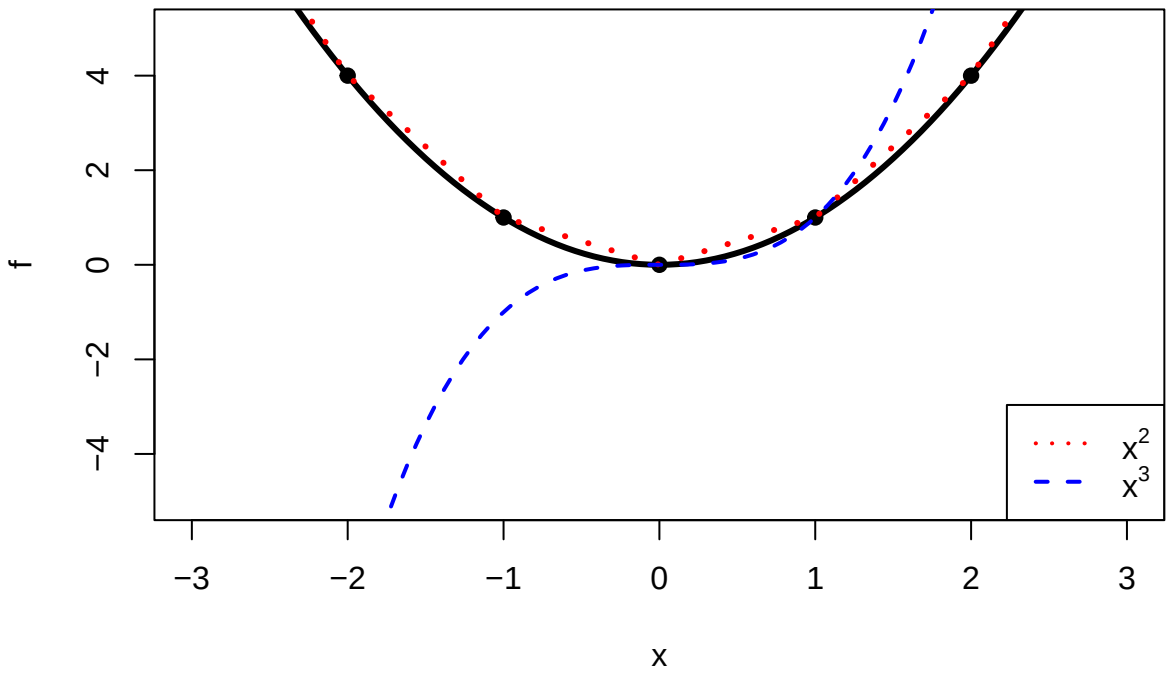

```
plot(tan, main="tan(x)" ,xlim = c(-pi,pi), ylim = c(-5,5))
abline(v=c(-pi/2, pi/2), col="red")
```



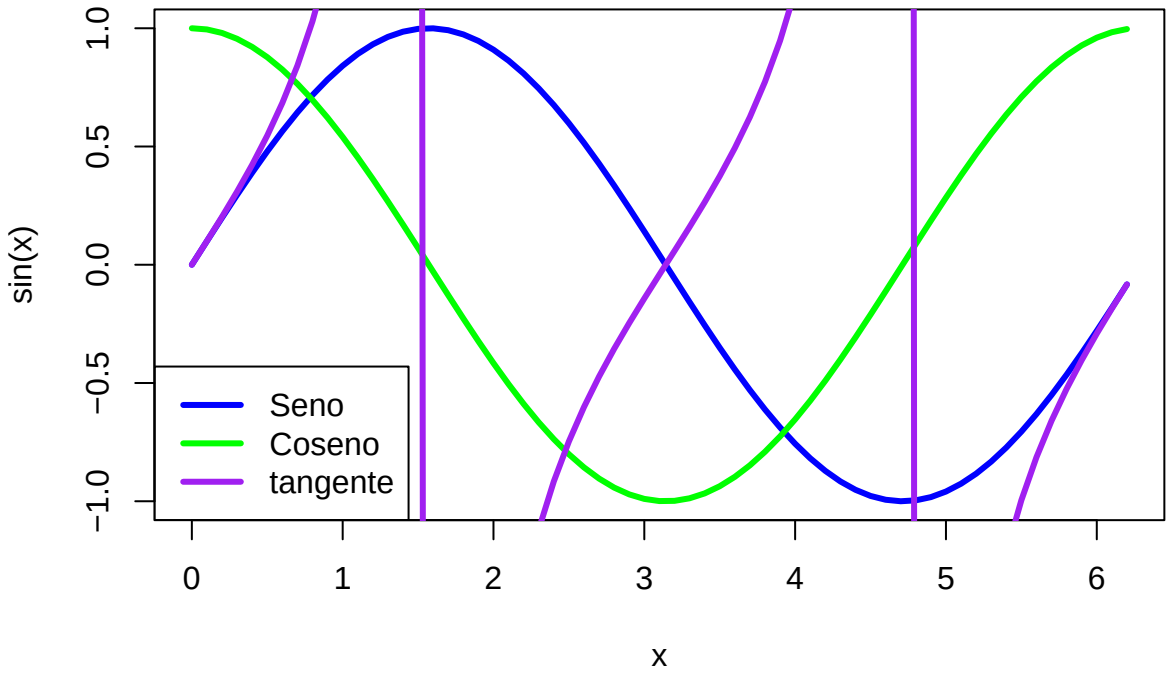

```
x=c(2,7,1,9,5)
y=c(-2,9,-4,1,-1)
plot(x,y, main = "Grafico de puntos ", xlim = c(0,11))
text(x,y, labels=c("(2,-2)", "(7,9)", "(1,-4)", "(9,1)", "(5,-1)"), pos = c(1,2,3,4,4))
```




```
f <- function(x){x^2}
plot(f, xlim = c(-3,3), ylim = c(-5,5), lwd = 3)
points(-3:3, f(-3:3), pch = 19)
lines(-3:3, f(-3:3), lwd=3, lty="dotted", col="red")
curve(x^3, lty = "dashed", col= "blue", add=TRUE, lwd=2)
legend("bottomright", legend = c(expression(x^2), expression(x^3)), lwd=2, col=c("red","blue"), lty=c("
```




```
x=seq(0, 2*pi, 0.1)
plot(x, sin(x), type="l", col="blue", lwd=3)
lines(x, cos(x), col="green", lwd=3)
lines(x, tan(x), col="purple", lwd=3)
legend("bottomleft", col=c("blue", "green", "purple"),  legend=c("Seno", "Coseno", "tangente"),  lwd=3,
```



DataFrames

Un data frame es una tabla de doble entrada, formada por variables en las columnas y observaciones de estas variables en las filas, de manera que cada fila contiene los valores de las variables para un mismo caso o un mismo individuo.

`data()`: para abrir una ventana con la lista de los objetos de datos a los que tenemos acceso en la sesión actual de R (los que lleva la instalación básica de R y los que aportan los paquetes que tengamos cargados.

- Si entramos `data(package=.packages(all.available = TRUE))` obtendremos la lista de todos los objetos de datos a los que tenemos acceso, incluyendo los de los paquetes que tengamos instalados, pero que no estén cargados en la sesión actual.

Obteniendo información del data frame

- `head(d.f,n)`: para mostrar las n primeras filas del data frame. Por defecto se muestran las 6 primeras filas
- `tail(d.f,n)`: para mostrar las n últimas filas del data frame. Por defecto semuestran las 6 últimas
- `str(d.f)`: para conocer la estructura global de un data frame
- `names(d.f)`: para producir un vector con los nombres de las columnas

Acontinuacion, se muestra un datafreme llamado iris, que viene por defecto

```r
df = iris
head(df,5)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

```r
tail(df,5)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```r
names(df)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```r
str(df)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Estructura y filtrado de DataFrames

Obteniendo información del data frame

- `rownames(d.f)`: para producir un vector con los identificadores de las filas `R` entiende siempre que estos identificadores son palabras, aunque sean números, de ahí que los imprima entre comillas
- `colnames(d.f)`: para producir un vector con los identificadores de las columnas
- `dimnames(d.f)`: para producir una list formada por dos vectores (el de los identificadores de las filas y el de los nombres de las columnas)
- `nrow(d.f)`: para consultar el número de filas de un data frame
- `ncol(d.f)`: para consultar el número de columnas de un data frame
- `dim(d.f)`: para producir un vector con el número de filas y el de columnas
- `d.f$nombre_variable` : para obtener una columna concreta de un dataframe

  - El resultado será un vector o un factor, según cómo esté definida la columna dentro del data frame
  - Las variables de un data frame son internas, no están definidas en el entorno global de trabajo de R

```
df$Petal.Length[1:10]
```

```
##  [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5
```

```
df$Species[1:10]
```

```
##  [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa setosa
## Levels: setosa versicolor virginica
```

Sub-data frames

`d.f[n,m]`: para extraer "trozos" del data frame por filas y columnas (funciona exactamente igual que en matrices) donde n y m pueden definirse como:

- intervalos
- condiciones
- números naturales
- no poner nada
- Si sólo queremos definir la subtabla quedándonos con algunas variables, basta aplicar el nombre del data frame al vector de variables
- Estas construcciones se pueden usar también para reordenar las filas o columnas

```
df[1:10,]
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
```

```
df[1:10,2:4]
```

```
##    Sepal.Width Petal.Length Petal.Width
## 1          3.5          1.4         0.2
## 2          3.0          1.4         0.2
## 3          3.2          1.3         0.2
## 4          3.1          1.5         0.2
```

```
## 5          3.6          1.4         0.2
## 6          3.9          1.7         0.4
## 7          3.4          1.4         0.3
## 8          3.4          1.5         0.2
## 9          2.9          1.4         0.2
## 10         3.1          1.5         0.1
```

```
df[df$Species == "setosa" & df$Sepal.Width > 4,  ]
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 16          5.7         4.4          1.5         0.4  setosa
## 33          5.2         4.1          1.5         0.1  setosa
## 34          5.5         4.2          1.4         0.2  setosa
```

Leyendo tablas de datos

- `read.table()`: para definir un data frame a partir de una tabla de datos contenida en un fichero
  - Este fichero puede estar guardado en nuestro ordenador o bien podemos conocer su url. Sea cual sea el caso, se aplica la función al nombre del fichero o a la dirección entre comillas

Parámetros de read.table()

- `header = TRUE`: para indicar si la tabla que importamos tiene una primera fila con los nombres de las columnas. El valor por defecto es FALSE
- `col.names = c(...)`: para especificar el nombre de las columnas. No olvidéis que cada nombre debe ir entre comillas
- `sep`: para especificar las separaciones entre columnas en el fichero (si no es un espacio en blanco). Si es así, hay que introducir el parámetro pertinente entre comillas
- `dec`: para especificar el signo que separa la parte entera de la decimal (si no es un punto. Si es así, hay que introducir el parámetro pertinente entre comillas

Dataset desde URL

```
path = "https://raw.githubusercontent.com/joanby/r-basic/master/data/StudentsData"
students = read.table(path,
  col.names = c("technicalDisciplines","aptitude","maths", "language",
                "generalKnowledge"))
head(students,8)
```

```
##   technicalDisciplines aptitude maths language generalKnowledge
## 1                    1       23    50       59               10
## 2                    1       70    30       79               18
## 3                    1       25    50       86               22
## 4                    1        0    40       67               16
## 5                    1        0    25       70               20
## 6                    1       20    60       78               22
## 7                    1       97    65       75               22
## 8                    1        1    45       61               22
```

Más parámetros de read.table()

- `stringsAsFactors`: para prohibir la transformación de las columnas de palabras en factores debemos usar `stringsAsFactors=FALSE` (ya que por defecto, R realiza dicha transformación)
- Para importar un fichero de una página web segura (cuyo url empiece con https), no podemos entrar directamente la dirección en `read.table()`; una solución es instalar y cargar el paquete RCurl y entonces usar la instrucción `read.table (textConnection(getURL("url ")),...)`.

Leyendo diferentes tipos de fichero

- `read.csv()`: para importar ficheros en formato CSV
- `read.xls()` o `read.xlsx()`: para importar hojas de cálculo tipo Excel u OpenOffice en formato XLS o XLSX, respectivamente. Se necesita el paquete xlsx
- `read.mtb()`: para importar tablas de datos Minitab. Se necesita el paquete foreign
- `read.spss()`: para importar tablas de datos SPSS. Se necesita el paquete foreign

Exportando datos a ficheros

- `write.table(df, file = "")`: para exportar un data frame a un fichero
  - `file = ""`: es donde indicaremos el nombre que queremos darle al fichero
  - Podemos usar el parámetro `sep` para indicar el símbolo de separación de columnas. Siempre entre comillas
  - También podemos utilizar el parámetro `dec` para indicar la separación entre la parte entera y decimal de los datos

Construyendo data frames

- `data.frame(vector_1,...,vector_n)`: para construir un data frame a partir de vectores introducidos en el orden en el que queremos disponer las columnas de la tabla
  - R considera del mismo tipo de datos todas las entradas de una columna de un data frame
  - Las variables tomarán los nombres de los vectores. Estos nombres se pueden especificar en el argumento de `data.frame` entrando una construcción de la forma `nombre_variable = vector`
  - `rownames`: para especificar los identificadores de las filas
  - También en esta función podemos hacer uso del parámetro `stringsAsFactors` para evitar la transformación de las columnas de tipo palabra en factores

```
Algebra = c(1,2,0,5,4,6,7,5,5,8)
Analysis = c(3,3,2,7,9,5,6,8,5,6)
Statistics = c(4,5,4,8,8,9,6,7,9,10)
grades = data.frame(Alg = Algebra, An = Analysis, Stat = Statistics)
str(grades)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ Alg : num  1 2 0 5 4 6 7 5 5 8
##  $ An  : num  3 3 2 7 9 5 6 8 5 6
##  $ Stat: num  4 5 4 8 8 9 6 7 9 10
```

```
gender = c("H", "H", "H", "M", "M")
age =    c( 23,  45,  20,  30,  18)
family = c(  2,   3,   4,   2,   5)
df3=data.frame(genero=gender, edad=age, familia=family)
row.names(df3) = c("P1", "P2", "P3", "P4", "P5")
df3
```

```
##    genero edad familia
## P1      H   23       2
## P2      H   45       3
## P3      H   20       4
## P4      M   30       2
## P5      M   18       5
```

```
str(df3)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ genero : chr  "H" "H" "H" "M" ...
##  $ edad   : num  23 45 20 30 18
##  $ familia: num  2 3 4 2 5
```

```
dimnames(df3)=list(
  c("Antonio", "Ricardo","JuanGabriel","Maria","Margarita"),
  c("Sexo", "años","MiembrosFamilia"))
df3
```

```
##             Sexo años MiembrosFamilia
## Antonio        H   23               2
## Ricardo        H   45               3
## JuanGabriel    H   20               4
## Maria          M   30               2
## Margarita      M   18               5
```

```
df4=df3
```

Construyendo data frames

- `fix(d.f)`: para crear / editar un data frame con el editor de datos
- `names(d.f)`: para cambiar los nombres de las variables
- `rownames(d.f)`: para modificar los identificadores de las filas. Han de ser todos diferentes
- `dimnames(d.f)=list(vec_nom_fil, vec_nom_col)`: para modificar el nombre de las filas y de las columnas simultáneamente
- `d.f[núm_fila,] = c(...)`: para añadir una fila a un data frame
  - Las filas que añadimos de esta manera son vectores, y por tanto sus entradas han de ser todas del mismo tipo
  - Si no añadimos las filas inmediatamente siguientes a la última fila del data frame, los valores entre su última fila y las que añadimos quedarán no definidos y aparecerán como NA

  - Para evitar el problema anterior, vale más usar la función `rbind()` para concatenar el data frame con la nueva fila

```
df3 = rbind(df3, c("H", 30, 1))
df3
```

```
##             Sexo años MiembrosFamilia
## Antonio        H   23               2
## Ricardo        H   45               3
## JuanGabriel    H   20               4
## Maria          M   30               2
## Margarita      M   18               5
## 6              H   30               1
```

- `d.f$new_var`: para añadir una nueva variable al data frame
  - Podemos concatenar columnas con un data frame existente mediante la función `cbind()`. De este modo se puede añadir la columna directamente sin necesidad de convertirla antes a data frame
  - Esta nueva variable ha de tener la misma longitud que el resto de columnas del data frame original. Si no, se añadirán valores NA a las variables del data frame original o a la nueva variable hasta completar la misma longitud

```
df3$Sexo=as.character(df3$Sexo)
## Podemos añadir una nueva columna
df3$Ingresos = c(10000, 12000, 15000, 20000, 21000,11000)
df3
```

```
##             Sexo años MiembrosFamilia Ingresos
## Antonio        H   23               2    10000
## Ricardo        H   45               3    12000
## JuanGabriel    H   20               4    15000
## Maria          M   30               2    20000
## Margarita      M   18               5    21000
## 6              H   30               1    11000
```

```
as.character(df3$años)
```

```
## [1] "23" "45" "20" "30" "18" "30"
```

```
as.numeric(df3$años)
```

```
## [1] 23 45 20 30 18 30
```

```
df4
```

```
##             Sexo años MiembrosFamilia
## Antonio        H   23               2
## Ricardo        H   45               3
## JuanGabriel    H   20               4
## Maria          M   30               2
## Margarita      M   18               5
```

```
df4[df4$Sexo=="M",] -> df_m
df_m
```

```
##           Sexo años MiembrosFamilia
## Maria        M   30               2
## Margarita    M   18               5
```

Cambiando los tipos de datos

- `as.character`: para transformar todos los datos de un objeto en palabras
- `as.integer`: para transformar todos los datos de un objeto a números enteros
- `as.numeric`: para transformar todos los datos de un objeto a números reales

Sub-data frames

- `droplevels(d.f)`: para borrar los niveles sobrantes de todos los factores, ya que las columnas que son factores heredan en los sub-data frames todos los niveles del factor original, aunque no aparezcan en el trozo que hemos extraído
- `select(d.f, parámetros)`: para especificar que queremos extraer de un data frame
  - `starts_with("x")`: extrae del data frame las variables cuyo nombre empieza con la palabra "x"
  - `ends_with("x")`: extrae del data frame las variables cuyo nombre termina con la palabra "x"
  - `contains("x")`: extrae del data frame las variables cuyo nombre contiene la palabra "x"
  - Se necesita el paquete `dplyr` o mejor aún `tidyverse`

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
iris_petal=select(iris, starts_with("Petal"))## columnas que comiencen en petal
head(iris_petal)
```

```
##   Petal.Length Petal.Width
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
## 4          1.5         0.2
## 5          1.4         0.2
## 6          1.7         0.4
```

```
iris_length=select(iris, ends_with("Length"))
head(iris_length)
```

```
##   Sepal.Length Petal.Length
## 1          5.1          1.4
## 2          4.9          1.4
## 3          4.7          1.3
## 4          4.6          1.5
## 5          5.0          1.4
## 6          5.4          1.7
```

- `subset(df,condición,select = columnas)`: para extraer del data frame las filas que cumplen la condición y las columnas especificadas
  - Si queremos todas las filas, no hay que especificar ninguna condición
  - Si queremos todas las columnas, no hace especificar el parámetro `select`
  - Las variables en la condición se especifican con su nombre, sin añadir antes el nombre del data frame

```
subset(iris, Species=="setosa") -> setosa
head(setosa)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
subset(iris, Species=="versicolor", select=c(1,3)) -> versicolor
head(versicolor)
```

```
##    Sepal.Length Petal.Length
## 51          7.0          4.7
## 52          6.4          4.5
## 53          6.9          4.9
## 54          5.5          4.0
## 55          6.5          4.6
## 56          5.7          4.5
```

```
## Para renombrar los indices
rownames(versicolor)=1:nrow(versicolor)
head(versicolor)
```

```
##   Sepal.Length Petal.Length
## 1          7.0          4.7
## 2          6.4          4.5
## 3          6.9          4.9
## 4          5.5          4.0
## 5          6.5          4.6
## 6          5.7          4.5
```

```
str(versicolor)
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ Sepal.Length: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
##  $ Petal.Length: num  4.7 4.5 4.9 4 4.6 4.5 4.7 3.3 4.6 3.9 ...
```

Samply, Aggregate, y Attach/Detach a DataFrames

Aplicando funciones a data frames

- `sapply(d.f, función)`: para aplicar una función a todas las columnas de un data frame en un solo paso
  - `na.rm=TRUE`: para evitar que el valor que devuelva la función para las columnas que contengan algún NA sea NA
- `aggregate(variables~factors,data=d.f,FUN=función)`: para aplicar una función a variables de un data frame clasificadas por los niveles de un, o más de un, factor
  - Si queremos aplicar la función a más de una variable, tenemos que agruparlas con un `cbind`
  - Si queremos separar las variables mediante más de un factor, tenemos que agruparlos con signos +

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
sapply(subset(iris, select = 1:4), mean) ## aplicamos la función promedio para las columnas 1--4
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

```
sapply(iris[, 1:4], sum )
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##        876.5        458.6        563.7        179.9
```

```
f=function(x){sqrt(sum(x^2))}
sapply(iris, f)
```

```
## Warning in Ops.factor(x, 2): '^' not meaningful for factors
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##     72.27621     37.82063     50.82037     17.38764           NA
```

```r
df=data.frame(C1=c(1,2, NA, 4), C2=c(5,NA,2,3))
# Si algun data tiene NA, los eliminamos
sapply(df, mean, na.rm=TRUE)
```

```
##       C1       C2
## 2.333333 3.333333
```

```r
aggregate(cbind(Sepal.Length, Petal.Length)~ Species, data=iris, FUN = mean, na.rm=TRUE)
```

```
##      Species Sepal.Length Petal.Length
## 1     setosa        5.006        1.462
## 2 versicolor        5.936        4.260
## 3  virginica        6.588        5.552
```

data frame de automoviles default en r, mtcars

```r
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```r
# Convetir una columma de numeros a factores
mtcars$cyl=as.factor(mtcars$cyl)
mtcars$gear=as.factor(mtcars$gear)
mtcars$carb=as.factor(mtcars$carb)
str(mtcars) ###Ver como cambia
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
```

```
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: Factor w/ 3 levels "3","4","5": 2 2 2 1 1 1 1 2 2 2 ...
##  $ carb: Factor w/ 6 levels "1","2","3","4",..: 4 4 1 1 2 1 4 2 2 4 ...
```

```
aggregate(mpg~cyl+gear+carb, data=mtcars, FUN=mean, na.rm=TRUE)
```

```
##    cyl gear carb   mpg
## 1    4    3    1 21.50
## 2    6    3    1 19.75
## 3    4    4    1 29.10
## 4    8    3    2 17.15
## 5    4    4    2 24.75
## 6    4    5    2 28.20
## 7    8    3    3 16.30
## 8    8    3    4 12.62
## 9    6    4    4 19.75
## 10   8    5    4 15.80
## 11   6    5    6 19.70
## 12   8    5    8 15.00
```

variables globales

- `attach(d.f)`: para hacer que R entienda sus variables como globales y que las podamos usar por su nombre, sin necesidad de añadir delante el nombre del data frame y el símbolo $
  - Si ya hubiera existido una variable definida con el mismo nombre que una variable del data frame al que aplicamos `attach`, hubiéramos obtenido un mensaje de error al ejecutar esta función y no se hubiera reescrito la variable global original
- `detach(d.f)`: para devolver la situación original, eliminando del entorno global las variables del data frame

Estadistica Descriptiva Con Datos Cualitativos

Análisis estadístico de los datos

Cuando tenemos una serie de datos que describen algunos aspectos de un conjunto de individuos queremos llevar a cabo un análisis estadístico. Estos análisis estadísticos se clasifican en:

- Análisis exploratorio, o descriptivo, si nuestro objetivo es resumir, representar y explicar los datos concretos de los que disponemos. La estadística descriptiva es el conjunto de técnicas que se usan con este fin.
- Análisis inferencial, si nuestro objetivo es deducir (inferir), a partir de estos datos, información significativa sobre el total de la población o las poblaciones de interés. Las técnicas que se usan en este caso forman la estadística inferencial.

Existe relación entre ambos. Cualquier análisis inferencial se suele empezar explorando los datos que se usarán así cómo también muchas técnicas descriptivas permiten estimar propiedades de la población de la que se ha extraído la muestra.

Ejemplo

La media aritmética de las alturas de una muestra de individuos nos da un valor representativo de esta muestra, pero también estima la media de las alturas del total de la población

###Estudio de los datos Cualitativos #### Tipos de datos

Trabajamos con datos multidimensionales: observamos varias características de una serie de individuos.

Se registran en un archivo de ordenador con un formato preestablecido. Por ejemplo texto simple (codificado en diferentes formatos: ASCII, isolatin...), hojas de cálculo (archivos de Open Office o Excel), bases de datos, etc.

Una de las maneras básicas de almacenar datos es en forma de tablas de datos. En R hacemos uso de data frames.

En una tabla de datos cada columna expresa una variable, mientras que cada fila corresponde a las observaciones de estas variables para un individuo concreto.

- Los datos de una misma columna tienen que ser del mismo tipo, porque corresponden a observaciones de una misma propiedad.
- Las filas en principio son de naturaleza heterogénea, porque pueden contener datos de diferentes tipos.

Los tipos de datos que consideramos son los siguientes:

- Datos de tipo atributo, o cualitativos: Expresan una cualidad del individuo. En R guardaremos las listas de datos cualitativos en vectores (habitualmente, de palabras), o en factores si vamos a usarlos para clasificar individuos.
- Datos ordinales: Similares a los cualitativos, con la única diferencia de que se pueden ordenar de manera natural. Por ejemplo, las calificaciones en un control (suspenso, aprobado, notable, sobresaliente). En R guardaremos las listas de datos ordinales en factores ordenados.
- Datos cuantitativos: Se refieren a medidas, tales como edades, longitudes, etc. En R guardaremos las listas de datos cuantitativos en vectores numéricos.

¿Qué son los datos cualitativos? Los datos cualitativos corresponden a observaciones sobre cualidades de un objeto o individuo.

Suelen codificarse por medio de palabras, pero también se pueden usar números que jueguen el papel de etiquetas.

Ejemplo

Es habitual representar No (o Falso, Fracaso, Ausente...) con un 0, y Sí (o Verdadero, Éxito, Presente...) con un 1

Los datos cualitativos son aquellos que pueden ser iguales o diferentes, pero que no admiten ningún otro tipo de comparación significativa.

Es decir, que no tenga ningún sentido preguntarse si uno es más grande que otro, ni efectuar operaciones aritméticas con ellos, aunque estén representados por números.

Por lo tanto, un mismo conjunto de datos puede ser cualitativo o de otro tipo, según el análisis que vayamos a hacer de él.

Ejemplo

Si hemos anotado durante unos años los días de la semana en los que ha llovido y queremos contar cuántas veces ha ocurrido en lunes, cuántas en martes, etc., esta lista de nombres (o números) serán datos cualitativos. Si, en cambio, queremos estudiar cómo se comportan los días de lluvia según avanza la semana, y por lo tanto el orden de los días es relevante, serán datos ordinales

Variable cualitativa: lista de observaciones de un tipo de datos cualitativos sobre un conjunto concreto de objetos.

Niveles: diferentes valores que pueden tomar estos datos. Por ejemplo, los dos niveles de una variable Sexo serían `M` (Macho) y `H` (Hembra), o sinónimos.

Con R, usaremos vectores y factores para representar variables cualitativas. Los factores nos servirán para agrupar las observaciones según los niveles de la variable. De esta manera podremos segmentar la población que representa la variable en grupos o subpoblaciones, asignando un grupo a cada nivel, y podremos comparar el comportamiento de otras variables sobre estos grupos.

Frecuencias absolutas y relativas

Dada una variable cualitativa, para cada uno de sus niveles podemos contar cuántos datos hay en ese nivel (frecuencia absoluta) y qué fracción del total representan (frecuencia relativa).

Ejemplo

Supongamos que tenemos un tipo de datos cualitativos con niveles

$$l_1, l_2, \cdots, l_k$$

Efectuamos n observaciones de este tipo de datos, y denotamos por

$$x_1, x_2, \cdots, x_n$$

los resultados que obtenemos con

$$x_j \in \{l_1, l_2, \cdots, l_k\}$$

Estas observaciones forman una variable cualitativa

Con estas notaciones:

La frecuencia absoluta, n_j, del nivel l_j en esta variable cualitativa es el número de observaciones en las que x_i toma el valor l_j.

La frecuencia relativa del nivel l_j en esta variable cualitativa es la fracción

$$f_j = \frac{n_j}{n}$$

Es decir, la frecuencia relativa del nivel l_j es la fracción (en tanto por uno) de observaciones que corresponden a este nivel.

La moda de esta variable cualitativa es su nivel, o niveles, de mayor frecuencia (absoluta o relativa).

Ejemplo

Supongamos que se ha realizado un seguimiento a 20 personas asistentes a un congreso. Uno de los datos que se han recogido sobre estas personas ha sido su sexo. El resultado ha sido una variable cualitativa formada por las 20 observaciones siguientes:

`Mujer, Mujer, Hombre, Mujer, Mujer, Mujer, Mujer, Mujer, Hombre, Mujer, Hombre, Hombre, Mujer, Mujer, Hombre, Mujer, Mujer, Mujer, Mujer, Hombre`

Sus dos niveles son `Hombre` y `Mujer`. En esta variable hay 14 mujeres y 6 hombres. Éstas son las frecuencias absolutas de estos niveles.

Puesto que en total hay 20 individuos, sus frecuencias relativas son

$$\text{Hombre} = \frac{6}{20} = 0.3, \qquad \text{Mujer} = \frac{14}{20} = 0.7$$

En este caso $l_1 =$ Hombre y $l_2 =$ Mujer, $n = 20$ (el número de observaciones efectuadas), y $x_1, \cdots, x_{20}$ formarían la muestra de sexos

Ejemplo

La tabla siguiente resume las frecuencias absolutas y relativas de la variable cualitativa del ejemplo anterior, con las notaciones que acabamos de introducir.

Sexo	n_i	f_i	%
Hombre	6	0.3	30%
Mujer	14	0.7	70%
Total	20	1	100%

Su moda es el nivel `Mujer`

Tablas de frecuencias unidimensionales

Supongamos que tenemos una variable cualitativa guardada en un vector o un factor como la siguiente:

```
x = sample(1:5, size = 12, replace = TRUE)
x
```

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

```
Respuestas=factor(sample(c("Si", "No"), size = 12, replace = TRUE))
Respuestas
```

```
##  [1] No Si No No No No Si Si No Si Si Si
## Levels: No Si
```

Con R, la tabla de frecuencias absolutas de un vector que representa una variable cualitativa se calcula con la función `table()`.

```
table(x)
```

```
## x
## 1 2 3 4 5
## 1 4 3 1 3
```

```
table(Respuestas)
```

```
## Respuestas
## No Si
##  6  6
```

Al aplicar `table()` a un vector obtenemos una tabla unidimensional formada por una fila con los niveles de la variable y una segunda fila donde, debajo de cada nivel, aparece su frecuencia absoluta en el vector.

Los nombres de las columnas de una tabla unidimensional se obtienen con la función `names()`.

```
names(table(x))
```

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

```
names(table(Respuestas))
```

```
## [1] "No" "Si"
```

En la `table` de un vector sólo aparecen los nombres de los niveles presentes en el vector. Si el tipo de datos cualitativos usado tenía más niveles y queremos que aparezcan explícitamente en la tabla (con frecuencia 0), hay que transformar el vector en un factor con los niveles deseados.

```
z=factor(x, levels=1:7) #Los niveles serán 1,2,3,4,5,6,7
z
```

```
##  [1] 3 3 4 2 3 1 5 2 5 2 2 5
## Levels: 1 2 3 4 5 6 7
```

```
table(z)
```

```
## z
## 1 2 3 4 5 6 7
## 1 4 3 1 3 0 0
```

Podemos pensar que una tabla unidimensional es como un vector de números donde cada entrada está identificada por un nombre: el de su columna. Para referirnos a una entrada de una tabla unidimensional, podemos usar tanto su posición como su nombre (entre comillas, aunque sea un número).

```
table(x)[3] #La tercera columna de table(x)
```

```
## 3
## 3
```

```
table(x)["7"] #¿La columna de table(x) con nombre 7?
```

```
## <NA>
##   NA
```

```
table(x)["5"] #La columna de table(x) con nombre 5
```

```
## 5
## 3
```

```
3*table(x)[2] #El triple de la segunda columna de table(x)
```

```
##  2
## 12
```

Las tablas de contingencia aceptan la mayoría de las funciones que ya hemos utilizado para vectores.

```r
sum(table(x)) #Suma de las entradas de table(x)
```

```
## [1] 12
```

```r
sqrt(table(Respuestas)) #Raíces cuadradas de las entradas de table(Respuestas)
```

```
## Respuestas
##      No      Si
## 2.44949 2.44949
```

La tabla de frecuencias relativas de un vector se puede calcular aplicando la función `prop.table()` a su `table`. El resultado vuelve a ser una tabla de contingencia unidimensional.

```r
prop.table(table(x))
```

```
## x
##          1          2          3          4          5
## 0.08333333 0.33333333 0.25000000 0.08333333 0.25000000
```

```r
prop.table(table(Respuestas))
```

```
## Respuestas
##  No  Si
## 0.5 0.5
```

¡CUIDADO! La función `prop.table()` se tiene que aplicar al resultado de `table`, no al vector original. Si aplicamos `prop.table()` a un vector de palabras o a un factor, dará un error, pero si la aplicamos a un vector de números, nos dará una tabla.

Esta tabla no es la tabla de frecuencias relativas de la variable cualitativa representada por el vector, sino la tabla de frecuencias relativas de una variable que tuviera como tabla de frecuencias absolutas este vector de números, entendiendo que cada entrada del vector representa la frecuencia de un nivel diferente.

```r
prop.table(x)
```

```
##  [1] 0.08108108 0.08108108 0.10810811 0.05405405 0.08108108 0.02702703
##  [7] 0.13513514 0.05405405 0.13513514 0.05405405 0.05405405 0.13513514
```

```r
X=c(1,1,1)
prop.table(table(X))
```

```
## X
## 1
## 1
```

```r
prop.table(X)
```

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

También podemos calcular la tabla de frecuencias relativas de un vector dividiendo el resultado de `table` por el número de observaciones.

```
table(x)/length(x)
```

```
## x
##          1          2          3          4          5
## 0.08333333 0.33333333 0.25000000 0.08333333 0.25000000
```

Dados un vector x y un número natural n, la instrucción

`names(which(table(x)==n))`

nos da los niveles que tienen frecuencia absoluta n en x.

```
table(x)
```

```
## x
## 1 2 3 4 5
## 1 4 3 1 3
```

```
names(which(table(x)==1))
```

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

En particular, por lo tanto,

`names(which(table(x)==max(table(x))))`

nos da los niveles de frecuencia máxima en x: su moda.

```
names(which(table(x)==max(table(x))))
```

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

```
names(which(table(Respuestas)==max(table(Respuestas))))
```

```
## [1] "No" "Si"
```

```
datos=factor(c("H","M","M","M","H","H","M","M"))
table(datos)
```

```
## datos
## H M
## 3 5
```

```
table(datos)["M"]
```

```
## M
## 5
```

```r
sum(table(datos))
```

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

Frecuencias Relativas

```r
prop.table(table(datos))
```

```
## datos
##     H     M
## 0.375 0.625
```

```r
100*prop.table(table(datos))
```

```
## datos
##    H    M
## 37.5 62.5
```

tambien lo podemos hacer a mano y obtenemos los mismos resultados

$$f_i = \frac{n_i}{n}$$

```r
table(datos)/length(datos)
```

```
## datos
##     H     M
## 0.375 0.625
```

```r
names(table(datos))
```

```
## [1] "H" "M"
```

```r
names(which(table(datos)==3))
```

```
## [1] "H"
```

```r
moda <- function(d){
  names(which(table(d)==max(table(d))))
}
m_t =moda(datos)
m_t
```

```
## [1] "M"
```

la moda del data frame es: M‘

Tablas de frecuencias bidimensionales

La función `table()` también permite construir tablas de frecuencias conjuntas de dos o más variables.

Supongamos que el vector `Respuestas` anterior contiene las respuestas a una pregunta dadas por unos individuos cuyos sexos tenemos almacenados en un vector `Sexo`, en el mismo orden que sus respuestas. En este caso, podemos construir una tabla que nos diga cuántas personas de cada sexo han dado cada respuesta.

```
Sexo= sample(c("H", "M"), size = length(Respuestas), replace = T) #H = hombre, M = mujer
table(Respuestas ,Sexo)
```

```
##           Sexo
## Respuestas H M
##         No 4 2
##         Si 2 4
```

Para referirnos a una entrada de una tabla bidimensional podemos usar el sufijo `[ , ]` como si estuviéramos en una matriz o un data frame. Dentro de los corchetes, tanto podemos usar los índices como los nombres (entre comillas) de los niveles.

```
table(Respuestas ,Sexo)[1,2]
```

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

```
table(Respuestas ,Sexo)["No","M"]
```

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

Como en el caso unidimensional, la función `prop.table()` sirve para calcular tablas bidimensionales de frecuencias relativas conjuntas de pares de variables. Pero en el caso bidimensional tenemos dos tipos de frecuencias relativas:

Frecuencias relativas globales: para cada par de niveles, uno de cada variable, la fracción de individuos que pertenecen a ambos niveles respecto del total de la muestra.

Frecuencias relativas marginales: dentro de cada nivel de una variable y para cada nivel de la otra, la fracción de individuos que pertenecen al segundo nivel respecto del total de la subpoblación definida por el primer nivel.

Dadas dos variables, se pueden calcular dos familias de frecuencias relativas marginales, según cuál sea la variable que defina las subpoblaciones en las que calculemos las frecuencias relativas de los niveles de la otra variable; no es lo mismo la fracción de mujeres que han contestado que sí respecto del total de mujeres, que la fracción de mujeres que han contestado que sí respecto del total de personas que han dado esta misma respuesta.

La tabla de frecuencias relativas globales se calcula aplicando sin más la función `prop.table()` a la `table`.

```
prop.table(table(Sexo,Respuestas)) #frec. Relativa Global
```

```
##     Respuestas
## Sexo        No        Si
##    H 0.3333333 0.1666667
##    M 0.1666667 0.3333333
```

De este modo, la tabla `prop.table(table(Sexo,Respuestas))` nos da la fracción del total que representa cada pareja (sexo, respuesta).

Para obtener las marginales, debemos usar el parámetro `margin` al aplicar la función `prop.table()` a la `table`. Con `margin=1` obtenemos las frecuencias relativas de las filas y con `margin=2`, de las columnas.

```
prop.table(table(Sexo,Respuestas), margin=1) #Por sexo
```

```
##     Respuestas
## Sexo        No        Si
##    H 0.6666667 0.3333333
##    M 0.3333333 0.6666667
```

```
prop.table(table(Sexo,Respuestas), margin=2) #Por respuesta
```

```
##     Respuestas
## Sexo        No        Si
##    H 0.6666667 0.3333333
##    M 0.3333333 0.6666667
```

La función `CrossTable()` del paquete `gmodels` permite producir (especificando el parámetro `prop.chisq=FALSE`) un resumen de la tabla de frecuencias absolutas y las tres tablas de frecuencias relativas de dos variables en un formato adecuado para su visualización.

La leyenda *Cell Contents* explica los contenidos de cada celda de la tabla: la frecuencia absoluta, la frecuencia relativa por filas, la frecuencia relativa por columnas, y la frecuencia relativa global. Esta función dispone de muchos parámetros que permiten modificar el contenido de las celdas, y que puedes consultar en `help(CrossTable)`.

install.packages("gmodels", dep=TRUE) library(gmodels)

```
sex=factor(c("H","M","M","M","H","H","M","M"))
ans=factor(c("S","N","S","S","S","N","N","S"))
#CrossTable(sex, ans, prop.chisq = FALSE)
```

Una tabla de contingencia bidimensional es, básicamente, una matriz con algunos atributos extra. En particular, podemos usar sobre estas tablas la mayoría de las funciones para matrices que tengan sentido para tablas:

- `rowSums()` y `colSums()` se pueden aplicar a una tabla y suman sus filas y sus columnas, respectivamente.
- También podemos usar sobre una tabla bidimensional (o, en general, multidimensional) la función `apply()` con la misma sintaxis que para matrices.

```
table(sex,ans )
```

```
##    ans
## sex N S
##   H 1 2
##   M 2 3
```

```r
colSums(table(sex,ans))
```

```
## N S
## 3 5
```

```r
rowSums(table(sex,ans))
```

```
## H M
## 3 5
```

```r
colSums(prop.table(table(sex,ans)))
```

```
##     N     S
## 0.375 0.625
```

```r
rowSums(prop.table(table(sex,ans)))
```

```
##     H     M
## 0.375 0.625
```

```r
tt <- table(sex, ans)
tt # Frec absolutas
```

```
##    ans
## sex N S
##   H 1 2
##   M 2 3
```

```r
prop.table(tt)## Frecu Rel Global
```

```
##    ans
## sex     N     S
##   H 0.125 0.250
##   M 0.250 0.375
```

```r
prop.table(tt,margin = 1) #Frec rel por sexo
```

```
##    ans
## sex         N         S
##   H 0.3333333 0.6666667
##   M 0.4000000 0.6000000
```

```r
prop.table(tt,margin=2) #Frec Rel por respuesta
```

```
##    ans
## sex         N         S
##   H 0.3333333 0.4000000
##   M 0.6666667 0.6000000
```

```r
colSums(tt)
```

```
## N S
## 3 5
```

```r
rowSums(tt)
```

```
## H M
## 3 5
```

```r
colSums(prop.table(tt))## ya esta arriba
```

```
##     N     S
## 0.375 0.625
```

```r
rowSums(prop.table(tt))## ""
```

```
##     H     M
## 0.375 0.625
```

```r
apply(tt, FUN=sum, MARGIN = 1)
```

```
## H M
## 3 5
```

Multivariante

```r
ans=sample(c("si","No"), size = 100, replace = TRUE)
sex=sample(c("H","M"), size = 100, replace = TRUE)
place=sample(c("San Francisco", "Barcelona", "Valencia", "Cobija", "Asturias"), size=100, replace = TRU
table(sex,ans,place)
```

ejemplo de tres dimensiones

```
## , , place = Asturias
##
##    ans
## sex No si
##   H  6  5
##   M  4  5
##
## , , place = Barcelona
##
##    ans
## sex No si
##   H  6  8
##   M  3  6
```

```
##
## , , place = Cobija
##
##    ans
## sex No si
##   H  7  6
##   M  2  3
##
## , , place = San Francisco
##
##    ans
## sex No si
##   H  7  6
##   M  3  3
##
## , , place = Valencia
##
##    ans
## sex No si
##   H  5  4
##   M  7  4
```

```
ftable(sex,ans,place)
```

```
##         place Asturias Barcelona Cobija San Francisco Valencia
## sex ans
## H   No              6         6      7             7        5
##     si              5         8      6             6        4
## M   No              4         3      2             3        7
##     si              5         6      3             3        4
```

```
ftable(sex,ans,place, col.vars = c("sex","ans"))
```

```
##               sex  H     M
##               ans No si No si
## place
## Asturias           6  5  4  5
## Barcelona          6  8  3  6
## Cobija             7  6  2  3
## San Francisco      7  6  3  3
## Valencia           5  4  7  4
```

####filtrar las tablas

```
table(sex, ans, place)["M","si", "San Francisco"]
```

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

```
table(sex, ans, place)[ ,"si", "Valencia"]
```

```
## H M
## 4 4
```

```r
table(sex, ans, place)[ ,"No", ]
```

```
##    place
## sex Asturias Barcelona Cobija San Francisco Valencia
##   H        6         6      7             7        5
##   M        4         3      2             3        7
```

Frecuencias relativas

```r
prop.table(table(sex,ans, place)) #Frec Rel Globales
```

```
## , , place = Asturias
##
##    ans
## sex   No   si
##   H 0.06 0.05
##   M 0.04 0.05
##
## , , place = Barcelona
##
##    ans
## sex   No   si
##   H 0.06 0.08
##   M 0.03 0.06
##
## , , place = Cobija
##
##    ans
## sex   No   si
##   H 0.07 0.06
##   M 0.02 0.03
##
## , , place = San Francisco
##
##    ans
## sex   No   si
##   H 0.07 0.06
##   M 0.03 0.03
##
## , , place = Valencia
##
##    ans
## sex   No   si
##   H 0.05 0.04
##   M 0.07 0.04
```

```r
prop.table(table(sex,ans, place), margin = 3) #Frec Rel Marginal por pais
```

```
## , , place = Asturias
##
```

```
##    ans
## sex        No        si
##   H 0.3000000 0.2500000
##   M 0.2000000 0.2500000
##
## , , place = Barcelona
##
##    ans
## sex        No        si
##   H 0.2608696 0.3478261
##   M 0.1304348 0.2608696
##
## , , place = Cobija
##
##    ans
## sex        No        si
##   H 0.3888889 0.3333333
##   M 0.1111111 0.1666667
##
## , , place = San Francisco
##
##    ans
## sex        No        si
##   H 0.3684211 0.3157895
##   M 0.1578947 0.1578947
##
## , , place = Valencia
##
##    ans
## sex        No        si
##   H 0.2500000 0.2000000
##   M 0.3500000 0.2000000
```

```
prop.table(table(sex,ans, place), margin = c(1,3)) #Frec Rel Marginal por sexo y pais
```

```
## , , place = Asturias
##
##    ans
## sex        No        si
##   H 0.5454545 0.4545455
##   M 0.4444444 0.5555556
##
## , , place = Barcelona
##
##    ans
## sex        No        si
##   H 0.4285714 0.5714286
##   M 0.3333333 0.6666667
##
## , , place = Cobija
##
##    ans
## sex        No        si
##   H 0.5384615 0.4615385
```

```
##   M 0.4000000 0.6000000
##
## , , place = San Francisco
##
##    ans
## sex        No        si
##   H 0.5384615 0.4615385
##   M 0.5000000 0.5000000
##
## , , place = Valencia
##
##    ans
## sex        No        si
##   H 0.5555556 0.4444444
##   M 0.6363636 0.3636364
```

```
ftable(prop.table(table(sex,ans, place)))
```

```
##         place Asturias Barcelona Cobija San Francisco Valencia
## sex ans
## H   No            0.06      0.06   0.07          0.07     0.05
##     si            0.05      0.08   0.06          0.06     0.04
## M   No            0.04      0.03   0.02          0.03     0.07
##     si            0.05      0.06   0.03          0.03     0.04
```

Analizando una tabla de frecuencias que contiene R

```
HairEyeColor
```

```
## , , Sex = Male
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    32   11    10     3
##   Brown    53   50    25    15
##   Red      10   10     7     7
##   Blond     3   30     5     8
##
## , , Sex = Female
##
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    36    9     5     2
##   Brown    66   34    29    14
##   Red      16    7     7     7
##   Blond     4   64     5     8
```

```
sum(HairEyeColor) -> total
```

El total de individuos de la tabla es `t total`

```r
prop.table(HairEyeColor, margin = 3)
```

```
## , , Sex = Male
##
##        Eye
## Hair          Brown        Blue       Hazel       Green
##   Black 0.114695341 0.039426523 0.035842294 0.010752688
##   Brown 0.189964158 0.179211470 0.089605735 0.053763441
##   Red   0.035842294 0.035842294 0.025089606 0.025089606
##   Blond 0.010752688 0.107526882 0.017921147 0.028673835
##
## , , Sex = Female
##
##        Eye
## Hair          Brown        Blue       Hazel       Green
##   Black 0.115015974 0.028753994 0.015974441 0.006389776
##   Brown 0.210862620 0.108626198 0.092651757 0.044728435
##   Red   0.051118211 0.022364217 0.022364217 0.022364217
##   Blond 0.012779553 0.204472843 0.015974441 0.025559105
```

```r
prop.table(HairEyeColor, margin = c(1,2))
```

```
## , , Sex = Male
##
##        Eye
## Hair        Brown      Blue     Hazel     Green
##   Black 0.4705882 0.5500000 0.6666667 0.6000000
##   Brown 0.4453782 0.5952381 0.4629630 0.5172414
##   Red   0.3846154 0.5882353 0.5000000 0.5000000
##   Blond 0.4285714 0.3191489 0.5000000 0.5000000
##
## , , Sex = Female
##
##        Eye
## Hair        Brown      Blue     Hazel     Green
##   Black 0.5294118 0.4500000 0.3333333 0.4000000
##   Brown 0.5546218 0.4047619 0.5370370 0.4827586
##   Red   0.6153846 0.4117647 0.5000000 0.5000000
##   Blond 0.5714286 0.6808511 0.5000000 0.5000000
```

se puede cambiar el formato de las tablas ó varables de la siguiente manera

```r
aperm(HairEyeColor, perm=c("Sex","Hair","Eye"))
```

```
## , , Eye = Brown
##
##         Hair
## Sex      Black Brown Red Blond
##   Male      32    53  10     3
##   Female    36    66  16     4
##
```

```
## , , Eye = Blue
##
##         Hair
## Sex      Black Brown Red Blond
##   Male      11    50  10    30
##   Female     9    34   7    64
##
## , , Eye = Hazel
##
##         Hair
## Sex      Black Brown Red Blond
##   Male      10    25   7     5
##   Female     5    29   7     5
##
## , , Eye = Green
##
##         Hair
## Sex      Black Brown Red Blond
##   Male       3    15   7     8
##   Female     2    14   7     8
```

Para darle formato a las tablas se usa la libreria kableExtra, la mandamos a kable(datos)

Tablas a partir de data frames de variables cualitativas

Como ya hemos comentado en varias ocasiones, la manera natural de organizar datos multidimensionales en `R` es en forma de data frame.

En esta sección explicaremos algunas instrucciones para calcular tablas de frecuencias absolutas a partir de un data frame de variables cualitativas.

Para ilustrarla, usaremos el fichero que se encuentra en el la carpeta de datos:

`"data/EnergyDrink"`

Este fichero consiste en una tabla de datos con la siguiente información sobre 122 estudiantes de una Universidad de España: su sexo (variable `sexo`), el estudio en el que están matriculados (variable `estudio`) y si consumen habitualmente bebidas energéticas para estudiar (variable `bebe`).

```
Beb_Energ=read.table("r-basic-master/data/EnergyDrink",header=TRUE)
str(Beb_Energ)
```

```
## 'data.frame':    122 obs. of  3 variables:
##  $ estudio: chr  "Informatica" "Mates" "Industriales" "Informatica" ...
##  $ bebe   : chr  "No" "No" "Si" "Si" ...
##  $ sexo   : chr  "Mujer" "Hombre" "Mujer" "Hombre" ...
```

```
head(Beb_Energ, 4)
```

```
##        estudio bebe   sexo
## 1  Informatica   No  Mujer
## 2        Mates   No Hombre
## 3 Industriales   Si  Mujer
## 4  Informatica   Si Hombre
```

Aplicando la función `summary()` a un data frame de variables cualitativas, obtenemos, a modo de resumen, una tabla con las frecuencias absolutas de cada variable.

```
summary(Beb_Energ)
```

```
##    estudio              bebe               sexo
##  Length:122         Length:122         Length:122
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
```

```
table(Beb_Energ)
```

```
## , , sexo = Hombre
##
##               bebe
## estudio        No Si
##   Industriales 19  6
##   Informatica  30  7
##   Mates         8  1
##   Telematica   10  2
##
## , , sexo = Mujer
##
##               bebe
## estudio        No Si
##   Industriales 10  2
##   Informatica  11  5
##   Mates         6  1
##   Telematica    3  1
```

```
table(Beb_Energ[c(1,3)])
```

```
##               sexo
## estudio        Hombre Mujer
##   Industriales     25    12
##   Informatica      37    16
##   Mates             9     7
##   Telematica       12     4
```

```
ftable(Beb_Energ)
```

```
##                    sexo Hombre Mujer
## estudio      bebe
## Industriales No            19    10
##              Si             6     2
## Informatica  No            30    11
##              Si             7     5
## Mates        No             8     6
##              Si             1     1
## Telematica   No            10     3
##              Si             2     1
```

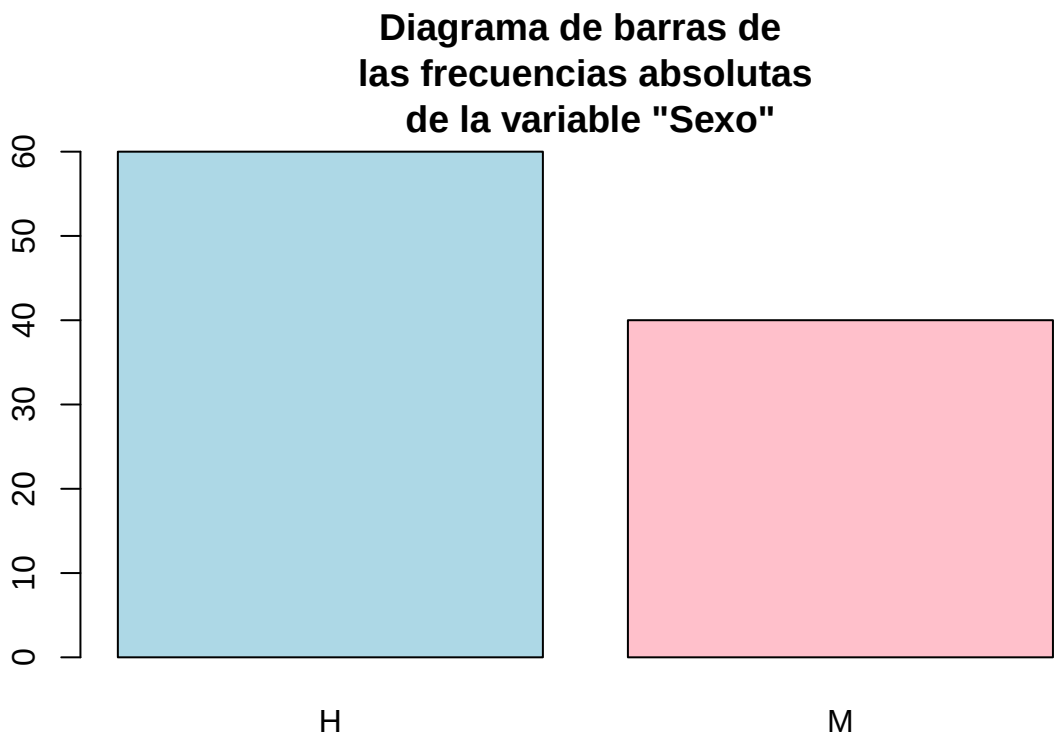
Diagrama de barras

El tipo de gráfico más usado para representar variables cualitativas son los diagramas de barras (`bar plots`). Como su nombre indica, un diagrama de barras contiene, para cada nivel de la variable cualitativa, una barra de altura su frecuencia.

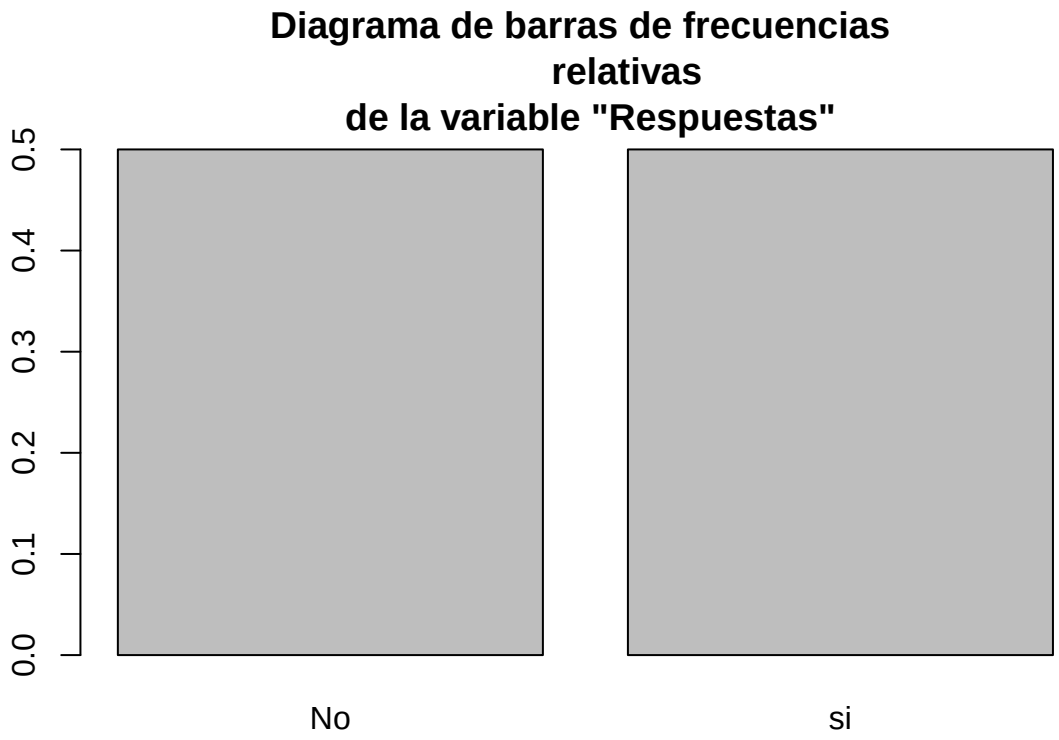
La manera más sencilla de dibujar un diagrama de barras de las frecuencias absolutas o relativas de una variable cualitativa es usando la instrucción `barplot()` aplicada a la tabla correspondiente.

¡Atención! Como pasaba con `prop.table()`, el argumento de `barplot` ha de ser una tabla, y, por consiguiente, se ha de aplicar al resultado de `table()` o de `prop.table()`, nunca al vector de datos original.

```
barplot(table(sex), col=c("lightblue","pink"), main="Diagrama de barras de
las frecuencias absolutas\n de la variable \"Sexo\"")
```




```
barplot(prop.table(table(ans)), main="Diagrama de barras de frecuencias
        relativas\n de la variable \"Respuestas\"")
```



Observado que en las funciones `barplot()` anteriores hemos usado el parámetro `main` para poner título a los diagramas; en general, la función `barplot()` admite los parámetros de `plot` que tienen sentido en el contexto de los diagramas de barras: `xlab`, `ylab`, `main`, etc. Los parámetros disponibles se pueden consultar en `help(barplot)`. Aquí sólo vamos a comentar algunos.

```
par(mfrow=c(1,2))
barplot(table(Respuestas), col=c("green"))
barplot(table(Respuestas), col=c("red","blue"))
```

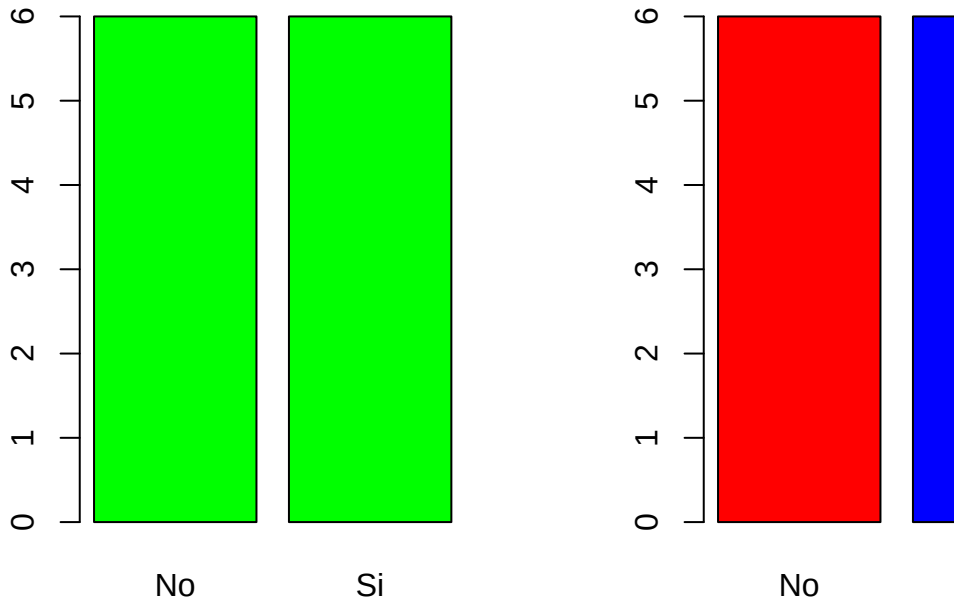
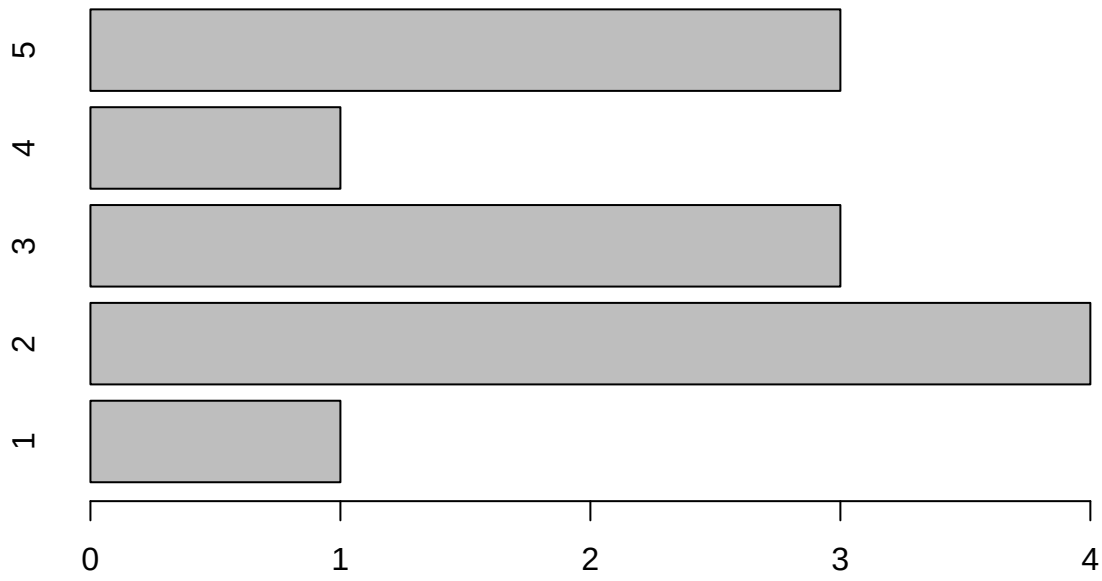


Diagrama de barras - Colores

```
par(mfrow=c(1,1))
```

```
barplot(table(x), horiz=TRUE)
```




```
barplot(table(Respuestas,Sexo ), legend.text = TRUE)
```

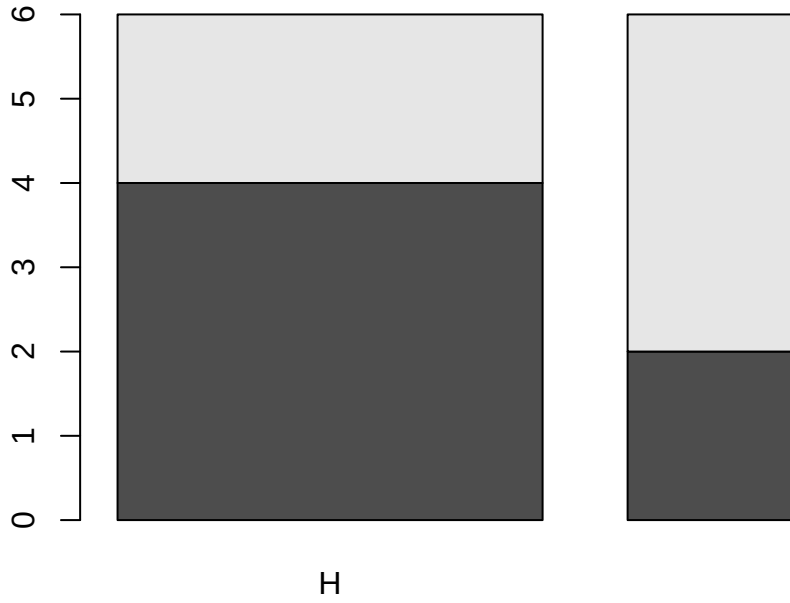
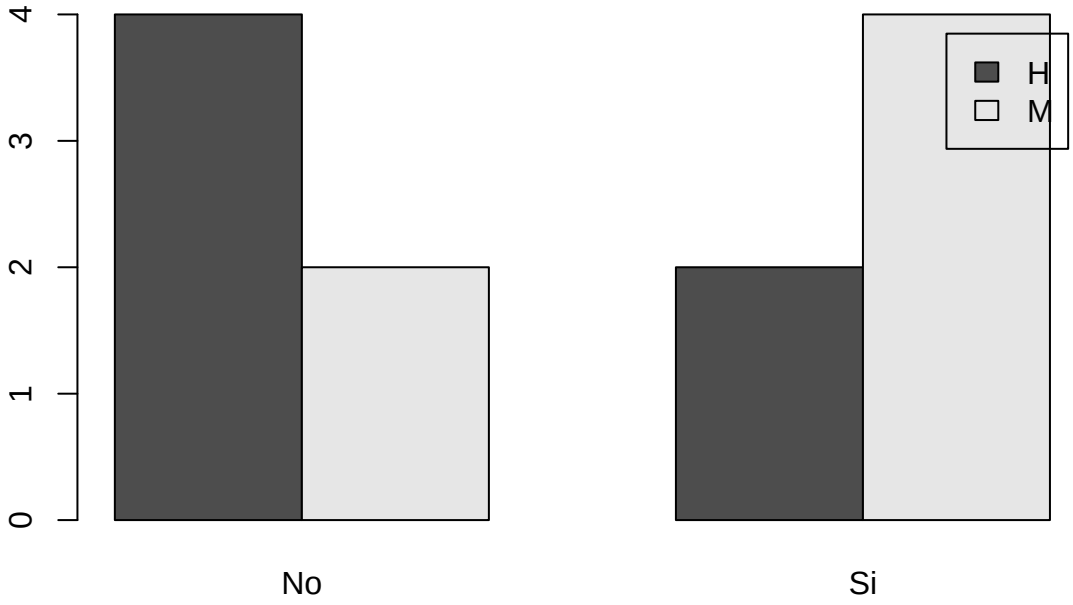


Diagrama de barras - Tabla bidimensional

```
barplot(table(Sexo,Respuestas), beside=TRUE, legend.text=TRUE)
```




```
barplot(table(Respuestas,Sexo), beside=TRUE, names=c("Men", "Women"),
        col=c("yellow","lightblue"), legend.text=c("No","Yes"))
```

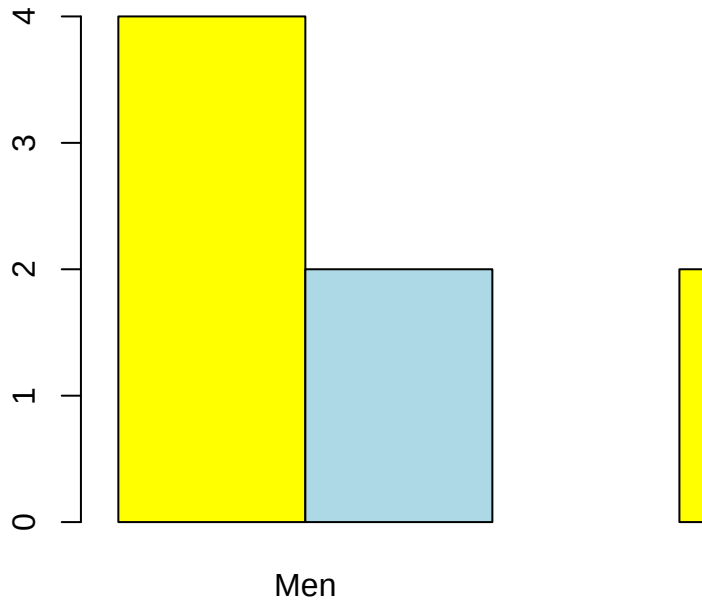


Diagrama de barras - Parámetros de las leyendas

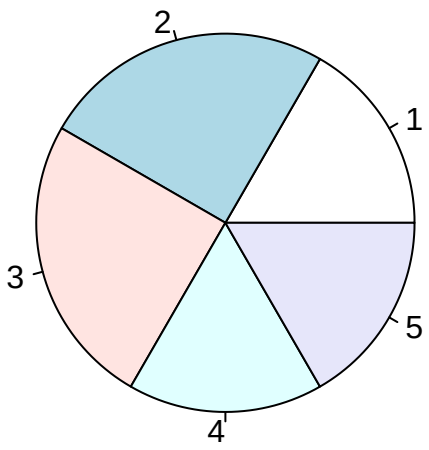
Diagrama circular

Un tipo muy popular de representación gráfica de variables cualitativas son los diagramas circulares. En un diagrama circular (`pie chart`) se representan los niveles de una variable cualitativa como sectores circulares de un círculo, de manera que el ángulo (o equivalentemente, el área) de cada sector sea proporcional a la frecuencia del nivel al que corresponde.

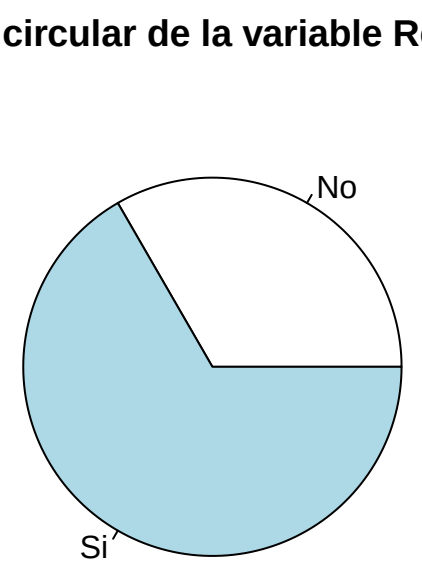
Con `R`, este tipo de diagramas se producen con la instrucción `pie`, de nuevo aplicada a una tabla de frecuencias y no al vector original.

La función `pie` admite muchos parámetros para modificar el resultado: se pueden cambiar los colores con `col`, se pueden cambiar los nombres de los niveles con `names`, se puede poner un título con `main`, etc.; podemos consultar la lista completa de parámetros en `help(pie)`.

```
x = c(4,2,3,5,1,4,3,1,5,2,3,2)
pie(table(x), main="Diagrama circular de la variable x")
```




```
Respuestas1=c("No","Si","Si","Si","Si","Si","No","No","Si","Si","No","Si")
pie(table(Respuestas1), main="Diagrama circular de la variable Respuestas")
```



Pese a su popularidad, es poco recomendable usar diagramas circulares porque a veces es difícil, a simple vista, comprender las relaciones entre las frecuencias que representan.

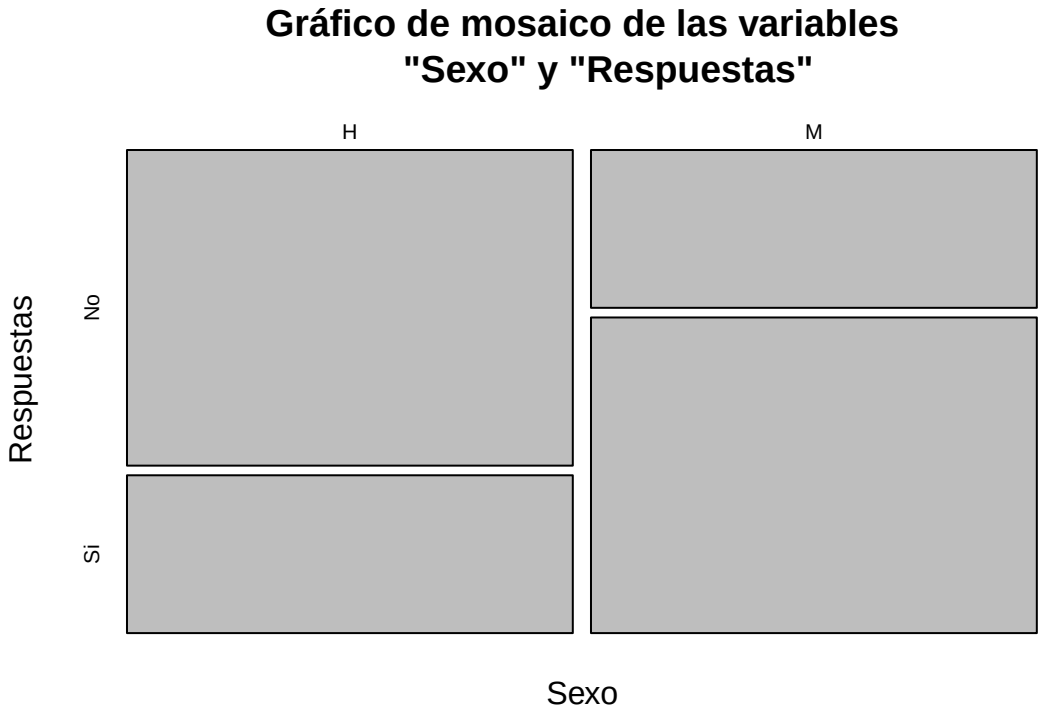
Gráficos de mosaico

Otra representación de las tablas multidimensionales de frecuencias son los gráficos de mosaico. Estos gráficos se obtienen sustituyendo cada entrada de la tabla de frecuencias por una región rectangular de área proporcional a su valor.

En concreto, para obtener el gráfico de mosaico de una tabla bidimensional, se parte de un cuadrado de lado 1, primero se divide en barras verticales de amplitudes iguales a las frecuencias relativas de una variable, y luego cada barra se divide, a lo alto, en regiones de alturas proporcionales a las frecuencias relativas marginales de cada nivel de la otra variable, dentro del nivel correspondiente de la primera variable.

Un gráfico de mosaico de una tabla se obtiene con `R` aplicando la función `plot` a la tabla, o también la función `mosaicplot`. Esta última también se puede aplicar a matrices.

```
plot(table(Sexo,Respuestas), main="Gráfico de mosaico de las variables
     \"Sexo\" y \"Respuestas\"")
```



En el gráfico de mosaico de una tabla tridimensional, primero se divide el cuadrado en barras verticales de amplitudes iguales a las frecuencias relativas de una variable.

Luego cada barra se divide, a lo alto, en regiones de alturas proporcionales a las frecuencias relativas marginales de cada nivel de una segunda variable, dentro del nivel correspondiente de la primera variable.

Finalmente, cada sector rectangular se vuelve a dividir a lo ancho en regiones de amplitudes proporcionales a las frecuencias relativas marginales de cada nivel de la tercera variable dentro de la combinación correspondiente de niveles de las otras dos.

```
plot(HairEyeColor, main="Gráfico de mosaico de la tabla HairEyeColor",
     col=c("pink","lightblue"))
```

Además de sus parámetros usuales, la función `plot` admite algunos parámetros específicos cuando se usa para producir el gráfico de mosaico de una tabla. Estos parámetros se pueden consultar en `help(mosaicplot)`.

Los paquetes `vcd` y `vcdExtra` incluyen otras funciones que producen representaciones gráficas interesantes de tablas tridimensionales.

- La función `cotabplot` de `vcd` produce un diagrama de mosaico para cada nivel de la tercera variable.
- La función `mosaic3d` de `vcdExtra` produce un diagrama de mosaico tridimensional en una ventana de una aplicación para gráficos 3D interactivos.

El objeto de datos `HairEyeColor` que lleva predefinido `R` es una tabla de frecuencias absolutas de tres variables cualitativas: color de cabello (`Hair`), color de los ojos (`Eye`) y sexo (`Sex`).

Vamos a extraer de esta tabla una tabla bidimensional de frecuencias absolutas de las variables `Eye` y `Hair`, sin distinguir según el sexo. La manera más sencilla de obtener esta tabla es sumando las subtablas de frecuencias para hombres y mujeres, y aplicando `as.table()` al resultado para transformarlo en una `table` por si no lo es.

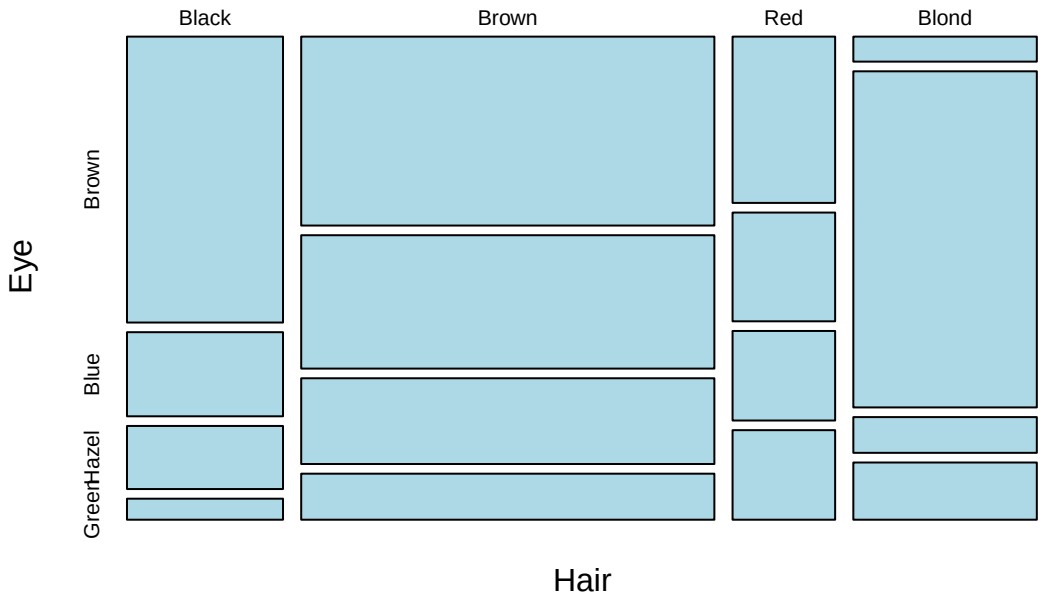
```
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    68   20    15     5
##   Brown   119   84    54    29
##   Red      26   17    14    14
##   Blond     7   94    10    16
```

```r
male <- HairEyeColor[, ,"Male"]
female <- HairEyeColor[, ,"Female"]
data <- as.table(male+female)
data
```

```
##        Eye
## Hair    Brown Blue Hazel Green
##   Black    68   20    15     5
##   Brown   119   84    54    29
##   Red      26   17    14    14
##   Blond     7   94    10    16
```

```r
plot(data, col=c("lightblue" ), main="Diagrama de moisaico de la tabla bidimensional de frecuencias de
```

a de moisaico de la tabla bidimensional de frecuencias de colores de c




```r
sum(data)
```

```
## [1] 592
```

```r
colSums(data)
```

```
## Brown  Blue Hazel Green
##   220   215    93    64
```

```
rowSums(data)
```

```
## Black Brown   Red Blond
##   108   286    71   127
```

```
round(prop.table(colSums(data)),3)
```

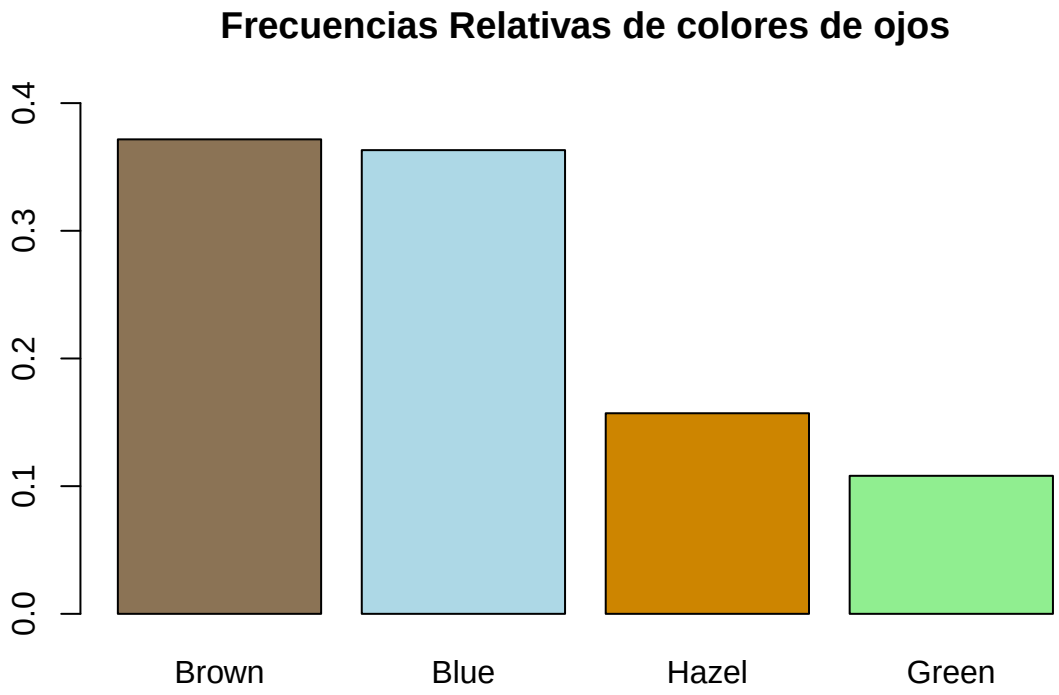
```
## Brown  Blue Hazel Green
## 0.372 0.363 0.157 0.108
```

```
round(prop.table(rowSums(data)),3)
```

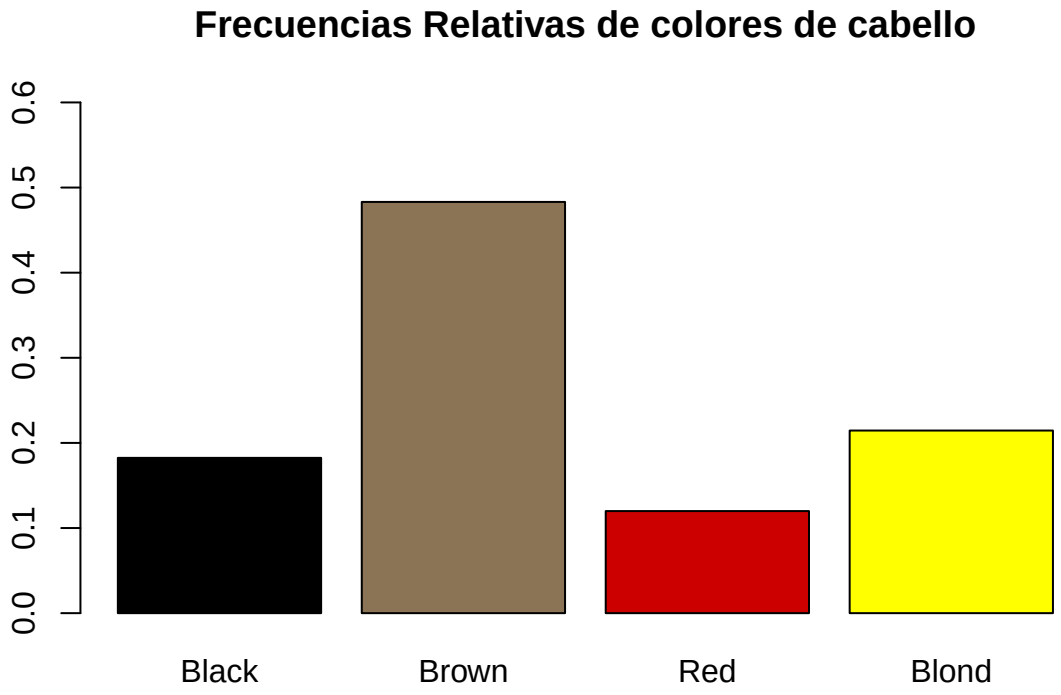
```
## Black Brown   Red Blond
## 0.182 0.483 0.120 0.215
```

representaremos los siguientes datos en diagrama de barras

```
barplot(prop.table(colSums(data)), ylim=c(0, 0.4), main="Frecuencias Relativas de colores de ojos",
        col=c("burlywood4", "lightblue", "orange3","lightgreen"))
```




```
barplot(prop.table(rowSums(data)), ylim=c(0, 0.6), main="Frecuencias Relativas de colores de cabello",
        col=c("black", "burlywood4", "red3","yellow"))
```



Frecuencias relativas globales y Marginales

```
round(prop.table(data),3) ## Frecuencias relatvas
```

```
##        Eye
## Hair    Brown  Blue Hazel Green
##   Black 0.115 0.034 0.025 0.008
##   Brown 0.201 0.142 0.091 0.049
##   Red   0.044 0.029 0.024 0.024
##   Blond 0.012 0.159 0.017 0.027
```

```
round(prop.table(data, margin=1),3) ## Marginal Por filas y se lee  de las personas de pelo negro el 63
```

```
##        Eye
## Hair    Brown  Blue Hazel Green
##   Black 0.630 0.185 0.139 0.046
##   Brown 0.416 0.294 0.189 0.101
##   Red   0.366 0.239 0.197 0.197
##   Blond 0.055 0.740 0.079 0.126
```

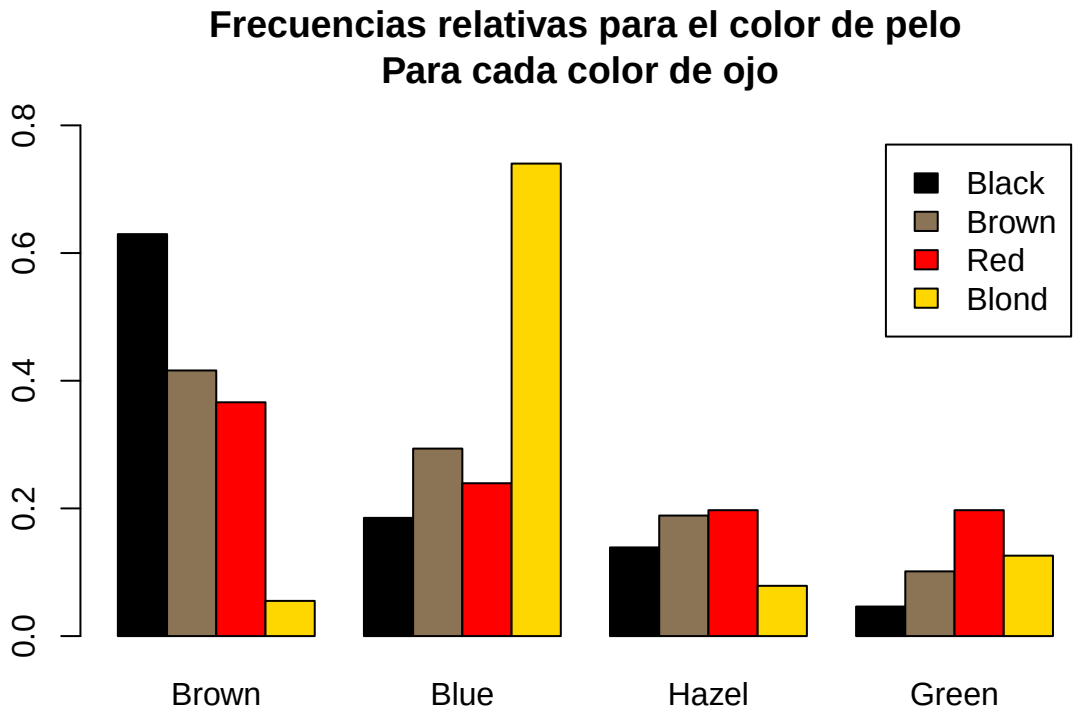
```
round(prop.table(data, margin=2),3) ## Marginal por columnas y se lee De las personas que tienen los oj
```

```
##        Eye
## Hair    Brown  Blue Hazel Green
```

```
##    Black 0.309 0.093 0.161 0.078
##    Brown 0.541 0.391 0.581 0.453
##    Red   0.118 0.079 0.151 0.219
##    Blond 0.032 0.437 0.108 0.250
```

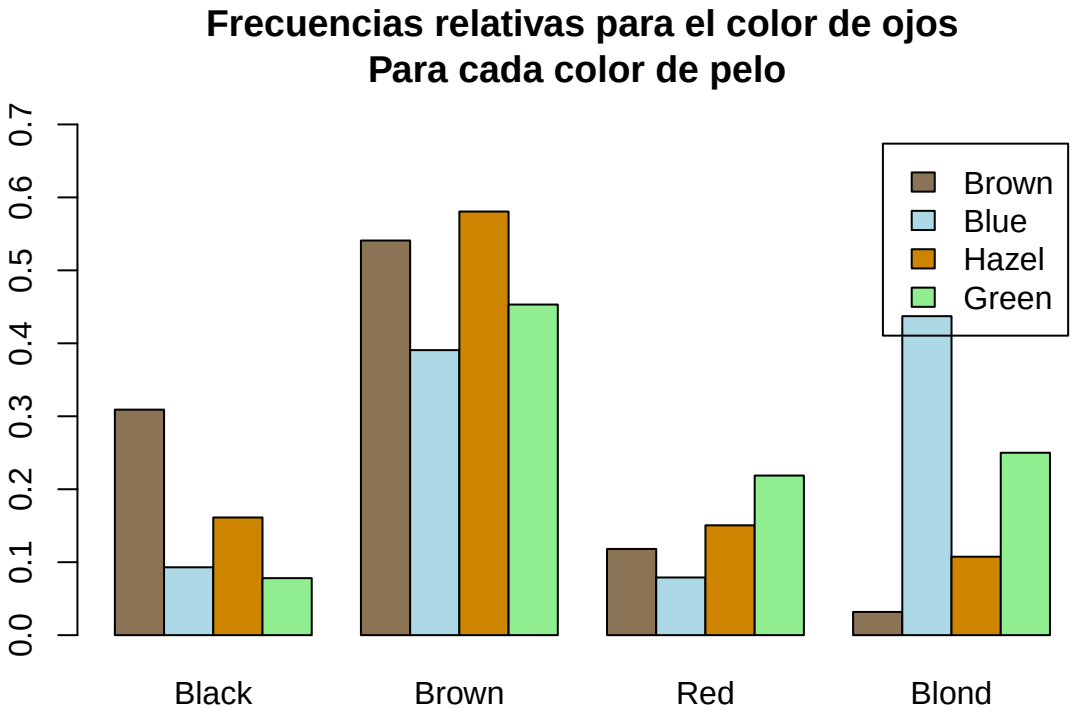
Grafico de las frecuencias marginales

```
barplot(prop.table(data, margin=1), beside=TRUE, legend.text = TRUE, ylim = c(0, 0.8), col=c("black", "
```




```
## Beside es para no tener las columnas apiladas si no en bloques (separadas)
```

```
barplot(t(prop.table(data, margin=2)), beside=TRUE, legend.text = TRUE, ylim = c(0, 0.7), col=c("burlyw
```




```
## Beside es para no tener las columnas apiladas si no en bloques (separadas)
```

Estadistica Descriptiva Con Datos Ordinales

Datos ordinales

Los datos ordinales son parecidos a los cualitativos, en el sentido de que son cualidades de los individuos u objetos.

La diferencia existente entre los datos cualitativos y los ordinales reside en las características que expresan. En el caso de los ordinales, éstas tienen un orden natural que permite "acumular" observaciones.

Frecuencias para datos ordinales (acumulada)

Al trabajar con datos ordinales, el orden de los niveles de los datos nos permite calcular no solo frecuencias absolutas y relativas, sino también frecuencias acumuladas.

Es decir, podemos contar cuantas veces hemos observado un dato menor o igual a este.

Ejemplo 1 Ejemplo 1

Suponga que tenemos una muestra de 15 estudiantes de los cuales sabemos su nota en el examen de Estadística. Clasificamos todos estos resultados en Suspenso (S), Aprobado (A), Notable (N) y Excelente (Ex) y consideramos su orden natural $S < A < N < Ex$.

Las notas obtenidas han sido las siguientes

$$S,\ A,\ N,\ Ex,\ S,\ S,\ Ex,\ Ex,\ N,\ A,\ A,\ A,\ A,\ N,\ S$$

Recuerde, que para saber cuantas hay de cada una (su frecuencia absoluta), utilizamos la función `table()`

```
notas = ordered(c("S","A", "N", "Ex", "S", "S", "Ex", "Ex", "N", "A", "A", "A",
                  "A", "N", "S"), levels = c("S", "A", "N", "Ex"))
table(notas)
```

```
## notas
##  S  A  N Ex
##  4  5  3  3
```

vea que hay 4 S, 5 A, 3 N y 3 Ex.

En lo referente a **frecuencias absolutas acumuladas**, hay

- 4 estudiantes con S o menos. Ello implica que la frecuencia acumulada de S es 4
- 9 estudiantes que han obtenido A o menos. Entonces, la frecuencia acumulada de A es 9
- 12 estudiantes los cuales han obtenido N o menos. Así, la frecuencia acumulada de N es 12
- 15 estudiantes (todos) que han obtenido Ex o menos. De este modo, la frecuencia acumulada de Ex es 15, o sea, el total.

Frecuencia relativa acumulada. Es la fracción del total de las observaciones en tanto por 1 que representa su frecuencia absoluta acumulada

Así, las recuencias relativas acumuladas respectivas son

- $S:\ \frac{4}{15} \approx 0.27$
- $A:\ \frac{9}{15} \approx 0.6$
- $N:\ \frac{12}{15} \approx 0.8$
- $Ex:\ \frac{15}{15} = 1$

En general, supongamos que realizamos n observaciones

$$x_1, \ldots, x_n$$

de un cierto tipo de datos ordinales, cuyos posibles niveles ordenados son

$$l_1 < l_2 < \cdots < l_k$$

Por tanto, cada una de las observaciones x_j es igual a algún l_i. Diremos que todas estas observaciones forman una variable ordinal. En nuestro ejemplo anterior, los 4 niveles eran

$$S < A < N < Ex$$

Además, nuestro $n = 15$ y $x_1, \ldots, x_{15}$ son las calificaciones obtenidas por los alumnos.

De este modo, con estas notaciones

- Las definiciones de frecuencias absolutas n_j y las relativas f_j, para cada nivel l_j son las mismas que en una variable cualitativa.

- Las frecuencia absoluta acumulada del nivel l_j en esta variable ordinal es el número N_j de observaciones x_i tales que $x_i \leq l_j$. Es decir,

$$N_j = \sum_{i=1}^{j} n_i$$

- La frecuencia relativa acumulada del nivel l_j en esta variable ordinal es la fracción en tanto por 1 F_j de observaciones x_i tales que $x_i \leq l_j$. Es decir,

$$F_j = \frac{N_j}{n} = \sum_{i=1}^{j} f_i$$

Ejemplo 2 Ejemplo 2

En un estudio, a un grupo de clientes de un restaurante se les hizo la siguiente pregunta:

"¿Estás contento con el trato ofrecido por los trabajadores del establecimiento?"

Las posibles respuestas forman una escala ordinal con $1 < 2 < 3 < 4 < 5$.

Supongamos que se recogieron las siguientes respuestas de 50 técnicos:

```
set.seed(2018) ## Para que salgan los mismos resultados cada vez que se ejecute
clientes = sample(1:5, 50, replace = TRUE)
clientes
```

```
##  [1] 3 4 5 2 5 1 3 4 2 4 3 3 1 1 5 3 1 3 3 5 1 4 2 5 3 4 5 1 2 2 1 5 5 2 1 2 5 5
## [39] 2 1 2 1 3 2 1 2 3 3 1 2
```

```
# set.seed(NULL) para que vuelva adquirir valores
```

En este caso tenemos 5 niveles ($k = 5$) y 50 observaciones ($n = 50$) que forman una variable ordinal a la que hemos llamado `clientes`.

Hemos calculado todas sus frecuencias (absoluta, relativa, acumulada y relativa acumulada) y las hemos representado en la siguiente tabla.

```
##   Absoluta Relativa Acumulada Rel. Acumulada
## 1       12     0.24        12           0.24
## 2       12     0.24        24           0.48
## 3       11     0.22        35           0.70
## 4        5     0.10        40           0.80
## 5       10     0.20        50           1.00
```

Frecuencia relativa acumulada

Los gráficos para frecuencias absolutas y relativas absolutas de variables ordinales son exactamente los mismos que para las variables cualitativas.

También podemos utilizar diagramas de barras para describir frecuencias acumuladas: en este caso, la altura de cada barra debe ser igual a la frecuencia acumulada del nivel respectivo. Además, estos niveles deben de aparecer ordenados de manera ascendente, de forma que las alturas de las barras también tengan un orden ascendente.

No obstante, se recomienda no hacer uso de diagramas circulares a la hora de representar frecuencias acumuladas, debido a que éstos no representan la información sobre la acumulación de datos de forma fácil de entender a simple vista.

Función cumsum() Recordemos la función `cumsum()` esta puede ser utilizada a la hora de calcular frecuencias acumuladas.

Retomemos el ejemplo anterior de las notas de los estudiantes y calculemos y representemos en un diagrama de barras las frecuencias acumuladas de la muestra de notas.

```
notas
```

```
##  [1] S  A  N  Ex S  S  Ex Ex N  A  A  A  A  N  S
## Levels: S < A < N < Ex
```

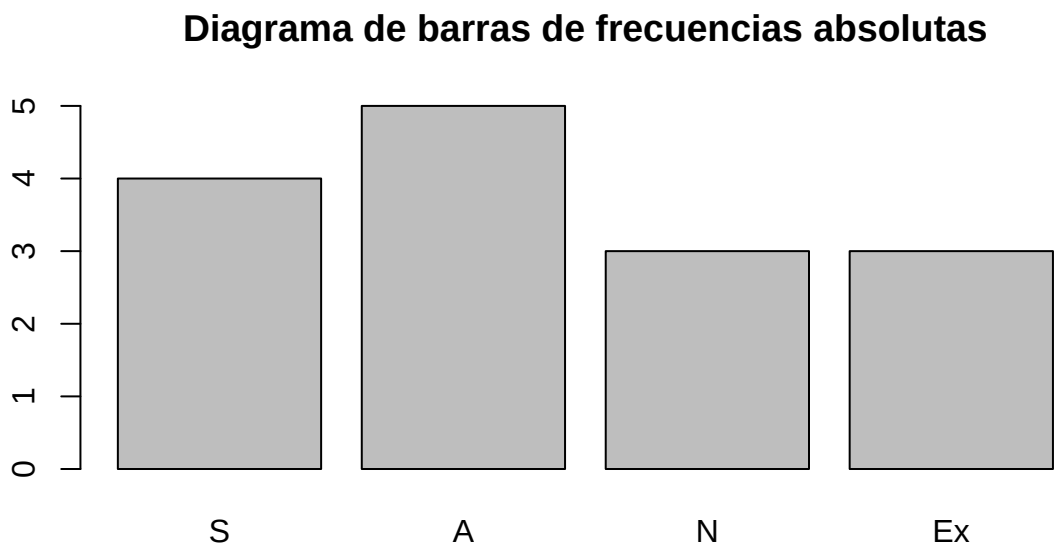
```
fAbs = table(notas) #Frec. abs.
cumsum(fAbs) #Frec. abs. acumuladas
```

```
##  S  A  N Ex
##  4  9 12 15
```

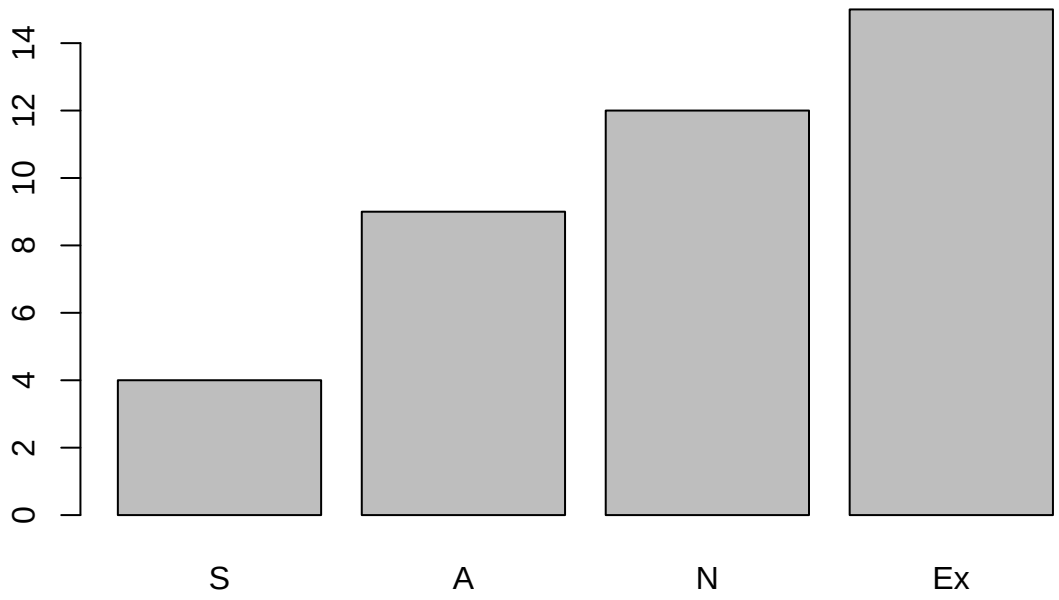
```
cumsum(prop.table(fAbs)) #Frec. relativas acumuladas
```

```
##         S         A         N        Ex
## 0.2666667 0.6000000 0.8000000 1.0000000
```

```
barplot(fAbs, main = "Diagrama de barras de frecuencias absolutas")
```




```
barplot(cumsum(fAbs), main = "Diagrama de barras de frecuencias absolutas acumuladas")
```



Podríamos haber calculado las frecuencias relativas acumuladas de la forma

```
cumsum(table(notas))/length(notas)
```

```
##         S         A         N        Ex
## 0.2666667 0.6000000 0.8000000 1.0000000
```

```
cumsum(table(notas)/length(notas))
```

```
##         S         A         N        Ex
## 0.2666667 0.6000000 0.8000000 1.0000000
```

Pero no podemos hacer `prop.table(cumsum(table(notas)))`.

Ejemplo 3 Ejemplo 3

Se ha evaluado el tamaño de los cuellos de 100 jirafas. Los niveles que se han utilizado se los considera ordenados de la siguiente manera:

$$\text{Muy.corto} < \text{Corto} < \text{Normal} < \text{Largo} < \text{Muy.largo}$$

Los valores obtenidos en dicho estudio han sido los siguientes

```
longitud
```

```
##   [1] Normal    Largo     Muy.largo Corto     Muy.largo Muy.corto Normal
##   [8] Largo     Corto     Largo     Normal    Normal    Muy.corto Muy.corto
##  [15] Muy.largo Normal    Muy.corto Normal    Normal    Muy.largo Muy.corto
##  [22] Largo     Corto     Muy.largo Normal    Largo     Muy.largo Muy.corto
##  [29] Corto     Corto     Muy.corto Muy.largo Muy.largo Corto     Muy.corto
##  [36] Corto     Muy.largo Muy.largo Corto     Muy.corto Corto     Muy.corto
##  [43] Normal    Corto     Muy.corto Corto     Normal    Normal    Muy.corto
##  [50] Corto     Normal    Muy.corto Largo     Largo     Corto     Muy.corto
##  [57] Corto     Normal    Normal    Normal    Normal    Muy.corto Normal
##  [64] Muy.corto Corto     Largo     Muy.corto Corto     Muy.corto Muy.largo
##  [71] Muy.corto Corto     Muy.largo Largo     Muy.largo Normal    Corto
##  [78] Corto     Normal    Largo     Largo     Corto     Corto     Muy.largo
##  [85] Largo     Largo     Normal    Normal    Muy.corto Normal    Corto
##  [92] Normal    Muy.corto Corto     Muy.corto Normal    Corto     Corto
##  [99] Muy.corto Corto
## Levels: Muy.corto < Corto < Normal < Largo < Muy.largo
```

Estudiemos sus frecuencias

```
Fr.Abs = table(longitud)
Fr.Abs
```

```
## longitud
## Muy.corto     Corto    Normal     Largo Muy.largo
##        23        26        24        13        14
```

```
Fr.Rel = prop.table(Fr.Abs)
Fr.Rel
```

```
## longitud
## Muy.corto     Corto    Normal     Largo Muy.largo
##      0.23      0.26      0.24      0.13      0.14
```

```
Fr.Acum = cumsum(Fr.Abs)
Fr.Acum
```

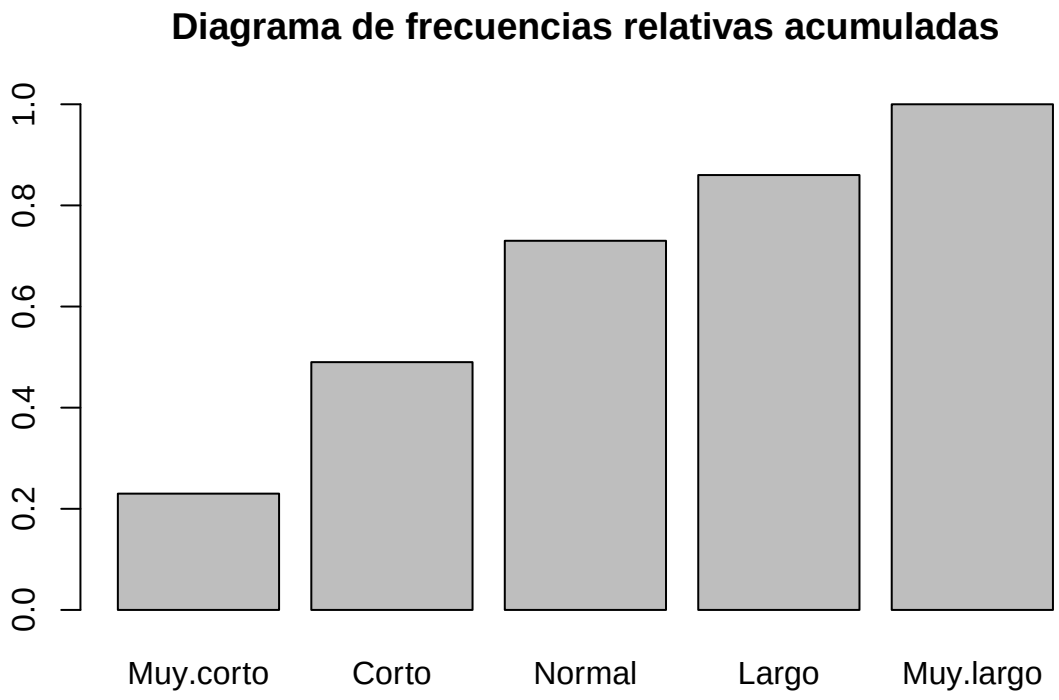
```
## Muy.corto     Corto    Normal     Largo Muy.largo
##        23        49        73        86       100
```

```
Fr.RAcum = cumsum(Fr.Rel)
Fr.RAcum
```

```
## Muy.corto     Corto    Normal     Largo Muy.largo
##      0.23      0.49      0.73      0.86      1.00
```

La instrucción `barplot` produce el siguiente diagrama de barras de frecuencias relativas acumuladas

```
barplot(Fr.RAcum, main = "Diagrama de frecuencias relativas acumuladas")
```



Para calcular frecuencias acumuladas en una tabla multidimensional, hay que aplicar a la tabla la función `cumsum` mediante la función `apply` que ya explicábamos para matrices. En este caso en concreto, la sintaxis de la instrucción sería

```
apply(tabla, MARGIN=..., FUN=cumsum)
```

donde el valor `MARGIN` ha de ser el de la dimensión en la que queremos acumular las frecuencias: 1 si queremos hacerlo por filas, 2 para hacerlo por columnas, etc. Lo veremos todo más claro con un ejemplo

Ejemplo 4 Ejemplo 4

Supongamos que en el ejemplo anterior, el de las jirafas, estas provienen de 4 zonas diferentes, A,B,C y D, de manera que las 30 primeras son de la zona A, las 25 siguientes de la B, las 35 siguientes de la C y las 10 últimas de la D. Nos interesa estudiar la distribución de las longitudes según la zona.

Vamos a organizar todos estos datos en un data frame llamado `jirafas`. Para que nos sea más fácil visualizar la información, es conveniente que las filas de las tablas de frecuencias correspondan a las zonas. Por lo tanto, al definir el data frame, entraremos como primera variable la de la muestra las zonas. Así, conseguiremos que éstas aparezcan en las filas al aplicarle la función table.

```
zonas = rep(c("A","B","C","D"), c(30,25,35,10)) #se crean  A 30 veces, B 25 veces asi sucesivamente
jirafas = data.frame(zonas,longitud) #se crea un data Frame
str(jirafas)
```

```
## 'data.frame':    100 obs. of  2 variables:
```

```
##  $ zonas   : chr  "A" "A" "A" "A" ...
##  $ longitud: Ord.factor w/ 5 levels "Muy.corto"<"Corto"<..: 3 4 5 2 5 1 3 4 2 4 ...
```

```
head(jirafas)
```

```
##   zonas  longitud
## 1     A    Normal
## 2     A     Largo
## 3     A Muy.largo
## 4     A     Corto
## 5     A Muy.largo
## 6     A Muy.corto
```

Para calcular la tabla de frecuencias absolutas acumuladas de las longitudes por zonas y como las zonas definen las filas de la tabla anterior, debemos utilizar la función `apply` con `MARGIN = 1`.

```
apply(table(jirafas), MARGIN = 1, FUN = cumsum)
```

```
##            zonas
## longitud     A  B  C  D
##   Muy.corto  6  7  7  3
##   Corto     11 15 15  8
##   Normal    19 19 25 10
##   Largo     24 21 31 10
##   Muy.largo 30 25 35 10
```

Observe que la tabla se ha traspuesto. Resulta que cuando se aplica `apply` a una `table` bidimensional, `R` intercambia, en caso de ser necesario, filas por columnas en el resultado para que la dimensión de la tabla resultante en la que se haya aplicado la función sea la de las columnas.

Con lo cual, para volver a tener las zonas en las filas, hay que trasponer el resultado de la función `apply`.

```
t(apply(table(jirafas), MARGIN = 1, FUN = cumsum))
```

```
##      longitud
## zonas Muy.corto Corto Normal Largo Muy.largo
##     A         6    11     19    24        30
##     B         7    15     19    21        25
##     C         7    15     25    31        35
##     D         3     8     10    10        10
```

Vamos ahora a calcular la tabla de frecuencias relativas acumuladas de las longitudes de cuello por zonas. Para conseguirlo, y en una única instrucción, primero calculamos la tabla de frecuencias relativas por filas, a continuación, con las funciones `apply` y `cumsum` las acumulamos y, finalmente, trasponemos el resultado.

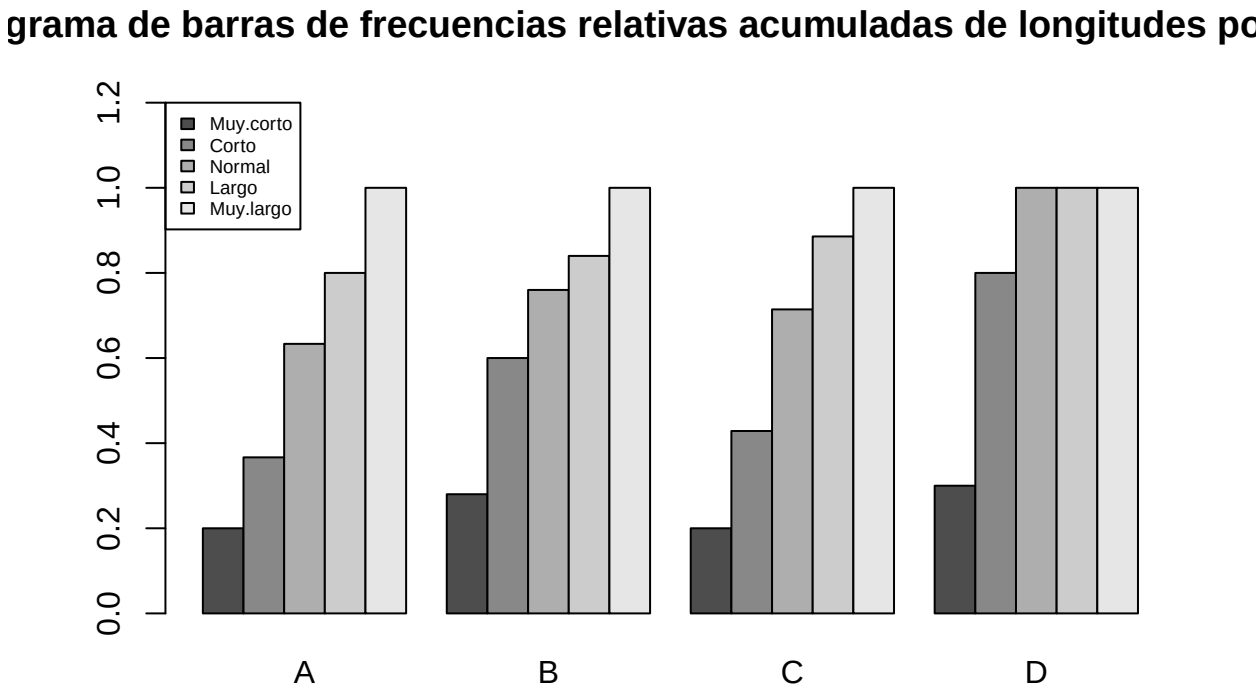
```
t(apply(prop.table(table(jirafas), margin = 1), MARGIN = 1, FUN = cumsum))
```

```
##      longitud
## zonas Muy.corto     Corto    Normal     Largo Muy.largo
##     A      0.20 0.3666667 0.6333333 0.8000000         1
##     B      0.28 0.6000000 0.7600000 0.8400000         1
##     C      0.20 0.4285714 0.7142857 0.8857143         1
##     D      0.30 0.8000000 1.0000000 1.0000000         1
```

Vamos ahora a dibujar el diagrama de barras por bloques de esta tabla. Nos interesa que las barras de este diagrama se agrupen por zonas. Entonces, tendremos que aplicar `barplot` a la tabla sin trasponer.

Además, vamos a colocar la leyenda en la esquina superior izquierda para que no se superponga a ninguna barra. También reduciremos el tamaño del texto de la leyenda para quede visible completamente.

```
Diagrama = apply(prop.table(table(jirafas), margin = 1), MARGIN = 1, FUN = cumsum)
barplot(Diagrama, ylim=c(0,1.2), beside = TRUE, legend = TRUE, main = "Diagrama de barras de frecuencias
args.legend=list(x="topleft", cex=0.55))  # cex hace la letra mas pequeña
```



Ejemplo 5 **Ejemplo 5**

Consideremos el data frame `datacrab` y arreglemos los datos.

```
crabs = read.table("./r-basic-master/data/datacrab.txt", header = TRUE)
crabs = crabs[,-1] #Omitimos la primera columna
str(crabs)
```

```
## 'data.frame':    173 obs. of  5 variables:
##  $ color : int  3 4 2 4 4 3 2 4 3 4 ...
##  $ spine : int  3 3 1 3 3 3 1 2 1 3 ...
##  $ width : num  28.3 22.5 26 24.8 26 23.8 26.5 24.7 23.7 25.6 ...
##  $ satell: int  8 0 9 0 4 0 0 0 0 0 ...
##  $ weight: int  3050 1550 2300 2100 2600 2100 2350 1900 1950 2150 ...
```

La variable numérica `width` contiene la anchura de cada cangrejo

```
table(crabs$width)
```

```
##
##   21   22 22.5 22.9   23 23.1 23.2 23.4 23.5 23.7 23.8 23.9   24 24.1 24.2 24.3
##    1    1    3    3    2    3    1    1    1    3    3    1    2    1    2    2
## 24.5 24.7 24.8 24.9   25 25.1 25.2 25.3 25.4 25.5 25.6 25.7 25.8 25.9   26 26.1
##    7    5    1    3    6    2    2    1    3    3    2    6    7    1    6    2
## 26.2 26.3 26.5 26.7 26.8   27 27.1 27.2 27.3 27.4 27.5 27.6 27.7 27.8 27.9   28
##    8    1    6    3    3    5    2    2    1    3    6    1    2    2    2    3
## 28.2 28.3 28.4 28.5 28.7 28.9   29 29.3 29.5 29.7 29.8   30 30.2 30.3 30.5 31.7
##    4    3    2    4    2    1    6    2    1    1    1    3    1    1    1    1
## 31.9 33.5
##    1    1
```

Un cangrejo mide 21, otro 22, tres mas 22.5 así sucesivamente

Vamos a convertir a la variable `width` en una variable ordinal que agrupe las entradas de la variable original en niveles.

La manera más sencilla de llevarlo a cabo es utilizando la función `cut`, que estudiaremos en detalle en lecciones posteriores. Por ahora, basta con saber que la instrucción dividirá el vector numérico `crabs$width` en intervalos de extremos los puntos especificados en el argumento `breaks`. El parámetro `right = FALSE` sirve para indicar que los puntos de corte pertenecen la intervalo de su derecha, e `Inf` indica ∞.

Por lo tanto, podemos utilizar la siguiente instrucción

```
intervalos = cut(crabs$width, breaks = c(21,25,29,33,Inf), right = FALSE,
                 labels = c("21-25", "25-29", "29-33", "33-..."))
```

El resultado de la instrucción es un factor que tiene como niveles estos intervalos, identificados con las etiquetas especificadas en el parámetro `labels`. Como nostros vamos a usar estos intervalos como niveles de una variable ordinal, además convertiremos este factor en ordenado.

```
crabs$width.rank = ordered(intervalos)
str(crabs)
```

```
## 'data.frame':    173 obs. of  6 variables:
##  $ color     : int  3 4 2 4 4 3 2 4 3 4 ...
##  $ spine     : int  3 3 1 3 3 3 1 2 1 3 ...
##  $ width     : num  28.3 22.5 26 24.8 26 23.8 26.5 24.7 23.7 25.6 ...
##  $ satell    : int  8 0 9 0 4 0 0 0 0 0 ...
##  $ weight    : int  3050 1550 2300 2100 2600 2100 2350 1900 1950 2150 ...
##  $ width.rank: Ord.factor w/ 4 levels "21-25"<"25-29"<..: 2 1 2 1 2 1 2 1 1 2 ...
```

Nos interesa estudiar la distribución de las anchuras de los cangrejos según el número de colores. Por lo tanto, vamos a calcular las tablas bidimensionales de frecuencias relativas y relativas acumuladas de los intervalos de las anchuras en cada nivel de `color` y las representaremos por medio de diagramas de barras.

La tabla de frecuencias absolutas de los pares se puede obtener aplicando `table` al data frame formado por la primera y última columnas.

```r
Tabla = table(crabs[,c(1,6)])
Tabla
```

```
##       width.rank
## color 21-25 25-29 29-33 33-...
##     2     1     9     2      0
##     3    19    62    13      1
##     4    17    24     3      0
##     5     9    12     1      0
```

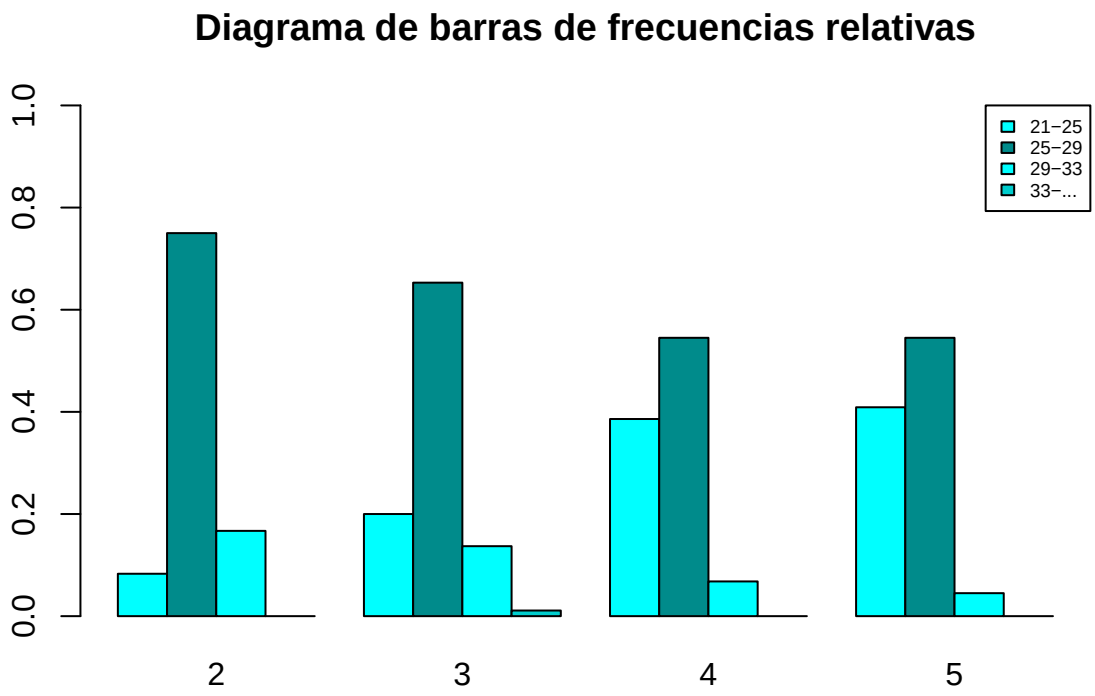
```r
Fr.rel = round(prop.table(Tabla,margin = 1),3)
Fr.rel
```

```
##       width.rank
## color 21-25 25-29 29-33 33-...
##     2 0.083 0.750 0.167  0.000
##     3 0.200 0.653 0.137  0.011
##     4 0.386 0.545 0.068  0.000
##     5 0.409 0.545 0.045  0.000
```

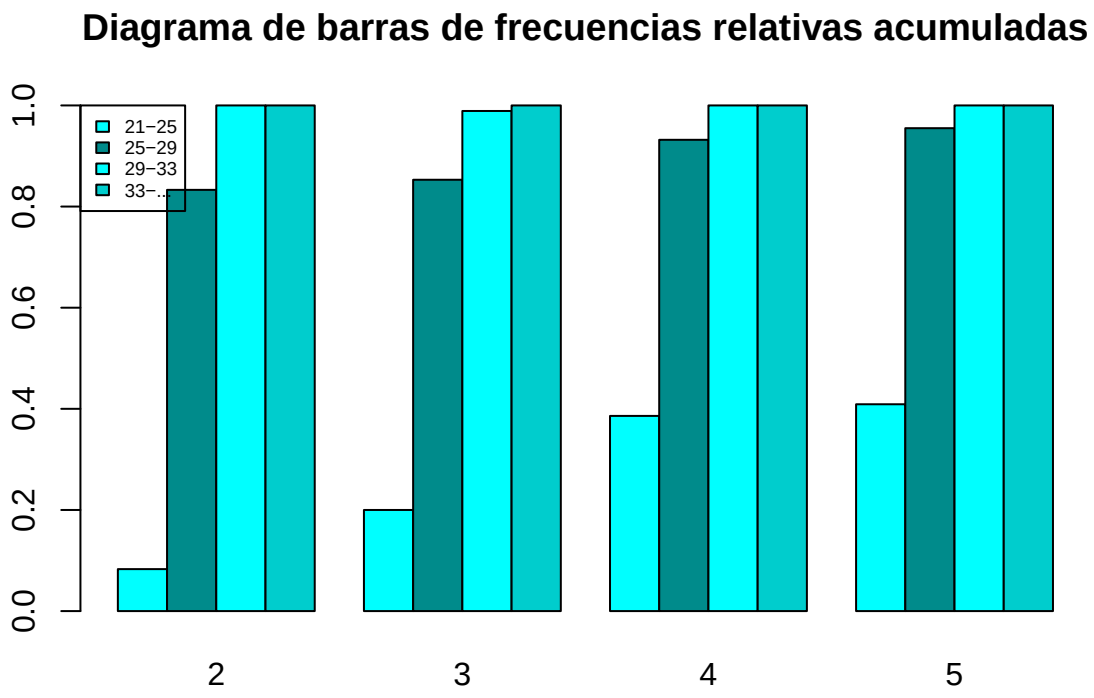
```r
Fr.rel.acu = round(apply(prop.table(Tabla, margin = 1), MARGIN = 1, FUN = cumsum), 3)
t(Fr.rel.acu)
```

```
##       width.rank
## color 21-25 25-29 29-33 33-...
##     2 0.083 0.833 1.000      1
##     3 0.200 0.853 0.989      1
##     4 0.386 0.932 1.000      1
##     5 0.409 0.955 1.000      1
```

```r
azul = c("cyan", "cyan4", "cyan1", "cyan3")

barplot(t(Fr.rel), beside = TRUE, legend = TRUE, ylim = c(0,1), col = azul,
        main = "Diagrama de barras de frecuencias relativas",
        args.legend=list(x = "topright", cex=0.55))
```




```
barplot(Fr.rel.acu, beside = TRUE, legend = TRUE, col = azul,
        main = "Diagrama de barras de frecuencias relativas acumuladas",
        args.legend=list(x = "topleft", cex=0.55))
```



Estadistica Descriptiva Con Datos Cuantitativos

Datos cuantitativos

Los datos cuantitativos son los que expresan cantidades que se representan mediante números. Éstos se suelen clasificar en continuos y discretos.

- Los datos continuos son los que, si existiese la posibilidad de medirlos con precisión infinita, en principio podrían tomar todos los valores de un intervalo de la recta real. A modo de ejemplo, el peso, la altura, el tiempo... son datos de este tipo.
- Por su parte, los datos discretos son los que pueden tomar un solo conjunto contable de valores. El número de colores de un gato, el número de individuos que conforman una población son algunos ejemplos de este tipo de datos.

Conviene tener en cuenta que esta división es solo teórica. Es decir, en la práctica, todos estos datos son discretos puesto que la precisión infinita no existe. Sin embargo, es necesario de vez en cuando suponer los datos de tipo continuo para así poder utilizar técnicas específicas en su análisis.

A la hora de estudiar variables cuantitativas, podemos utilizar las frecuencias que hemos visto hasta el momento: absoluta, relativa, acumulada y relativa acumulada. Esto se debe a que podemos ordenar los datos cuantitativos en el orden natural de los números reales.

En este caso, disponemos de muchas otras técnicas descriptivas aparte de las frecuencias, puesto que estamos trabajando con números reales y podemos operar con ellos.

Los datos cuantitativos admiten dos tipos de tratamiento según trabajemos con los raw data (datos brutos u originales) o bien los agrupemos en clases o intervalos.

En esta lección trabajaremos sobre la primera situación. En la siguiente, estudiaremos la descripción de datos cuantitativos agrupados.

Frecuencias

Frecuencias de datos cuantitativos El tratamiento de las frecuencias de datos cuantitativos es similar al de los datos ordinales. La cosa cambia ligeramente debido a que no se tienen en cuenta todos los niveles posibles, sino únicamente los observados.

Ejemplo 1 Ejemplo 1

Se han pedido las edades a 20 clientes de un museo. Las respuestas obtenidas han sido las siguientes:

```
edad = c(15,18,25,40,30,29,56,40,13,27,42,23,11,26,25,32,30,40,33,29)
```

Recordemos que solamente nos interesan las frecuencias de las edades observadas. Es decir, solamente nos interesan

```
table(edad)
```

```
## edad
## 11 13 15 18 23 25 26 27 29 30 32 33 40 42 56
##  1  1  1  1  1  2  1  1  2  2  1  1  3  1  1
```

Calculemos el resto de frecuencias como ya sabemos

```
round(prop.table(table(edad)),3)#frec relativas
```

```
## edad
##    11    13    15    18    23    25    26    27    29    30    32    33    40    42    56
## 0.05 0.05 0.05 0.05 0.05 0.10 0.05 0.05 0.10 0.10 0.05 0.05 0.15 0.05 0.05
```

```
cumsum(table(edad))
```

```
## 11 13 15 18 23 25 26 27 29 30 32 33 40 42 56
##  1  2  3  4  5  7  8  9 11 13 14 15 18 19 20
```

```
round(cumsum(prop.table(table(edad))),3)
```

```
##    11    13    15    18    23    25    26    27    29    30    32    33    40    42    56
## 0.05 0.10 0.15 0.20 0.25 0.35 0.40 0.45 0.55 0.65 0.70 0.75 0.90 0.95 1.00
```

Frecuencias de datos cuantitativos En general, supongamos que tenemos n observaciones de una propiedad que se mide con un número real y obtenemos la variable cuantitativa formada por los datos

$$x_1, \ldots, x_n$$

Sean ahora $X_1, \ldots, X_k$ los valores distintos que aparecen en esta lista de datos y considerémoslos ordenados

$$X_1 < X_2 < \cdots < X_k$$

Entonces, en esta variable cuantitativa

- La frecuencia absoluta de X_i es el número n_i de elementos que son iguales a X_i
- La frecuencia relativa de X_i es $f_i = \frac{n_i}{n}$
- La frecuencia absoluta acumulada de X_i es $N_i = \sum_{j=1}^{i} n_j$
- La frecuencia relativa acumulada de X_i es $F_i = \frac{N_i}{n}$

Ejemplo 2 Ejemplo 2

Lanzamos 25 veces un dado de 6 caras y anotamos las puntuaciones obtenidas en cada tirada.

En este caso, $n = 25$ y, los distintos valores observados son

$$X_1 = 1,\ X_2 = 2,\ X_3 = 3,\ X_4 = 4,\ X_5 = 5,\ X_6 = 6$$

Nos interesa ahora calcular las frecuencias de este experimento. Además, las organizaremos en un data frame para observarlas de forma más clara y sencilla en una tabla.

```
set.seed(162017)
dados = sample(1:6,25,replace = TRUE)
dados
```

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

```
set.seed(NULL)
```

```
table(dados)
```

```
## dados
##  1  2  3  4  5  6
## 10  3  3  2  5  2
```

```
round(prop.table(table(dados)),2)
```

```
## dados
##    1    2    3    4    5    6
## 0.40 0.12 0.12 0.08 0.20 0.08
```

```
cumsum(table(dados))
```

```
##  1  2  3  4  5  6
## 10 13 16 18 23 25
```

```
round(cumsum(prop.table(table(dados))),2)
```

```
##    1    2    3    4    5    6
## 0.40 0.52 0.64 0.72 0.92 1.00
```

```
dados.df = data.frame(Puntuacion = 1:6,
                      Fr.abs = as.vector(table(dados)),
                      Fr.rel = as.vector(round(prop.table(table(dados)),2)),
                      Fr.acu = as.vector(cumsum(table(dados))),
                      Fr.racu = as.vector(round(cumsum(prop.table(table(dados))),2)))
```

nota: los datos los convertimos como vector, dado que si no lo hacemos, tendremos filas repetidas con los nombres 1,2,3,4...

```
dados.df
```

```
##   Puntuacion Fr.abs Fr.rel Fr.acu Fr.racu
## 1          1     10   0.40     10    0.40
## 2          2      3   0.12     13    0.52
## 3          3      3   0.12     16    0.64
## 4          4      2   0.08     18    0.72
## 5          5      5   0.20     23    0.92
## 6          6      2   0.08     25    1.00
```

¡OJO! Para entrar una tabla unidimensional como una variable en un data frame, es conveniente transformarla en vector con `as.vector`. Si no, cada `table` y cada `prop.table` añadirían una columna extra con los nombres de los niveles.

Medidas de tendencia central

Las medidas de tendencia central son las que dan un valor representativo a todas las observaciones. Algunas de las más importantes son:

- La media aritmética o valor medio

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{\sum_{j=1}^{k} n_j X_j}{n} = \sum_{j=1}^{k} f_j X_j$$

- La mediana, que representa el valor central en la lista ordenada de observaciones.
- La moda es el valor (o valores) de máxima frecuencia (absoluta o relativa, el resultado será el mismo).

Mediana

La definición formal de la mediana es la siguiente. Denotando por

$$x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)}$$

los datos de la variable cuantitativa ordenados de menor a mayor, la mediana es

- Si n par, la medio de los dos datos centrales

$$\frac{x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}}{2}$$

- Si n impar, el dato central

$$x_{(\frac{n+1}{2})}$$

Ejemplo 1 Recordemos el ejemplo de las edades.

```
sort(edad) #Ordenamos los datos por su orden natural
```

```
##  [1] 11 13 15 18 23 25 25 26 27 29 29 30 30 32 33 40 40 40 42 56
```

```
table(edad)
```

```
## edad
## 11 13 15 18 23 25 26 27 29 30 32 33 40 42 56
##  1  1  1  1  1  2  1  1  2  2  1  1  3  1  1
```

En este caso, la moda es 40, la mediana es $\frac{29+29}{2} = 29$ y la media aritmética es $\frac{11+13+15+18+23+25+25+26+27+29+29+30+30+32+33+40+40+40+42+56}{20} = 29.2$

Ejemplo 2 Recordemos el ejemplo de los dados.

```
dados.df
```

```
##   Puntuacion Fr.abs Fr.rel Fr.acu Fr.racu
## 1          1     10   0.40     10    0.40
## 2          2      3   0.12     13    0.52
## 3          3      3   0.12     16    0.64
## 4          4      2   0.08     18    0.72
## 5          5      5   0.20     23    0.92
## 6          6      2   0.08     25    1.00
```

En este caso, la moda son dos valores: el 2 y el 3. La mediana es $x_{(13)} = 3$ y la media aritmética es 2.8

Medidas de tendencia central en R

Vamos a calcular la media aritmética, mediana y moda de los dos ejemplos anteriores con instrucciones de R.

```
mean(edad) #La media aritmética
```

```
## [1] 29.2
```

```
mean(dados)
```

```
## [1] 2.8
```

```
median(edad) #La mediana
```

```
## [1] 29
```

```
median(dados)
```

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

```
as.numeric(names(which(table(edad)==max(table(edad))))) #La moda
```

```
## [1] 40
```

```
as.numeric(names(which(table(dados)==max(table(dados)))))
```

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

Cuando trabajamos con datos cuantitativos, es conveniente que el resultado lo demos como un número. De ahí que hayamos aplicado la función `as.numeric`.

Medidas de posición

Las medidas de posición estiman qué valores dividen las observaciones en unas determinadas proporciones.

Los valores que determinan estas posiciones son conocidos como los cuantiles.

Pensándolo de este modo, la mediana puede interpretarse como una medida de posición, debido a que divide la variable cuantitativa en dos mitades.

Dada una proporción $p \in (0, 1)$, el cuantil de orden p de una variable cuantitativa, Q_p, es el valor más pequeño tal que su frecuencia relativa acumulada es mayor o igual a p.

Dicho de otro modo, si tenemos un conjunto de observaciones $x_1, \ldots, x_n$ y los ordenamos de menor a mayor, entonces Q_p será el número más pequeño que deja a su izquierda (incluyéndose a sí mismo) como mínimo a la fracción p de los datos. Es decir, $p \cdot n$.

Así, ahora es más claro ver que la mediana vendría a ser $Q_{0.5}$, el cuantil de orden 0.5.

Ejemplo 3 Ejemplo 3

Consideremos un experimento en el que lanzamos 50 veces un dado de 4 caras y obtenemos los siguientes resultados

```
set.seed(260798)
dado = sample(1:4, 50, replace = TRUE)
set.seed(NULL)
length(dado)
```

```
## [1] 50
```

```
dado = sort(dado) #Los ordenamos de menor a mayor
dado
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 4 4
## [39] 4 4 4 4 4 4 4 4 4 4 4 4
```

```
df.dado = data.frame(Puntuacion = 1:4,
                     Fr.abs = as.vector(table(dado)),
                     Fr.rel = as.vector(round(prop.table(table(dado)),2)),
                     Fr.acu = as.vector(cumsum(table(dado))),
                     Fr.racu = as.vector(round(cumsum(prop.table(table(dado))),2)))
df.dado
```

```
##   Puntuacion Fr.abs Fr.rel Fr.acu Fr.racu
## 1          1     16   0.32     16    0.32
## 2          2     15   0.30     31    0.62
## 3          3      5   0.10     36    0.72
## 4          4     14   0.28     50    1.00
```

Si nos piden el cuantil $Q_{0.3}$, sabemos que este es el primer elemento de la lista cuya frecuencia relativa acumulada es mayor o igual a 0.3. Este se corresponde con la puntuación 1.

También podríamos hallarlo de otro modo: fijándonos en la lista ordenada de puntuaciones, el cuantil $Q_{0.3}$ sería el primer elemento de dicha lista tal que fuera mayor o igual que, como mínimo, el 30% de los datos. Si calculamos el 30% de 50, obtenemos que es 15. Esto lo que nos dice es que el cuantil que buscamos es el número que se encuentra en la quinceava posición de la lista ordenada.

```
dado[15]
```

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

Cuantiles Algunos cuantiles tienen nombre propio:

- Los cuartiles son los cuantiles $Q_{0.25}, Q_{0.5}$ y $Q_{0.75}$. Respectivamente, son llamados primer, segundo y tercer cuartil. El primer cuartil, $Q_{0.25}$, será el menor valor que es mayor o igual a una cuarta parte de las observaciones y $Q_{0.75}$, el menor valor que es mayor o igual a tres cuartas partes de los datos observados.
- El cuantil $Q_{0.5}$ es la mediana
- Los deciles son los cuantiles Q_p con p un múltiplo de 0.1.
- Los percentiles son los cuantiles Q_p con p un múltiplo de 0.01.

La definición de cuantil anteriormente dada es orientativa. La realidad es que, exceptuando el caso de la mediana, no hay consenso sobre cómo deben calcularse los cuantiles. En verdad, existen diferentes métodos que pueden dar lugar a soluciones distintas.

Nuestro objetivo no es el de encontrar el primer valor de una muestra cuya frecuencia relativa acumulada en la variable sea mayor o igual a p, sino estimar el valor de esta cantidad para el total de la población.

Para calcular los cuantiles de orden p de una variable cualitativa x con `R`, se utiliza la instrucción `quantile(x,p)`, la cual dispone de 9 métodos diferentes que se especifican con el parámetro `type`. El valor por defecto es `type = 7` y no hace falta especificarlo, como veremos en el siguiente ejemplo.

```
set.seed(0)
dados2 = sample(1:6,15, replace = TRUE)
dados2
```

Ejemplo 4

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

```
set.seed(NULL)
quantile(dados2,0.25) #Primer cuartil
```

```
## 25%
##   2
```

```
quantile(dados2,0.8)
```

```
## 80%
##   5
```

Medidas de dispersión Las medidas de dispersión evalúan lo dispersos que están los datos. Algunas de las más importantes son:

- El rango o recorrido, que es la diferencia entre el máximo y el mínimo de las observaciones.
- El rango intercuartílico, que es la diferencia entre el tercer y primer cuartil, $Q_{0.75} - Q_{0.25}$.
- La varianza, a la que denotaremos por s^2, es la media aritmética de las diferencias al cuadrado entre los datos x_i y la media aritmética de las observaciones, $\bar{x}$.

$$s^2 = \frac{\sum_{j=1}^{n}(x_j - \bar{x})^2}{n} = \frac{\sum_{j=1}^{k} n_j(X_j - \bar{x})^2}{n} = \sum_{j=1}^{k} f_j(X_j - \bar{x})^2$$

  .
- La desviación típica es la raíz cuadrada positiva de la varianza, $s = \sqrt{s^2}$.
- La varianza muestral es la corrección de la varianza. La denotamos por $\tilde{s}^2$ y se corresponde con

$$\tilde{s}^2 = \frac{n}{n-1}s^2 = \frac{\sum_{j=1}^{n}(x_i - \bar{x})^2}{n-1}$$

- La desviación típica muestral, que es la raíz cuadrada positiva de la varianza muestral, $\tilde{s} = \sqrt{\tilde{s}^2}$

Propiedades de la varianza

Propiedades de la varianza.

- $s^2 \geq 0$. Esto se debe a que, por definición, es una suma de cuadrados de números reales.
- $s^2 = 0 \Longrightarrow x_j - \bar{x} = 0 \ \forall j = 1, \ldots, n$. En consecuencia, si $s^2 = 0$, entonces todos los datos son iguales.
- $s^2 = \frac{\sum_{j=1}^{n} x_j^2}{n} - \bar{x}^2$. Es decir, la varianza es la media de los cuadrados de los datos menos el cuadrado de la media aritmética de estos.

Varianza y varianza muestral

La diferencia entre ambas definiciones viene por la interrelación entre la estadística descriptiva y la inferencial.

Por un lado, es normal medir cómo varían los datos cuantitativos mediante su varianza definida como la media aritmética de las distancias al cuadrado de los datos a su valor medio. No obstante, por otro lado, el conjunto de nuestras observaciones, por lo normal, será una muestra de una población mucho mayor y nos interesará estimar entre otras muchas cosas su variabilidad.

La varianza de una muestra suele dar valores más pequños que la varianza de la población, mientras que la varianza muestral tiende a dar valores alrededor de la varianza de la población.

Esta corrección, para el caso de una muestra grande no es notable. Dividir n entre $n-1$ en el caso de n ser grande no significa una gran diferencia y aún menos si tenemos en cuenta que lo que tratamos es de estimar la varianza de la población, no de calcularla de forma exacta.

En cambio, si la muestra es relativamente pequeña (digamos $n < 30$), entonces la varianza muestral aproxima significativamente mejor la varianza de la población que la varianza.

La diferencia entre desviación típica y desviación típica muestral es análoga.

Con `R`, calcularemos la varianza y la desviación típica **muestrales**. Con lo cual, si queremos calcular las que no son muestrales, tendremos que multiplicarlas por $\frac{n-1}{n}$, donde n es el tamaño de la muestra. Lo veremos a continuación.

Varianza y desviación típica

Nótese que tanto la varianza como la desviación típica dan una información equivalente. Entonces, es comprensible preguntarse por qué se definen ambas medidas si con una basta. Pues bien, las unidades de la varianza (metros, litros, años...), ya sea muestral o no, están al cuadrado, mientras que las de la desviación típica no.

Medidas de dispersión con R

Medida de dispersión	Instrucción
Valores mínimo y máximo	`range(x)`
Rango	`diff(range(x))`
Rango intercuartílico	`IQR(x, type = ...)`
Varianza muestral	`var(x)`
Desviación típica muestral	`sd(x)`
Varianza	`var(x)*(length(x)-1)/length(x)`
Desviación típica	`sd(x)*sqrt((length(x)-1)/length(x))`

```
dados2
```

Ejemplo 4

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

```
diff(range(dados2))
```

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

```
IQR(dados2)
```

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

```
var(dados2)
```

```
## [1] 3.209524
```

```
sd(dados2)
```

```
## [1] 1.791514
```

```
n = length(dados2)
var(dados2)*(n-1)/n
```

```
## [1] 2.995556
```

```
sd(dados2)*sqrt((n-1)/n)
```

```
## [1] 1.730767
```

Función summary()

La función `summary` aplicada a un vector numérico o a una variable cuantitativa nos devuelve un resumen estadístico con los valores mínimo y máximo del vector, sus tres cuartiles y su media.

Al aplicar esta función a un data frame, esta se aplica a todas sus variables de forma simultánea. De este modo, podemos observar rápidamente si hay diferencias notables entre sus variables numéricas.

```
cangrejos = read.table("r-basic-master/data/datacrab.txt", header = TRUE) #Cargamos el data frame
cangrejos = cangrejos[-1] #Eliminamos la primera columna
summary(cangrejos) #Aplicamos la función summary
```

```
##      color           spine           width          satell           weight
##  Min.   :2.000   Min.   :1.000   Min.   :21.0   Min.   : 0.000   Min.   :1200
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:24.9   1st Qu.: 0.000   1st Qu.:2000
##  Median :3.000   Median :3.000   Median :26.1   Median : 2.000   Median :2350
##  Mean   :3.439   Mean   :2.486   Mean   :26.3   Mean   : 2.919   Mean   :2437
##  3rd Qu.:4.000   3rd Qu.:3.000   3rd Qu.:27.7   3rd Qu.: 5.000   3rd Qu.:2850
##  Max.   :5.000   Max.   :3.000   Max.   :33.5   Max.   :15.000   Max.   :5200
```

Ejemplo 5 Si nos interesa comparar numéricamente los pesos y las anchuras de los cangrejos con 3 colores con los que tienen 5 colores, utilizaríamos las siguientes instrucciones:

```
summary(subset(cangrejos, color == 3,c("weight","width")))
```

```
##      weight         width
##  Min.   :1300   Min.   :22.5
##  1st Qu.:2100   1st Qu.:25.1
##  Median :2500   Median :26.5
##  Mean   :2538   Mean   :26.7
##  3rd Qu.:3000   3rd Qu.:28.2
##  Max.   :5200   Max.   :33.5
```

```
summary(subset(cangrejos, color == 5,c("weight","width")))
```

```
##      weight         width
##  Min.   :1300   Min.   :21.00
##  1st Qu.:1900   1st Qu.:23.90
##  Median :2125   Median :25.50
##  Mean   :2174   Mean   :25.28
##  3rd Qu.:2400   3rd Qu.:26.57
##  Max.   :3225   Max.   :29.30
```

Y deducimos así que los cangrejos con 5 colores pesan ligeramente menos y tienen menos anchura que los que tienen 3 colores.

Función by()

La función `by()` se utiliza para aplicar una determinada función a algunas columnas de un data frame segmentándolas según los niveles de un factor.

La sintaxis de esta función es `by(columnas, factor, FUN = función)`.

Con lo cual, haciendo uso de la función `by` y especificando `FUN = summary`, podremos calcular el resumen estadístico anteriormente comentado a subpoblaciones definidas por los niveles de un factor.

Ejemplo 6 Para este ejemplo, haremos uso del famoso dataset iris.

Si nos interesase calcular de forma rápida y sencilla las longitudes de sépalos y petalos en función de la especie, necesitaríamos hacer uso de la instrucción mostrada a continuación.

Por motivos de espacio, no se muestran los resultados proporcionados por R.

```
by(iris[,c(1,3)], iris$Species, FUN = summary)
```

Función aggregate()

Tanto la función `by` como la función `aggregate` son equivalentes. No obstante, los resultados se muestran de forma diferente en función de cual utilicemos.

En el caso del ejemplo anterior, convenía más hacer uso de la función `by`.

Podemos comprobarlo introduciendo por consola la siguiente instrucción:

```
aggregate(cbind(Sepal.Length,Petal.Length)~Species, data=iris, FUN=summary)
```

NA La mayoría de las funciones vistas a lo largo de este tema no funcionan bien con valores `NA`.

Para no tenerlos en cuenta a la hora de aplicar estas funciones, hay que especificar el parámetro `na.rm = TRUE` en el argumento de la función.

```
dadosNA = c(dados2,NA)
dadosNA
```

Ejemplo 7

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

```
mean(dadosNA)
```

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

```
mean(dadosNA, na.rm = TRUE)
```

```
## [1] 3.266667
```

Diagramas de caja

El conocido diagrama de caja o box plot es un tipo de gráfico que básicamente, remarca 5 valores estadísticos:

- La mediana, representada por la línea gruesa que divide la caja
- El primer y tercer cuartil, que son los lados inferior y superior, respectivamente. De este modo, la altura de la caja es el rango intercuantílico
- Los extremos, los valores b_{inf}, b_{sup}, son los bigotes (whiskers) del gráfico. Si m y M son el mínimo y máximo de la variable cuantitativa, entonces los extremos se calculan del siguiente modo:

$$b_{inf} = \max\{m, Q_{0.25} - 1.5(Q_{0.75} - Q_{0.25})\}$$

$$b_{sup} = \min\{M, Q_{0.75} + 1.5(Q_{0.75} - Q_{0.25})\}$$

- Valores atípicos o outliers, que son los que están más allá de los bigotes. Se marcan como puntos aislados.

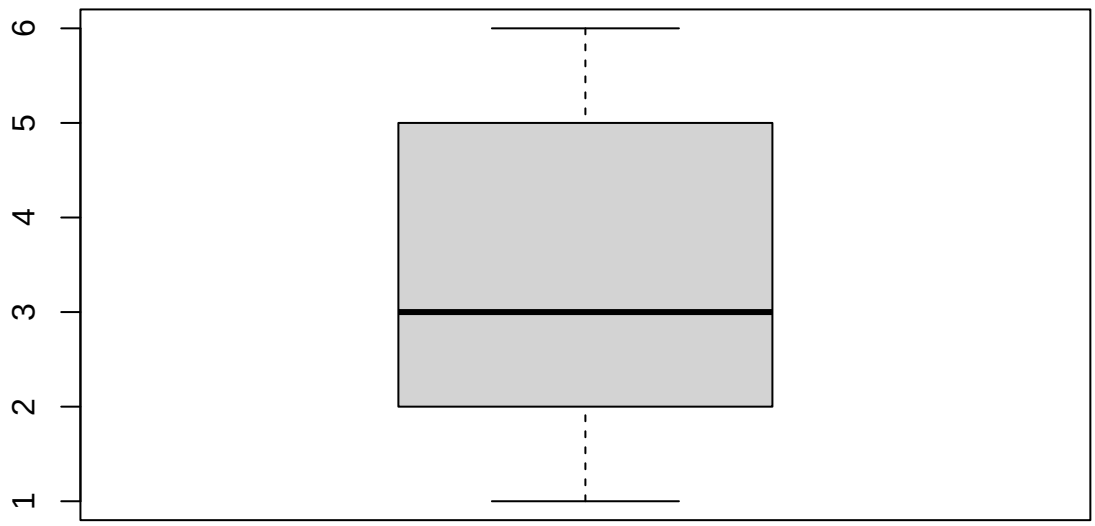
Más sobre los bigotes Por su definición, concluimos que los bigotes marcan el mínimo y máximo de la variable cuantitativa, a no ser que haya datos muy alejados de la caja intercuantílica.

En tal caso, el bigote inferior marca el valor 1.5 veces el rango intercuantílico por debajo de $Q_{0.25}$, mientras que el superior marca el valor 1.5 veces el rango intercuantílico por encima de $Q_{0.75}$

Función boxplot

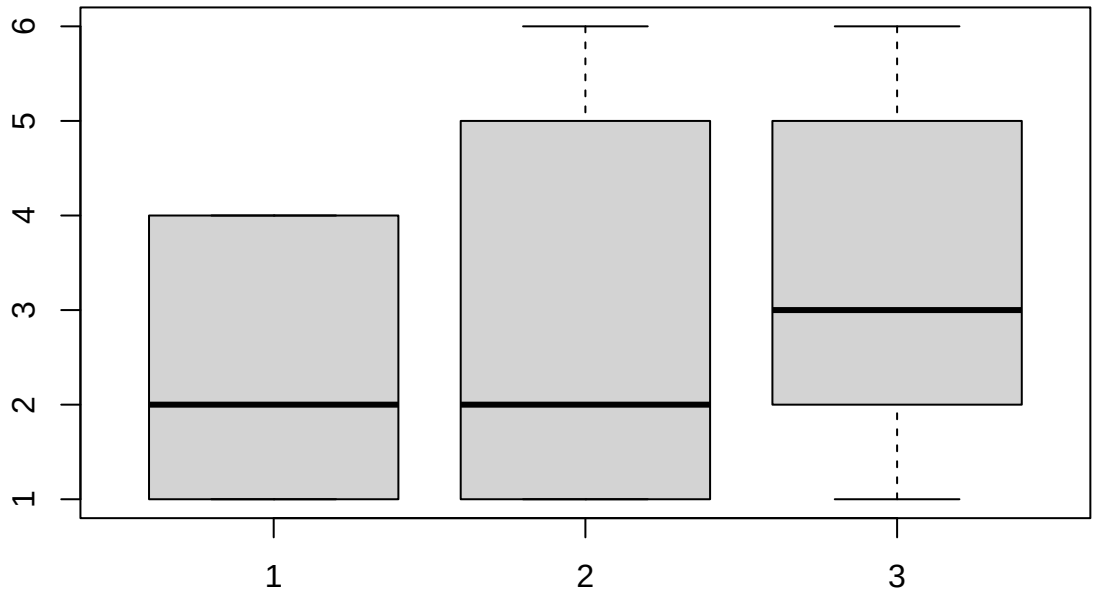
La instrucción `boxplot()` dibuja diagramas de caja en R.

```
boxplot(dados2, main = "Un diagrama de caja")
```



También podemos dibujar diversos diagramas de caja en un mismo gráfico. De este modo, se pueden comparar con mayor facilidad:

```
boxplot(dado,dados,dados2)
```

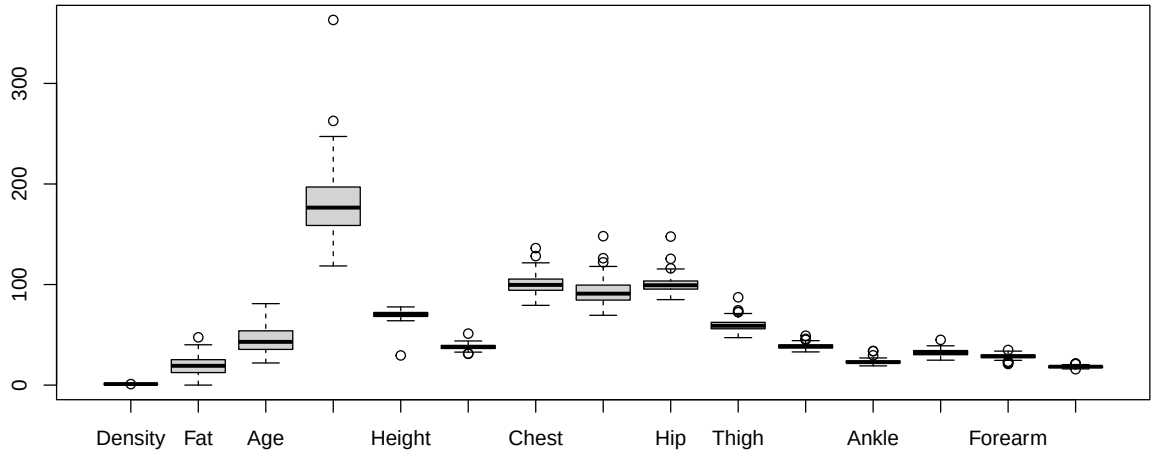


Además, podemos dibujar el diagrama de caja de todas las variables de un data frame en un solo paso aplicando la instrucción `boxplot(data.frame)`.

La mayoría de veces, dicho gráfico no será del todo satisfactorio. Dibujar diagramas de factores no tiene sentido alguno. Estos gráficos se pueden manipular incluyendo solo las variables de interés, cambiando los nombres...

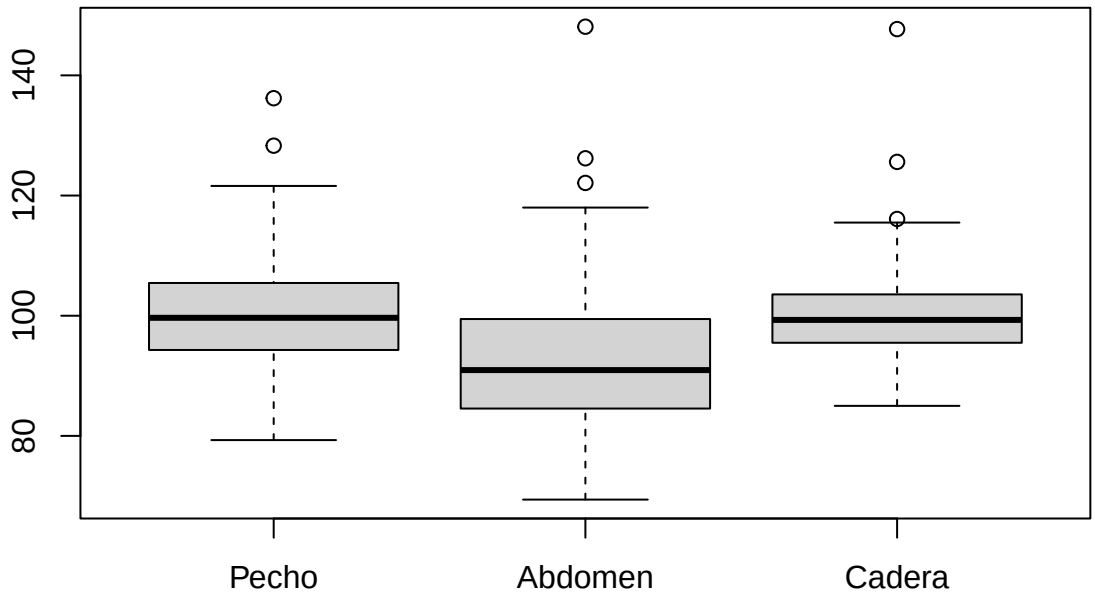
Veamos un ejemplo:

```
body = read.table("r-basic-master/data/bodyfat.txt", header = TRUE)
boxplot(body)
```



Ejemplo 8

```
boxplot(body[,7:9], names = c("Pecho", "Abdomen", "Cadera"))
```

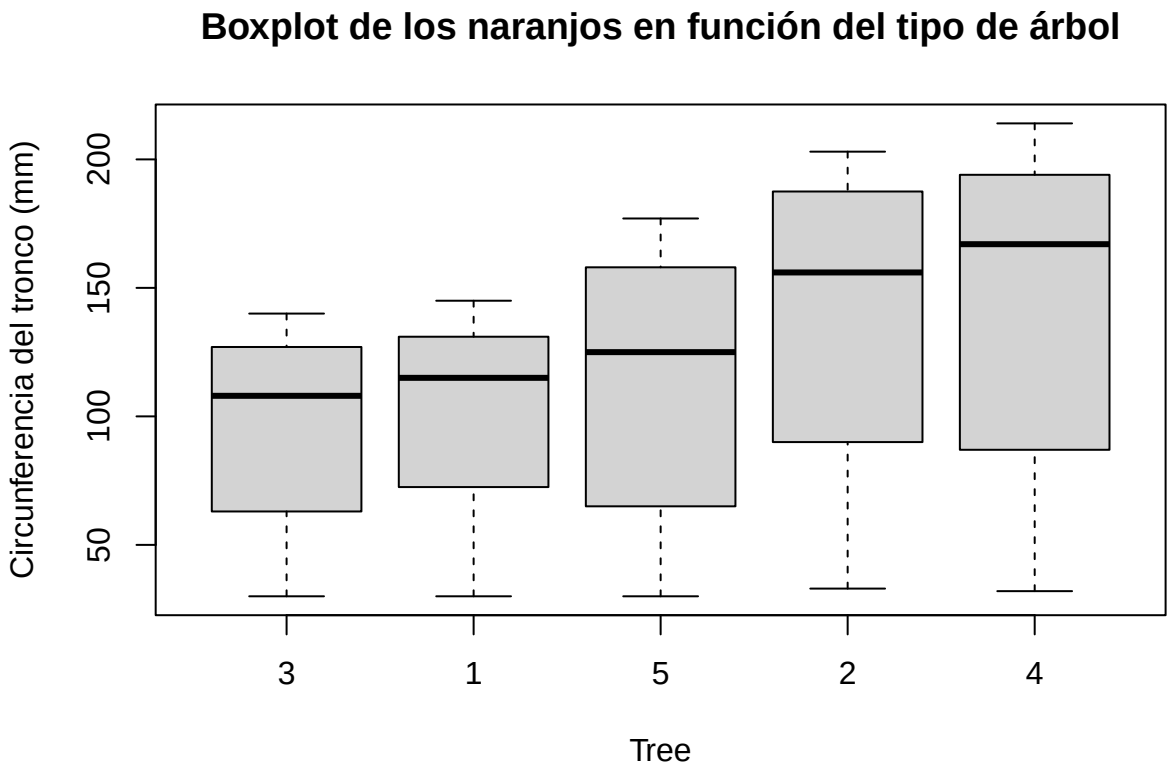


Agrupar varios diagramas de caja en un solo gráfico tiene por objetivo poder compararlos visualmente, lo cual tiene sentido cuando las variables tienen significados parecidos o cuando comparamos una misma variable de poblaciones distintas.

La mayoría de las veces, queremos comparar diagramas de cajas de una misma variable cuantitativa segmentada por los niveles de un factor.

La sintaxis de la instrucción para dibujar en un único gráfico los diagramas de caja de una variable numérica de un data frame en función de los niveles de un factor del mismo data frame es `boxplot(var.numérica~factor, data = data frame)`

```
boxplot(circumference~Tree, data = Orange, ylab = "Circunferencia del tronco (mm)",
        main = "Boxplot de los naranjos en función del tipo de árbol")
```



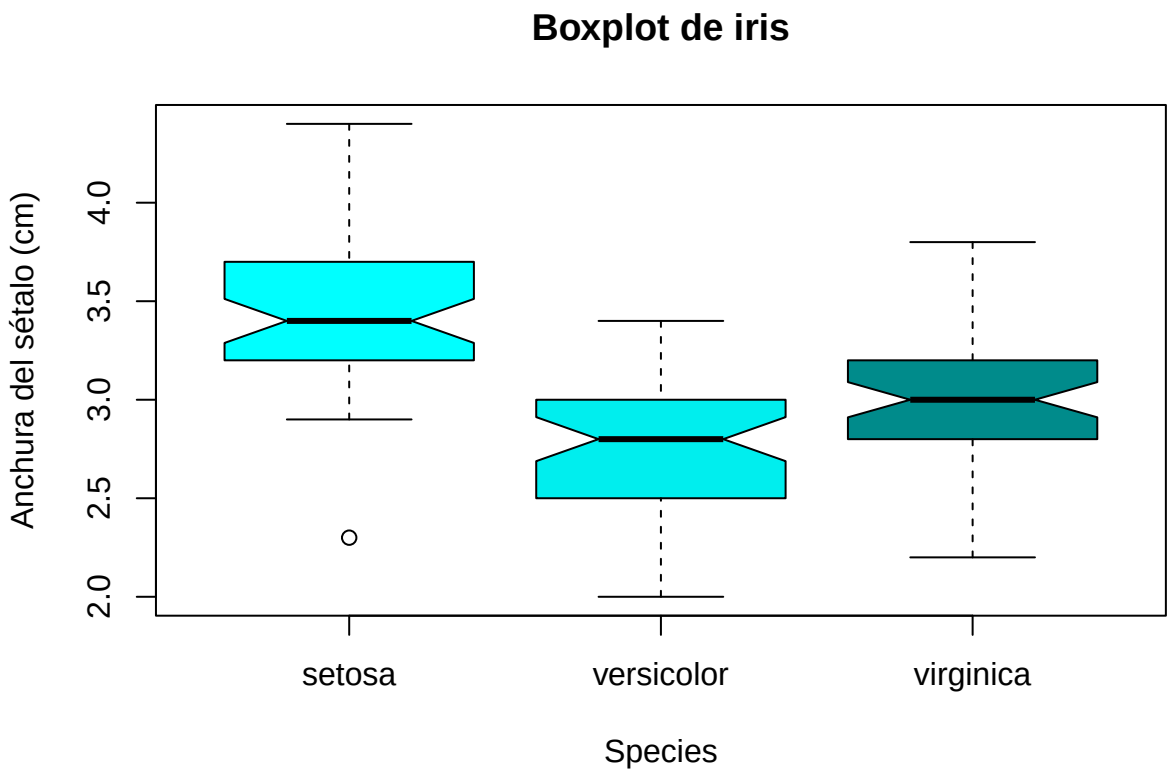
Ejemplo 9

Parámetros de la función boxplot Todos los parámetros de la función `plot()` que tengan sentido pueden ser utilizados en los argumentos de la función `boxplot()`.

Aparte, la función `boxplot()` dispone de algunos parámetros específicos, de los cuales mencionaremos:

- `notch` igualado a `TRUE` añade una muesca en la mediana de la caja. Si se da el caso en que las muescas de dos diagramas de cajas no se solapan, entonces con alto grado de confianza, concluimos que las medianas de las poblaciones correspondientes son diferentes.

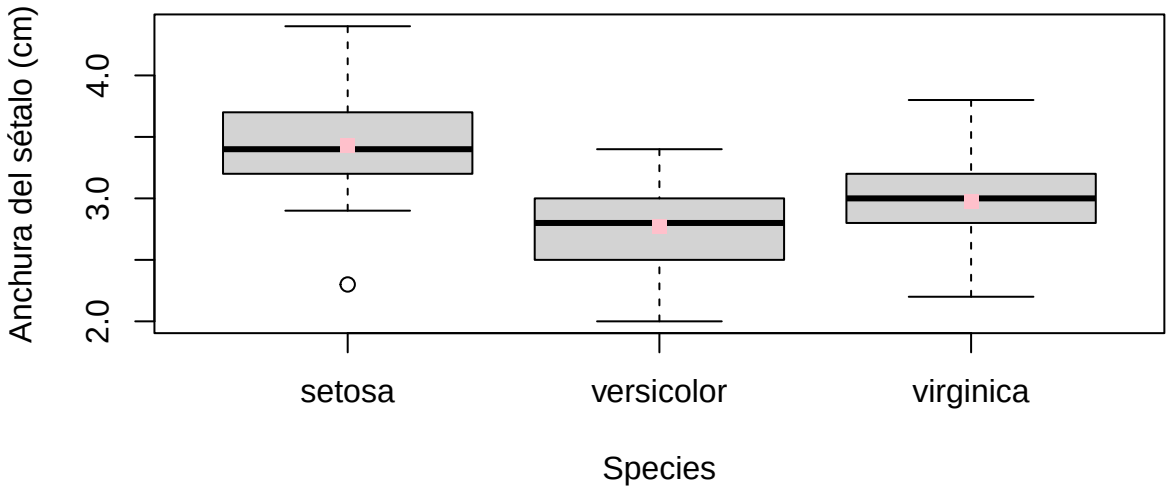
```
boxplot(Sepal.Width~Species, data = iris, ylab = "Anchura del sétalo (cm)",
        notch = TRUE, col = c("cyan","cyan2","cyan4"),
        main = "Boxplot de iris")
```



Ejemplo 10

Si quisiéramos marcar de alguna forma en un diagrama de caja, cosa que puede ser muy útil en ocasiones, la media aritmética de la variable correspondiente, podríamos hacerlo mediante la función `points`:

```
boxplot(Sepal.Width~Species, data = iris, ylab = "Anchura del sétalo (cm)")
medias = aggregate(Sepal.Width~Species, data = iris, FUN = mean)
points(medias, col = "pink", pch = 15)
```



La primera instrucción del chunk anterior genera el diagrama de cajas de las anchuras de los sépalos en función de la especie. Por su parte, la segunda instrucción lo que hace es calcular las medias aritméticas de las anchuras según la especie. Finalmente, la tercera instrucción lo que hace es añadir al diagrama un punto cuadrado a cada caja en la ordenada correspondiente a su media aritmética.

La estructura interna de boxplot Como ya sabemos, podemos estudiar la función interna de algunos objetos con la función `str`.

Dicha función aplicada a un boxplot, nos produce una list. Podéis ver esta list si introducís por consola la siguiente instrucción: `str(boxplot(circumference~Tree, data = Orange))` Destacaremos dos de sus componenetes aquí:

- `stats` nos devuelve los valores $b_{inf},\ Q_{0.25},\ Q_{0.5},\ Q_{0.75},\ b_{sup}$
- `out` nos retorna los valores atípicos. En caso de haber diversos diagramas en un plot, la componente `group` nos indica a qué diagramas pertenecen estos ouliers.

###Datos cuantitativos agrupados

Aunque no seamos completamente conscientes de ello, tendemos a agrupar datos cuantitativos constantemente.

Sin ir más lejos, calificamos de excelente a todas las notas que están sobre el 9. También decimos que una persona tiene 20 años cuando se encuentra en el intervalo [20,21). Es decir, cuando ha cumplido los 20 pero aún no tiene los 21.

En estadística, existen innumerables motivos por los cuales nos interesa agrupar los datos cuando estos son cuantitativos. Uno de estos motivos puede ser perfectamente que los datos sean muy heterogéneos. En este caso, nos encontraríamos con que las frecuencias de los valores individuales serían todas muy similares, lo que daría lugar a un diagrama de barras muy difícil de interpretar, tal y como mostramos en el siguiente ejemplo.

Ejemplo 1 Ejemplo 1

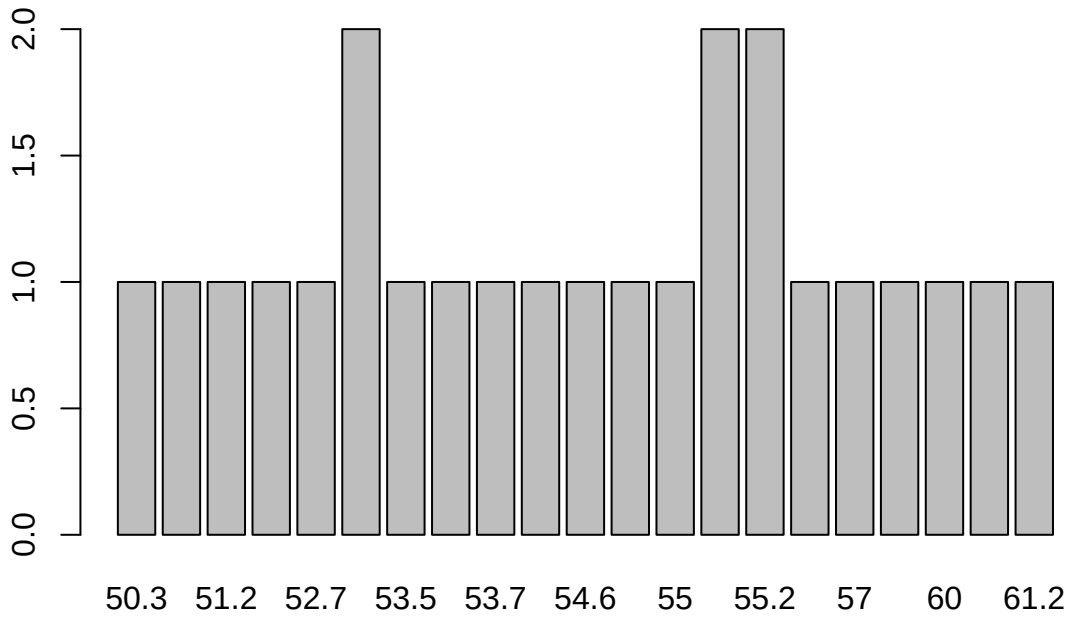
Consideremos la siguiente muestra de 24 pesos de estudiantes:

```
pesos = c(55.2,54.0,55.2,53.7,60.2,53.2,54.6,55.1,51.2,53.2,54.8,52.3,56.9,57.0,55.0,
          53.5,50.9,55.1,53.6,61.2,59.5,50.3,52.7,60.0)
```

El diagrama de barras de sus frecuencias absolutas, tomando como posibles niveles todos los pesos entre su mínimo y maximo

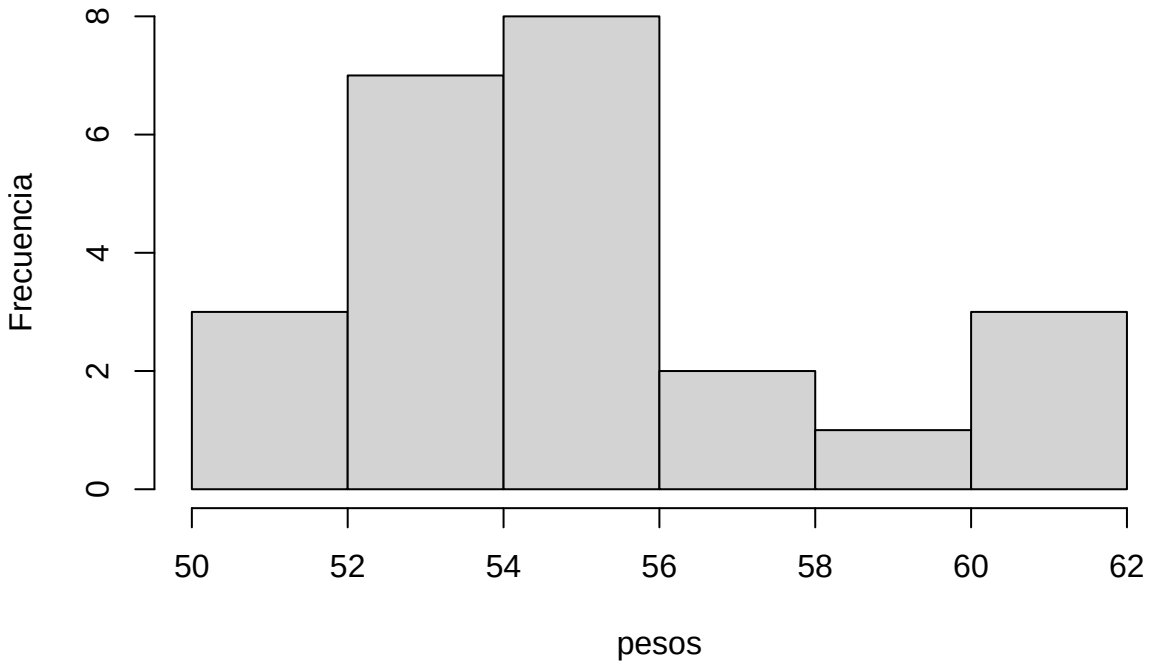
Como vemos, todas estas frecuencias se encuentran entre 0 y 2, cosa que no nos da mucha información.

```
barplot(table(pesos))
```



En cambio, si dividimos todos estos posibles valores que puede tomar la variable cuantitativa en intervalos y tomamos como sus frecuencias las de todos los valores que caen en dicho intervalo, la cosa cambia.

En este caso, sería mucho más fácil interpretar los resultados, ya que estos darán mucha más información. Más adelante veremos como crear estos intervalos.



Otro de los motivos por el que necesitamos muchas veces agrupar los datos cuantitativos es porque, como ya dijimos en temas anteriores, la precisión infinita no existe. Por tanto, esta imposibilidad de medir de manera exacta muchas de las magnitudes continuas (tiempo, peso, altura...) nos obliga a trabajar con aproximaciones o redondeos de valores reales y que cada uno de estos represente todo un intervalo de posibles valores.

Por lo general, existen 3 situaciones en las cuales conviene sin lugar a dudas agrupar datos cuantitativos en intervalos, también llamados clases

- Cuando los datos son continuos, su redondeo ya define un agrupamiento debido a la inexistencia de precisión infinita
- Cuando los datos son discretos, pero con un número considerablemente grande de posibles valores
- Cuando tenemos muchísimos datos y estamos interesados en estudiar las frecuencias de sus valores

Cómo agrupar datos

Este proceso consta de 4 pasos:

1. Decidir el número de intervalos que vamos a utilizar
2. Decidir la amplitud de estos intervalos
3. Acumular los extremos de los intervalos
4. Calcular el valor representativo de cada intervalo, su marca de clase

No hay una forma de agrupar datos mejor que otra. Cada uno de los diferentes agrupamientos para un conjunto de datos podría sacar a la luz características diferentes del conjunto.

Sí calificamos de excelente a todas las notas que están sobre el 9. También decimos que una persona tiene 20 años cuando se encuentra en el intervalo [20,21). Es decir, cuando ha cumplido los 20 pero aún no tiene los 21.

En estadística, existen innumerables motivos por los cuales nos interesa agrupar los datos cuando estos son cuantitativos. Uno de estos motivos puede ser perfectamente que los datos sean muy heterogéneos. En este caso, nos encontraríamos con que las frecuencias de los valores individuales serían todas muy similares, lo que daría lugar a un diagrama de barras muy difícil de interpretar, tal y como mostramos en el siguiente ejemplo.

Ejemplo 1 Ejemplo 1

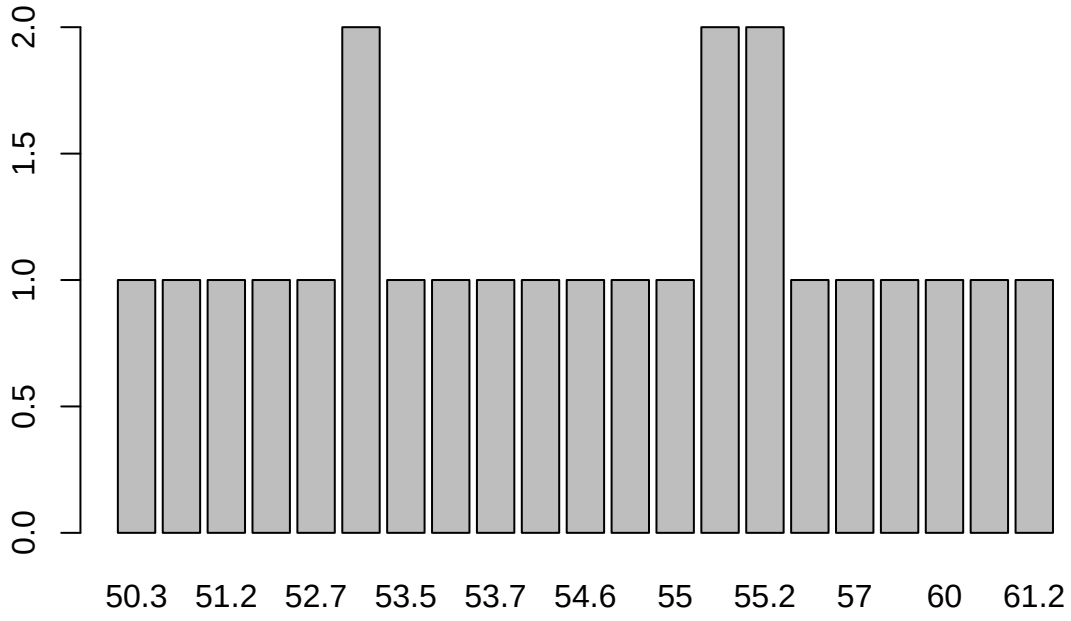
Consideremos la siguiente muestra de 24 pesos de estudiantes:

```
pesos = c(55.2,54.0,55.2,53.7,60.2,53.2,54.6,55.1,51.2,53.2,54.8,52.3,56.9,57.0,55.0,
          53.5,50.9,55.1,53.6,61.2,59.5,50.3,52.7,60.0)
```

El diagrama de barras de sus frecuencias absolutas, tomando como posibles niveles todos los pesos entre su mínimo y máximo se muestra en la siguiente diapositiva.

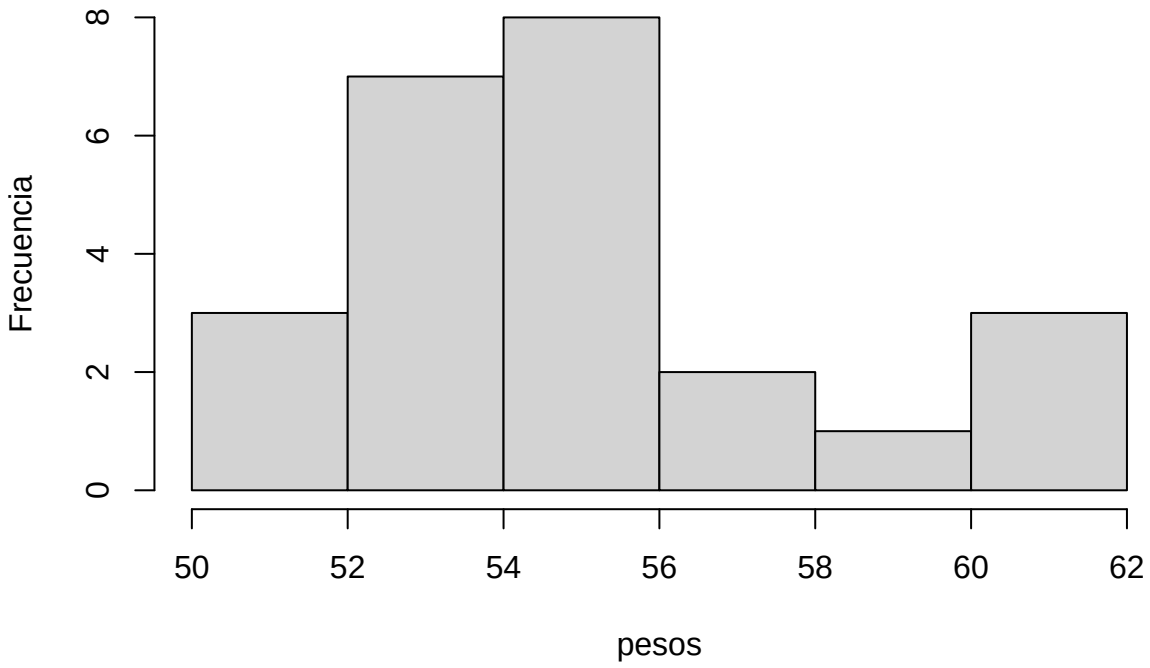
Como vemos, todas estas frecuencias se encuentran entre 0 y 2, cosa que no nos da mucha información.

```
barplot(table(pesos))
```



En cambio, si dividimos todos estos posibles valores que puede tomar la variable cuantitativa en intervalos y tomámos como sus frecuencias las de todos los valores que caen en dicho intervalo, hay un cambio

En este caso, sería mucho más fácil interpretar los resultados, ya que estos darán mucha más información. Más adelante veremos como crear estos intervalos.



Otro de los motivos por el que necesitamos muchas veces agrupar los datos cuantitativos es porque, como ya dijimos en temas anteriores, la precisión infinita no existe. Por tanto, esta imposibilidad de medir de manera exacta muchas de las magnitudes continuas (tiempo, peso, altura...) nos obliga a trabajar con aproximaciones o redondeos de valores reales y que cada uno de estos represente todo un intervalo de posibles valores.

Por lo general, existen 3 situaciones en las cuales conviene agrupar datos cuantitativos en intervalos, también llamados clases

- Cuando los datos son continuos, su redondeo ya define un agrupamiento debido a la inexistencia de precisión infinita
- Cuando los datos son discretos, pero con un número considerablemente grande de posibles valores
- Cuando tenemos muchísimos datos y estamos interesados en estudiar las frecuencias de sus valores

La función hist() La función de `R` por excelencia para estudiar datos agrupados es `hist`. Dicha función implementa los 4 pasos del proceso.

Si le indicamos como argumentos el vector de datos y el número de intervalos que deseamos, o bien el método para determinarlo (cosa que veremos a continuación), la función agrupará los datos en el número de clases que le hemos introducido, más o menos. sin control de ningún tipo por nuestra parte sobre los intervalos que produce.

Esto puede ser util en algunos casos, pero no en otros.

Estableciendo el número de clases En este tema explicaremos una receta para agrupar datos. Lo dicho, ni mejor ni peor que el resto.

Lo primero es establecer el número k de clases en las que vamos a dividir nuestros datos. Podemos decidir en función de nuestros intereses o podemos hacer uso de alguna de las reglas existentes. Destacaremos las más populares. Sea n el número total de datos de la muestra

- Regla de la raíz cuadrada: $k = \lceil \sqrt{n} \ \rceil$
- Regla de Sturges: $k = \lceil 1 + \log_2(n) \rceil$
- Regla de Scott: Se determina primero la amplitud teórica, A_S de las clases

  $$A_S = 3.5 \cdot \tilde{s} \cdot n^{-\frac{1}{3}}$$

  donde $\tilde{s}$ es la desviación típica muestral. Luego se toma

  $$k = \left\lceil \frac{\max(x) - \min(x)}{A_S} \right\rceil$$

- Regla de Freedman-Diaconis: Se determina primero la amplitud teórica, A_{FD} de las clases

  $$A_{FD} = 2 \cdot (Q_{0.75} - Q_{0.25}) \cdot n^{-\frac{1}{3}}$$

  (donde, recordemos, $Q_{0.75} - Q_{0.25}$, es el rango intercuantílico) y entonces

  $$k = \left\lceil \frac{\max(x) - \min(x)}{A_{FD}} \right\rceil$$

Observe, las dos primeras solo dependen de n, mientras que las dos últimas también tienen en cuenta, de formas diferentes, la dispersión de los datos. De nuevo, no hay ninguna mejor que las demás. Pero sí puede ocurrir que métodos diferentes den lugar a la observación de características diferentes en los datos.

Estableciendo el número de clases con R

Las instrucciones para llevar a cabo las 3 últimas reglas con `R` son, respectivamente,

- `nclass.Sturges`
- `nclass.scott`
- `nclass.FD`

Puede ocurrir que las difrentes reglas den valores diferentes, o no.

Decidiendo la amplitud Una vez determinado k, hay que decidir su amplitud.

La forma más fácil y la que nosotros utilizaremos por defecto es que la amplitud de todos los intervalos sea la misma, A. Esta forma no es la única.

Para calcular A, lo que haremos será dividir el rango de los datos entre k, el número de clases, y redondearemos por exceso a un valor de la precisión de la medida.

Si se da el improbable caso en que el cociente de exacto, tomaremos como A ese cociente más una unidad de precisión.

Extremos de los intervalos Para calcular los extremos de los intervalos. Nosotros tomaremos estos intervalos siempre cerrados por su izquierda y abiertos por la derecha, debido a que esta es la forma en que R los construye y porque es así como se utilizan en Teoría de Probabilidades al definir la distribución de una variable aleatoria discreta y también en otras muchas situaciones cotidianas.

Utilizaremos la siguiente notación

$$[L_1, L_2), [L_2, L_3), \ldots, [L_k, L_{k+1})$$

donde los L_i denotan los extremos de los intervalos. Estos se calculan de la siguiente forma:

$$L_1 = \min(x) - \frac{1}{2} \cdot \text{precisión}$$

Extremos de los intervalos A partir de L_1, el resto de intervalos se obtiene de forma recursiva:

$$\begin{aligned} L_2 &= L_1 + A \\ L_3 &= L_2 + A \\ &\vdots \\ L_{k+1} &= L_k + A \end{aligned}$$

Si nos fijamos bien, los extremos forman una progresión aritmética de salto A:

$$L_i = L_1 + (i-1)A, \qquad i = 2, \ldots, k+1$$

De esta forma garantizamos que los extremos de los intervalos nunca coincidan con valores del conjunto de datos, puesto que tinen una precisión mayor.

Solo nos queda determinar la marca de clase, X_i, de cada intervalo $[L_i, L_{i+1})$.

Este no es más que un valor del intervalo que utilizaremos para identificar la clase y para calcular algunos estadísticos.

Generalmente,

$$X_i = \frac{L_i + L_{i+1}}{2}$$

es decir, X_i será el punto medio del intervalo, para así garantizar que el error máximo cometido al describir cualquier elemento del intervalo por medio de su marca de clase sea mínimo o igual a la mitad de la amplitud del respectivo intervalo.

Es sencillo concluir que, al tener todos los intervalos amplitud A, la distancia entre X_i y X_{i+1} tambien será A. Por consiguiente,

$$X_i = X_1 + (i-1)A, \qquad i = 2, \ldots, k$$

donde

$$X_1 = \frac{L_1 + L_2}{2}$$

Decidiendo la amplitud Una vez determinado k, hay que decidir su amplitud.

La forma más fácil y la que nosotros utilizaremos por defecto es que la amplitud de todos los intervalos sea la misma, A. Esta forma no es la única.

Para calcular A, lo que haremos será dividir el rango de los datos entre k, el número de clases, y redondearemos por exceso a un valor de la precisión de la medida.

Si se da el improbable caso en que el cociente de exacto, tomaremos como A ese cociente más una unidad de precisión.

Marca de clase Solo nos queda determinar la marca de clase, X_i, de cada intervalo $[L_i, L_{i+1})$.

Este no es más que un valor del intervalo que utilizaremos para identificar la clase y para calcular algunos estadísticos.

Generalmente,

$$X_i = \frac{L_i + L_{i+1}}{2}$$

es decir, X_i será el punto medio del intervalo, para así garantizar que el error máximo cometido al describir cualquier elemento del intervalo por medio de su marca de clase sea mínimo o igual a la mitad de la amplitud del respectivo intervalo.

Es sencillo concluir que, al tener todos los intervalos amplitud A, la distancia entre X_i y X_{i+1} tambien será A. Por consiguiente,

$$X_i = X_1 + (i-1)A, \qquad i = 2, \ldots, k$$

donde

$$X_1 = \frac{L_1 + L_2}{2}$$

Ejemplo 2 Vamos a considerar el conjunto de datos de `datacrab`. Para nuestro estudio, trabajaremos únicamente con la variable `width`.

Llevaremos a cabo los 4 pasos explicados con anterioridad: Para el cálculo del número de intervalos, determinación de la amplitud, de los extremos y las marcas de clase.

Solución En primer lugar, cargamos los datos en un data frame:

```
crabs = read.table("r-basic-master/data/datacrab.txt", header = TRUE)
str(crabs)
```

```
## 'data.frame':    173 obs. of  6 variables:
##  $ input : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ color : int  3 4 2 4 4 3 2 4 3 4 ...
##  $ spine : int  3 3 1 3 3 3 1 2 1 3 ...
##  $ width : num  28.3 22.5 26 24.8 26 23.8 26.5 24.7 23.7 25.6 ...
##  $ satell: int  8 0 9 0 4 0 0 0 0 0 ...
##  $ weight: int  3050 1550 2300 2100 2600 2100 2350 1900 1950 2150 ...
```

```
cw = crabs$width
```

A continuación, definimos la variable `cw` que contiene los datos de la variable `width`.

Calculemos el número de clases según las diferentes reglas que hemos visto:

- Regla de la raíz cuadrada:

```
n = length(cw)
k1 = ceiling(sqrt(n))
k1
```

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

- Regla de Sturges:

```
k2 = ceiling(1+log(n,2))
k2
```

```
## [1] 9
```

- Regla de Scott:

```
As = 3.5*sd(cw)*n^(-1/3) #Amplitud teórica
k3 = ceiling(diff(range(cw))/As)
k3
```

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

- Regla de Freedman-Diaconis:

```
#Amplitud teórica
Afd = 2*(quantile(cw,0.75, names = FALSE)-quantile(cw,0.25,names = FALSE))*n^(-1/3)
k4 = ceiling(diff(range(cw))/Afd)
k4
```

```
## [1] 13
```

Podemos comprobar nuestros 3 últimos resultados con `R`:

```
nclass.Sturges(cw)
```

```
## [1] 9
```

```
nclass.scott(cw)
```

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

```
nclass.FD(cw)
```

```
## [1] 13
```

De momento, vamos a seguir la Regla de Scott. Es decir, vamos a considerar 10 intervalos.

A continuación, debemos elegir la amplitud de los intervalos.

```
A = diff(range(cw)) / 10
A
```

```
## [1] 1.25
```

Como nuestros datos están expresados en mm con una precisión de una cifra decimal, debemos redondear por exceso a un cifra decimal el resultado obtenido. Por lo tanto, nuestra amplitud será de

```
A = 1.3
```

Recordemos que si el cociente nos hubiera dado un valor exacto con respecto a la precisión, tendríamos que haberle sumado una unidad de precisión.

Solución Ahora nos toca calcular los extremos $L_1, \ldots, L_{11}$ de los intervalos.

Recordemos que nuestros intervalos tendrán la siguiente forma:

$$[L_1, L_2), \ \ldots, \ [L_{10}, L_{11})$$

Calculamos el primer extremo:

```
L1 = min(cw)-1/2*0.1
L1
```

```
## [1] 20.95
```

donde 0.1 es nuestra precisión (décimas de unidad, en este caso) Y, el resto de extremos se calculan del siguiente modo:

```
L2 = L1 + A
L3 = L2 + A
L4 = L3 + A
L5 = L4 + A
L6 = L5 + A
L7 = L6 + A
L8 = L7 + A
L9 = L8 + A
L10 = L9 + A
L11 = L10 + A
L = c(L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11)
L
```

```
##  [1] 20.95 22.25 23.55 24.85 26.15 27.45 28.75 30.05 31.35 32.65 33.95
```

O bien, si queremos facilitarnos el trabajo, también los podemos calcular mucho más rápido del siguiente modo:

```
L = L1 + A*(0:10)
L
```

```
##  [1] 20.95 22.25 23.55 24.85 26.15 27.45 28.75 30.05 31.35 32.65 33.95
```

Así, nuestros intervalos serán los siguientes:

$$[20.95, 22.25), \ [22.25, 23.55), \ [23.55, 24.85), \ [24.85, 26.15), \ [26.15, 27.45),$$
$$[27.45, 28.75), \ [28.75, 30.05), \ [30.05, 31.35), \ [31.35, 32.65), \ [32.65, 33.95)$$

Y hemos llegado al úlitmo paso: calcular las marcas de clase.

Recordemos que $X_i = \frac{L_i + L_{i+1}}{2} \quad \forall i = 1, \ldots, 10$

Empecemos calculando X_1

```
X1 = (L[1]+L[2])/2
X1
```

```
## [1] 21.6
```

Y, el resto de marcas de clase se calculan del siguiente modo:

```
X2 = X1 + A
X3 = X2 + A
X4 = X3 + A
X5 = X4 + A
X6 = X5 + A
X7 = X6 + A
X8 = X7 + A
X9 = X8 + A
X10 = X9 + A
X = c(X1,X2,X3,X4,X5,X6,X7,X8,X9,X10)
X
```

```
##  [1] 21.6 22.9 24.2 25.5 26.8 28.1 29.4 30.7 32.0 33.3
```

O bien, si queremos facilitarnos el trabajo, también los podemos calcular mucho más rápido como sucesión:

```
X = X1 + A*(0:9)
X
```

```
##  [1] 21.6 22.9 24.2 25.5 26.8 28.1 29.4 30.7 32.0 33.3
```

o también, como punto medio del intervalo

```
X = (L[1:length(L)-1]+L[2:length(L)])/2
X
```

```
##  [1] 21.6 22.9 24.2 25.5 26.8 28.1 29.4 30.7 32.0 33.3
```

Ejercicio Repetir este proceso para el número de clases obtenido con

- la regla de la raíz
- la regla de Sturges
- la regla de Freedman-Diaconis

Agrupando los datos con R

Al agrupar los datos, lo que hacemos es convertir nuestra variable cuantitativa en un factor cuyos niveles son las clases en que ha sido dividida e identificamos cada dato con su clase.

Cuando etiquetamos los niveles, podemos elegir 3 codificaciones:

- Los intervalos
- Las marcas de clase (el punto medio de cada intervalo)
- El número de orden de cada intervalo

La función cut Esta función es la básica en `R` para agrupar un vector de datos numéricos y codificar sus valores con clases a las que pertenecen.

Su sintaxis básica es

```
cut(x, breaks=..., labels=..., right=...)
```

- `x` es el vector numérico, nuestra variable cuantitativa
- `breaks` puede ser un vector numérico formado por los extremos de los intervalos en los que queremos agrupar nuestros datos y que habremos calculado previamente. También puede ser un número k, en cuyo caso `R` agrupa los datos en k clases. Para este caso, `R` divide el intervalo comprendido entre los valores mínimo y máximo de x en k intervalos y, a continuación, desplaza ligeramente el extremo inferior del primer intervalo a la izquierda y el extremo del último, a la derecha.
- `labels` es un vector con las etiquetas de los intervalos. Su valor por defecto es utilizar la etiqueta de los mismos intervalos. Si especificamos `labels = FALSE`, obtendremos los intervalos etiquetados por medio de los números naturales correlativos, empezando por 1. Para utilizar como etiqueta las marcas de clase o cualquier otra codificación, hay que entrarlo como valor de este parámetro.
- `right` es un parámetro que igualado a `FALSE` hace que los intervalos que consideremos sean cerrados por la izquierda y abiertos por la derecha. Este no es su valor por defecto.
- `include.lowest` igualdo a `TRUE` combinado con `right = FALSE` hace que el último intervalo sea cerrado. Puede ser útil en algunos casos.

En cualquier caso, el resultado de la función `cut` es una lista con los elementos del vector original codificados con las etiquetas de las clases a las que pertenecen. Bien puede ser un factor o un vector.

```
petals = iris$Petal.Length
cut(petals, breaks=5)
```

```
##   [1] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##   [6] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [11] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [16] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [21] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [26] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [31] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [36] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [41] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [46] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18] (0.994,2.18]
##  [51] (4.54,5.72]  (3.36,4.54]  (4.54,5.72]  (3.36,4.54]  (4.54,5.72]
##  [56] (3.36,4.54]  (4.54,5.72]  (2.18,3.36]  (4.54,5.72]  (3.36,4.54]
##  [61] (3.36,4.54]  (3.36,4.54]  (3.36,4.54]  (4.54,5.72]  (3.36,4.54]
##  [66] (3.36,4.54]  (3.36,4.54]  (3.36,4.54]  (3.36,4.54]  (3.36,4.54]
##  [71] (4.54,5.72]  (3.36,4.54]  (4.54,5.72]  (4.54,5.72]  (3.36,4.54]
##  [76] (3.36,4.54]  (4.54,5.72]  (4.54,5.72]  (3.36,4.54]  (3.36,4.54]
##  [81] (3.36,4.54]  (3.36,4.54]  (3.36,4.54]  (4.54,5.72]  (3.36,4.54]
##  [86] (3.36,4.54]  (4.54,5.72]  (3.36,4.54]  (3.36,4.54]  (3.36,4.54]
##  [91] (3.36,4.54]  (4.54,5.72]  (3.36,4.54]  (2.18,3.36]  (3.36,4.54]
##  [96] (3.36,4.54]  (3.36,4.54]  (3.36,4.54]  (2.18,3.36]  (3.36,4.54]
## [101] (5.72,6.91]  (4.54,5.72]  (5.72,6.91]  (4.54,5.72]  (5.72,6.91]
## [106] (5.72,6.91]  (3.36,4.54]  (5.72,6.91]  (5.72,6.91]  (5.72,6.91]
## [111] (4.54,5.72]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]
## [116] (4.54,5.72]  (4.54,5.72]  (5.72,6.91]  (5.72,6.91]  (4.54,5.72]
## [121] (4.54,5.72]  (4.54,5.72]  (5.72,6.91]  (4.54,5.72]  (4.54,5.72]
## [126] (5.72,6.91]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]  (5.72,6.91]
```

```
## [131] (5.72,6.91]  (5.72,6.91]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]
## [136] (5.72,6.91]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]
## [141] (4.54,5.72]  (4.54,5.72]  (4.54,5.72]  (5.72,6.91]  (4.54,5.72]
## [146] (4.54,5.72]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]  (4.54,5.72]
## Levels: (0.994,2.18] (2.18,3.36] (3.36,4.54] (4.54,5.72] (5.72,6.91]
```

```
cut(petals, breaks=ceiling(sqrt(length(petals))), right=FALSE)
```

```
##   [1] [0.994,1.45) [0.994,1.45) [0.994,1.45) [1.45,1.91)  [0.994,1.45)
##   [6] [1.45,1.91)  [0.994,1.45) [1.45,1.91)  [0.994,1.45) [1.45,1.91)
##  [11] [1.45,1.91)  [1.45,1.91)  [0.994,1.45) [0.994,1.45) [0.994,1.45)
##  [16] [1.45,1.91)  [0.994,1.45) [0.994,1.45) [1.45,1.91)  [1.45,1.91)
##  [21] [1.45,1.91)  [1.45,1.91)  [0.994,1.45) [1.45,1.91)  [1.45,1.91)
##  [26] [1.45,1.91)  [1.45,1.91)  [1.45,1.91)  [0.994,1.45) [1.45,1.91)
##  [31] [1.45,1.91)  [1.45,1.91)  [1.45,1.91)  [0.994,1.45) [1.45,1.91)
##  [36] [0.994,1.45) [0.994,1.45) [0.994,1.45) [0.994,1.45) [1.45,1.91)
##  [41] [0.994,1.45) [0.994,1.45) [0.994,1.45) [1.45,1.91)  [1.45,1.91)
##  [46] [0.994,1.45) [1.45,1.91)  [0.994,1.45) [1.45,1.91)  [0.994,1.45)
##  [51] [4.63,5.08)  [4.18,4.63)  [4.63,5.08)  [3.72,4.18)  [4.18,4.63)
##  [56] [4.18,4.63)  [4.63,5.08)  [3.27,3.72)  [4.18,4.63)  [3.72,4.18)
##  [61] [3.27,3.72)  [4.18,4.63)  [3.72,4.18)  [4.63,5.08)  [3.27,3.72)
##  [66] [4.18,4.63)  [4.18,4.63)  [3.72,4.18)  [4.18,4.63)  [3.72,4.18)
##  [71] [4.63,5.08)  [3.72,4.18)  [4.63,5.08)  [4.63,5.08)  [4.18,4.63)
##  [76] [4.18,4.63)  [4.63,5.08)  [4.63,5.08)  [4.18,4.63)  [3.27,3.72)
##  [81] [3.72,4.18)  [3.27,3.72)  [3.72,4.18)  [5.08,5.54)  [4.18,4.63)
##  [86] [4.18,4.63)  [4.63,5.08)  [4.18,4.63)  [3.72,4.18)  [3.72,4.18)
##  [91] [4.18,4.63)  [4.18,4.63)  [3.72,4.18)  [3.27,3.72)  [4.18,4.63)
##  [96] [4.18,4.63)  [4.18,4.63)  [4.18,4.63)  [2.82,3.27)  [3.72,4.18)
## [101] [5.99,6.45)  [5.08,5.54)  [5.54,5.99)  [5.54,5.99)  [5.54,5.99)
## [106] [6.45,6.91)  [4.18,4.63)  [5.99,6.45)  [5.54,5.99)  [5.99,6.45)
## [111] [5.08,5.54)  [5.08,5.54)  [5.08,5.54)  [4.63,5.08)  [5.08,5.54)
## [116] [5.08,5.54)  [5.08,5.54)  [6.45,6.91)  [6.45,6.91)  [4.63,5.08)
## [121] [5.54,5.99)  [4.63,5.08)  [6.45,6.91)  [4.63,5.08)  [5.54,5.99)
## [126] [5.99,6.45)  [4.63,5.08)  [4.63,5.08)  [5.54,5.99)  [5.54,5.99)
## [131] [5.99,6.45)  [5.99,6.45)  [5.54,5.99)  [5.08,5.54)  [5.54,5.99)
## [136] [5.99,6.45)  [5.54,5.99)  [5.08,5.54)  [4.63,5.08)  [5.08,5.54)
## [141] [5.54,5.99)  [5.08,5.54)  [5.08,5.54)  [5.54,5.99)  [5.54,5.99)
## [146] [5.08,5.54)  [4.63,5.08)  [5.08,5.54)  [5.08,5.54)  [5.08,5.54)
## 13 Levels: [0.994,1.45) [1.45,1.91) [1.91,2.36) [2.36,2.82) ... [6.45,6.91)
```

```
cut(petals, breaks=c(1,2,3,4,5,6,7), right=FALSE)
```

```
##   [1] [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2)
##  [13] [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2)
##  [25] [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2)
##  [37] [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2) [1,2)
##  [49] [1,2) [1,2) [4,5) [4,5) [4,5) [4,5) [4,5) [4,5) [4,5) [3,4) [4,5) [3,4)
##  [61] [3,4) [4,5) [4,5) [4,5) [3,4) [4,5) [4,5) [4,5) [4,5) [3,4) [4,5) [4,5)
##  [73] [4,5) [4,5) [4,5) [4,5) [4,5) [5,6) [4,5) [3,4) [3,4) [3,4) [3,4) [5,6)
##  [85] [4,5) [4,5) [4,5) [4,5) [4,5) [4,5) [4,5) [4,5) [4,5) [3,4) [4,5) [4,5)
##  [97] [4,5) [4,5) [3,4) [4,5) [6,7) [5,6) [5,6) [5,6) [5,6) [6,7) [4,5) [6,7)
## [109] [5,6) [6,7) [5,6) [5,6) [5,6) [5,6) [5,6) [5,6) [5,6) [6,7) [6,7) [5,6)
## [121] [5,6) [4,5) [6,7) [4,5) [5,6) [6,7) [4,5) [4,5) [5,6) [5,6) [6,7) [6,7)
```

```
## [133] [5,6) [5,6) [5,6) [6,7) [5,6) [5,6) [4,5) [5,6) [5,6) [5,6) [5,6) [5,6)
## [145] [5,6) [5,6) [5,6) [5,6) [5,6) [5,6)
## Levels: [1,2) [2,3) [3,4) [4,5) [5,6) [6,7)
```

Frecuencias

Una primera consideración es tratar las clases obtenidas como los niveles de una variable ordinal y calcular sus frecuencias.

- La frecuencia absoluta de una clase será el número de datos originales que pertenecen a la clase
- La frecuencia absoluta acumulada de una clase será el número de datos que pertenecen a dicha clase o alguna de las anteriores

Tabla de frecuencias

Normalmente, las frecuencias de un conjunto de datos agrupados se suele representar de la siguiente forma

Intervalos	X_j	n_j	N_j	f_j	F_j
$[L_1, L_2)$	X_1	n_1	N_1	f_1	F_1
$[L_2, L_3)$	X_2	n_2	N_2	f_2	F_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$[L_k, L_{k+1})$	X_k	n_k	N_k	f_k	F_k

El cálculo de las frecuencias con `R` podemos hacerlo mediante las funciones `table`, `prop.table` y `cumsum`.

También podemos utilizar la función `hist`, que internamente genera una lista cuya componente `count` es el vector de frecuencias absolutas de las clases. Por consiguiente, para calcular estas frecuencias, podemos utilizar la sintaxis

```
hist(x, breaks=..., right=FALSE, plot=FALSE)$count
```

Conviene igualar el parámetro `breaks` al vector de los extremos del intervalo debido a que `cut` y `hist` hacen uso de diferentes métodos para agrupar los datos cuando se especifica solamente el número k de clases.

El resultado de `hist` incluye la componente `mids` que contiene el vector de puntos medios de los intervalos, es decir, nuestras marcas de clase.

Tabla de frecuencias con R

Podemos automatizar el cálculo de la ya tan mencionada tabla de frecuencias, utilizando las dos funciones que mostramos a continuación.

La primera sirve en el caso en que vayamos a tomar todas las clases de la misma amplitud. Sus parámetros son: x, el vector con los datos cuantitativos; k, el número de clases; A, su amplitud; y p, la precisión de los datos ($p = 1$ si la precisión son unidades, $p = 0.1$ si la precisión son décimas de unidad...).

Por su parte, la segunda es para cuando conocemos los extremos de las clases. Sus parámetros son: x, el vector con los datos cuantitativos; L, el vector de extremos de clases; y V , un valor lógico, que ha de ser `TRUE` si queremos que el último intervalo sea cerrado, y `FALSE` en caso contrario.

```r
#Primera función
TablaFrecs = function(x,k,A,p){
  L = min(x)-p/2+A*(0:k)
  x_cut = cut(x, breaks = L, right=FALSE)
  intervals = levels(x_cut)
  mc = (L[1]+L[2])/2+A*(0:(k-1))
  Fr.abs = as.vector(table(x_cut))
  Fr.rel = round(Fr.abs/length(x),4)
  Fr.cum.abs = cumsum(Fr.abs)
  Fr.cum.rel = cumsum(Fr.rel)
  tabla = data.frame(intervals, mc, Fr.abs, Fr.cum.abs, Fr.rel, Fr.cum.rel)
  tabla
  }
```

```r
TablaFrecs.L = function(x,L,V){
  x_cut = cut(x, breaks=L, right=FALSE, include.lowest=V)
  intervals = levels(x_cut)
  mc = (L[1:(length(L)-1)]+L[2:length(L)])/2
  Fr.abs = as.vector(table(x_cut))
  Fr.rel = round(Fr.abs/length(x),4)
  Fr.cum.abs = cumsum(Fr.abs)
  Fr.cum.rel = cumsum(Fr.rel)
  tabla = data.frame(intervals, mc, Fr.abs, Fr.cum.abs, Fr.rel, Fr.cum.rel)
  tabla
  }
```

Ejemplo 2 **Ejemplo 2**

Siguiendo con el ejemplo de las anchuras de los cangrejos, vamos a calcular sus tablas de frecuencias haciendo uso de todo lo aprendido anteriormente.

Solución La tabla queda del siguiente modo:

Intervalos	X_j	n_j	N_j	f_j	F_j
[20.95, 22.25)	21.6	2	2	0.0116	0.0116
[22.25, 23.55)	22.9	14	16	0.0809	0.0925
[23.55, 24.85)	24.2	27	43	0.1561	0.2486
[24.85, 26.15)	25.5	44	87	0.2543	0.5029
[26.15, 27.45)	26.8	34	121	0.1965	0.6994
[27.45, 28.75)	28.1	31	152	0.1792	0.8786

Intervalos	X_j	n_j	N_j	f_j	F_j
[28.75, 30.05)	29.4	15	167	0.0867	0.9653
[30.05, 31.35)	30.7	3	170	0.0173	0.9826
[31.35, 32.65)	32	2	172	0.0116	0.9942
[32.65, 33.95)	33.3	1	173	0.0058	1

Y, ahora, lo haremos con las funciones que hemos proporcionado:

```
TablaFrecs(cw,10,1.3,0.1)
```

```
##      intervals   mc Fr.abs Fr.cum.abs Fr.rel Fr.cum.rel
## 1  [20.9,22.2) 21.6      2          2 0.0116     0.0116
## 2  [22.2,23.6) 22.9     14         16 0.0809     0.0925
## 3  [23.6,24.9) 24.2     27         43 0.1561     0.2486
## 4  [24.9,26.1) 25.5     44         87 0.2543     0.5029
## 5  [26.1,27.4) 26.8     34        121 0.1965     0.6994
## 6  [27.4,28.8) 28.1     31        152 0.1792     0.8786
## 7    [28.8,30) 29.4     15        167 0.0867     0.9653
## 8    [30,31.4) 30.7      3        170 0.0173     0.9826
## 9  [31.4,32.6) 32.0      2        172 0.0116     0.9942
## 10   [32.6,34) 33.3      1        173 0.0058     1.0000
```

```
TablaFrecs.L(cw,L,FALSE)
```

```
##      intervals   mc Fr.abs Fr.cum.abs Fr.rel Fr.cum.rel
## 1  [20.9,22.2) 21.6      2          2 0.0116     0.0116
## 2  [22.2,23.6) 22.9     14         16 0.0809     0.0925
## 3  [23.6,24.9) 24.2     27         43 0.1561     0.2486
## 4  [24.9,26.1) 25.5     44         87 0.2543     0.5029
## 5  [26.1,27.4) 26.8     34        121 0.1965     0.6994
## 6  [27.4,28.8) 28.1     31        152 0.1792     0.8786
## 7    [28.8,30) 29.4     15        167 0.0867     0.9653
## 8    [30,31.4) 30.7      3        170 0.0173     0.9826
## 9  [31.4,32.6) 32.0      2        172 0.0116     0.9942
## 10   [32.6,34) 33.3      1        173 0.0058     1.0000
```

Veamos que los intervalos no terminan de ser los que hemos calculado nosotros, pero eso se debe a como funciona la función `cut`.

Ejemplo 3

Enunciado Ejemplo 3

Se han recogido las notas de un examen de historia a los 100 alumnos de primero de bachillerato de un instituto.

Vamos a hacer uso de todo lo aprendido para obtener la mayor información posible utilizando las funciones `cut` e `hist` y también, las proporcionadas por nosotros.

Los resultados obtenidos en la encuesta han sido:

```
notas
```

```
##   [1]  7 10  2  2  6  2  5  4  9  2  7  5  1  7  0  3 10  2 10  4  1  4  5  4  0
##  [26]  5 10  4  3  0  7  5 10  3  4  8  1  9  3  7  9  1  9 10  5 10 10  9  5  0
##  [51]  3  1  3  2  0  6  6  4  7  4  7  3  9  0  7  0  3  0  3  3  1  4 10  9  1
##  [76]  4  0  6 10  0 10  1  0  2  6  4  8  2  3  7  7  3  3  8  2  6  6  2  8  9
```

Vamos a agrupar las notas en los siguientes intervalos:

$$[0,5),\ [5,7),\ [7,9),\ [9,10]$$

Claramente, estos 4 intervalos no tienen la misma amplitud.

Fijémonos también en que el último intervalo está cerrado por la derecha.

```
#Definimos vector de extremos
L = c(0,5,7,9,10)
#Definimos notas1 como el resultado de la codificación en intervalos utilizando como
#etiquetas los propios intervalos
notas1 = cut(notas, breaks = L, right = FALSE, include.lowest = TRUE)
## break sob los intervalos
## right no se cierra por la derecha
## include.lowest = TRUE Se incluye el mas bajo de los numeros Asi queda cerrado por la  izquierda
##
notas1 ## se representa en factor de 4 niveles
```

```
##   [1] [7,9)  [9,10] [0,5)  [0,5)  [5,7)  [0,5)  [5,7)  [0,5)  [9,10] [0,5)
##  [11] [7,9)  [5,7)  [0,5)  [7,9)  [0,5)  [0,5)  [9,10] [0,5)  [9,10] [0,5)
##  [21] [0,5)  [0,5)  [5,7)  [0,5)  [0,5)  [5,7)  [9,10] [0,5)  [0,5)  [0,5)
##  [31] [7,9)  [5,7)  [9,10] [0,5)  [0,5)  [7,9)  [0,5)  [9,10] [0,5)  [7,9)
##  [41] [9,10] [0,5)  [9,10] [9,10] [5,7)  [9,10] [9,10] [9,10] [5,7)  [0,5)
##  [51] [0,5)  [0,5)  [0,5)  [0,5)  [0,5)  [5,7)  [5,7)  [0,5)  [7,9)  [0,5)
##  [61] [7,9)  [0,5)  [9,10] [0,5)  [7,9)  [0,5)  [0,5)  [0,5)  [0,5)  [0,5)
##  [71] [0,5)  [0,5)  [9,10] [9,10] [0,5)  [0,5)  [0,5)  [5,7)  [9,10] [0,5)
##  [81] [9,10] [0,5)  [0,5)  [0,5)  [5,7)  [0,5)  [7,9)  [0,5)  [0,5)  [7,9)
##  [91] [7,9)  [0,5)  [0,5)  [7,9)  [0,5)  [5,7)  [5,7)  [0,5)  [7,9)  [9,10]
## Levels: [0,5) [5,7) [7,9) [9,10]
```

```
#Definimos las marcas de clase
MC = (L[1:length(L)-1]+L[2:length(L)])/2
#Definimos notas2 como el resultado de la codificación en intervalos utilizando como
#etiquetas las marcas de clase
notas2 = cut(notas, breaks = L, labels = MC, right = FALSE, include.lowest = TRUE)
notas2 ## salen las marcas de clase
```

```
##   [1] 8   9.5 2.5 2.5 6   2.5 6   2.5 9.5 2.5 8   6   2.5 8   2.5 2.5 9.5 2.5
##  [19] 9.5 2.5 2.5 2.5 6   2.5 2.5 6   9.5 2.5 2.5 2.5 8   6   9.5 2.5 2.5 8
##  [37] 2.5 9.5 2.5 8   9.5 2.5 9.5 9.5 6   9.5 9.5 9.5 6   2.5 2.5 2.5 2.5 2.5
##  [55] 2.5 6   6   2.5 8   2.5 8   2.5 9.5 2.5 8   2.5 2.5 2.5 2.5 2.5 2.5 2.5
##  [73] 9.5 9.5 2.5 2.5 2.5 6   9.5 2.5 9.5 2.5 2.5 2.5 6   2.5 8   2.5 2.5 8
##  [91] 8   2.5 2.5 8   2.5 6   6   2.5 8   9.5
## Levels: 2.5 6 8 9.5
```

```
#Definimos notas3 como el resultado de la codificación en intervalos utilizando como
#etiquetas la posición ordenada del intervalo (1, 2, 3 o 4)
notas3 = cut(notas, breaks = L, labels = FALSE, right = FALSE, include.lowest = TRUE)
notas3 ## Se representa con un xaracter ordinal es la posicion del intervalo que esta la nota
```

```
##   [1] 3 4 1 1 2 1 2 1 4 1 3 2 1 3 1 1 4 1 4 1 1 1 2 1 1 2 4 1 1 1 3 2 4 1 1 3 1
##  [38] 4 1 3 4 1 4 4 2 4 4 4 2 1 1 1 1 1 1 2 2 1 3 1 3 1 4 1 3 1 1 1 1 1 1 1 4 4
##  [75] 1 1 1 2 4 1 4 1 1 1 2 1 3 1 1 3 3 1 1 3 1 2 2 1 3 4
```

```
#Definimos notas4 como el resultado de la codificación en intervalos utilizando como
#etiquetas Susp, Aprob, Not y Exc
notas4 = cut(notas, breaks = L, labels = c("Susp", "Aprob", "Not", "Exc"), right = FALSE, include.lowes
notas4
```

```
##   [1] Not   Exc   Susp  Susp  Aprob Susp  Aprob Susp  Exc   Susp  Not   Aprob
##  [13] Susp  Not   Susp  Susp  Exc   Susp  Exc   Susp  Susp  Susp  Aprob Susp
##  [25] Susp  Aprob Exc   Susp  Susp  Susp  Not   Aprob Exc   Susp  Susp  Not
##  [37] Susp  Exc   Susp  Not   Exc   Susp  Exc   Exc   Aprob Exc   Exc   Exc
##  [49] Aprob Susp  Susp  Susp  Susp  Susp  Susp  Aprob Aprob Susp  Not   Susp
##  [61] Not   Susp  Exc   Susp  Not   Susp  Susp  Susp  Susp  Susp  Susp  Susp
##  [73] Exc   Exc   Susp  Susp  Susp  Aprob Exc   Susp  Exc   Susp  Susp  Susp
##  [85] Aprob Susp  Not   Susp  Susp  Not   Not   Susp  Susp  Not   Susp  Aprob
##  [97] Aprob Susp  Not   Exc
## Levels: Susp Aprob Not Exc
```

El resultado de `cut` ha sido, en cada caso, una lista con los elementos del vector original codificados con las etiquetas de las clases a las que pertenecen.

Las dos primeras aplicaciones de la función `cut` han producido factores (cuyos niveles son los intervalos y las marcas de clase, respectivamente, en ambos casos ordenados de manera natural), mientras que aplicándole `labels = FALSE` hemos obtenido un vector.

¿Qué habría ocurrido si le hubiéramos pedido a R que cortase los datos en 4 intervalos?

Pues en este caso no nos hubiera servido de mucho, sobre todo porque la amplitud de nuestros intervalos era, desde buen inicio, diferente.

```
cut(notas, breaks = 4, right = FALSE, include.lowest = TRUE)
```

```
##   [1] [5,7.5)      [7.5,10]     [-0.01,2.5) [-0.01,2.5) [5,7.5)      [-0.01,2.5)
##   [7] [5,7.5)      [2.5,5)      [7.5,10]    [-0.01,2.5) [5,7.5)      [5,7.5)
##  [13] [-0.01,2.5) [5,7.5)      [-0.01,2.5) [2.5,5)     [7.5,10]    [-0.01,2.5)
##  [19] [7.5,10]    [2.5,5)      [-0.01,2.5) [2.5,5)     [5,7.5)     [2.5,5)
##  [25] [-0.01,2.5) [5,7.5)      [7.5,10]    [2.5,5)     [2.5,5)     [-0.01,2.5)
##  [31] [5,7.5)     [5,7.5)      [7.5,10]    [2.5,5)     [2.5,5)     [7.5,10]
##  [37] [-0.01,2.5) [7.5,10]     [2.5,5)     [5,7.5)     [7.5,10]    [-0.01,2.5)
##  [43] [7.5,10]    [7.5,10]     [5,7.5)     [7.5,10]    [7.5,10]    [7.5,10]
##  [49] [5,7.5)     [-0.01,2.5) [2.5,5)     [-0.01,2.5) [2.5,5)     [-0.01,2.5)
##  [55] [-0.01,2.5) [5,7.5)      [5,7.5)     [2.5,5)     [5,7.5)     [2.5,5)
##  [61] [5,7.5)     [2.5,5)      [7.5,10]    [-0.01,2.5) [5,7.5)     [-0.01,2.5)
##  [67] [2.5,5)     [-0.01,2.5) [2.5,5)     [2.5,5)     [-0.01,2.5) [2.5,5)
##  [73] [7.5,10]    [7.5,10]     [-0.01,2.5) [2.5,5)     [-0.01,2.5) [5,7.5)
##  [79] [7.5,10]    [-0.01,2.5) [7.5,10]    [-0.01,2.5) [-0.01,2.5) [-0.01,2.5)
##  [85] [5,7.5)     [2.5,5)      [7.5,10]    [-0.01,2.5) [2.5,5)     [5,7.5)
##  [91] [5,7.5)     [2.5,5)      [2.5,5)     [7.5,10]    [-0.01,2.5) [5,7.5)
##  [97] [5,7.5)     [-0.01,2.5) [7.5,10]    [7.5,10]
## Levels: [-0.01,2.5) [2.5,5) [5,7.5) [7.5,10]
```

R ha repartido los datos en 4 intervalos de longitud 2.5, y ha desplazado ligeramente a la izquierda el extremo izquierdo del primer intervalo.

Trabajaremos ahora con `notas4` y calcularemos sus frecuencias:

```r
table(notas4) #Fr. Abs
```

```
## notas4
##  Susp Aprob   Not   Exc
##    53    14    14    19
```

```r
prop.table(table(notas4)) #Fr. Rel
```

```
## notas4
##  Susp Aprob   Not   Exc
##  0.53  0.14  0.14  0.19
```

```r
cumsum(table(notas4)) #Fr. Abs. Cum
```

```
##  Susp Aprob   Not   Exc
##    53    67    81   100
```

```r
cumsum(prop.table(table(notas4))) #Fr. Rel. Cum
```

```
##  Susp Aprob   Not   Exc
##  0.53  0.67  0.81  1.00
```

Podríamos haber obtenido todo lo anterior haciendo uso de la función `hist`.

```r
notasHist = hist(notas, breaks = L, right = FALSE, include.lowest = TRUE, plot = FALSE)
FAbs = notasHist$count
FRel = prop.table(FAbs)
FAbsCum = cumsum(FAbs)
FRelCum = cumsum(FRel)
```

Podemos crear un data frame con todas estas frecuencias:

```r
intervalos = c("[0,5)","[5,7)","[7,9)","[9,10]")
calificacion = c("Suspenso", "Aprobado", "Notable", "Excelente")
marcas = notasHist$mids
tabla.Fr = data.frame(intervalos,calificacion,marcas,FAbs,FAbsCum,FRel,FRelCum)
tabla.Fr
```

```
##   intervalos calificacion marcas FAbs FAbsCum FRel FRelCum
## 1      [0,5)     Suspenso    2.5   53      53 0.53    0.53
## 2      [5,7)     Aprobado    6.0   14      67 0.14    0.67
## 3      [7,9)      Notable    8.0   14      81 0.14    0.81
## 4     [9,10]    Excelente    9.5   19     100 0.19    1.00
```

O bien, podríamos haber utilizado las funciones que hemos proporcionado:

```r
TablaFrecs.L(notas, L, TRUE)
```

```
##   intervals  mc Fr.abs Fr.cum.abs Fr.rel Fr.cum.rel
## 1     [0,5) 2.5     53         53   0.53       0.53
## 2     [5,7) 6.0     14         67   0.14       0.67
## 3     [7,9) 8.0     14         81   0.14       0.81
## 4    [9,10] 9.5     19        100   0.19       1.00
```

Al tener una muestra de datos numéricos, conviene calcular los estadísticos antes de realizar los agrupamientos, puesto que de lo contrario podemos perder información.

Hay situaciones en que los datos los obtenemos ya agrupados. En estos casos, aún sigue siendo posible calcular los estadísticos y utilizarlos como aproximaciones de los estadísticos de los datos "reales", los cuales no conocemos.

La media $\bar{x}$, la varianza, s^2, la varianza muestral, $\tilde{s}^2$, la desviación típica, s, y la desviación típica muestral, $\tilde{s}$ de un conjunto de datos agrupados se calculan mediante las mismas fórmulas que para los datos no agrupados con la única diferencia de que sustituimos cada clase por su marca de clase y la contamos con su frecuencia.

Es decir, si tenemos k clases, con sus respectivas marcas $X_1, \ldots, X_k$ con frecuencias absolutas $n_1, \ldots, n_k$ de forma que $n = \sum_{j=1}^{k} n_j$. Entonces

$$\bar{x} = \frac{\sum_{j=1}^{k} n_j X_j}{n}, \quad s^2 = \frac{\sum_{j=1}^{k} n_j X_j^2}{n} - \bar{x}^2, \quad \tilde{s}^2 = \frac{n}{n-1} \cdot s^2$$
$$s = \sqrt{s^2}, \quad \tilde{s} = \sqrt{\tilde{s}^2}$$

Intervalo modal En lo referente a la moda, esta se sustituye por el intervalo modal, que es la clase con mayor frecuencia (absoluta o relativa).

En el caso en que un valor numérico fuera necesario, se tomaría su marca de clase.

Intervalo crítico para la mediana Se conoce como intervalo crítico para la mediana, $[L_c, L_{c+1})$, al primer intervalo donde la frecuencia relativa acumulada sea mayor o igual que 0.5

Denotemos por n_c su frecuencia absoluta, por $A_c = L_{c+1} - L_c$ su amplitud y por N_{c-1} la frecuencia acumalada del intervalo inmediantamente anterior (en caso de ser $[L_c, L_{c+1}) = [L_1, L_2)$, entonces $N_{c-1} = 0$). Y por lo tanto, M será una aproximación para la mediana de los datos "reales" a partir de los agrupados

$$M = L_c + A_c \cdot \frac{\frac{n}{2} - N_{c-1}}{n_c}$$

Aproximación de los cuantiles

La fórmula anterior nos permite aproximar el cuantil Q_p de los datos "reales" a partir de los datos agrupados:

$$Q_p = L_p + A_p \cdot \frac{p \cdot n - N_{p-1}}{n_p}$$

donde el intervalo $[L_p, L_{p+1})$ denota el primer intervalo cuya frecuencia relativa acumulada es mayor o igual a p

####Ejemplo 2

Vamos a seguir trabajando con nuestra variable `cw`, esta vez, lo que haremos será calcular los estadísticos de la variable con los datos agrupados y, estimaremos la mediana y algunos cuantiles.

Solución Recordemos todo lo que habíamos obtenido sobre nuestra variable `cw`:

```
##  [1] 20.95 22.25 23.55 24.85 26.15 27.45 28.75 30.05 31.35 32.65 33.95
```

```
##        intervals   mc Fr.abs Fr.cum.abs Fr.rel Fr.cum.rel
## 1  [20.95,22.25) 21.6      2          2 0.0116     0.0116
## 2  [22.25,23.55) 22.9     14         16 0.0809     0.0925
## 3  [23.55,24.85) 24.2     27         43 0.1561     0.2486
## 4  [24.85,26.15) 25.5     44         87 0.2543     0.5029
## 5  [26.15,27.45) 26.8     34        121 0.1965     0.6994
## 6  [27.45,28.75) 28.1     31        152 0.1792     0.8786
## 7  [28.75,30.05) 29.4     15        167 0.0867     0.9653
## 8  [30.05,31.35) 30.7      3        170 0.0173     0.9826
## 9  [31.35,32.65) 32.0      2        172 0.0116     0.9942
## 10 [32.65,33.95) 33.3      1        173 0.0058     1.0000
```

Ahora ya podemos calcular los estadísticos:

```
TOT = tabla$Fr.cum.abs[10] ## accedo a la ultima fila de la tabla anterior
TOT
```

```
## [1] 173
```

```
anchura.media = round(sum(tabla$Fr.abs*tabla$mc)/TOT,3)
anchura.media #Media
```

```
## [1] 26.312
```

```
anchura.var = round(sum(tabla$Fr.abs*tabla$mc^2)/TOT-anchura.media^2,3)
anchura.var #Varianza
```

```
## [1] 4.476
```

```
anchura.dt = round(sqrt(anchura.var),3)
anchura.dt #Desviación típica
```

```
## [1] 2.116
```

```
I.modal = tabla$intervals[which(tabla$Fr.abs == max(tabla$Fr.abs))]
I.modal #Intervalo modal
```

```
## [1] "[24.85,26.15)"
```

Por lo tanto, con los datos de los que disponemos, podemos afirmar que la anchura media de los cangrejos de la muestra es de 26.312mm, con una desviación típica de unos 4.476mm, y que el grupo de anchuras más numeroso era el de [24.85,26.15).

calculemos el intervalo crítico para la mediana.

```r
I.critic = tabla$intervals[which(tabla$Fr.cum.rel >= 0.5)]
I.critic[1] #Intervalo critic
```

```
## [1] "[24.85,26.15)"
```

Ahora, ya podemos calcular una estimación de la mediana de los datos "reales".

```r
n = TOT
Lc = L[4]
Lc.pos = L[5]
Ac = L[5]-L[4]
Nc.ant = tabla$Fr.cum.abs[3]
nc = tabla$Fr.abs[4]
M = Lc+Ac*((n/2)-Nc.ant)/nc
M #Aproximación de la mediana de los datos "reales"
```

```
## [1] 26.13523
```

```r
median(cw) #Mediana de los datos "reales"
```

```
## [1] 26.1
```

También podemos hacer aproximaciones de los cuantiles. Hemos creado una función `aprox.quantile.p` para no tener que copiar la operación cada vez que queramos calcular un cuantil aproximado.

```r
aprox.quantile.p = function(Lcrit,Acrit,n,p,Ncrit.ant,ncrit){
  round(Lcrit+Acrit*(p*n-Ncrit.ant)/ncrit,3)
}
aprox.quantile.p(Lc,Ac,n,0.25,Nc.ant,nc) #Primer cuartil
```

```
## [1] 24.857
```

```r
aprox.quantile.p(Lc,Ac,n,0.75,Nc.ant,nc) #Tercer cuartil
```

```
## [1] 27.413
```

Y ahora, calculemos los cuartiles de los datos "reales"

```r
quantile(cw,0.25)
```

```
##  25%
## 24.9
```

```r
quantile(cw,0.75)
```

```
##  75%
## 27.7
```

Histogramas

La mejor manera de representar datos agrupados es mediante unos diagramas de barras especiales conocidos como histogramas.

En ellos se dibuja sobre cada clase una barra cuya área representa su frecuencia. se puede comprobar que el producto de la base por la altura de cada barra es igual a la frecuencia de la clase correspondiente.

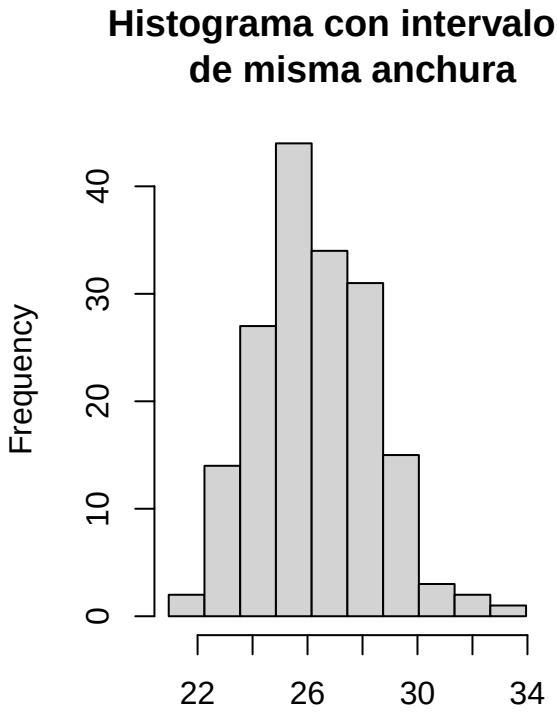
Si todas las clases tienen la misma amplitud, las alturas de estas barras son proporcionales a las frecuencias de sus clases, con lo cual podemos marcar sin ningún problema las frecuencias sobre el eje vertical. Pero si las amplitudes de las clases no son iguales, las alturas de las barras en un histograma no representan correctamente las frecuencias de las clases.

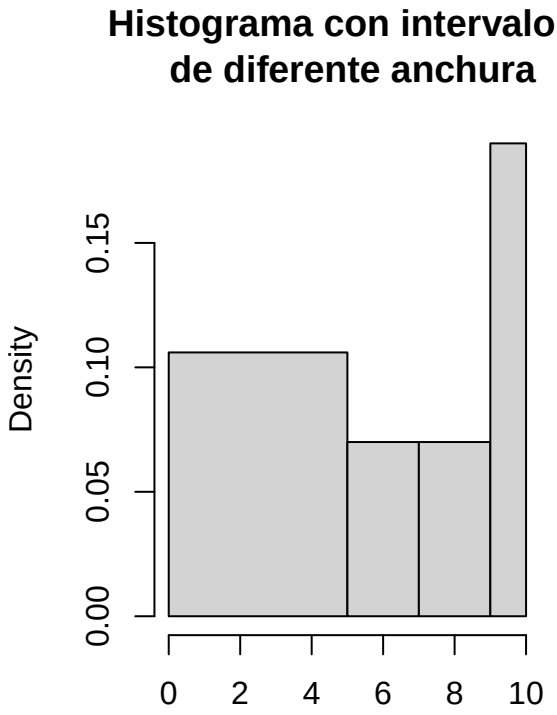
En este último caso, las alturas de las barras son las necesarias para que el área de cada barra sea igual a la frecuencia de la clase correspondiente y como las bases son de amplitudes diferentes, estas alturas no son proporcionales a las frecuencias de las clases, por lo que no tiene sentido marcar las frecuencias en el eje vertical

Los histogramas también son utilizados para representar frecuencias acumuladas de datos agrupados. En este caso, las alturas representan las frecuencias independientemente de la base debido a que éstas deben ir creciendo.

El eje de las abcisas representa los datos. Aquí marcamos los extremos de las clases y se dibuja una barra sobre cada una de ellas. Esta barra tiene significados diferentes en función del tipo de histograma, pero en general representa la frecuencia de su clase

- Histograma de frecuencias absolutas: la altura de cada barra es la necesaria para que el área de la barra sea igual a la frecuencia absoluta de la clase. Las amplitudes de las clases pueden ser todas iguales o no. En el primer caso, las alturas son proporcionales a las frecuencias. En el segundo caso, no existe tal proporcionalidad. De todas formas, sea cual sea el caso, conviene indicar de alguna forma la frecuencia que representa cada barra.





Interpretación de los histogramas

- Histograma de frecuencias relativas: la altura, densidad, de cada barra es la necesaria para que el área sea igual a la frecuencia relativa de la clase. La suma de todas las áreas debe ser 1. De nuevo, conviene indicar de alguna forma la frecuencia que representa cada barra.
- Histogramas de frecuencias acumuladas: las alturas de las barras son iguales a las frecuencias acumuladas de las clases, independientemente de su amplitud.

Frecuencias nulas No es conveniente que en un histograma aparezcan clases con frecuencia nula, exceptuando el caso en que represente poblaciones muy diferentes y separadas sin individuos intermedios.

Si apareciesen clases vacías, convendría utilizar un número menor de clases, o bien unir las clases vacías con alguna de sus adyacentes. De este último modo romperíamos nuestro modo de trabajar con clases de la misma amplitud.

Dibujando histogramas con R

Lo hacemos con la función `hist`, la cual ya conocemos. Su sintaxis es

```
hist(x, breaks=..., freq=..., right=..., ...)
```

- `x`: vector de los datos
- `breaks`: vector con los extremos de los intervalos o el número k de intervalos. Incluso podemos indicar, entre comillas, el método que deseemos para calcular el número de clases: `"Scott"`, `"Sturges"`... Eso

sí, para cualquiera de las dos últimas opciones, no siempre obtendréis el número deseado de intervalos, puesto que R lo considerará solo como sugerencia. Además, recordad que el método para calcular los intervalos es diferente al de la función `cut`. Por tanto, se recomienda hacer uso de la primera opción.
- `freq=TRUE`, que es su valor por defecto, produce el histograma de frecuencias absolutas si los intervalos son todos de la misma amplitud y de frecuencias relativas en caso contrario. `freq=FALSE` nos produce siempre el de frecuencias relativas.
- `right` funciona exactamente igual que en la función `cut`.
- `include.lowest = TRUE` también funciona exactamente igual que en la función `cut`.
- También podéis utilizar los parámetros de la función `plot` que tengan sentido

`hist` titula por defecto los histogramas del siguiente modo: "Histogram of" seguido del nombre del vector de datos. No suele quedar muy bien si no estamos haciendo nuestro análisis en inglés.

Recordemos que el parámetro `plot` igualado a `FALSE` no dibujaba, pero sí calculaba el histograma.

La función `hist` contiene mucha información en su estructura interna

- `breaks` contiene el vector de extremos de los intervalos: $L_1, \ldots, L_{k+1}$
- `mids` contiene los puntos medios de los intervalos, lo que nosotros consideramos las marcas de clase: $X_1, \ldots, X_k$
- `counts` contiene el vector de frecuencias absolutas de los intervalos: $n_1, \ldots, n_k$
- `density` contiene el vector de las densidades de los intervalos. Estas se corresponden con las alturas de las barras del histograma de frecuencias relativas. Recordemos, la densidad de un intervalo es su frecuencia relativa divida por su amplitud.

La siguiente función es útil para calcular histogramas de frecuencias absolutas más completas:

```
histAbs = function(x,L) {
  h = hist(x, breaks = L, right = FALSE, freq = FALSE,
           xaxt = "n", yaxt = "n", col = "lightgray",
           main = "Histograma de frecuencias absolutas",
           xlab = "Intervalos y marcas de clase",ylab = "Frecuencias absolutas")
  axis(1, at=L)
  text(h$mids, h$density/2, labels=h$counts, col="purple")
  }
```

- `xaxt="n"` e `yaxt="n"` especifican que, por ahora, la función no dibuje los ejes de abcisas y ordenadas, respectivamente.

Regresión Lineal

Introducción

Supongamos que tenemos una serie de puntos en el plano cartesiano $\mathbb{R}^2$, de la forma

$$(x_1, y_1),\ \ldots,\ (x_n, y_n)$$

que representan las observaciones de dos variables numéricas. Digamos que x es la edad e y el peso de n estudiantes.

Nuestro objetivo: describir la relación entre la variable independiente, x, y la variable dependiente, y, a partir de estas observaciones.

Para ello, lo que haremos será buscar una función $y = f(x)$ cuya gráfica se aproxime lo máximo posible a nuestros pares ordenados $(x_i, y_i)_{i=1,\ldots,n}$.

Esta función nos dará un modelo matemático de cómo se comportan estas observaciones, lo cual nos permitirá entender mejor los mecanismos que relacionan las variables estudiadas o incluso, nos dará la oportunidad de hacer predicccciones sobre futuras observaciones.

La primera opción es la más fácil. Consiste en estudiar si los puntos $(x_i, y_i)_{i=1,\dots,n}$ satisfacen una relación lineal de la forma

$$y = ax + b$$

con $a, b \in \mathbb{R}$.

En este caso, se busca la recta $y = ax + b$ que mejor aproxime los puntos dados imponiendo que la suma de los cuadrados de las diferencias entre los valores y_i y sus aproximaciones $\tilde{y}_i = ax_i + b$ sea mínima. Es decir, que

$$\sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

sea mínima

El objetivo de este tema no es otro más que como en R obtiene esta recta de regresión.

Veremos también cómo se puede evaluar numéricamente si esta recta se ajusta bien a las observaciones dadas.

Para ello, introduciremos algunas funciones de R y haremos uso de transformaciones logarítmicas para tratar casos en los que los puntos dados se aproximen mejor mediante una función exponencial o potencial.

Calculando rectas de regresión

Por lo general, cuando tenemos una serie de observaciones en parejas, $(x_i, y_i)_{i=1,\dots,n}$, la forma natural de almacenarlas en R es mediante una tabla de datos. Y la que más conocemos nostros es el data frame.

Ejemplo 1 Ejemplo 1

En este ejemplo, nosotros haremos uso del siguiente data frame:

```
body = read.table("r-basic-master/data/bodyfat.txt", header = TRUE)
head(body,3)
```

```
##   Density  Fat Age Weight Height Neck Chest Abdomen  Hip Thigh Knee Ankle
## 1  1.0708 12.3  23 154.25  67.75 36.2  93.1    85.2 94.5  59.0 37.3  21.9
## 2  1.0853  6.1  22 173.25  72.25 38.5  93.6    83.0 98.7  58.7 37.3  23.4
## 3  1.0414 25.3  22 154.00  66.25 34.0  95.8    87.9 99.2  59.6 38.9  24.0
##   Biceps Forearm Wrist
## 1   32.0    27.4  17.1
## 2   30.5    28.9  18.2
## 3   28.8    25.2  16.6
```

Más concretamente, trabajaremos con las variables `fat` y `weight`.

```
body2 = body[,c(2,4)]#nuevo data con columns peso y grasa
names(body2) = c("Grasa","Peso")  ##nobramos
str(body2)
```

```
## 'data.frame':    252 obs. of  2 variables:
##  $ Grasa: num  12.3 6.1 25.3 10.4 28.7 20.9 19.2 12.4 4.1 11.7 ...
##  $ Peso : num  154 173 154 185 184 ...
```

```
head(body2,3)
```

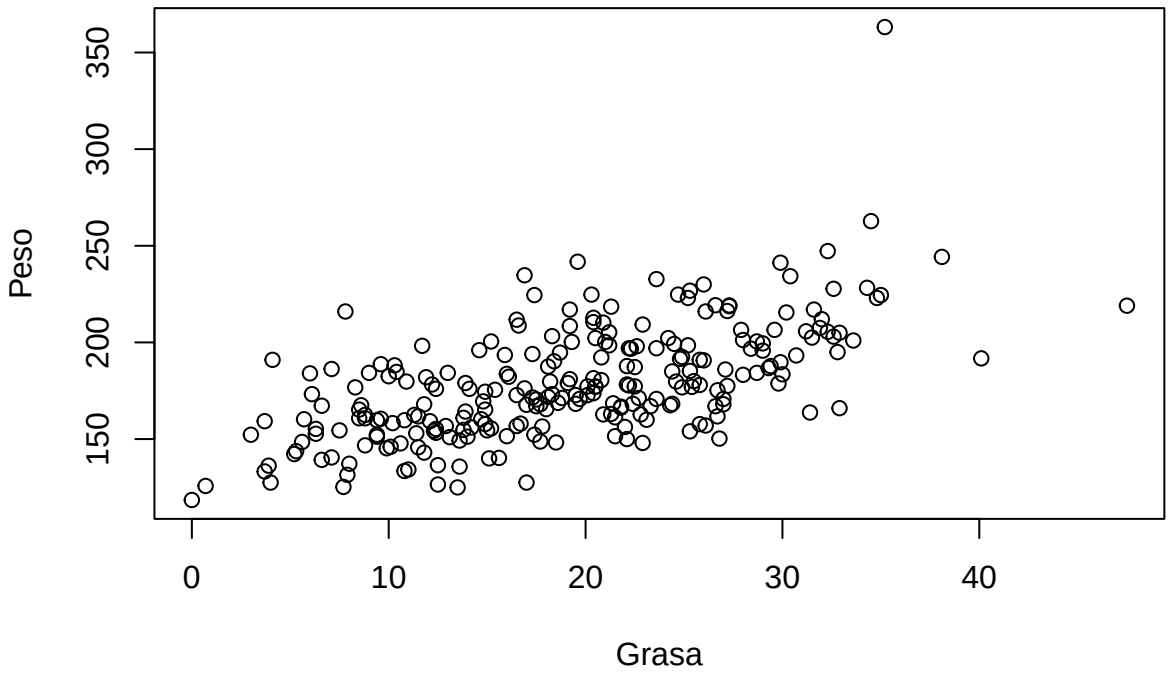
```
##   Grasa   Peso
## 1  12.3 154.25
## 2   6.1 173.25
## 3  25.3 154.00
```

Representación gráfica

Al analizar datos, siempre es recomendable empezar con una representación gráfica que nos permita hacernos a la idea de lo que tenemos.

Esto se consigue haciendo uso de la función `plot`, que ya hemos estudiado en detalle en lecciones anteriores. No obstante, para lo que necesitamos en este tema nos conformamos con un gráfico básico de estos puntos que nos muestre su distribución.

```
plot(body2)
```



Para calcular la recta de regresión con R de la familia de puntos $(x_i, y_i)_{i=1,\dots,n}$, si `x` es el vector $(x_i)_{i=1,\dots,n}$ e `y` es el vector $(y_i)_{i=1,\dots,n}$, entonces, su recta de regresión se calcula mediante la instrucción

`lm(y~x)`

Cuidado con la sintaxis: primero va el vector de las variables dependientes y, seguidamente después de una tilde `~`, va el vector de las variables independientes.

Esto se debe a que R toma el significado de la tilde como "en función de". Es decir, la interpretación de `lm(y~x)` en R es "la recta de regresión de y en función de x".

Si los vectores `y` y `x` son, en este orden, la primera y la segunda columna de un data frame de dos variables, entonces es suficiente aplicar la función `lm` al data frame.

En general, si `x` e `y` son dos variables de un data frame, para calcular la recta de regresión de `y` en función de `x` podemos usar la instrucción

```
lm(y~x, data = data fame)
```

```
lm(body2$Peso~body2$Grasa) #Opción 1
```

Ejemplo 1

```
##
## Call:
## lm(formula = body2$Peso ~ body2$Grasa)
##
## Coefficients:
## (Intercept)  body2$Grasa
##     137.738        2.151
```

```
lm(Peso~Grasa, data = body2) #Opción 2
```

```
##
## Call:
## lm(formula = Peso ~ Grasa, data = body2)
##
## Coefficients:
## (Intercept)        Grasa
##     137.738        2.151
```

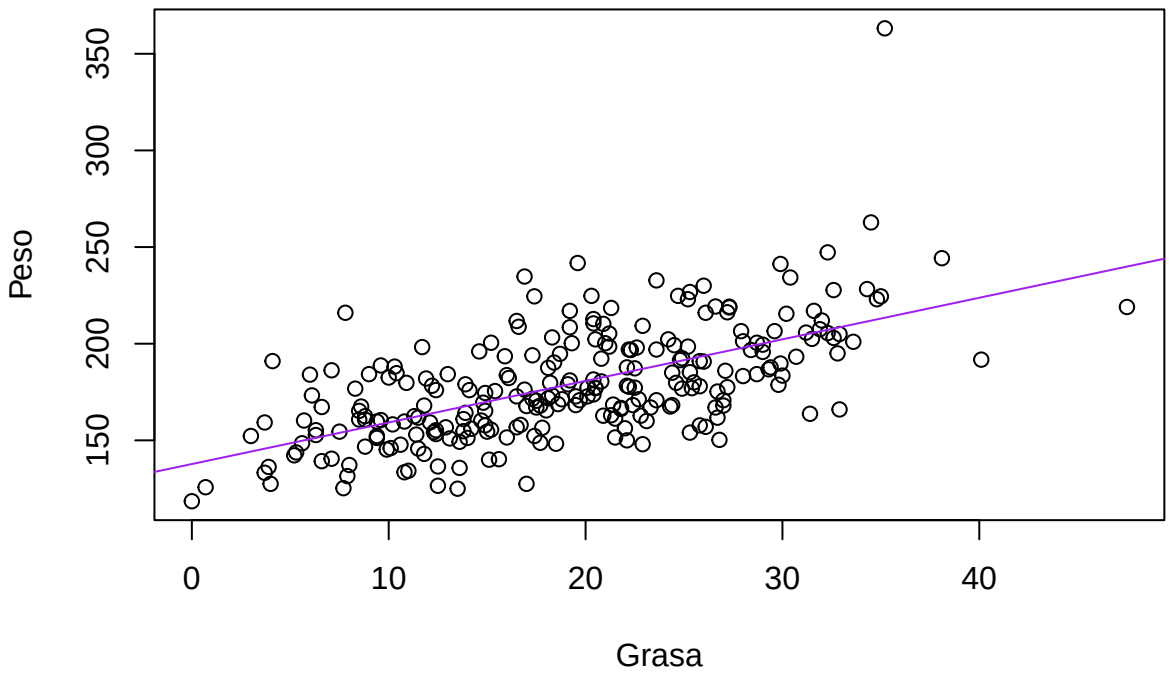
observemos, las dos formas de llamar a la función dan exactamente lo mismo. Ninguna es mejor que la otra.

El resultado obtenido en ambos casos significa que la recta de regresión para nuestros datos es

$$y = 2.151x + 137.738$$

Ahora, podemos superponer esta recta a nuestro gráfico anterior haciendo uso de la función `abline()`.

```
plot(body2)
abline(lm(Peso~Grasa, data = body2), col = "purple")
```



Observación Hay que tener en cuenta que el análisis llevado a cabo hasta el momento de los pares de valores $(x_i, y_i)_{i=1,\dots,n}$ ha sido puramente descriptivo.

Es decir, hemos mostrado que estos datos son consistentes con una función lineal, pero no hemos demostrado que la variable dependiente sea función aproximadamente lineal de la variable dependiente. Esto último necesitaría una demostración matemática, o bien un argumento biológico, pero no basta con una simple comprobación numérica.

Haciendo predicciones

Eso sí, podemos utilizar todo lo hecho hasta ahora para predecir valores $\tilde{y}_i$ en función de los x_i resolviendo una simple ecuación lineal

Coeficiente de determinación

El coeficiente de determinación, R^2, nos es útil para evaluar numéricamente si la relación lineal obtenida es significativa o no.

No explicaremos de momento como se define. Eso lo dejamos para curiosidad del usuario. Por el momento, es suficiente con saber que este coeficiente se encuentra en el intervalo $[0, 1]$. Si R^2 es mayor a 0.9, consideraremos que el ajuste es bueno. De lo contrario, no.

La función summary

La función `summary` aplicada a `lm` nos muestra los contenidos de este objeto. Entre ellos encontramos `Multiple R-squared`, que no es ni más ni menos que el coeficiente de determinación, R^2.

Para facilitarnos las cosas y ahorrarnos información que, de momento, no nos resulta de interés, podemos aplicar `summary(lm(...))$r.squared`

```
summary(lm(Peso~Grasa, data = body2))
```

Ejemplo 1

```
##
## Call:
## lm(formula = Peso ~ Grasa, data = body2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -46.799 -14.999  -3.469  11.860 149.709
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 137.7375     3.6684   37.55   <2e-16 ***
## Grasa         2.1507     0.1756   12.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.28 on 250 degrees of freedom
## Multiple R-squared:  0.3751, Adjusted R-squared:  0.3726
## F-statistic:   150 on 1 and 250 DF,  p-value: < 2.2e-16
```

```
summary(lm(Peso~Grasa, data = body2))$r.squared
```

```
## [1] 0.3750509
```

En este caso, hemos obtenido un coeficiente de determinación de 0.3751, cosa que confirma que la recta de regresión no aproxima nada bien nuestros datos.

Rectas de regresión y transformaciones logarítmicas

No siempre encontraremos dependencias lineales. A veces nos encontraremos otro tipo de dependencias, como por ejemplo pontencias o exponenciales.

Estas se pueden transformar a lineales mediante un cambio de escala

Escalas logarítmicas

Por lo general, es habitual encontrarnos gráficos con sus ejes en escala lineal. Es decir, las marcas en los ejes están igualmente espaciadas.

A veces, es conveniente dibujar alguno de los ejes en escala logarítmica, de modo que la misma distancia entre las marcas significa el mismo cociente entre sus valores. En otras palabras, un eje en escala logarítmica representa el logaritmo de sus valores en escala lineal.

Diremos que un gráfico está en escala semilogarítmica cuando su eje de abcisas está en escala lineal y, el de ordenadas, en escala logarítmica.

Diremos que un gráfico está en escala doble logarítmica cuando ambos ejes están en escala logarítmica.

Interpretación gráfica

Si al representar unos puntos $(x_i, y_i)_{i=1,\dots,n}$ en escala semilogarítmica observamos que siguen aproximadamente una recta, esto querrá decir que los valores $\log(y)$ siguen una ley aproximadamente lineal en los valores x, y, por lo tanto, que y sigue una ley aproximadamente exponencial en x.

En efecto, si $\log(y) = ax + b$, entonces,

$$y = 10^{\log(y)} = 10^{ax+b} = 10^{ax} \cdot 10^b = \alpha^x \beta$$

con $\alpha = 10^a$ y $\beta = 10^b$

Si al representar unos puntos $(x_i, y_i)_{i=1,\dots,n}$ en escala doble logarítmica observamos que siguen aproximadamente una recta, esto querrá decir que los valores $\log(y)$ siguen una ley aproximadamente lineal en los valores $\log(x)$, y, por lo tanto, que y sigue una ley aproximadamente potencial en x.

En efecto, si $\log(y) = a\log(x) + b$, entonces, por propiedades de logaritmos

$$y = 10^{\log(y)} = 10^{a\log(x)+b} = (10^{\log(x)})^a \cdot 10^b = x^a \beta$$

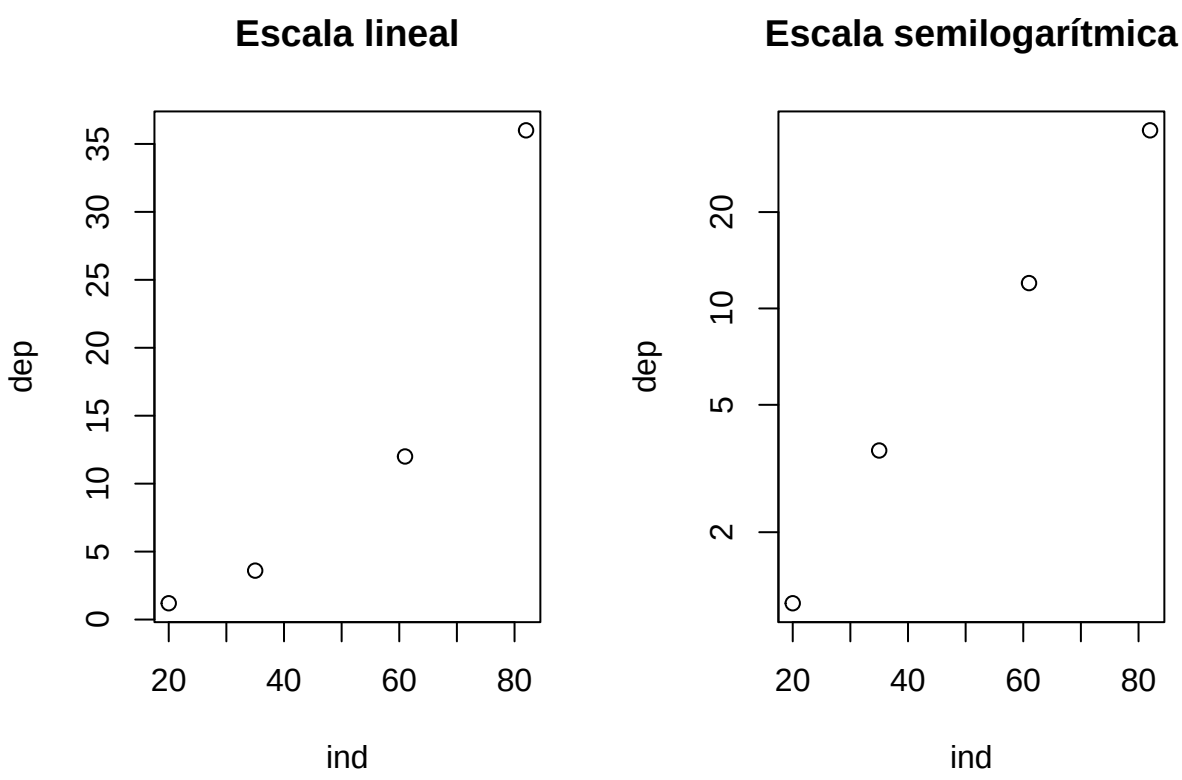
con $\beta = 10^b$

Ejemplo 2 Ejemplo 2

En este caso trabajaremos no con un data frame, sino directamente con los dos vectores siguientes:

```
dep = c(1.2,3.6,12,36)
ind = c(20,35,61,82)
```

```
plot(ind,dep, main = "Escala lineal")
plot(ind,dep, log = "y", main = "Escala semilogarítmica")
```




```
lm(log10(dep)~ind)
```

```
##
## Call:
## lm(formula = log10(dep) ~ ind)
##
## Coefficients:
## (Intercept)          ind
##    -0.32951      0.02318
```

```
summary(lm(log10(dep)~ind))$r.squared
```

```
## [1] 0.9928168
```

Lo que acabamos de obtener es que

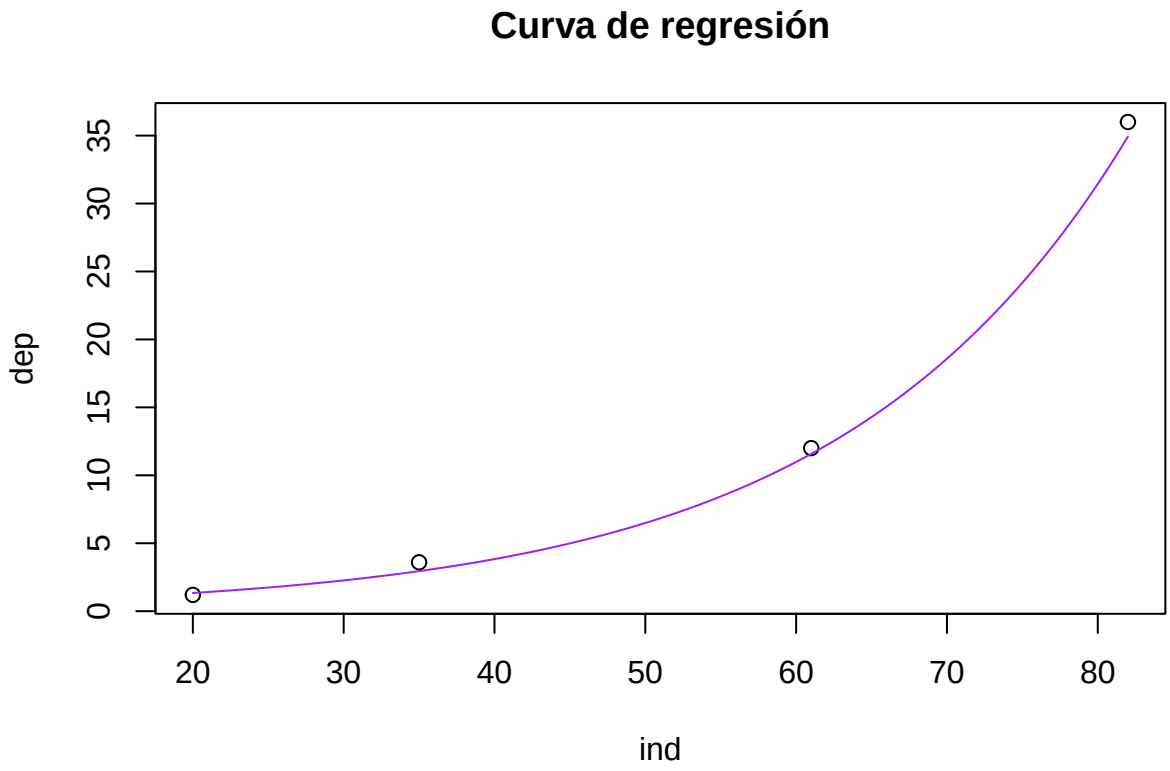
$$\log(dep) = 0.023 \cdot ind - 0.33$$

es una buena aproximación de nuestros datos.

Con lo cual

$$dep = 10^{0.023 \cdot ind} \cdot 10^{-0.33} = 1.054^{ind} \cdot 0.468$$

```
plot(ind,dep, main = "Curva de regresión")
curve(1.054^x*0.468, add = TRUE, col = "purple")
```

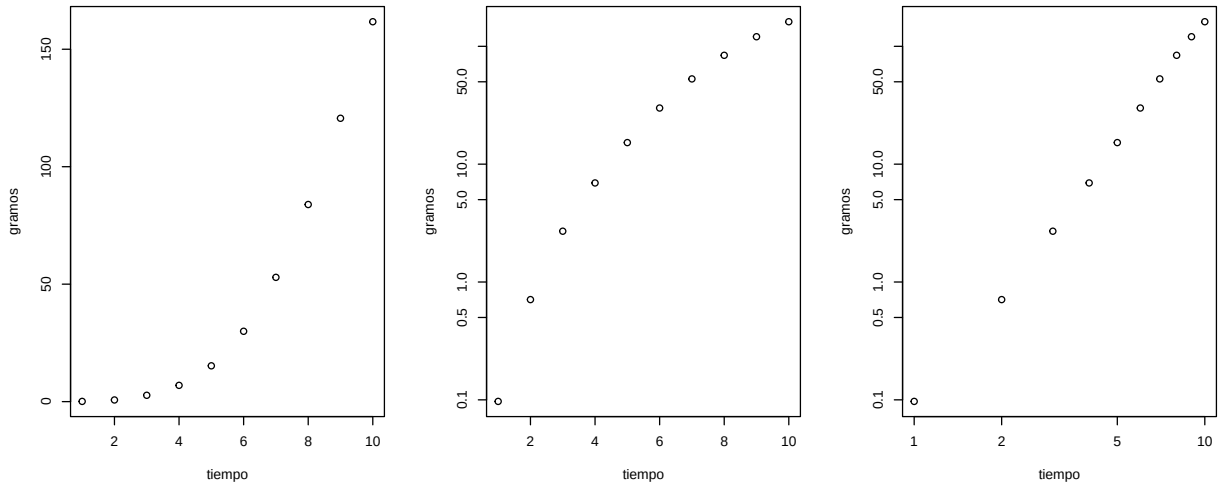


Ejemplo 3 Ejemplo 3

En este caso trabajaremos con el siguiente data frame:

```
tiempo = 1:10
gramos = c(0.097,0.709,2.698,6.928,15.242,29.944,52.902,83.903,120.612,161.711)
d.f = data.frame(tiempo,gramos)
```

```
plot(d.f)
plot(d.f, log = "y")
plot(d.f, log = "xy")
```




```
lm(log10(gramos)~log10(tiempo), data = d.f)
```

```
##
## Call:
## lm(formula = log10(gramos) ~ log10(tiempo), data = d.f)
##
## Coefficients:
##   (Intercept)  log10(tiempo)
##        -1.093          3.298
```

```
summary(lm(log10(gramos)~log10(tiempo), data = d.f))$r.squared
```

```
## [1] 0.9982009
```

Lo que acabamos de obtener es que

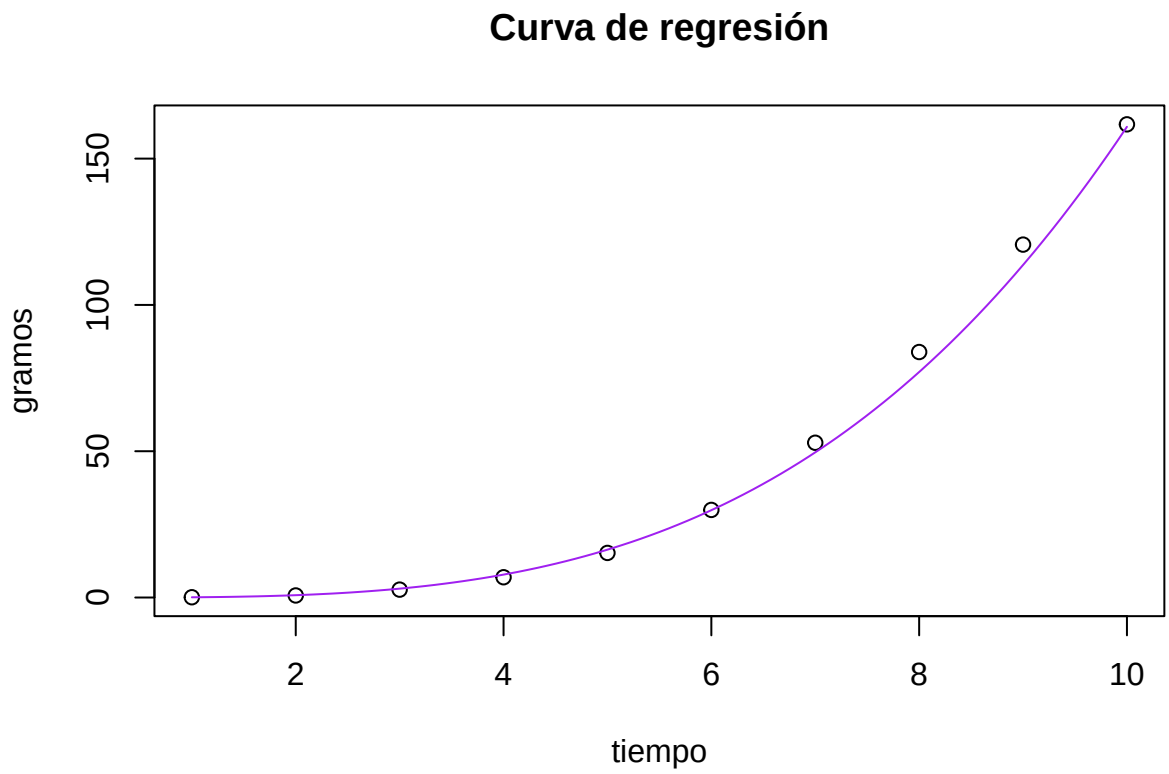
$$\log(gramos) = 3.298 \cdot \log(tiempo) - 1.093$$

es una buena aproximación de nuestros datos.

Con lo cual

$$gramos = 10^{3.298 \cdot \log(tiempo)} \cdot 10^{-1.093} = tiempo^{3.298} \cdot 0.081$$

```
plot(d.f, main = "Curva de regresión")
curve(x^(3.298)*0.081, add=TRUE, col = "purple")
```



Seguramente, en algún momento de nuestra vida ya sea hojeando un libro de matemáticas, curioseando artículos científicos... habréis visto una línea recta o algún otro tipo de curva en un gráfico que se ajusta a las observaciones representadas por medio de puntos en el plano.

En general, la situación es la siguiente: supongamos que tenemos una serie de puntos en el plano cartesiano $\mathbb{R}^2$, de la forma

$$(x_1, y_1),\ \ldots,\ (x_n, y_n)$$

que representan las observaciones de dos variables numéricas. Digamos que x es la edad e y el peso de n estudiantes.

Nuestro objetivo: describir la relación entre la variable independiente, x, y la variable dependiente, y, a partir de estas observaciones.

Para ello, lo que haremos será buscar una función $y = f(x)$ cuya gráfica se aproxime lo máximo posible a nuestros pares ordenados $(x_i, y_i)_{i=1,\ldots,n}$.

Esta función nos dará un modelo matemático de cómo se comportan estas observaciones, lo cual nos permitirá entender mejor los mecanismos que relacionan las variables estudiadas o incluso, nos dará la oportunidad de hacer predicccciones sobre futuras observaciones.

La primera opción es la más fácil. Consiste en estudiar si los puntos $(x_i, y_i)_{i=1,\ldots,n}$ satisfacen una relación lineal de la forma

$$y = ax + b$$

con $a, b \in \mathbb{R}$.

En este caso, se busca la recta $y = ax + b$ que mejor aproxime los puntos dados imponiendo que la suma de los cuadrados de las diferencias entre los valores y_i y sus aproximaciones $\tilde{y}_i = ax_i + b$ sea mínima. Es decir,

que

$$\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

sea mínima

El objetivo de este tema no es otro más que enseñaros como hacer uso de `R` para obtener esta recta de regresión.

Veamos también cómo se puede evaluar numéricamente si esta recta se ajusta bien a las observaciones dadas.

Para ello, introduciremos algunas funciones de `R` y haremos uso de transformaciones logarítmicas para tratar casos en los que los puntos dados se aproximen mejor mediante una función exponencial o potencial.

Como ya hemos dicho, el objetivo de este tema es estudiar si existe relación lineal entre las variables dependiente e independiente.

Por lo general, cuando tenemos una serie de observaciones emparejadas, $(x_i, y_i)_{i=1,\ldots,n}$, la forma natural de almacenarlas en `R` es mediante una tabla de datos. Y la que más conocemos nostros es el data frame.

Ejemplo 1 Ejemplo 1

En este ejemplo, nosotros haremos uso del siguiente data frame:

```
#body = read.table("../data/bodyfat.txt", header = TRUE)
#head(body,3)
```

Más concretamente, trabajaremos con las variables `fat` y `weight`.

```
body2 = body[,c(2,4)]
names(body2) = c("Grasa","Peso")
str(body2)
```

```
## 'data.frame':    252 obs. of  2 variables:
##  $ Grasa: num  12.3 6.1 25.3 10.4 28.7 20.9 19.2 12.4 4.1 11.7 ...
##  $ Peso : num  154 173 154 185 184 ...
```

```
head(body2,3)
```

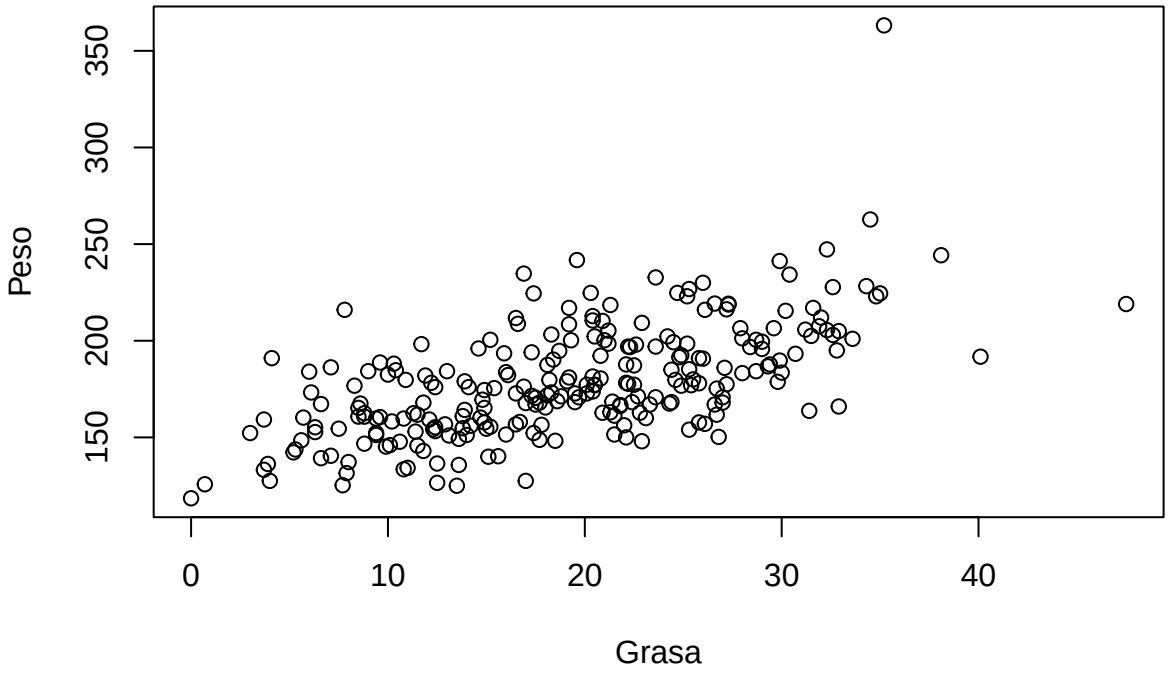
```
##   Grasa   Peso
## 1  12.3 154.25
## 2   6.1 173.25
## 3  25.3 154.00
```

Representación gráfica

Al analizar datos, siempre es recomendable empezar con una representación gráfica que nos permita hacernos a la idea de lo que tenemos.

Esto se consigue haciendo uso de la función `plot`, que ya hemos estudiado en detalle en lecciones anteriores. No obstante, para lo que necesitamos en este tema nos conformamos con un gráfico básico de estos puntos que nos muestre su distribución.

```
plot(body2)
```



Ejemplo 1

Para calcular la recta de regresión con R de la familia de puntos $(x_i, y_i)_{i=1,\dots,n}$, si `x` es el vector $(x_i)_{i=1,\dots,n}$ e `y` es el vector $(y_i)_{i=1,\dots,n}$, entonces, su recta de regresión se calcula mediante la instrucción

```
lm(y~x)
```

Cuidado con la sintaxis: primero va el vector de las variables dependientes y, seguidamente después de una tilde `~`, va el vector de las variables independientes.

Esto se debe a que `R` toma el significado de la tilde como "en función de". Es decir, la interpretación de `lm(y~x)` en R es "la recta de regresión de y en función de x"

Si los vectores `y` y `x` son, en este orden, la primera y la segunda columna de un data frame de dos variables, entonces es suficiente aplicar la función `lm` al data frame.

En general, si `x` e `y` son dos variables de un data frame, para calcular la recta de regresión de `y` en función de `x` podemos usar la instrucción

```
lm(y~x, data = data fame)
```

```
lm(body2$Peso~body2$Grasa) #Opción 1
```

Ejemplo 1

```
##
## Call:
## lm(formula = body2$Peso ~ body2$Grasa)
##
## Coefficients:
## (Intercept)  body2$Grasa
##     137.738        2.151
```

```
lm(Peso~Grasa, data = body2) #Opción 2
```

```
##
## Call:
## lm(formula = Peso ~ Grasa, data = body2)
##
## Coefficients:
## (Intercept)        Grasa
##     137.738        2.151
```

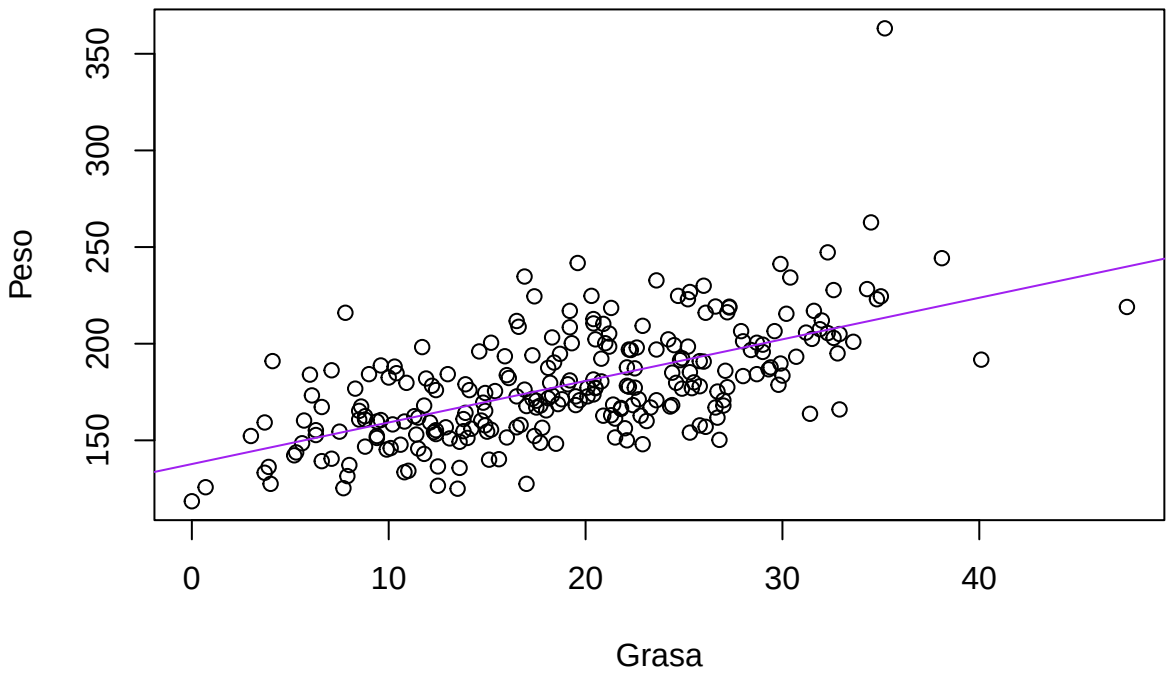
Como podemos observar, las dos formas de llamar a la función dan exactamente lo mismo. Ninguna es mejor que la otra.

El resultado obtenido en ambos casos significa que la recta de regresión para nuestros datos es

$$y = 2.151x + 137.738$$

Ahora, podemos superponer esta recta a nuestro gráfico anterior haciendo uso de la función `abline()`.

```
plot(body2)
abline(lm(Peso~Grasa, data = body2), col = "purple")
```



Observación Hay que tener en cuenta que el análisis llevado a cabo hasta el momento de los pares de valores $(x_i, y_i)_{i=1,\dots,n}$ ha sido puramente descriptivo.

Es decir, hemos mostrado que estos datos son consistentes con una función lineal, pero no hemos demostrado que la variable dependiente sea función aproximadamente lineal de la variable dependiente. Esto último necesitaría una demostración matemática, o bien un argumento biológico, pero no basta con una simple comprobación numérica.

Haciendo predicciones

Podemos utilizar todo lo hecho hasta ahora para predecir valores $\tilde{y}_i$ en función de los x_i resolviendo una simple ecuación lineal

Coeficiente de determinación

El coeficiente de determinación, R^2, nos es útil para evaluar numéricamente si la relación lineal obtenida es significativa o no.

No explicaremos de momento como se define. Eso lo dejamos para curiosidad del usuario. Por el momento, es suficiente con saber que este coeficiente se encuentra en el intervalo $[0, 1]$. Si R^2 es mayor a 0.9, consideraremos que el ajuste es bueno. De lo contrario, no.

La función summary

La función `summary` aplicada a `lm` nos muestra los contenidos de este objeto. Entre ellos encontramos `Multiple R-squared`, que es el coeficiente de determinación, R^2.

Para facilitarnos las cosas y ahorrarnos información que, de momento, no nos resulta de interés, podemos aplicar `summary(lm(...))$r.squared`

```
summary(lm(Peso~Grasa, data = body2))
```

Ejemplo 1

```
##
## Call:
## lm(formula = Peso ~ Grasa, data = body2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -46.799 -14.999  -3.469  11.860 149.709
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 137.7375     3.6684   37.55   <2e-16 ***
## Grasa         2.1507     0.1756   12.25   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.28 on 250 degrees of freedom
## Multiple R-squared:  0.3751, Adjusted R-squared:  0.3726
## F-statistic:   150 on 1 and 250 DF,  p-value: < 2.2e-16
```

```
summary(lm(Peso~Grasa, data = body2))$r.squared
```

```
## [1] 0.3750509
```

En este caso, hemos obtenido un coeficiente de determinación de 0.3751, cosa que confirma que la recta de regresión no aproxima nada bien nuestros datos.

Rectas de regresión y transformaciones logarítmicas

No siempre encontraremos dependencias lineales. A veces nos encontraremos otro tipo de dependencias, como por ejemplo pontencias o exponenciales.

Estas se pueden transformar a lineales mediante un cambio de escala

Escalas logarítmicas

Por lo general, es habitual encontrarnos gráficos con sus ejes en escala lineal. Es decir, las marcas en los ejes están igualmente espaciadas.

A veces, es conveniente dibujar alguno de los ejes en escala logarítmica, de modo que la misma distancia entre las marcas significa el mismo cociente entre sus valores. En otras palabras, un eje en escala logarítmica representa el logaritmo de sus valores en escala lineal.

Diremos que un gráfico está en escala semilogarítmica cuando su eje de abcisas está en escala lineal y, el de ordenadas, en escala logarítmica.

Diremos que un gráfico está en escala doble logarítmica cuando ambos ejes están en escala logarítmica.

Interpretación gráfica

Si al representar unos puntos $(x_i, y_i)_{i=1,\dots,n}$ en escala semilogarítmica observamos que siguen aproximadamente una recta, esto querrá decir que los valores $\log(y)$ siguen una ley aproximadamente lineal en los valores x, y, por lo tanto, que y sigue una ley aproximadamente exponencial en x.

En efecto, si $\log(y) = ax + b$, entonces,

$$y = 10^{\log(y)} = 10^{ax+b} = 10^{ax} \cdot 10^b = \alpha^x \beta$$

con $\alpha = 10^a$ y $\beta = 10^b$

Si al representar unos puntos $(x_i, y_i)_{i=1,\dots,n}$ en escala doble logarítmica observamos que siguen aproximadamente una recta, esto querrá decir que los valores $\log(y)$ siguen una ley aproximadamente lineal en los valores $\log(x)$, y, por lo tanto, que y sigue una ley aproximadamente potencial en x.

En efecto, si $\log(y) = a\log(x) + b$, entonces, por propiedades de logaritmos

$$y = 10^{\log(y)} = 10^{a\log(x)+b} = (10^{\log(x)})^a \cdot 10^b = x^a \beta$$

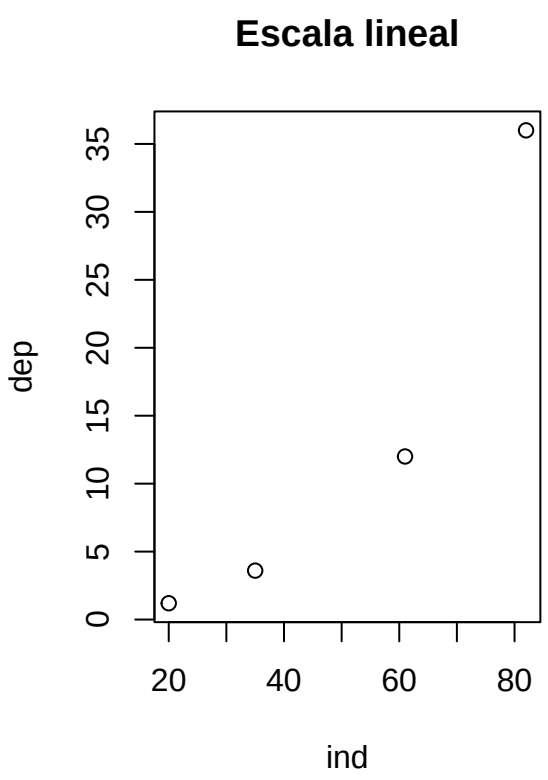
con $\beta = 10^b$

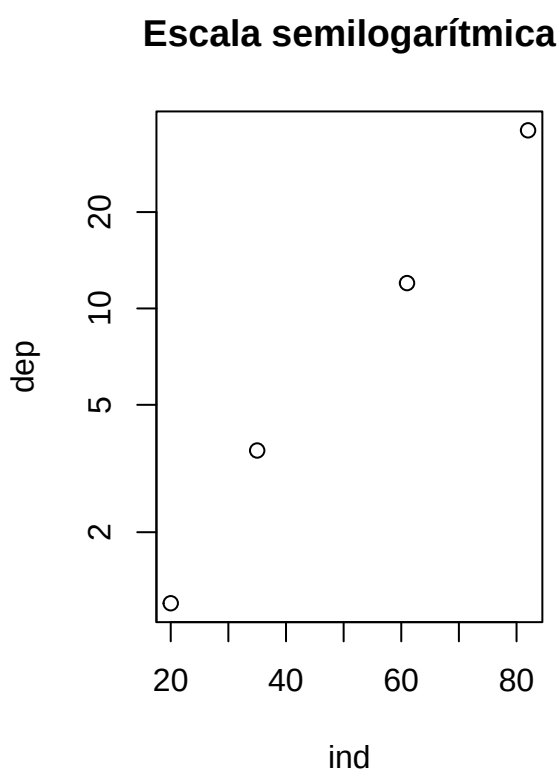
Ejemplo 2 Ejemplo 2

En este caso trabajaremos no con un data frame, sino directamente con los dos vectores siguientes:

```
dep = c(1.2,3.6,12,36)
ind = c(20,35,61,82)
```

```
plot(ind,dep, main = "Escala lineal")
plot(ind,dep, log = "y", main = "Escala semilogarítmica")
```






```
lm(log10(dep)~ind)
```

```
##
## Call:
## lm(formula = log10(dep) ~ ind)
##
## Coefficients:
## (Intercept)          ind
##    -0.32951      0.02318
```

```
summary(lm(log10(dep)~ind))$r.squared
```

```
## [1] 0.9928168
```

Lo que acabamos de obtener es que

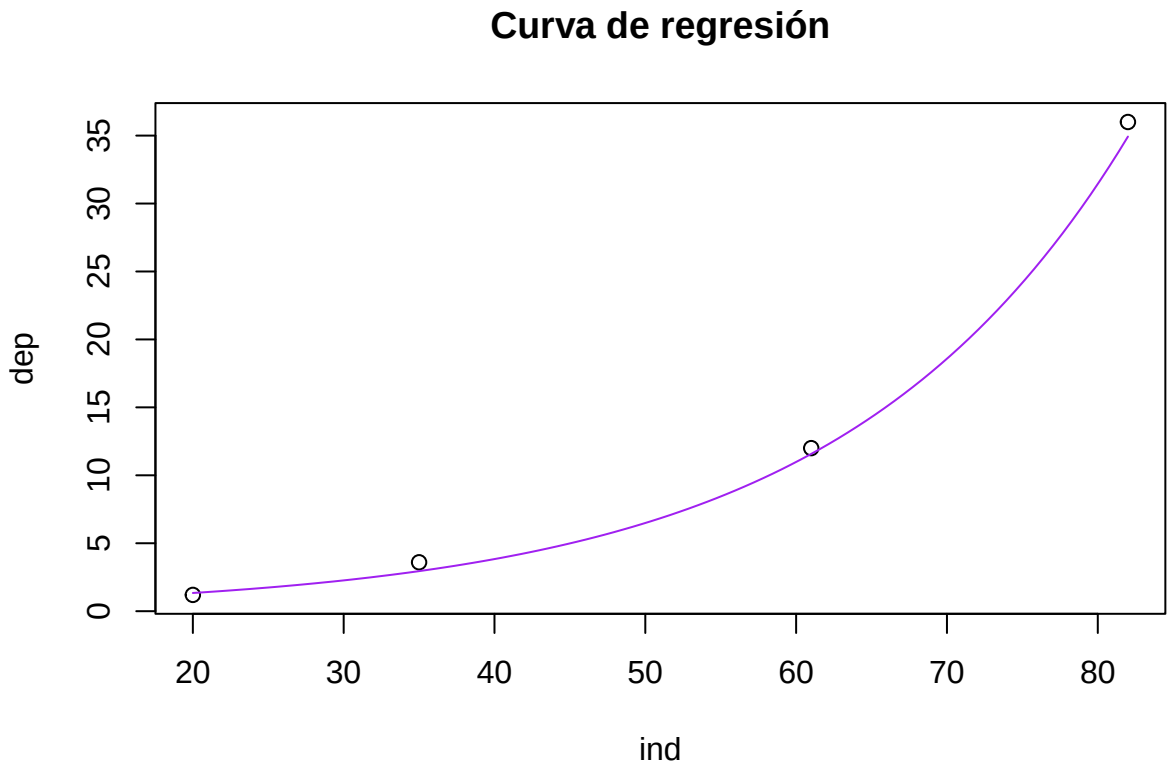
$$\log(dep) = 0.023 \cdot ind - 0.33$$

es una buena aproximación de nuestros datos.

Con lo cual

$$dep = 10^{0.023 \cdot ind} \cdot 10^{-0.33} = 1.054^{ind} \cdot 0.468$$

```
plot(ind,dep, main = "Curva de regresión")
curve(1.054^x*0.468, add = TRUE, col = "purple")
```

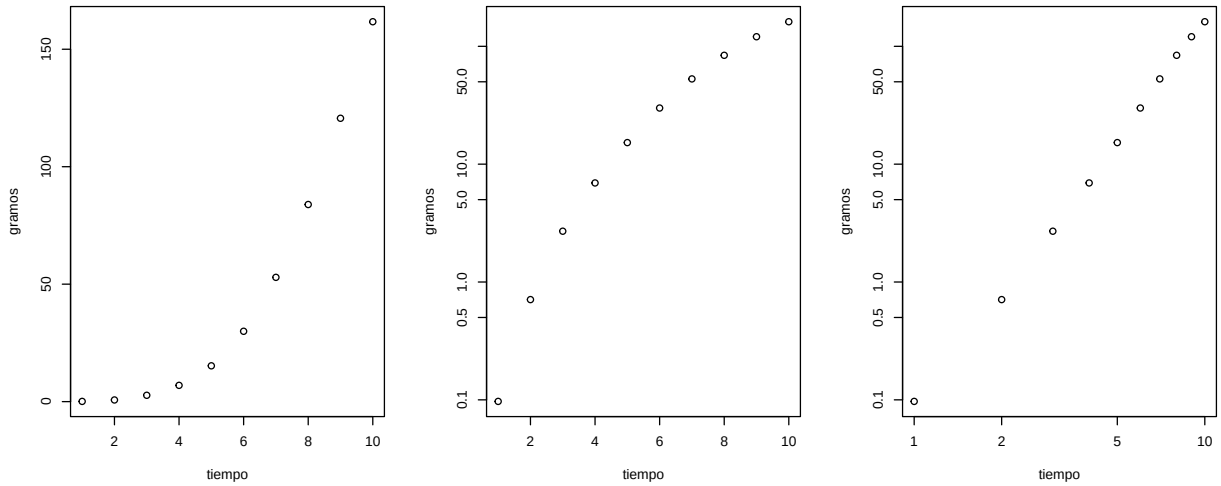


Ejemplo 3 Ejemplo 3

En este caso trabajaremos con el siguiente data frame:

```
tiempo = 1:10
gramos = c(0.097,0.709,2.698,6.928,15.242,29.944,52.902,83.903,120.612,161.711)
d.f = data.frame(tiempo,gramos)
```

```
plot(d.f)
plot(d.f, log = "y")
plot(d.f, log = "xy")
```




```
lm(log10(gramos)~log10(tiempo), data = d.f)
```

```
##
## Call:
## lm(formula = log10(gramos) ~ log10(tiempo), data = d.f)
##
## Coefficients:
##   (Intercept)  log10(tiempo)
##        -1.093          3.298
```

```
summary(lm(log10(gramos)~log10(tiempo), data = d.f))$r.squared
```

```
## [1] 0.9982009
```

Lo que acabamos de obtener es que

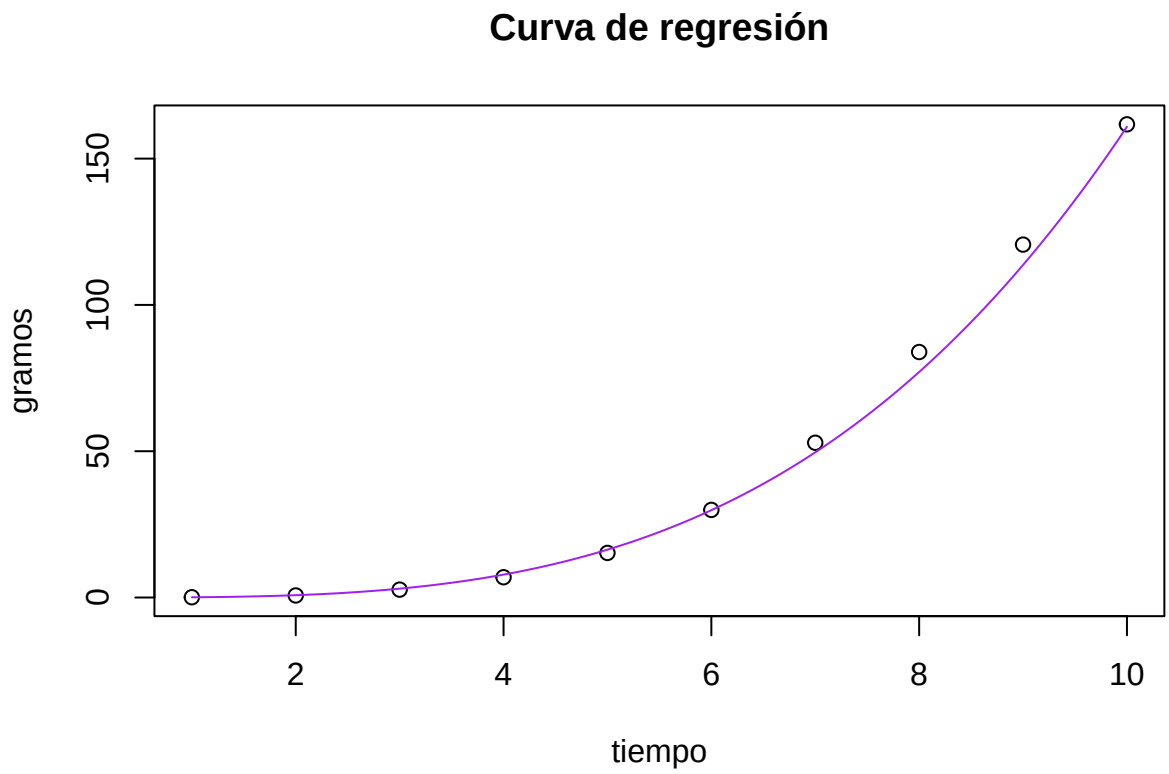
$$\log(gramos) = 3.298 \cdot \log(tiempo) - 1.093$$

es una buena aproximación de nuestros datos.

Con lo cual

$$gramos = 10^{3.298 \cdot \log(tiempo)} \cdot 10^{-1.093} = tiempo^{3.298} \cdot 0.081$$

```
plot(d.f, main = "Curva de regresión")
curve(x^(3.298)*0.081, add=TRUE, col = "purple")
```



Distribuciones de Probabilidad

Experimento aleatorio

Experimento aleatorio. Experimento que efectuado en las mismas condiciones puede dar lugar a resultados diferentes

Suceso elemental. Cada uno de los posibles resultados del experimento aleatorio

Espacio muestral. Conjunto Ω formado por todos los sucesos elementales del experimento aleatorio

Ejemplo

Lanzar una moneda es un experimento aleatorio

Los sucesos elementales son: sacar cara (C) y sacar cruz $(+)$

El espacio muestral de este experimento aleatorio es $\Omega = \{C, +\}$

Sucesos

Suceso. Subconjunto del espacio muestral

Suceso total o seguro. Ω

Suceso vacío o imposible. $\emptyset$

Ejemplo

Lanzar un dado es un experimento aleatorio

Algunos sucesos podrían ser: sacar número par ($\{2,4,6\}$), sacar mayor que 4 ($\{5,6\}$), sacar número múltiplo de 3 ($\{3,6\}$)...

El suceso total de este experimento aleatorio es $\Omega = \{1,2,3,4,5,6\}$

Un ejemplo de suceso imposible de este experimento aleatorio es $\emptyset = \{7\}$ (sacar 7)

Operaciones con sucesos. Sean $A, B \subseteq \Omega$ sucesos. Entonces,

- $A \cup B$ es el suceso unión (resultados pertenecen a A, o a B, o a ambos)
- $A \cap B$ es el suceso intersección (resultados pertenecen a A y B)
- A^c es el suceso complementario (resultados que no pertenecen a A)
- $A - B = A \cap B^c$ es el suceso diferencia (resultados que pertenecen a A pero no a B)

Sucesos incompatibles. Si $A \cap B = \emptyset$

Probabilidad

Probabilidad de un suceso. Número entre 0 y 1 (ambos incluidos) que mide la expectativa de que se dé este suceso

Ejemplo

- La probabilidad de sacar un 6 al lanzar un dado estándar no trucado es $\frac{1}{6}$
- La probabilidad de sacar un 6 al lanzar un dado de 4 caras es 0
- La probabilidad de sacar un 6 al lanzar un dado de 20 caras es $\frac{1}{20}$

Probabilidad. Sea Ω el espacio muestral de un experimento aleatorio. Suponiendo que Ω es **finito**, una probabilidad sobre Ω es una aplicación

$$p : \mathcal{P}(\Omega) \longrightarrow [0,1]$$

que satisface

- $0 \leq p(A) \leq 1 \ \forall A \in \mathcal{P}(\Omega)$
- $p(\Omega) = 1$
- Si $\{A_1, \ldots, A_n\}$ son sucesos incompatibles dos a dos ($A_i \cap A_j = \emptyset \ \forall i \neq j$), entonces

$$p(A_1 \cup \cdots \cup A_n) = p(A_1) + \cdots + p(A_n)$$

Notación: Si $a \in \Omega$, escribiremos $p(a)$ en vez de $p(\{a\})$

Variables aleatorias

Variable aleatoria. Una variable aleatoria (v.a.) sobre Ω es una aplicación

$$X : \Omega \longrightarrow \mathbb{R}$$

que asigna a cada suceso elemental ω un número real $X(\omega)$

Puede entenderse como una descripción numérica de los resultados de un experimento aleatorio

Dominio de una variable aleatoria. D_X, es el conjunto de los valores que puede tomar

Sucesos de variables aleatorias

Una variable aleatoria puede definir sucesos, de los cuales queremos conocer la probabilidad p

- $p(X = a) = p(\{\omega \in \Omega \mid X(\omega) = a\})$
- $p(X < b) = p(\{\omega \in \Omega \mid X(\omega) < b\})$
- $p(X \leq b) = p(\{\omega \in \Omega \mid X(\omega) \leq b\})$
- $p(a < X) = p(\{\omega \in \Omega \mid a < X(\omega)\})$
- $p(a \leq X) = p(\{\omega \in \Omega \mid a \leq X(\omega)\})$
- $p(a \leq X \leq b) = p(\{\omega \in \Omega \mid a \leq X(\omega) \leq b\})$
- $p(a < X < b) = p(\{\omega \in \Omega \mid a < X(\omega) < b\})$
- $p(X \in A) = p(\{\omega \in \Omega \mid X(\omega) \in A\})$

Función de distribución

Función de distribución de la v.a. X. Es una función

$$F : \mathbb{R} \longrightarrow [0, 1]$$

definida por $F(x) = p(X \leq x)$

Sea F una función de distribución de una v.a. X y digamos

$$F(a^-) = \lim_{x \to a^-} F(x)$$

- $p(X \leq a) = F(a)$
- $p(X < a) = \lim_{b \to a,\ b<a} p(X \leq b) = \lim_{b \to a,\ b<a} F(b) = F(a^-)$
- $p(X = a) = p(X \leq a) - p(X < a) = F(a) - F(a^-)$
- $p(a \leq X \leq b) = p(X \leq b) - p(X < a) = F(b) - F(a^-)$

Cuantiles

Cuantil de orden p de una v.a. X. Es el $x_p \in \mathbb{R}$ más pequeño tal que $F(x_p) \geq p$

Nótese que la mediana es el cuantil de orden 0.5

Variables aleatorias discretas

Variable aleatoria discreta

Variable aleatoria discreta. Una v.a. $X : \Omega \longrightarrow \mathbb{R}$ es discreta cuando D_X es finito o un subconjunto de $\mathbb{N}$

Función de probabilidad. Es la función $f : \mathbb{R} \longrightarrow [0, 1]$ definida por

$$f(x) = p(X = x)$$

Nótese que $f(x) = 0$ si $x \notin D_X$. Por tanto, interpretaremos la función de probabilidad como la función

$$f : D_X \longrightarrow [0, 1]$$

Esperanza

Esperanza de una v.a. discreta. Sea $f : D_X \longrightarrow [0,1]$ la función de probabilidad de X, entonces la esperanza respecto de la función de probabilidad es la suma ponderada de los elementos de D_X, multiplicando cada elemento x de D_X por su probabilidad,

$$E(X) = \sum_{x \in D_X} x \cdot f(x)$$

Si $g : D_X \longrightarrow \mathbb{R}$ es una aplicación

$$E(g(X)) = \sum_{x \in D_X} g(x) \cdot f(x)$$

Varianza

Varianza de una v.a. discreta. Sea $f : D_X \longrightarrow [0,1]$ la función de probabilidad de X, entonces la varianza respecto de la función de probabilidad es el valor esperado de la diferencia al cuadrado entre X y su valor medio $E(X)$,

$$Var(X) = E((X - E(X))^2)$$

La varianza mide como de variados son los resultados de X respecto de la media

Ejercicio. Demostrar la siguiente igualdad.

$$Var(X) = E(X^2) - (E(X))^2$$

Si X es una v.a. discreta y $g : D_X \longrightarrow \mathbb{R}$ una función,

$$Var(g(X)) = E((g(X) - E(g(X)))^2) = E(g(X)^2) - (E(g(X)))^2$$

Desviación típica

Desviación típica de una v.a. discreta. Sea $f : D_X \longrightarrow [0,1]$ la función de probabilidad de X, entonces la desviación típica respecto de la función de probabilidad es

$$\sigma(X) = \sqrt{Var(X)}$$

Las unidades de la varianza son las de X al cuadrado. En cambio, las de la desviación típica son las mismas unidades que las de X

Si X es una v.a. discreta y $g : D_X \longrightarrow \mathbb{R}$ una función,

$$\sigma(g(X)) = \sqrt{Var(g(X))}$$

Distribuciones de probabilidad

Distribución de probabilidad

Distribución de probabilidad. En teoría de la probabilidad y estadística, la distribución de probabilidad de una variable aleatoria es una función que asigna a cada suceso definido sobre la variable la probabilidad de que dicho suceso ocurra.

Distribuciones en `R`

Dada cualquier variable aleatoria, `va`, `R` nos da cuatro funciones para poder trabajar con ellas:

- `dva(x,...)`: Función de densidad o de probabilidad $f(x)$ de la variable aleatoria para el valor x del dominio de definición.
- `pva(x,...)`: Función de distribución $F(x)$ de la variable aleatoria para el valor x del dominio de definición.
- `qva(p,...)`: Cuantil p-ésimo de la variable aleatoria (el valor de x más pequeño tal que $F(x) \geq p$).
- `rva(n,...)`: Generador de n observaciones siguiendo la distribución de la variable aleatoria.

Distribuciones en `Python`

Dada cualquier variable aleatoria, en `Python` tenemos las mismas cuatro funciones, sin que su nombre dependa de la misma:

- `pmf(k,...)` o `pdf(x,...)`: Función de probabilidad $f(k)$ o de densidad $f(x)$ de la variable aleatoria para los valores k o x del dominio.
- `cdf(x,...)`: Función de distribución $F(x)$ de la variable aleatoria para el valor k del dominio.
- `ppf(p,...)`: Cuantil p-ésimo de la variable aleatoria (el valor de x más pequeño tal que $F(x) \geq p$).
- `rvs(size,...)`: Generador de $size$ observaciones siguiendo la distribución de la variable aleatoria.

También vale la pena conocer la función `stats(moments='mvsk')` que nos devuelve cuatro valores con los estadísticos de la media `m`, la varianza `v`, el sesgo `s` y la curtosis `k` de la distribución.

Distribución de Bernoulli

Si X es variable aleatoria que mide el "número de éxitos" y se realiza un único experimento con dos posibles resultados (éxito, que toma valor 1, o fracaso, que toma valor 0), diremos que X se distribuye como una Bernoulli con parámetro p

$$X \sim \text{Be}(p)$$

donde p es la probabilidad de éxito y $q = 1 - p$ es la probabilidad de fracaso.

- El **dominio** de X será $D_X = \{0, 1\}$
- La **función de probabilidad** vendrá dada por

$$f(k) = p^k(1-p)^{1-k} = \begin{cases} p & \text{si } k = 1 \\ 1-p & \text{si } k = 0 \\ 0 & \text{en cualquier otro caso} \end{cases}$$

- La **función de distribución** vendrá dada por

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1-p & \text{si } 0 \leq x < 1 \\ 1 & \text{si } x \geq 1 \end{cases}$$

- **Esperanza** $E(X) = p$
- **Varianza** $Var(X) = pq$

El código de la distribución de Beroulli:

- En `R` tenemos las funciones del paquete `Rlab`: `dbenr(x,prob)`, `pbenr(q,prob)`, `qbenr(p,prob)`, `rbenr(n, prob)` donde `prob` es la probabilidad de éxito.
- En `Python` tenemos las funciones del paquete `scipy.stats.bernoulli`: `pmf(k,p)`, `cdf(k,p)`, `ppf(q,p)`, `rvs(p, size)` donde `p` es la probabilidad de éxito.

Función de densidad Sea $X = Be(p = 0.7)$, la distribución que modela la probabilidad de obtener una cara usando una moneda trucada.

$$f(k) = p^k(1-p)^{1-k}, \ k \in \{0, 1\}$$

```
library(Rlab)
```

En R

```
## Rlab 4.0 attached.

##
## Attaching package: 'Rlab'

## The following object is masked from 'package:dplyr':
##
##     count

## The following object is masked from 'package:tibble':
##
##     view

## The following objects are masked from 'package:stats':
##
##     dexp, dgamma, dweibull, pexp, pgamma, pweibull, qexp, qgamma,
##     qweibull, rexp, rgamma, rweibull

## The following object is masked from 'package:datasets':
##
##     precip
```

```
dbern(0, prob= 0.7)  #probabilidad de obbtener 0
```

```
## [1] 0.3
```

```
dbern(1, prob = 0.7) #probabilidad de obbtener 1   0.3 + 0.7=1
```

```
## [1] 0.7
```

```r
pbern(0, prob = 0.7) #probabilidad acumulada
```

```
## [1] 0.3
```

```r
pbern(1, prob = 0.7) #probabilidad acumulada
```

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

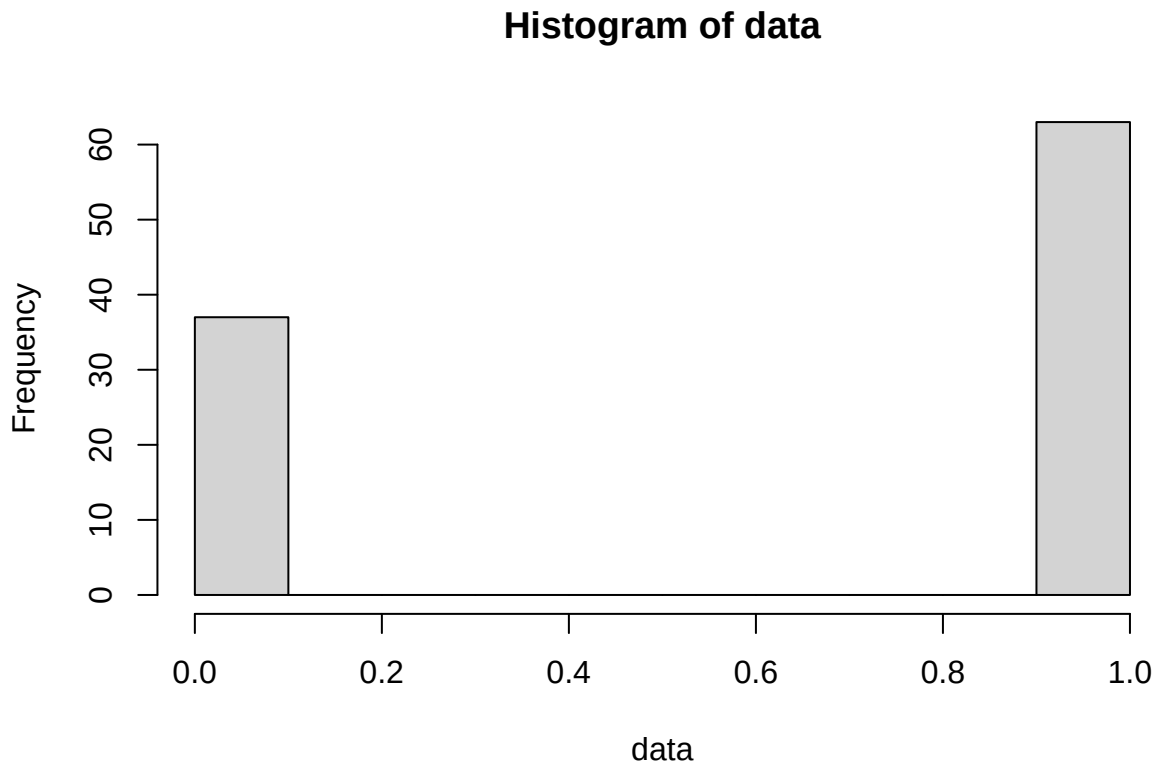
```r
qbern(0.5, prob = 0.7)    ## cuantil, en este caso es la mediana
```

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

```r
qbern(0.25, prob = 0.7)   ## 1er Cuartil
```

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

```r
rbern(100, prob = 0.7) -> data  ##Genera 70 unos y 30 ceros (aprox)
hist(data)
```



```
from scipy.stats import bernoulli
import matplotlib.pyplot as plt
p = 0.7
mean, var, skew, kurt = bernoulli.stats(p, moments = 'mvsk')
print("Media %f"%mean)
```

En Python

```
## Media 0.700000
```

```
print("Varianza %f"%var)
```

```
## Varianza 0.210000
```

```
print("Sesgo %f"%skew)
```

```
## Sesgo -0.872872
```

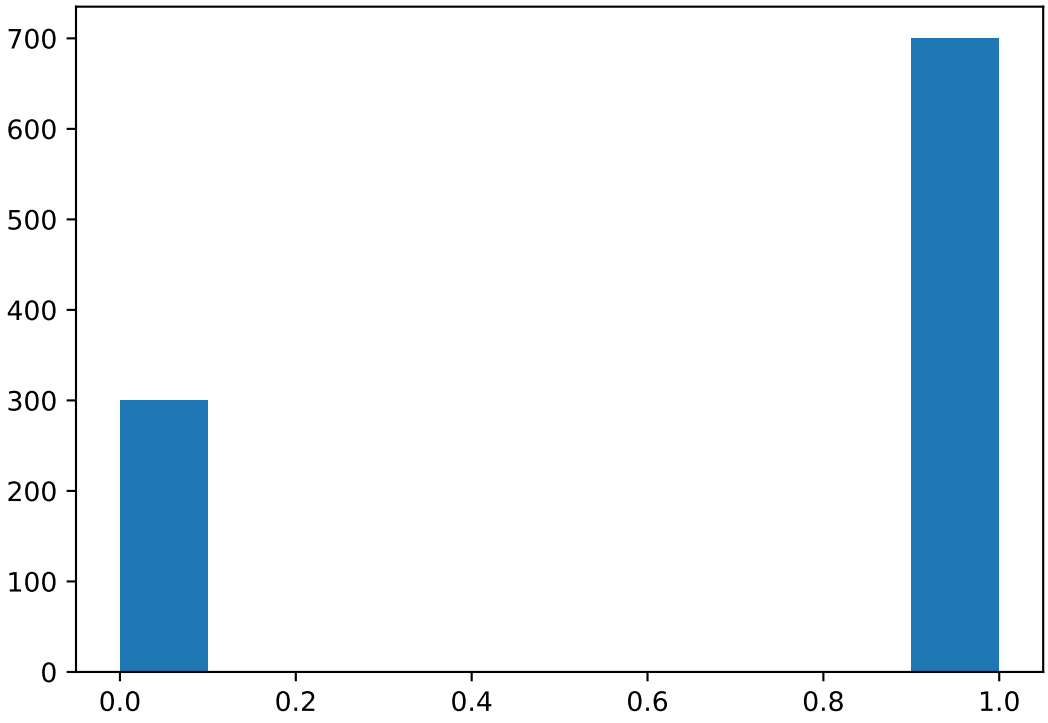
```
print("Curtosis %f"%kurt)
```

```
## Curtosis -1.238095
```

```
fix, ax = plt.subplots(1,1)
x = bernoulli.rvs(p, size = 1000)
ax.hist(x)
```

```
## (array([300.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 700.]), array([0. , 0.1, 0.2, 0.3, 0.4
```

```
plt.show()
```



Distribución Binomial

Si X es variable aleatoria que mide el "número de éxitos" y se realizan n ensayos de Bernoulli independientes entre sí, diremos que X se distribuye como una Binomial con parámetros n y p

$$X \sim \mathrm{B}(n, p)$$

donde p es la probabilidad de éxito y $q = 1 - p$ es la probabilidad de fracaso

- El **dominio** de X será $D_X = \{0, 1, 2, \ldots, n\}$
- La **función de probabilidad** vendrá dada por

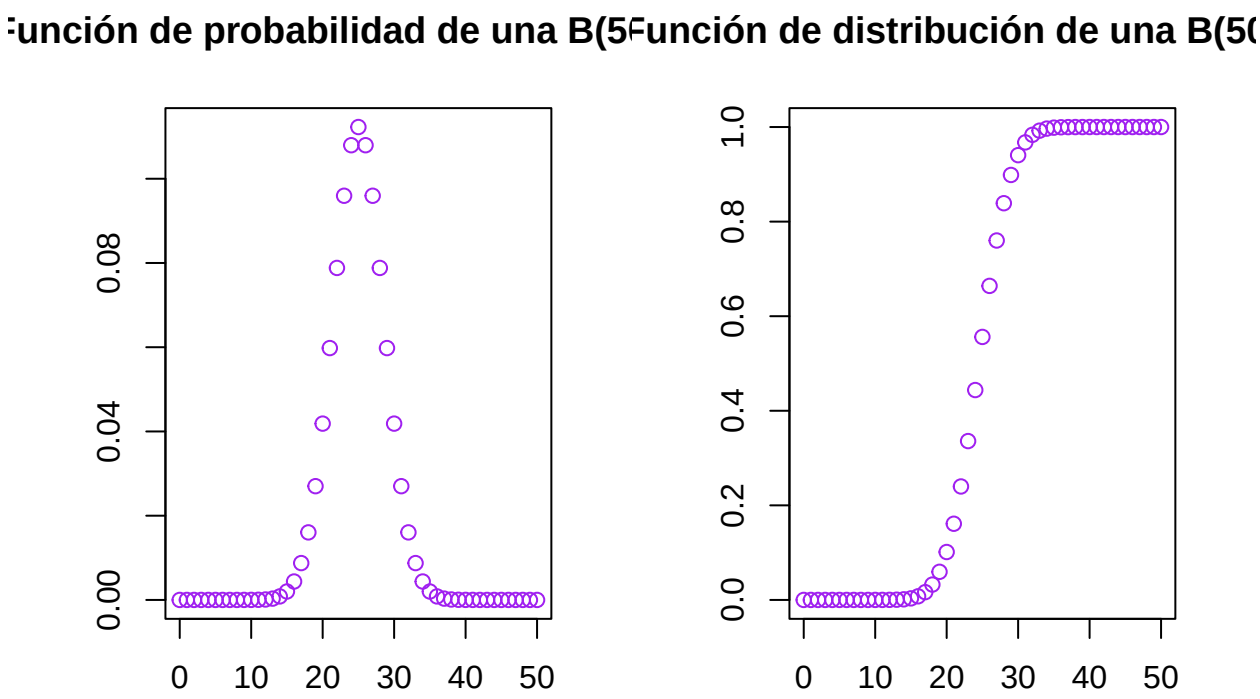
$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

- La **función de distribución** vendrá dada por

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ \sum_{k=0}^{x} f(k) & \text{si } 0 \leq x < n \\ 1 & \text{si } x \geq n \end{cases}$$

- **Esperanza** $E(X) = np$
- **Varianza** $Var(X) = npq$

Atención. Observemos que la distribución de Bernoulli es un caso particular de la Binomial. Basta tomar $n = 1$ y tendremos que $X \sim \text{Be}(p)$ y $X \sim \text{B}(1, p)$ son equivalentes.



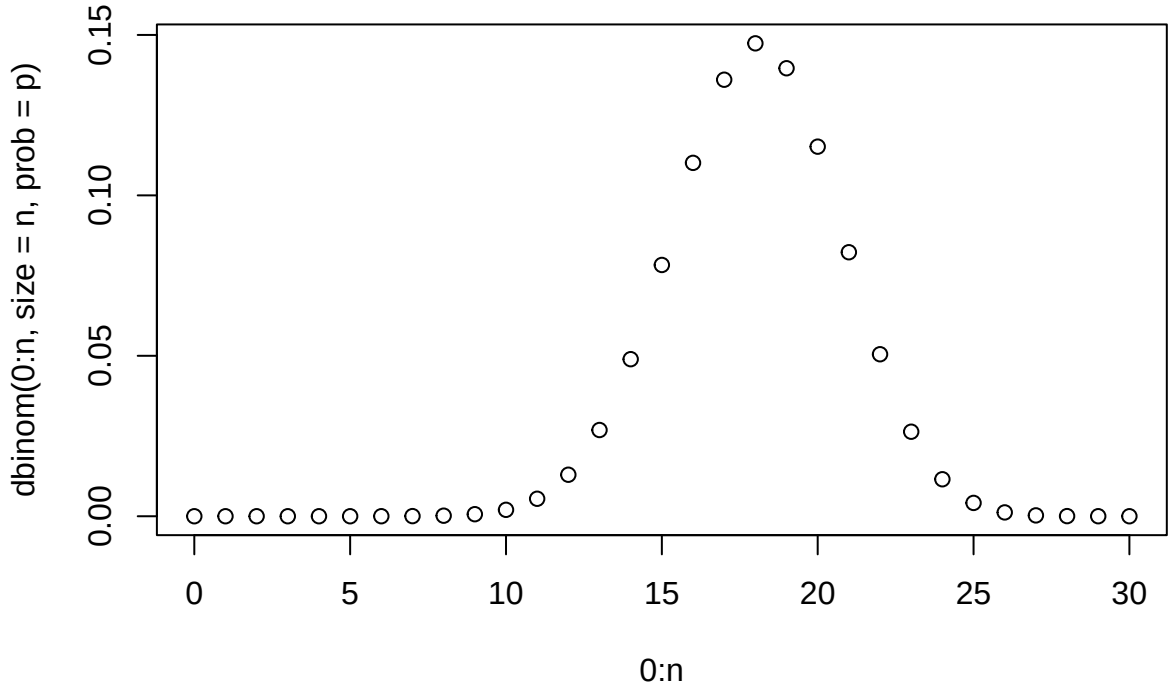
El código de la distribución Binomial:

- En `R` tenemos las funciones del paquete `Rlab`: `dbinom(x, size, prob)`, `pbinom(q,size, prob)`, `qbinom(p, size, prob)`, `rbinom(n, size, prob)` donde `prob` es la probabilidad de éxito y `size` el número de ensayos del experimento.
- En `Python` tenemos las funciones del paquete `scipy.stats.binom`: `pmf(k,n,p)`, `cdf(k,n,p)`, `ppf(q,n,p)`, `rvs(n, p, size)` donde `p` es la probabilidad de éxito y `n` el número de ensayos del experimento.

Función de densidad Sea $X = B(n = 30, p = 0.6)$,

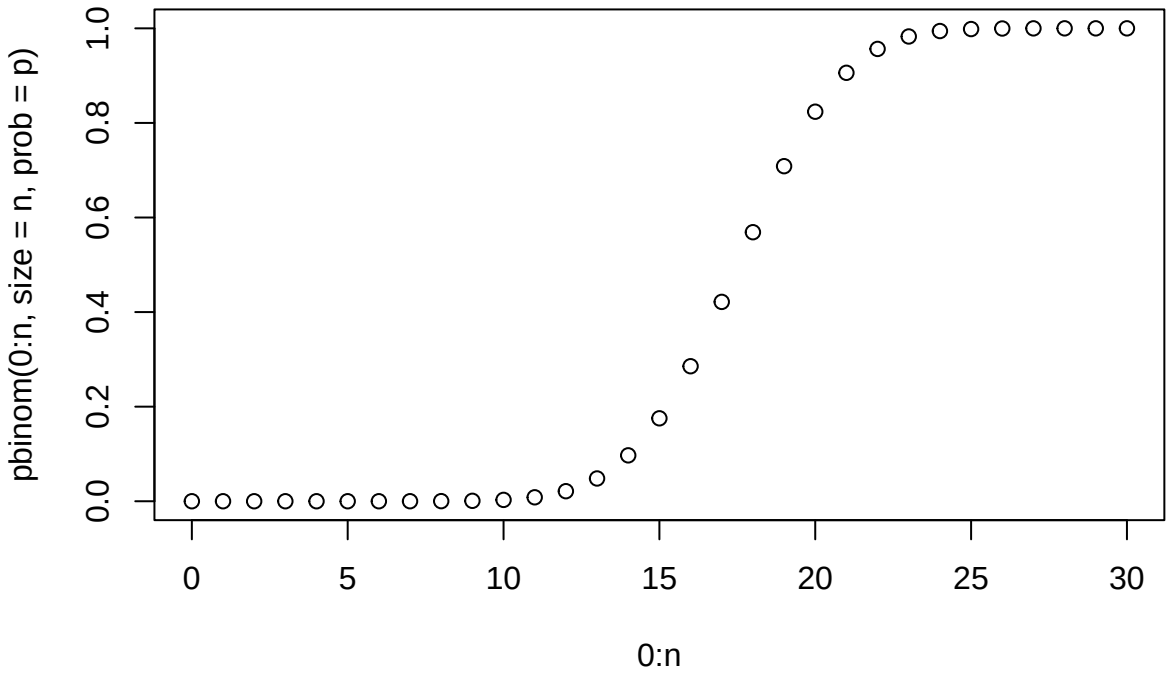
TODO: escribir la FDens y la FDistr

```
library(Rlab)
n = 30
p = 0.6
plot(0:n, dbinom(0:n, size = n, prob = p))
```



En R

```
plot(0:n, pbinom(0:n, size = n, prob = p))
```



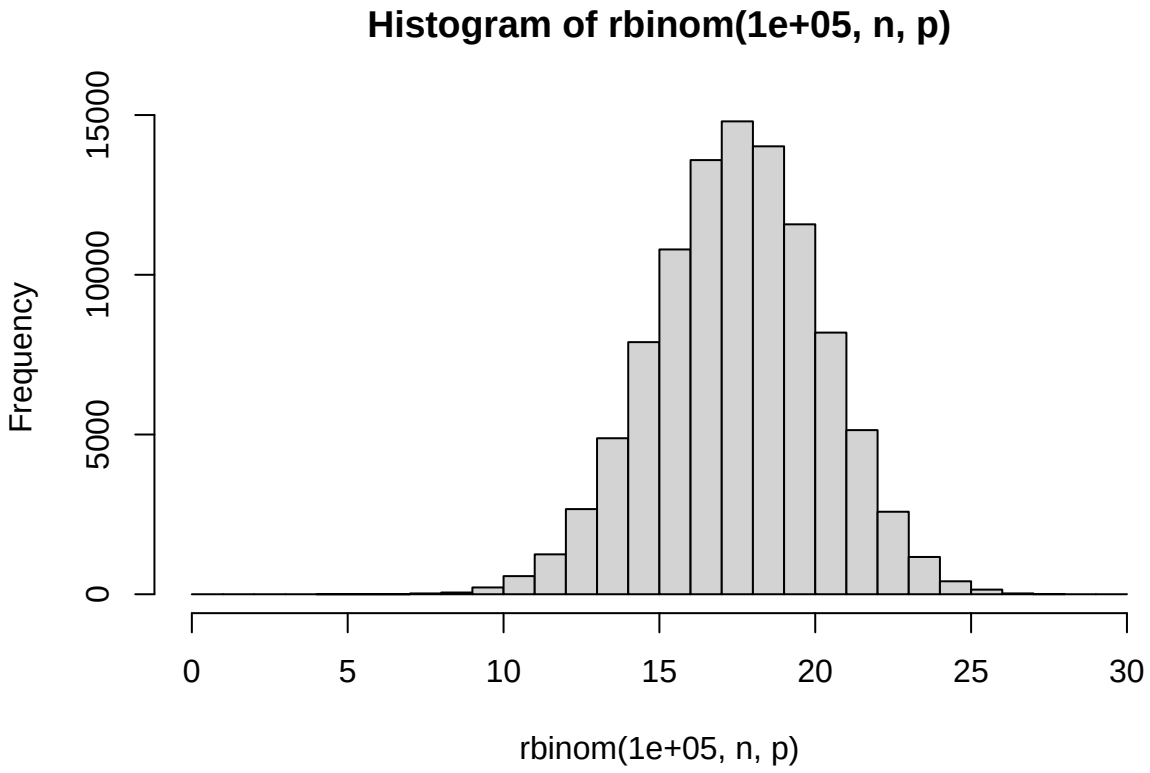

```
qbinom(0.5, n, p)
```

```
## [1] 18
```

```
qbinom(0.25, n, p)
```

```
## [1] 16
```

```
hist(rbinom(100000, n, p), breaks = 0:30)
```




```
from scipy.stats import binom
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(1,1)
n = 7
p = 0.4

mean, var, skew, kurt = binom.stats(n, p, moments = 'mvsk')

print("Media %f"%mean)
```

En Python

```
## Media 2.800000
```

```
print("Varianza %f"%var)
```

```
## Varianza 1.680000
```

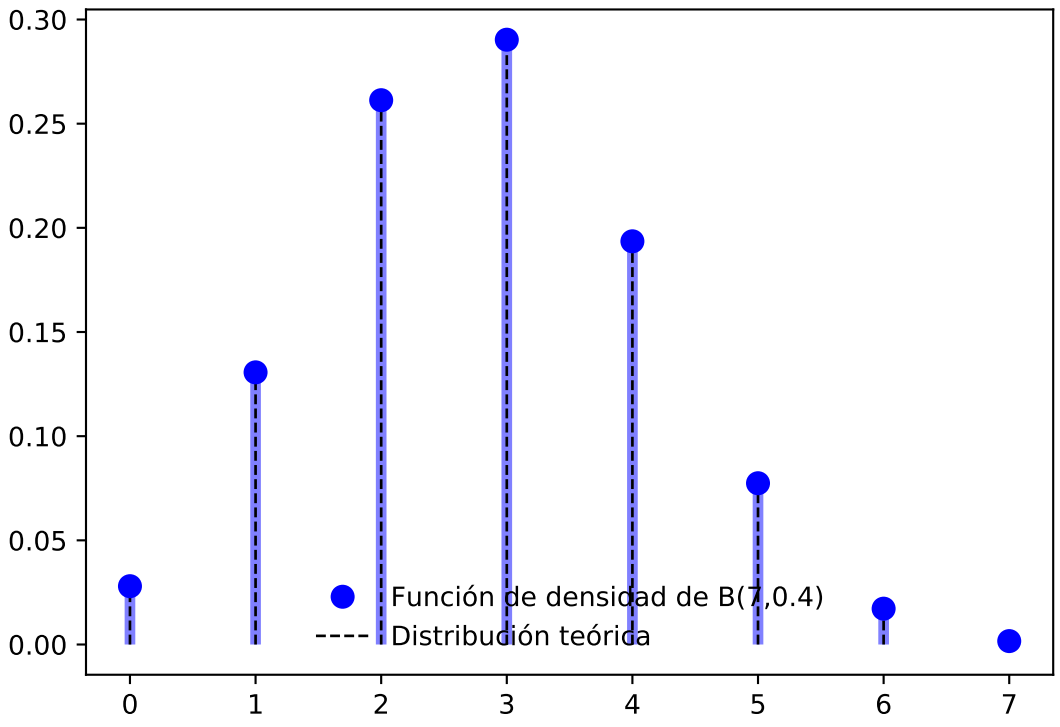
```
print("Sesgo %f"%skew)
```

```
## Sesgo 0.154303
```

```
print("Curtosis %f"%kurt)
```

```
## Curtosis -0.261905
```

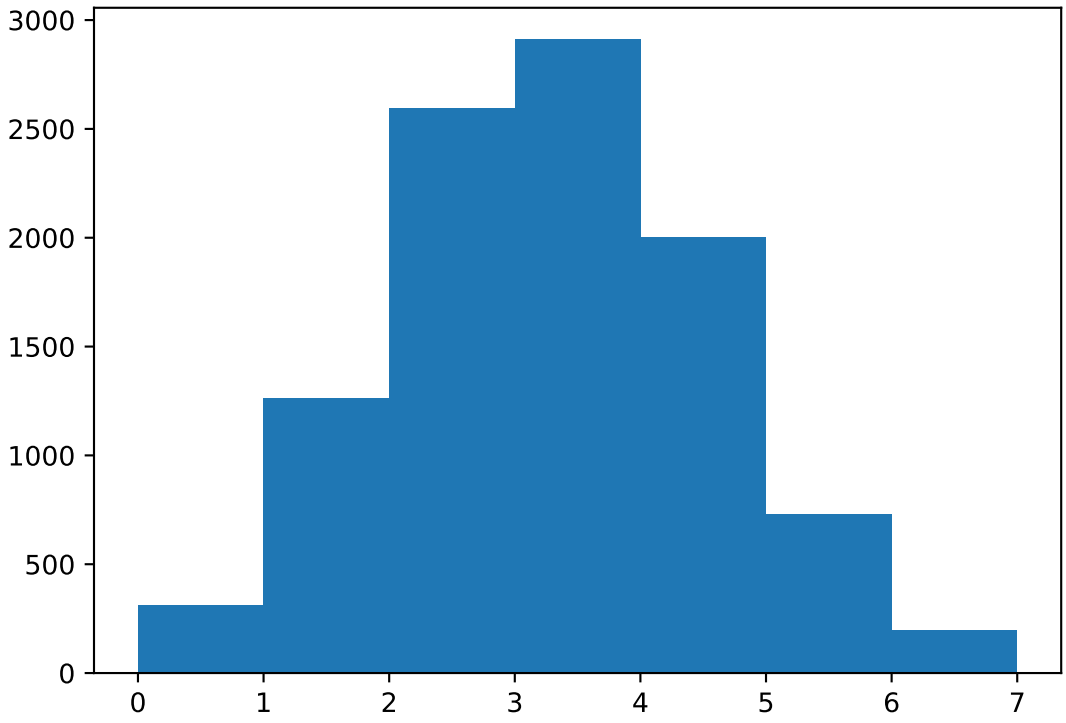
```
x = np.arange(0, n+1)
ax.plot(x, binom.pmf(x, n, p), 'bo', ms = 8, label = "Función de densidad de B(7,0.4)")
ax.vlines(x, 0, binom.pmf(x,n,p), colors = 'b', lw = 4, alpha = 0.5)

rv = binom(n,p)
ax.vlines(x,0, rv.pmf(x), colors = 'k', linestyles='--', lw = 1, label = "Distribución teórica")

ax.legend(loc = 'best', frameon = False)

plt.show()
```




```
fix, ax = plt.subplots(1,1)
r = binom.rvs(n, p, size = 10000)
ax.hist(r, bins = n)
```

```
## (array([ 310., 1260., 2592., 2911., 2001.,  730.,  196.]), array([0., 1., 2., 3., 4., 5., 6., 7.]),
```

```
plt.show()
```



Distribución Geométrica

Si X es variable aleatoria que mide el "número de repeticiones independientes del experimento hasta haber conseguido éxito", diremos que X se distribuye como una Geométrica con parámetro p

$$X \sim \text{Ge}(p)$$

donde p es la probabilidad de éxito y $q = 1 - p$ es la probabilidad de fracaso

- El **dominio** de X será $D_X = \{0, 1, 2, \dots\}$ o bien $D_X = \{1, 2, \dots\}$ en función de si empieza en 0 o en 1, respectivamente
- La **función de probabilidad** vendrá dada por

$$f(k) = (1-p)^k p \qquad \text{si empieza en 0}$$

$$f(k) = (1-p)^{k-1} p \qquad \text{si empieza en 1}$$

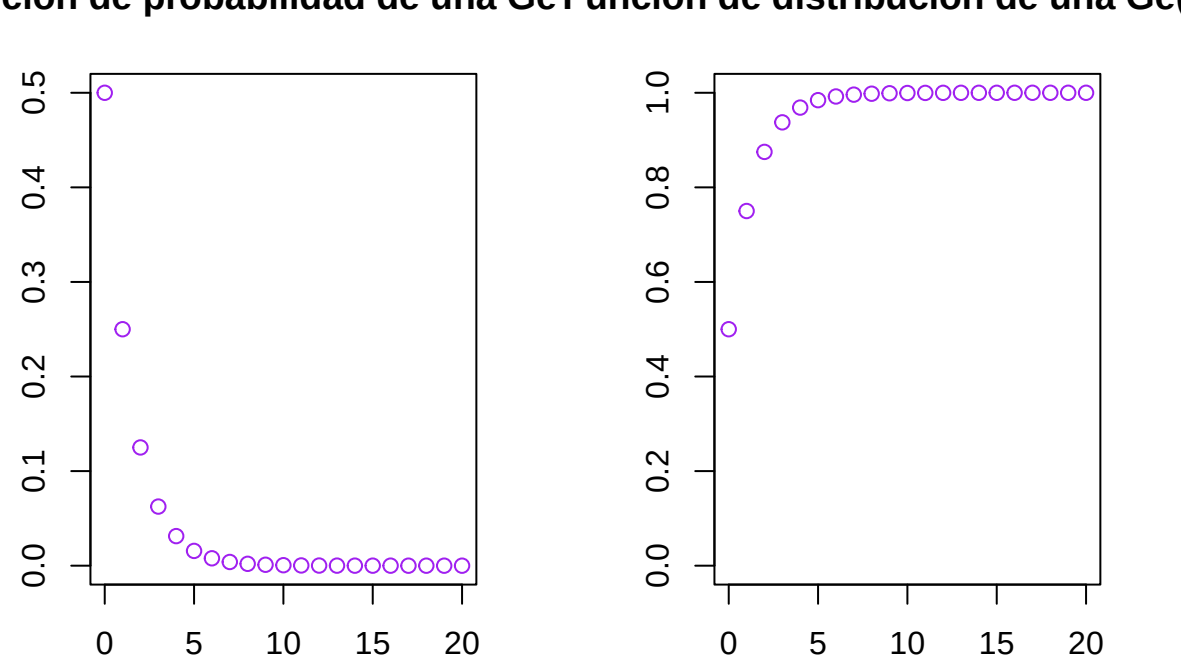
- La **función de distribución** vendrá dada por

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 - (1-p)^{k+1} & \text{si } k \leq x < k+1,\ k \in \mathbb{N} \end{cases}$$

- **Esperanza** $E(X) = \frac{1-p}{p}$ si empieza en 0 y $\text{E}(X) = \frac{1}{p}$ si empieza en 1

- **Varianza** $Var(X) = \frac{1-p}{p^2}$
- Propiedad de la falta de memoria. Si X es una v.a. Ge(p), entonces,

$$p\{X \geq m+n : \ X \geq n\} = p\{X \geq m\} \ \forall m, n = 0, 1, \ldots$$



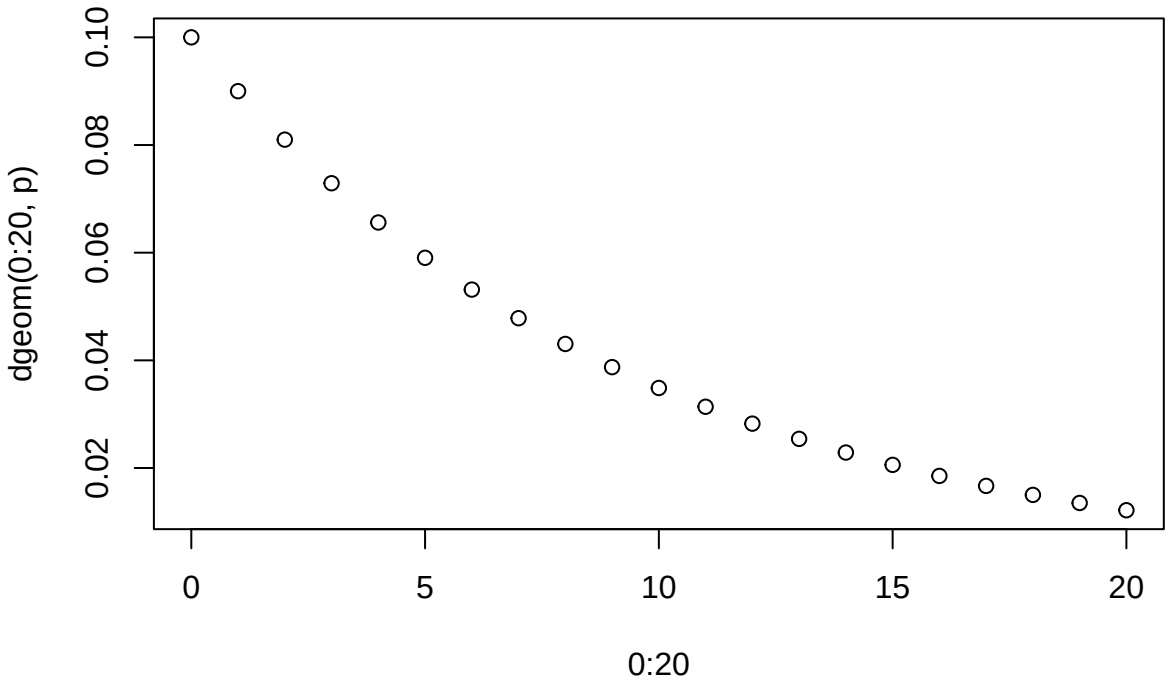
El código de la distribución Geométrica:

- En `R` tenemos las funciones del paquete `Rlab`: `dgeom(x, prob)`, `pgeom(q, prob)`, `qgeom(p, prob)`, `rgeom(n, prob)` donde `prob` es la probabilidad de éxito del experimento.
- En `Python` tenemos las funciones del paquete `scipy.stats.geom`: `pmf(k,p)`, `cdf(k,p)`, `ppf(q,p)`, `rvs(p, size)` donde `p` es la probabilidad de éxito del experimento.

Función de densidad Sea $X = Geom(p = 0.1)$ la distribución que modela la probabilidad de intentar abrir una puerta hasta conseguirlo.

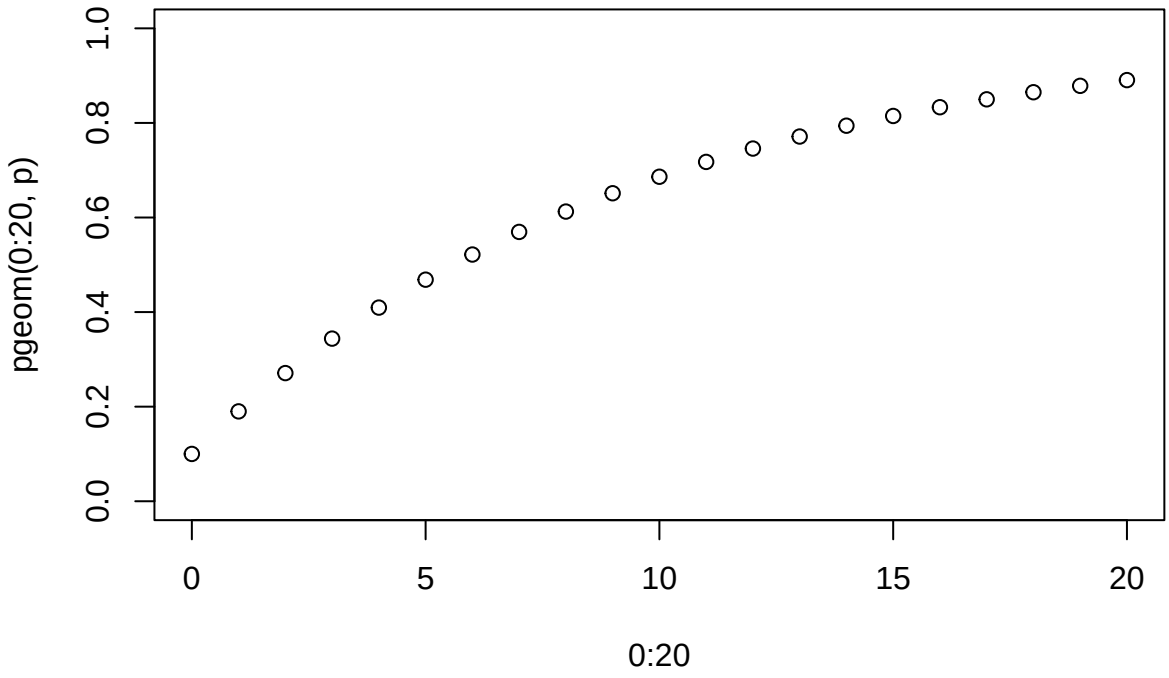
$$f(k) = (1-p)^{k-1}p$$

```
library(Rlab)
p = 0.1
plot(0:20, dgeom(0:20, p))
```



En R

```
plot(0:20, pgeom(0:20, p), ylim = c(0,1))
```



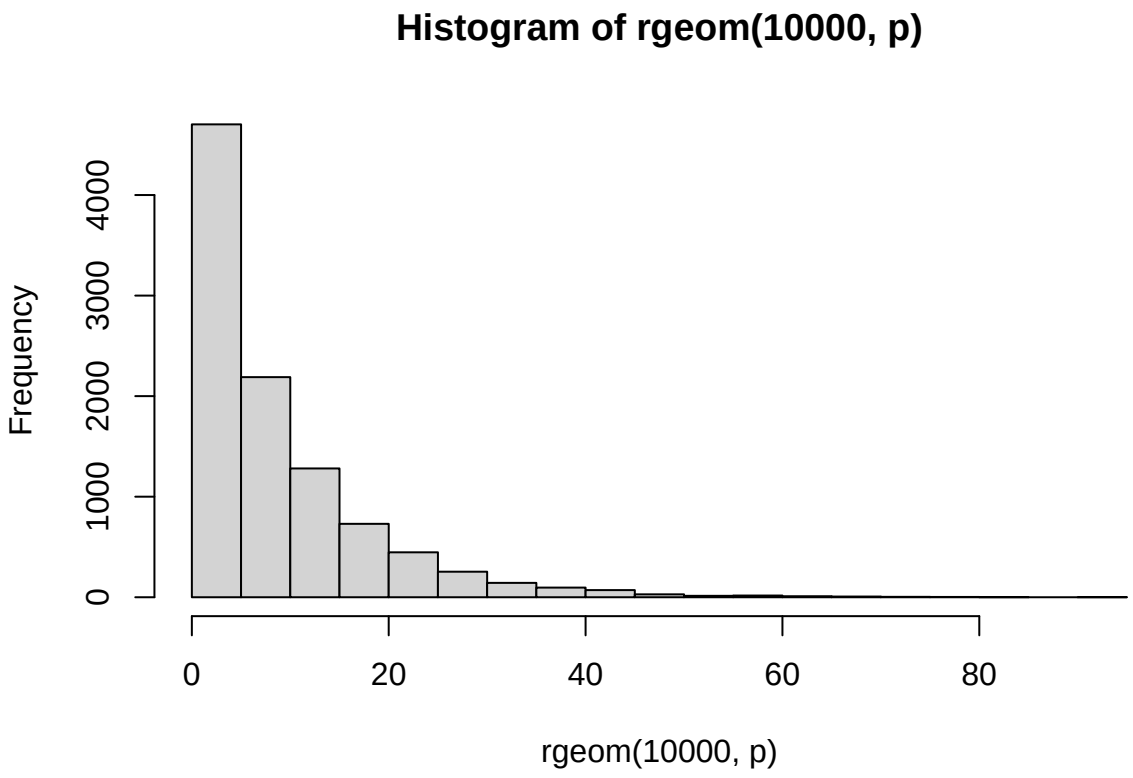

```
qgeom(0.5, p)
```

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

```
qgeom(0.75, p)
```

```
## [1] 13
```

```
hist(rgeom(10000, p))
```




```
from scipy.stats import geom
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(1,1)
p = 0.3
mean, var, skew, kurt = geom.stats(p, moments = 'mvsk')
print("Media %f"%mean)
```

En Python

```
## Media 3.333333
```

```
print("Varianza %f"%var)
```

```
## Varianza 7.777778
```

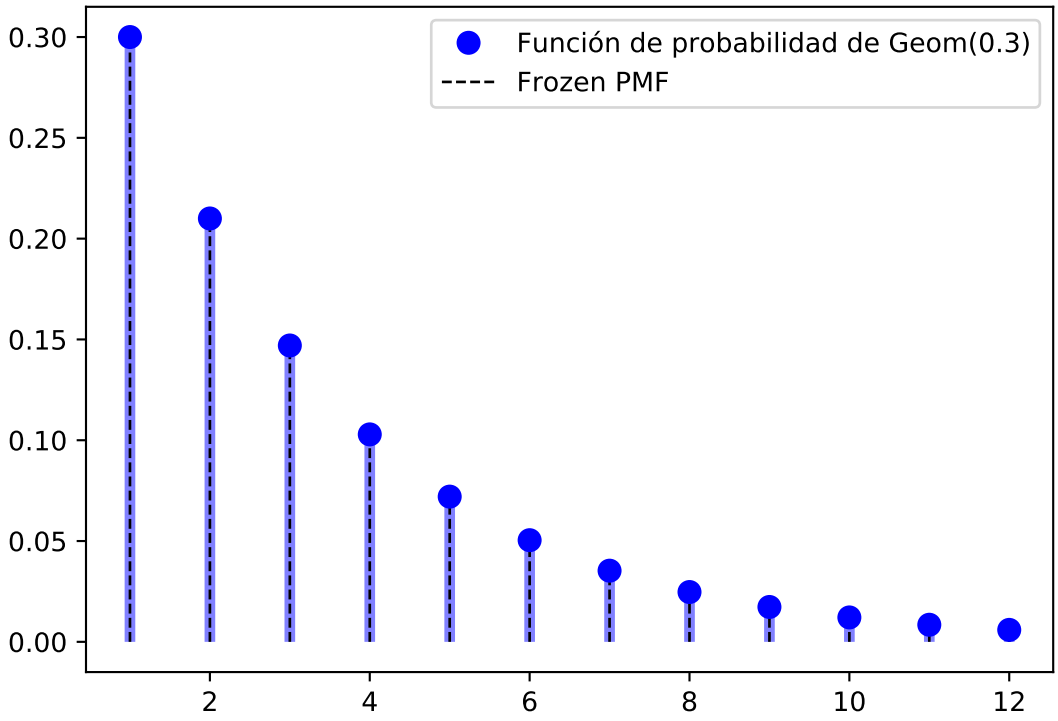
```
print("Sesgo %f"%skew)
```

```
## Sesgo 2.031889
```

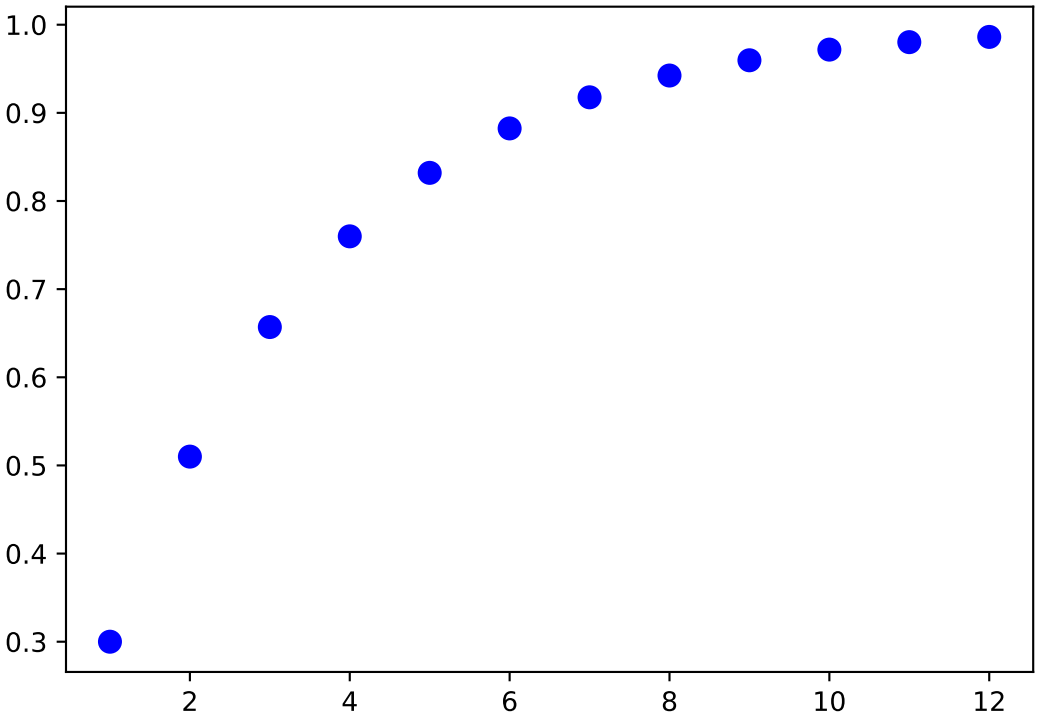
```
print("Curtosis %f"%kurt)
```

```
## Curtosis 6.128571
```

```
x = np.arange(geom.ppf(0.01,p), geom.ppf(0.99, p))
ax.plot(x, geom.pmf(x, p), 'bo', ms = 8, label = "Función de probabilidad de Geom(0.3)")
ax.vlines(x,0,geom.pmf(x,p),  colors = 'b', lw = 4, alpha = 0.5)

rv = geom(p)
ax.vlines(x,0,rv.pmf(x), colors = 'k', linestyles = '--', lw = 1, label = "Frozen PMF")
ax.legend(loc = 'best')
plt.show()
```



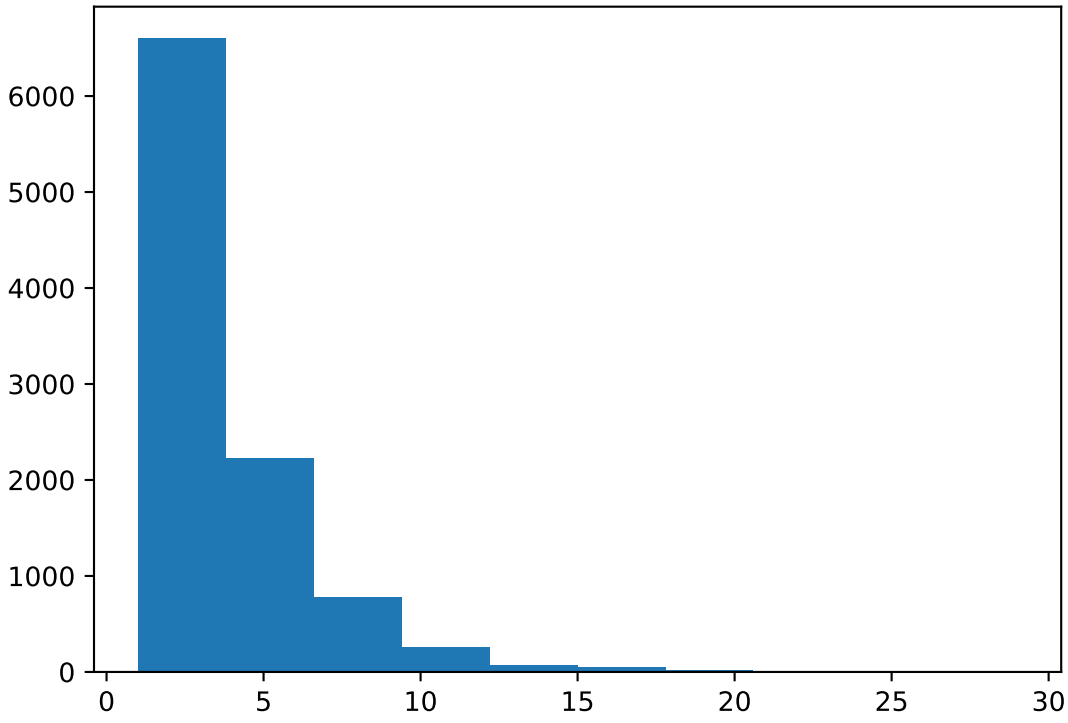

```
fig, ax = plt.subplots(1,1)
prob = geom.cdf(x,p)
ax.plot(x, prob, 'bo', ms = 8, label = "Función de distribución acumulada")
plt.show()
```




```
fig, ax = plt.subplots(1,1)
r = geom.rvs(p, size = 10000)
plt.hist(r)
```

```
## (array([6.601e+03, 2.223e+03, 7.730e+02, 2.600e+02, 6.800e+01, 4.900e+01,
##         1.700e+01, 5.000e+00, 1.000e+00, 3.000e+00]), array([ 1. ,  3.8,  6.6,  9.4, 12.2, 15. , 17.8
```

```
plt.show()
```



Distribución Hipergeométrica

Consideremos el experimento "extraer a la vez (o una detrás de otra, sin retornarlos) n objetos donde hay N de tipo A y M de tipo B". Si X es variable aleatoria que mide el "número de objetos del tipo A", diremos que X se distribuye como una Hipergeométrica con parámetros N, M, n

$$X \sim \mathrm{H}(N, M, n)$$

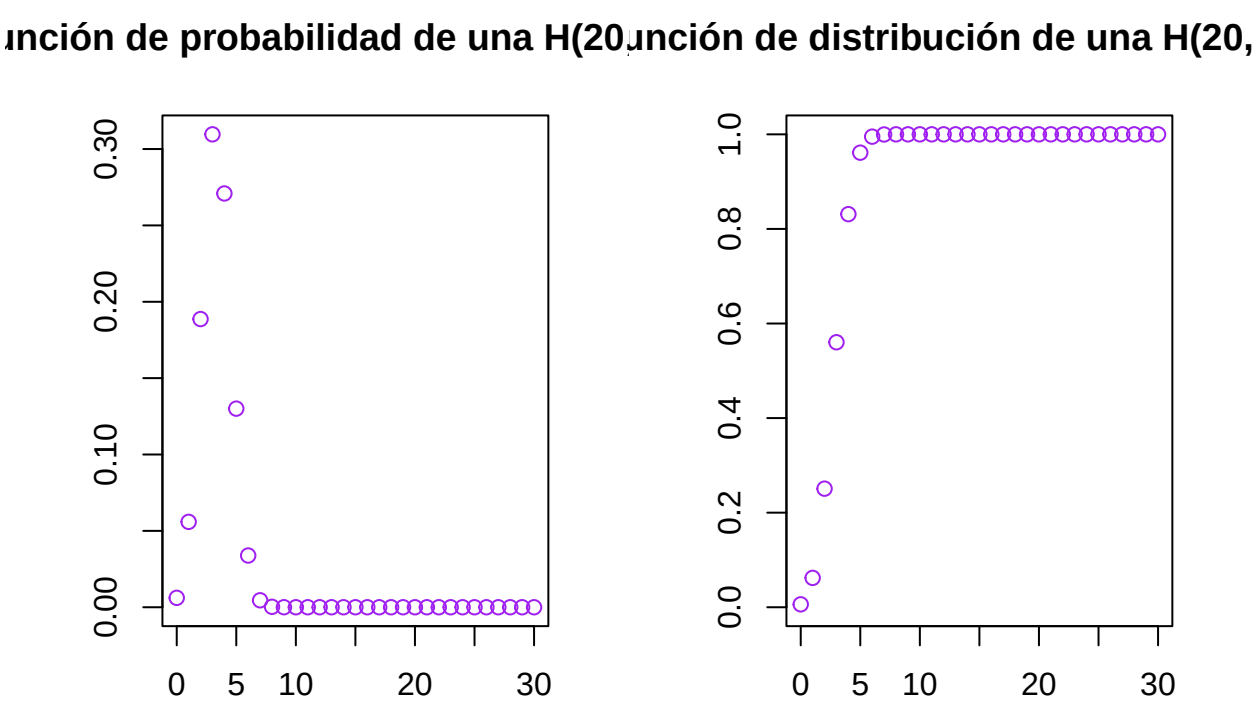
- El dominio de X será $D_X = \{0, 1, 2, \ldots, N\}$ (en general)
- La función de probabilidad vendrá dada por

$$f(k) = \frac{\binom{n}{k}\binom{M}{n-k}}{\binom{N+N}{n}}$$

La **función de distribución** vendrá dada por

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ \sum_{k=0}^{x} f(k) & \text{si } 0 \leq x < n \\ 1 & \text{si } x \geq n \end{cases}$$

- **Esperanza** $E(X) = \frac{nN}{N+M}$ - **Varianza** $Var(X) = \frac{nNM}{(N+M)^2} \cdot \frac{N+M-n}{N+M-1}$



El código de la distribución Hipergeométrica:

- En `R` tenemos las funciones del paquete `Rlab`: `dhyper(x, m, n, k)`, `phyper(q, m, n, k)`, `qhyper(p, m, n, k)`, `rhyper(nn, m, n, k)` donde `m` es el número de objetos del primer tipo, `n` el número de objetos del segundo tipo y `k` el número de extracciones realizadas.
- En `Python` tenemos las funciones del paquete `scipy.stats.hypergeom`: `pmf(k,M, n, N)`, `cdf(k,M, n, N)`, `ppf(q,M, n, N)`, `rvs(M, n, N, size)` donde `M` es el número de objetos del primer tipo, `N` el número de objetos del segundo tipo y `n` el número de extracciones realizadas.

Supongamos que tenemos 20 animales, de los cuales 7 son perros. Queremos medir la probabilidad de encontrar un número determinado de perros si elegimos $k = 12$ animales al azar.

```
library(Rlab)
M = 7
N = 13
k = 12
dhyper(x = 0:12, m = M, n = N, k = k)
```

En `R`

```
##  [1] 0.0001031992 0.0043343653 0.0476780186 0.1986584107 0.3575851393
##  [6] 0.2860681115 0.0953560372 0.0102167183 0.0000000000 0.0000000000
## [11] 0.0000000000 0.0000000000 0.0000000000
```

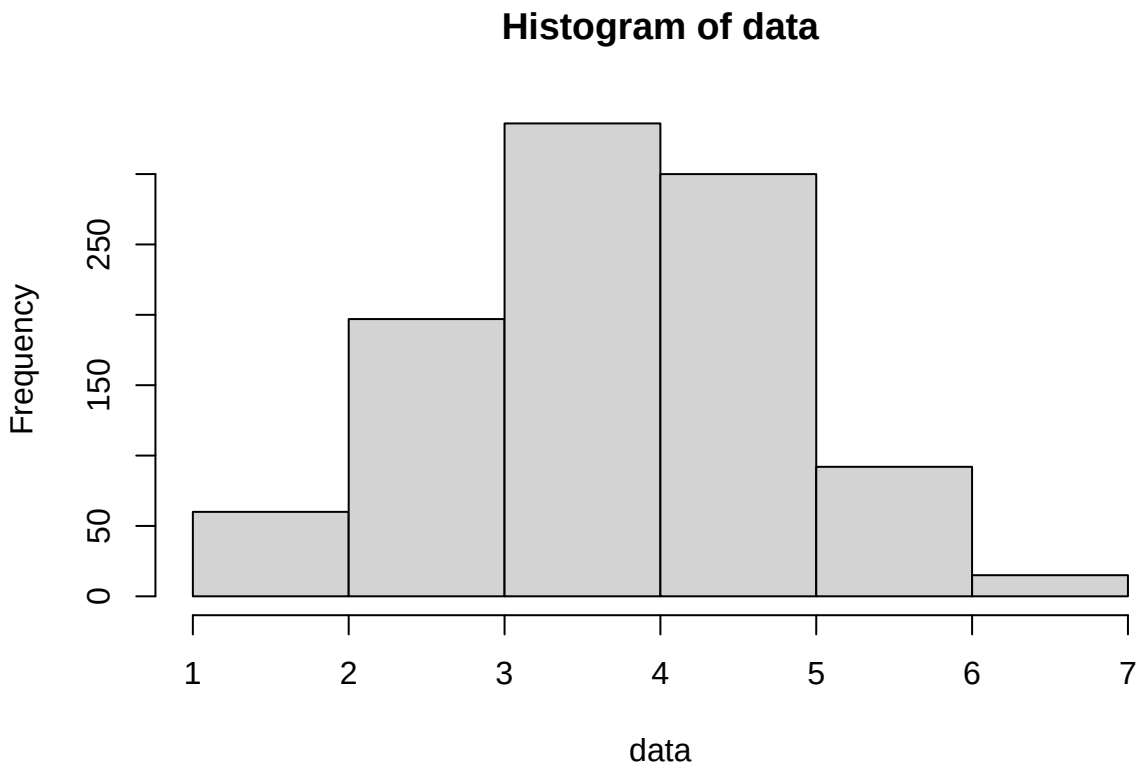
```r
phyper(q = 0:12, m = M, n = N, k = k)
```

```
##  [1] 0.0001031992 0.0044375645 0.0521155831 0.2507739938 0.6083591331
##  [6] 0.8944272446 0.9897832817 1.0000000000 1.0000000000 1.0000000000
## [11] 1.0000000000 1.0000000000 1.0000000000
```

```r
qhyper(p = 0.5, m = M, n = N, k = k)
```

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

```r
rhyper(nn = 1000, m = M, n = N, k = k) -> data
hist(data, breaks = 8)
```




```python
from scipy.stats import hypergeom
import matplotlib.pyplot as plt
import numpy as np

[M, n, N] = [20, 7, 6]
rv = hypergeom(M, n, N)
x = np.arange(0, n+1)
y = rv.pmf(x)
```

```
mean, var, skew, kurt = rv.stats(moments = 'mvsk')
print("Media %f"%mean)
```

En Python

```
## Media 2.100000
```

```
print("Varianza %f"%var)
```

```
## Varianza 1.005789
```

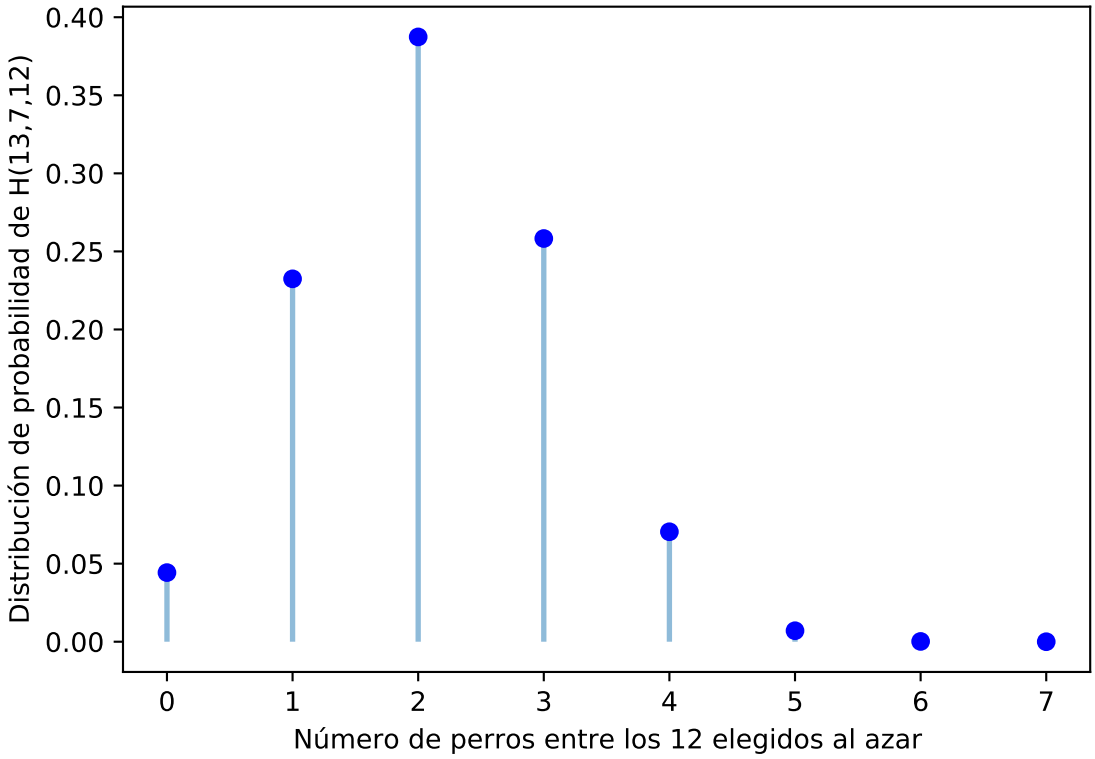
```
print("Sesgo %f"%skew)
```

```
## Sesgo 0.132949
```

```
print("Curtosis %f"%kurt)
```

```
## Curtosis -0.203835
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x, y, 'bo' )
ax.vlines(x,0,y, lw = 2, alpha = 0.5)
ax.set_xlabel("Número de perros entre los 12 elegidos al azar")
ax.set_ylabel("Distribución de probabilidad de H(13,7,12)")
plt.show()
```



Distribución de Poisson

Si X es variable aleatoria que mide el "número de eventos en un cierto intervalo de tiempo", diremos que X se distribuye como una Poisson con parámetro λ

$$X \sim \text{Po}(\lambda)$$

donde λ representa el número de veces que se espera que ocurra el evento durante un intervalo dado

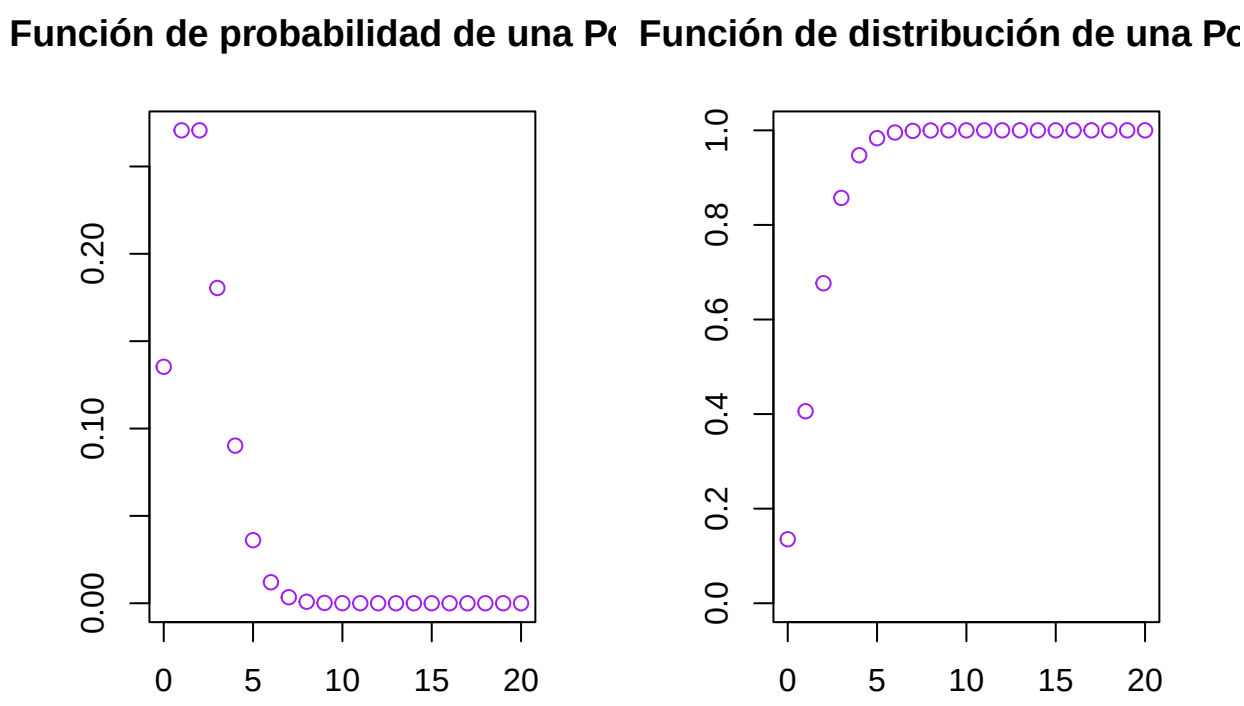
- El **dominio** de X será $D_X = \{0, 1, 2, \dots\}$
- La **función de probabilidad** vendrá dada por

$$f(k) = \frac{e^{-\lambda}\lambda^k}{k!}$$

La **función de distribución** vendrá dada por

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ \sum_{k=0}^{x} f(k) & \text{si } 0 \leq x < n \\ 1 & \text{si } x \geq n \end{cases}$$

- **Esperanza** $E(X) = \lambda$ - **Varianza** $Var(X) = \lambda$

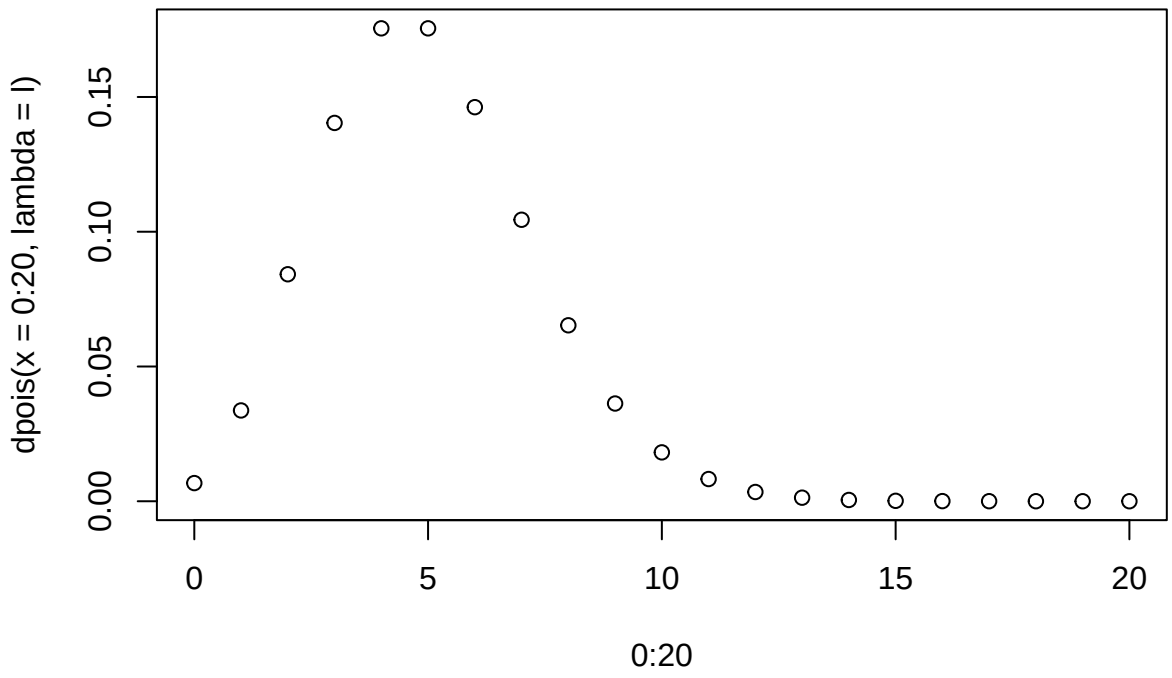


El código de la distribución de Poisson:

- En `R` tenemos las funciones del paquete `Rlab`: `dpois(x, lambda)`, `ppois(q,lambda)`, `qpois(p,lambda)`, `rpois(n, lambda)` donde `lambda` es el número esperado de eventos por unidad de tiempo de la distribución.
- En `Python` tenemos las funciones del paquete `scipy.stats.poisson`: `pmf(k,mu)`, `cdf(k,mu)`, `ppf(q,mu)`, `rvs(M,mu)` donde `mu` es el número esperado de eventos por unidad de tiempo de la distribución.

Supongamos que X modela el número de errores por página que tiene un valor esperado $\lambda = 5$.

```
l = 5
plot(0:20, dpois(x = 0:20, lambda = l))
```



En R

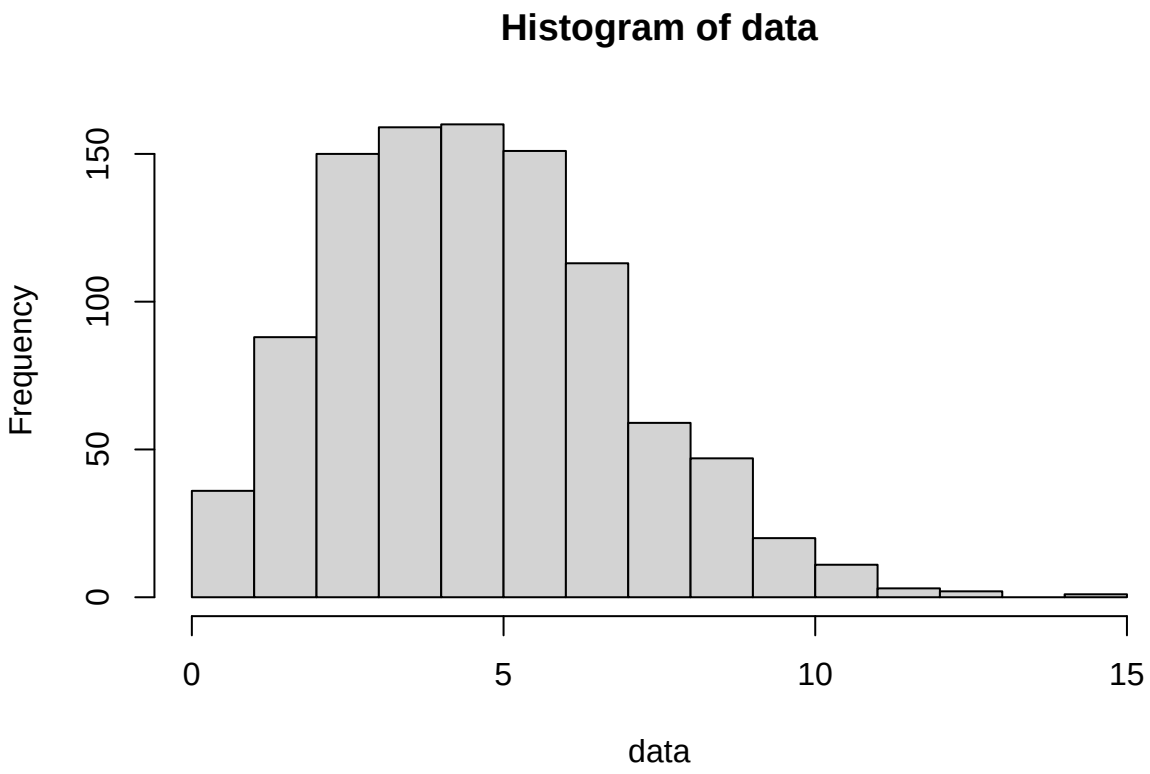
```
ppois(0:20, l)
```

```
##  [1] 0.006737947 0.040427682 0.124652019 0.265025915 0.440493285 0.615960655
##  [7] 0.762183463 0.866628326 0.931906365 0.968171943 0.986304731 0.994546908
## [13] 0.997981148 0.999302010 0.999773746 0.999930992 0.999980131 0.999994584
## [19] 0.999998598 0.999999655 0.999999919
```

```
qpois(0.5, 5)
```

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

```
rpois(1000, lambda = l) -> data
hist(data)
```




```
import numpy as np
from scipy.stats import poisson
import matplotlib.pyplot as plt


fig, ax = plt.subplots(1,1)
mu = 5
mean, var, skew, kurt = poisson.stats(mu, moments = 'mvsk')
print("Media %f"%mean)
```

En Python

```
## Media 5.000000
```

```
print("Varianza %f"%var)
```

```
## Varianza 5.000000
```

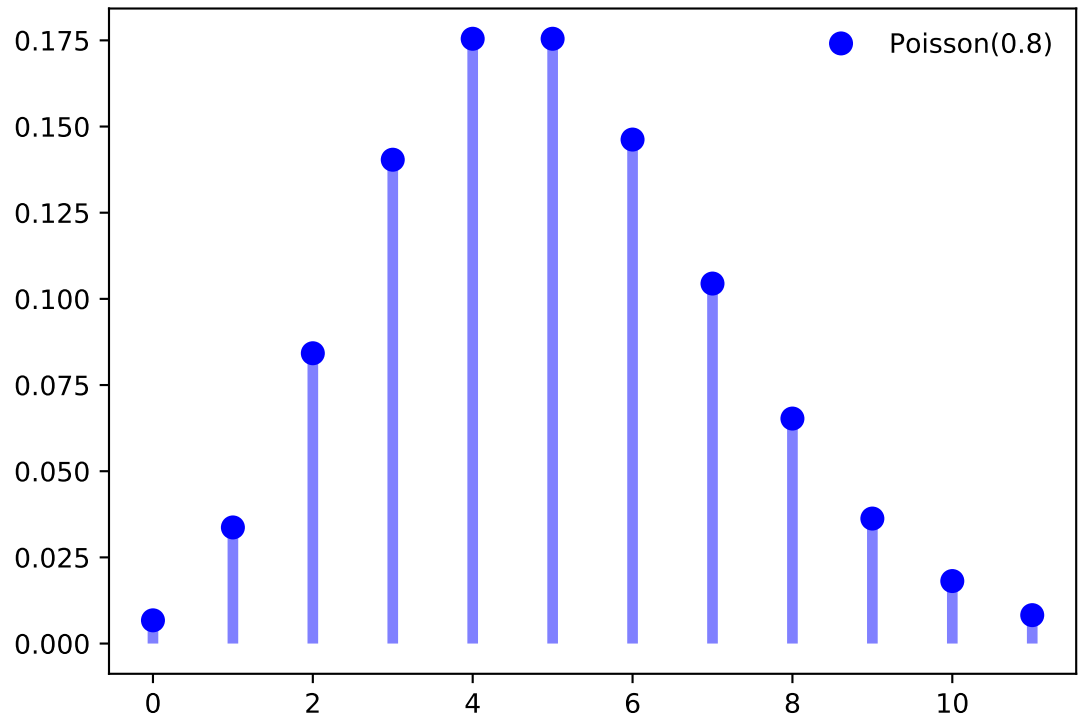
```
print("Sesgo %f"%skew)
```

```
## Sesgo 0.447214
```

```
print("Curtosis %f"%kurt)
```

```
## Curtosis 0.200000
```

```
x = np.arange(0, 12)
ax.plot(x, poisson.pmf(x, mu), 'bo', ms = 8, label = 'Poisson(0.8)')
ax.vlines(x,0, poisson.pmf(x,mu), colors = 'b', lw = 4, alpha = 0.5)
ax.legend(loc = "best", frameon = False)
plt.show()
```



Distribución Binomial Negativa

Si X es variable aleatoria que mide el "número de repeticiones hasta observar los r éxitos en ensayos de Bernoulli", diremos que X se distribuye como una Binomial Negativa con parámetros r y p,

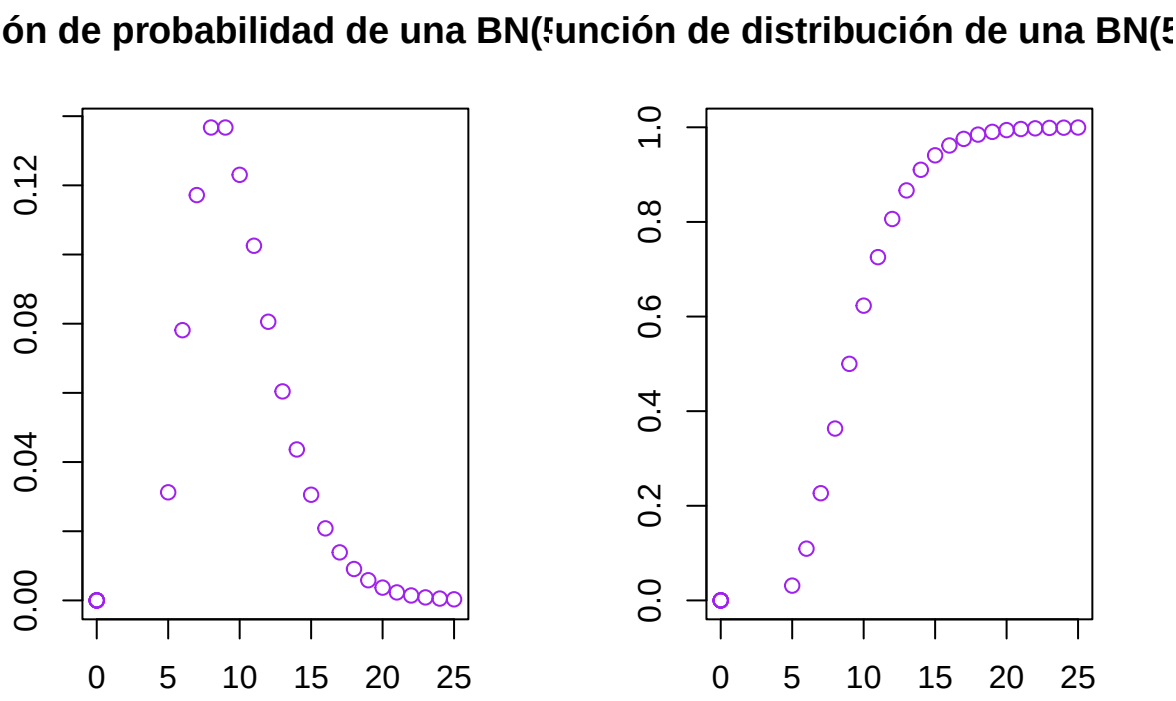
$$X \sim \text{BN}(r, p)$$

donde p es la probabilidad de éxito

- El **dominio** de X será $D_X = \{r, r+1, r+2, \dots\}$
- La **función de probabilidad** vendrá dada por

$$f(k) = \binom{k-1}{r-1} p^r (1-p)^{k-r}, k \geq r$$

- La **función de distribución** no tiene una expresión analítica.
- **Esperanza** $E(X) = \frac{r}{p}$
- **Varianza** $Var(X) = r\frac{1-p}{p^2}$



El código de la distribución Binomial Negativa:

- En `R` tenemos las funciones del paquete `Rlab`: `dnbinom(x, size, prop)`, `pnbinom(q, size, prop)`, `qnbinom(p, size, prop)`, `rnbinom(n, size, prop)` donde `size` es el número de casos exitosos y `prob` la probabilidad del éxito.
- En `Python` tenemos las funciones del paquete `scipy.stats.nbinom`: `pmf(k,n,p)`, `cdf(k,n,p)`, `ppf(q,n,p)`, `rvs(n,p)` donde `n`es el número de casos exitosos y `p` la probabilidad del éxito.

Distribuciones discretas en R

R conoce las distribuciones de probabilidad más importantes.

Distribución	Instrucción en R	Instrucción en Python	Parámetros
Bernoulli	`bern`	`scipy.stats.bernoulli`	probabilidad de éxito p
Binomial	`binom`	`scipy.stats.binom`	tamaño de la muestra n y probabilidad de éxito p
Geométrica	`geom`	`scipy.stats.geom`	probabilidad de éxito p

Distribución	Instrucción en R	Instrucción en Python	Parámetros
Hipergeométrica	`hyper`	`scipy.stats.hypergeom`	N, M, n
Poisson	`pois`	`scipy.stats.poisson`	esperanza λ
Binomial Negativa	`nbinom`	`scipy.stats.nbinom`	número de éxitos r y probabilidad de éxito p

Variables aleatorias continuas

Variable aleatoria continua. Una v.a. $X : \Omega \longrightarrow \mathbb{R}$ es continua cuando su función de distribución $F_X : \mathbb{R} \longrightarrow [0, 1]$ es continua

En este caso, $F_X(x) = F_X(x^-)$ y, por este motivo,

$$p(X = x) = 0 \ \forall x \in \mathbb{R}$$

pero esto no significa que sean sucesos imposibles

Función de densidad Función de densidad. Función $f : \mathbb{R} \longrightarrow \mathbb{R}$ que satisface

- $f(x) \geq 0 \ \forall x \in \mathbb{R}$
- $\int_{-\infty}^{+\infty} f(t)dt = 1$

Una función de densidad puede tener puntos de discontinuidad

Variable aleatoria continua Toda variable aleatoria X con función de distribución

$$F(x) = \int_{-\infty}^{x} f(t)dt \ \forall x \in \mathbb{R}$$

para cualquier densidad f es una v.a. continua

Diremos entonces que f es la función de densidad de X

A partir de ahora, considerareos solamente las v.a. X continuas que tienen función de densidad

Esperanza Esperanza de una v.a. continua. Sea X v.a. continua con densidad f_X. La esperanza de X es

$$E(X) = \int_{-\infty}^{+\infty} x \cdot f_X(x)dx$$

Si el dominio D_X de X es un intervalo de extremos $a < b$, entonces

$$E(X) = \int_{a}^{b} x \cdot f_X(x)dx$$

Sea $g : D_X \longrightarrow \mathbb{R}$ una función continua. Entonces,

$$E(g(X)) = \int_{-\infty}^{+\infty} g(x) \cdot f_X(x)dx$$

Si el dominio D_X de X es un intervalo de extremos $a < b$, entonces

$$E(g(X)) = \int_{a}^{b} g(x) \cdot f_X(x)dx$$

Varianza Varianza de una v.a. continua. Como en el caso discreto,

$$Var(X) = E((X - E(X))^2)$$

y se puede demostrar que

$$Var(X) = E(X^2) - (E(X))^2$$

Desviación típica Desviación típica de una v.a. continua. Como en el caso discreto,

$$\sigma = \sqrt{Var(X)}$$

Distribuciones continuas más conocidas

- Uniforme
- Exponencial
- Normal
- Khi cuadrado
- t de Student
- F de Fisher

Distribución Uniforme

Una v.a. continua X tiene distribución uniforme sobre el intervalo real $[a, b]$ con $a < b$, $X \sim \mathrm{U}(a, b)$ si su función de densidad es

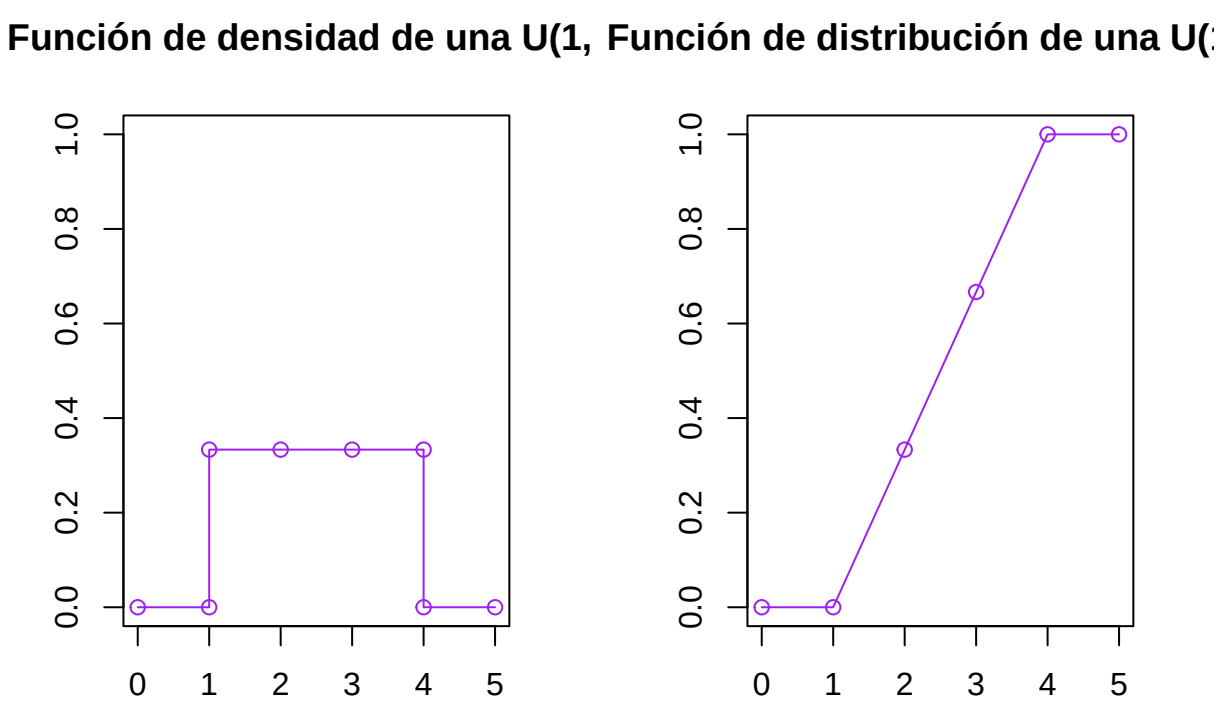
$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{si } a \leq x \leq b \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Modela el elegir un elemento del intervalo $[a, b]$ de manera equiprobable

- El **dominio** de X será $D_X = [a, b]$
- La **función de distribución** vendrá dada por

$$F_X(x) = \begin{cases} 0 & \text{si } x < a \\ \frac{x-a}{b-a} & \text{si } a \leq x < b \\ 1 & \text{si } x \geq b \end{cases}$$

- **Esperanza** $E(X) = \frac{a+b}{2}$
- **Varianza** $Var(X) = \frac{(b-a)^2}{12}$

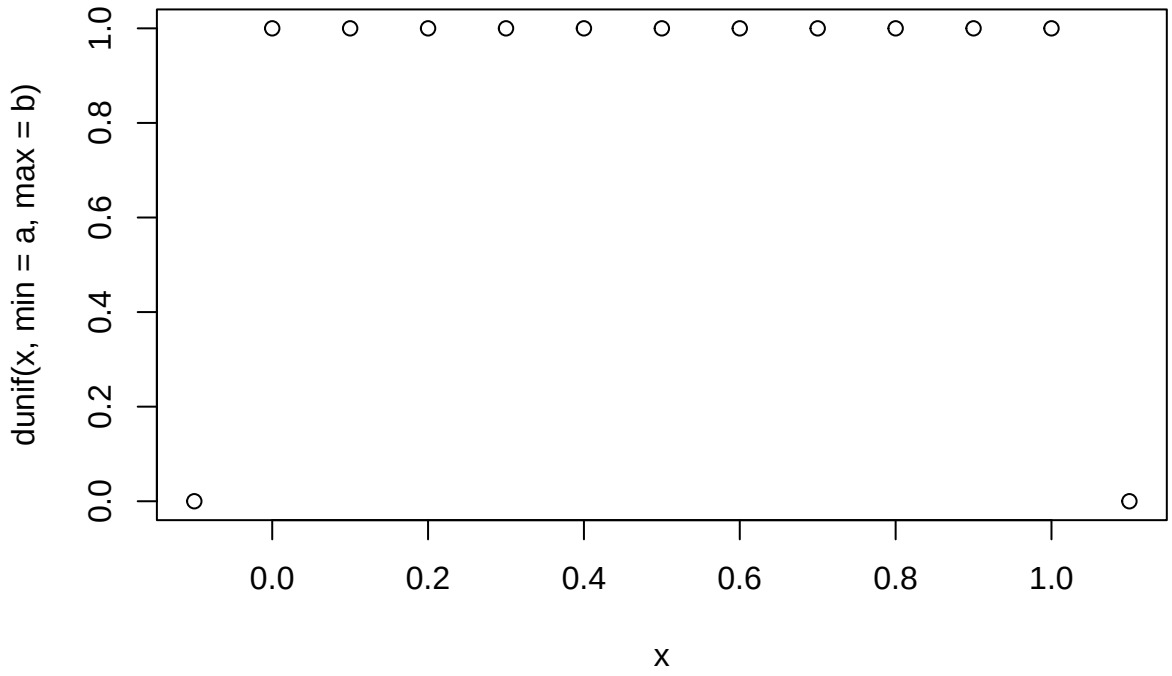


El código de la distribución Uniforme:

- En `R` tenemos las funciones del paquete `stats`: `dunif(x, min, max)`, `punif(q, min, max)`, `qunif(p, min, max)`, `runif(n, min, max)` donde `min` y `max` són los extremos de los intervalos de la distribución uniforme.
- En `Python` tenemos las funciones del paquete `scipy.stats.uniform`: `pdf(k,loc, scale)`, `cdf(k,loc, scale)`, `ppf(q,loc, scale)`, `rvs(n,loc, scaler)` donde la distribución uniforme está definida en el intervalo `[loc, loc+scale]`.

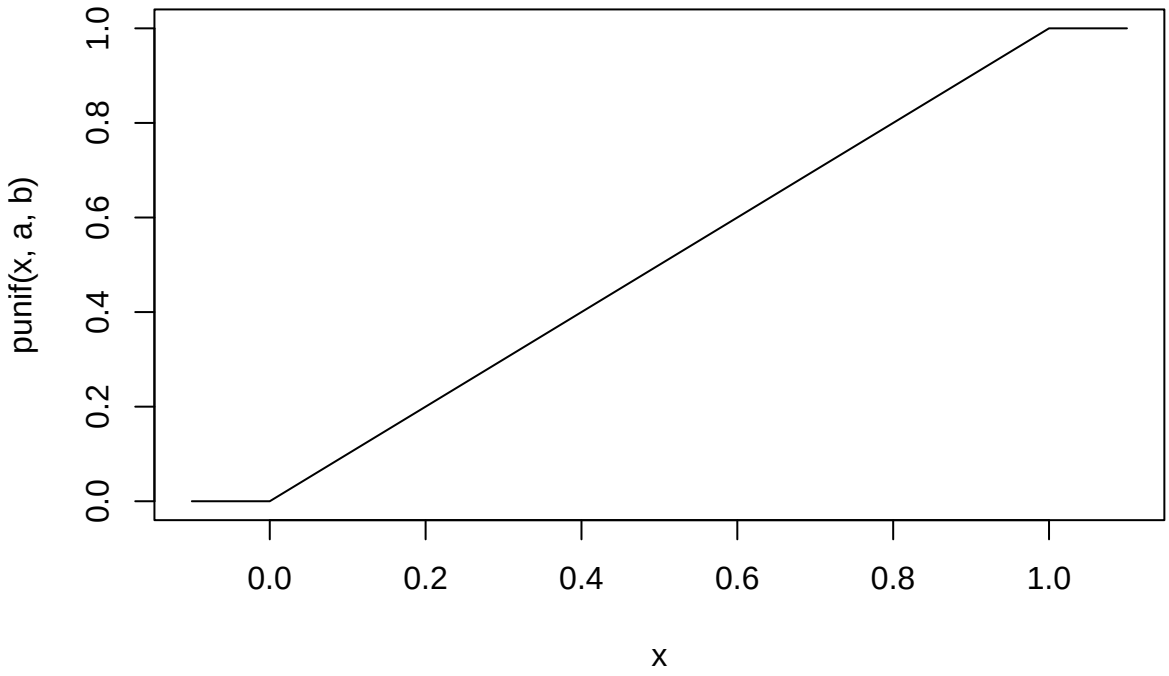
Supongamos que $X \sim U([0,1])$ entonces podemos estudiar sus parámetros

```
a = 0
b = 1

x = seq(-0.1, 1.1, 0.1)
plot(x, dunif(x, min = a, max = b))
```



En R

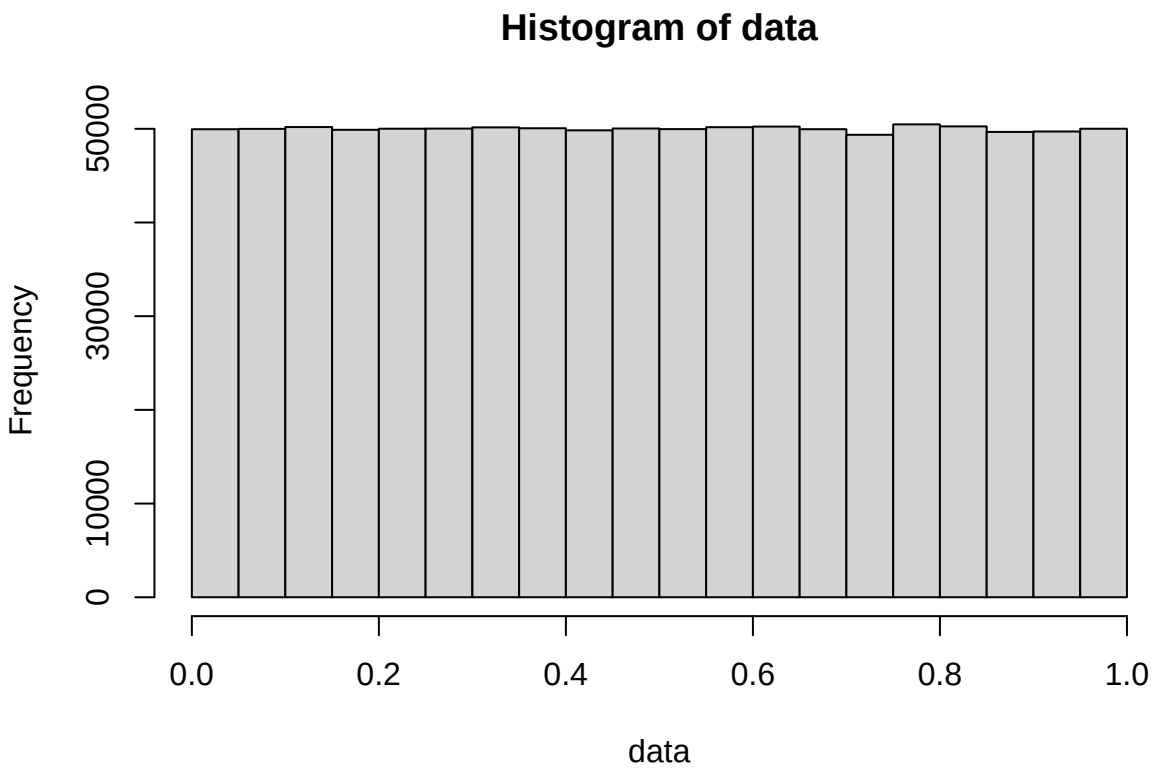
```
plot(x, punif(x, a, b), type = "l")
```




```
qunif(0.5, a, b)
```

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

```
runif(1000000, a, b) -> data
hist(data)
```




```
from scipy.stats import uniform
import matplotlib.pyplot as plt
import numpy as np

a = 0
b = 1

loc = a
scale = b-a

fig, ax = plt.subplots(1,1)

rv = uniform(loc = loc, scale = scale)

mean, var, skew, kurt = rv.stats(moments = 'mvsk')
print("Media %f"%mean)
```

En Python

```
## Media 0.500000
```

```
print("Varianza %f"%var)
```

```
## Varianza 0.083333
```

```
print("Sesgo %f"%skew)
```

```
## Sesgo 0.000000
```

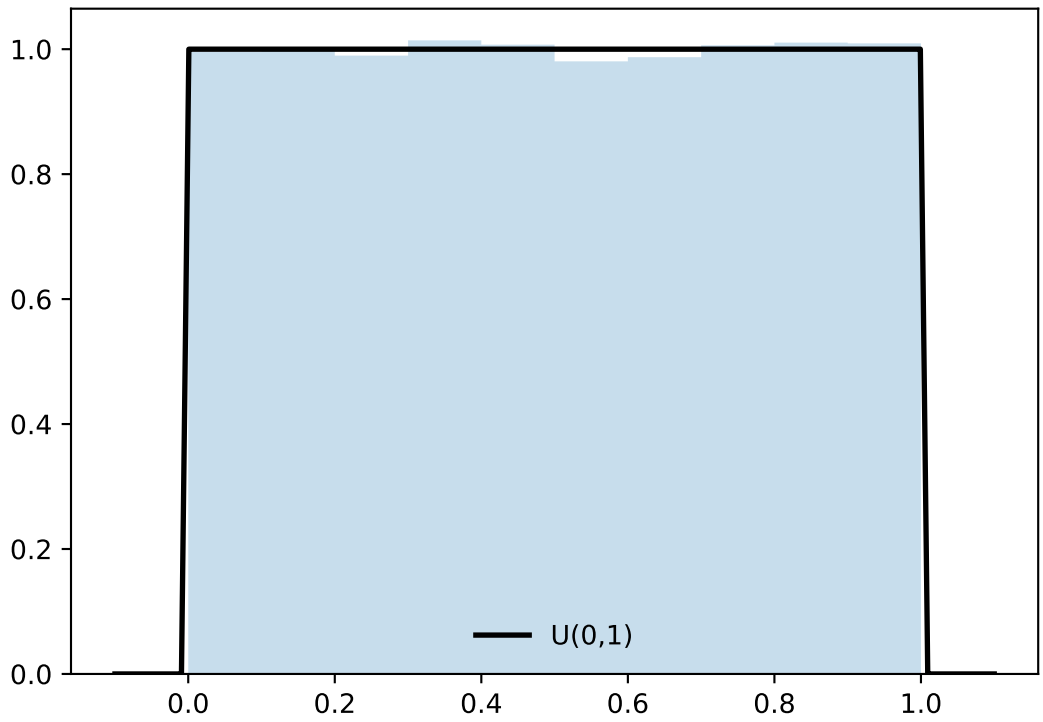
```
print("Curtosis %f"%kurt)
```

```
## Curtosis -1.200000
```

```
x = np.linspace(-0.1, 1.1, 120)
ax.plot(x, rv.pdf(x), 'k-', lw = 2, label = "U(0,1)")

r = rv.rvs(size = 100000)
ax.hist(r, density = True, histtype = "stepfilled", alpha = 0.25)
```

```
## (array([0.99582617, 0.99952627, 0.98972601, 1.01412665, 1.00722647,
##         0.98052577, 0.98722595, 1.00622645, 1.01052656, 1.00932653]), array([8.01596185e-06, 1.000053
##         3.99997503e-01, 4.99994874e-01, 5.99992246e-01, 6.99989618e-01,
##         7.99986990e-01, 8.99984361e-01, 9.99981733e-01]), [<matplotlib.patches.Polygon object at 0x00
```

```
ax.legend(loc = 'best', frameon = False)
plt.show()
```



Distribución Exponencial

Una v.a. X tiene distribución exponencial de parámetro λ, $X \sim \text{Exp}(\lambda)$, si su función de densidad es

$$f_X(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ \lambda \cdot e^{-\lambda x} & \text{si } x > 0 \end{cases}$$

Teorema. Si tenemos un proceso de Poisson de parámetro λ por unidad de tiempo, el tiempo que pasa entre dos sucesos consecutivos es una v.a. Exp(λ)

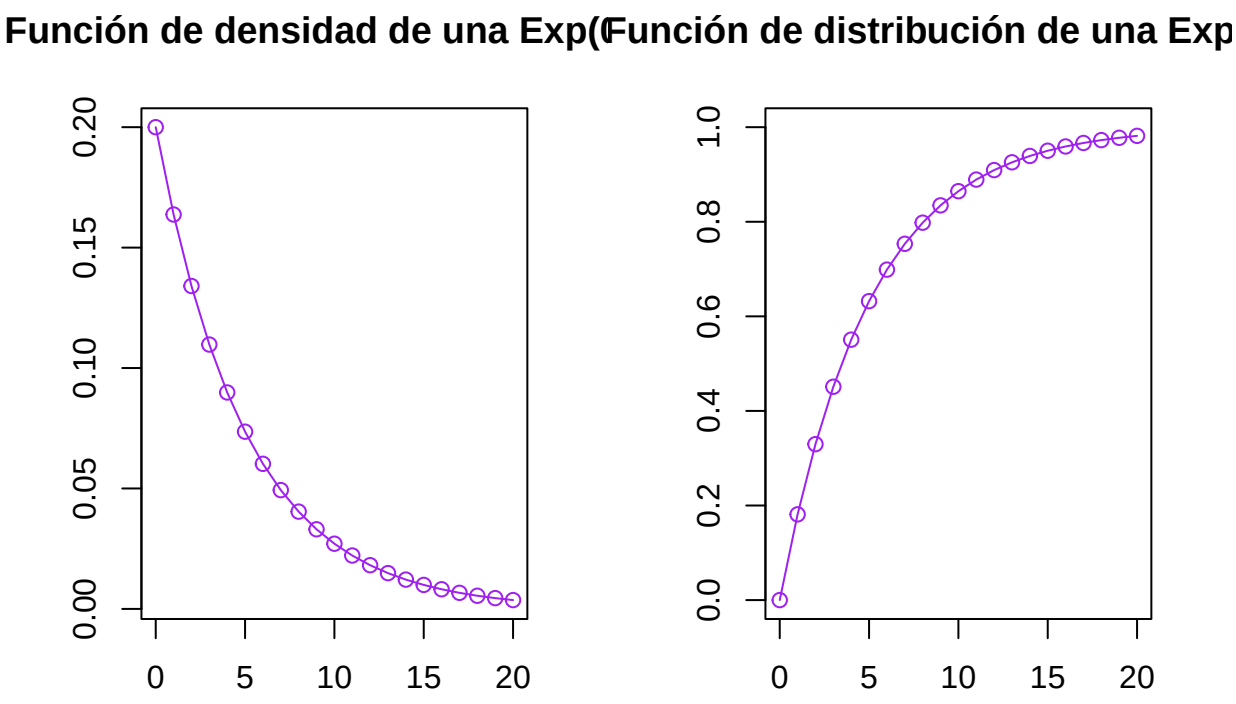
Propiedad de la pérdida de memoria. Si X es v.a. Exp(λ), entonces

$$p(X > s + t \ : \ X > s) = p(X > t) \ \forall s, t > 0$$

- El **dominio** de X será $D_X = [0, \infty)$
- La **función de distribución** vendrá dada por

$$F_X(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 - e^{-\lambda x} & \text{si } x > 0 \end{cases}$$

- **Esperanza** $E(X) = \frac{1}{\lambda}$
- **Varianza** $Var(X) = \frac{1}{\lambda^2}$



El código de la distribución Exponencial:

- En `R` tenemos las funciones del paquete `stats`: `dexp(x, rate)`, `pexp(q, rate)`, `qexp(p, rate)`, `rexp(n, rate)` donde `rate`= λ es el tiempo entre dos sucesos consecutivos de la distribución.
- En `Python` tenemos las funciones del paquete `scipy.stats.expon`: `pdf(k, scale)`, `cdf(k, scale)`, `ppf(q, scale)`, `rvs(n, scaler)` donde `scale`= $1/\lambda$ es la inversa del tiempo entre dos sucesos consecutivos de la distribución.

```
from scipy.stats import expon
import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1)

lam = 3
rv = expon(scale = 1/lam)

mean, var, skew, kurt = rv.stats(moments = 'mvsk')
print("Media %f"%mean)
```

En Python

```
## Media 0.333333
```

```
print("Varianza %f"%var)
```

```
## Varianza 0.111111
```

```
print("Sesgo %f"%skew)
```

```
## Sesgo 2.000000
```

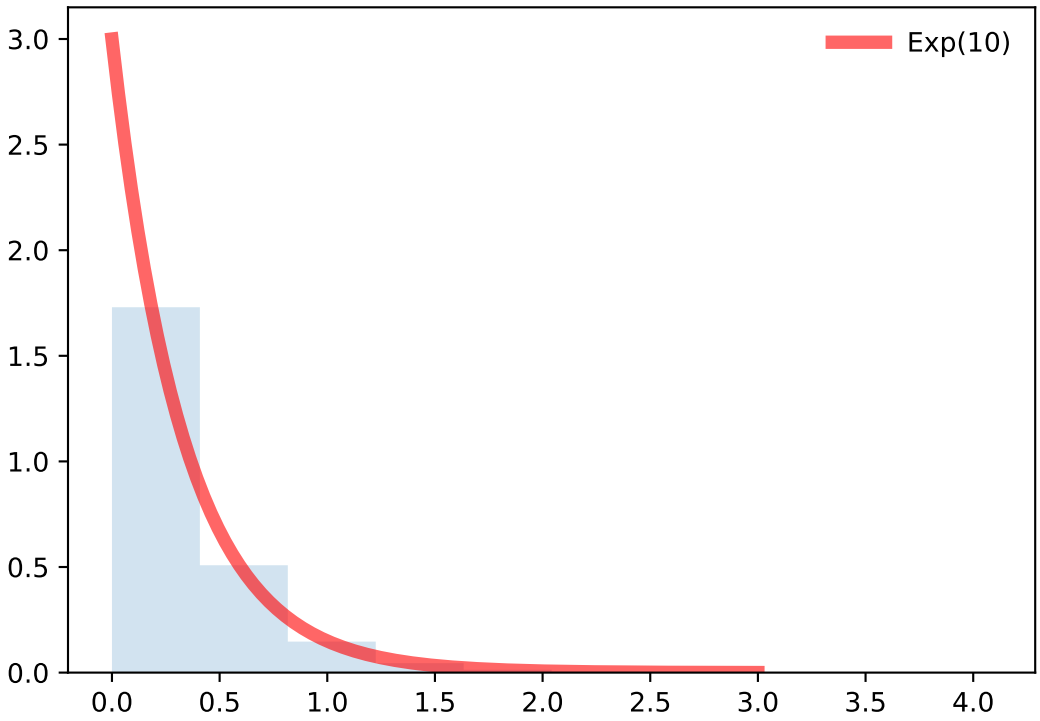
```
print("Curtosis %f"%kurt)
```

```
## Curtosis 6.000000
```

```
x = np.linspace(0, 3, 1000)
ax.plot(x, rv.pdf(x), 'r-', lw = 5, alpha = 0.6, label = "Exp(10)")

r = rv.rvs(size = 100000)
ax.hist(r, density = True, histtype = 'stepfilled', alpha = 0.2)
```

```
## (array([1.73002291e+00, 5.08344830e-01, 1.46780621e-01, 4.47149506e-02,
##         1.33948949e-02, 3.84460419e-03, 1.27337209e-03, 3.18343022e-04,
##         7.34637743e-05, 2.44879248e-05]), array([3.93644695e-06, 4.08368471e-01, 8.16733005e-01, 1.225
##         1.63346207e+00, 2.04182661e+00, 2.45019114e+00, 2.85855568e+00,
##         3.26692021e+00, 3.67528475e+00, 4.08364928e+00]), [<matplotlib.patches.Polygon object at 0x00
```

```
ax.legend(loc = "best", frameon= False)
plt.show()
```



Distribución Normal

Una v.a. X tiene distribución normal o gaussiana de parámetros μ y σ, $X \sim \mathcal{N}(\mu, \sigma)$ si su función de densidad es

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad \forall x \in \mathbb{R}$$

La gráfica de f_X es conocida como la Campana de Gauss

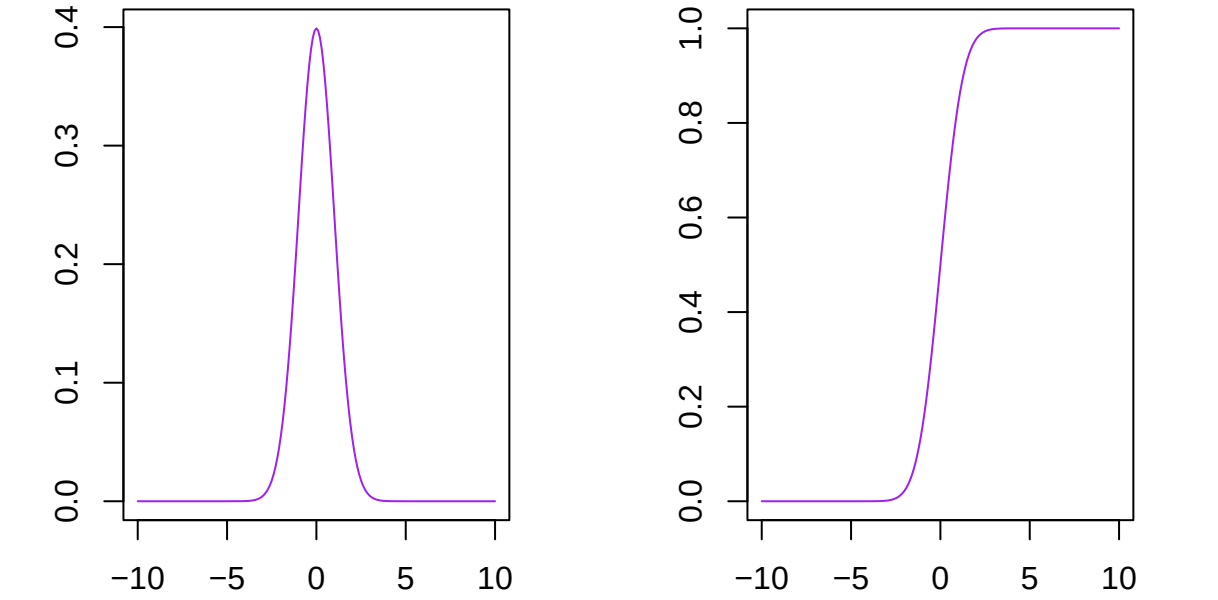
Cuando $\mu = 0$ y $\sigma = 1$, diremos que la v.a. X es estándar y la indicaremos usualmente como Z, la cual tendrá función de densidad

$$f_Z(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} \quad \forall z \in \mathbb{R}$$

- **Esperanza** $E(X) = \mu$
- **Varianza** $Var(X) = \sigma^2$

En particular, si Z sigue una distribución estándar,

- **Esperanza** $E(X) = 0$
- **Varianza** $Var(X) = 1$



El código de la distribución Normal:

- En `R` tenemos las funciones del paquete `stats`: `dnorm(x, mean, sd)`, `pnorm(q, mean, sd)`, `qnorm(p, mean, sd)`, `rnorm(n, mean, sd)` donde `mean` es la media y `sd` es la desviación estándar de la normal $N(\mu, \sigma)$.
- En `Python` tenemos las funciones del paquete `scipy.stats.normal`: `pdf(k, mu, scale)`, `cdf(k, mu, scale)`, `ppf(q, mu, scale)`, `rvs(n, mu, scale)` donde `mu` es la media y `scale` es la desviación estándar de la normal $N(\mu, \sigma)$.

Estandarización de una v.a. normal. Si X es una v.a. $\mathcal{N}(\mu, \sigma)$, entonces

$$Z = \frac{X - \mu}{\sigma} \sim \mathcal{N}(0, 1)$$

Las probabilidades de una normal estándar Z determinan las de cualquier X de tipo $\mathcal{N}(\mu, \sigma)$:

$$p(X \leq x) = p\left(\frac{X - \mu}{\sigma} \leq \frac{x - \mu}{\sigma}\right) = p\left(Z \leq \frac{x - \mu}{\sigma}\right)$$

F_Z no tiene expresión conocida.

Se puede calcular con cualquier programa, como por ejemplo R, o bien a mano utilizando las tablas de la $\mathcal{N}(0, 1)$

Con las tablas se pueden calcular tanto probabilidades como cuantiles

Distribución Normal en R y Python Si a la hora de llamar a alguna de las 4 funciones siguientes: `dnorm`, `pnorm`, `qnorm` o `rnorm` no especificásemos los parámetros de la media ni la desviación típica, R entiende que se trata de la normal estándar: la $\mathcal{N}(0,1)$.

Es decir, R interpreta $\mu = 0$ y $\sigma = 1$

En Python ocurre exactamente lo mismo.

Otras distribuciones importantes

- La distribución χ^2_k, donde k representa los grados de libertad de la misma y que procede de la suma de los cuadrados de k distribuciones normales estándar independientes:

$$X = Z_1^2 + Z_2^2 + \cdots + Z_k^2 \sim \chi^2_k$$

- La distribución t_k surge del problema de estimar la media de una población normalmente distribuida cuando el tamaño de la muestra es pequeña y procede del cociente

$$T = \frac{Z}{\sqrt{\chi^2_k/k}} \sim T_k$$

- La distribución F_{n_1,n_2} aparece frecuentemente como la distribución nula de una prueba estadística, especialmente en el análisis de varianza. Viene definida como el cociente

$$F = \frac{\chi^2_{n_1}/n_1}{\chi^2_{n_2}/n_2} \sim F_{n_1,n_2}$$

Distribuciones continuas en R

Distribución	Instrucción en R	Instrucción en Python	Parámetros
Uniforme	`unif`	`scipy.stats.uniform`	mínimo y máximo
Exponencial	`exp`	`scipy.stats.expon`	λ
Normal	`norm`	`scipy.stats.normal`	media μ, desviación típica σ
Khi cuadrado	`chisq`	`scipy.stats.chi2`	grados de libertad
t de Student	`t`	`scipy.stats.t`	grados de libertad
F de Fisher	`f`	`scipy.stats.f`	los dos grados de libertad