Capítulo 5

Análisis Estadísticos con R

El término **Análisis Estadístico** lleva implícitos los conceptos de categorización, manipulación y resumen de datos para dar respuesta a diferentes cuestiones.

Cuando hablamos de análisis estadísticos hacemos referencia a la capacidad de resumir, de forma coherente y entendible, un conjunto de datos entre los que existen relaciones que también hay que detectar y determinar, si los comportamientos observados en una muestra concreta son o no extensibles al total de una población.

Cualquier problema de análisis de datos puede resumirse en los siguientes puntos:

1. **Planteamiento del problema**. Todo nace de un problema, de una necesidad real.

2. **Recopilación de información**. Es preciso recoger datos (según un sistema estructurado) que permitan la realización del análisis. Estos datos serán sometidos a un proceso de depuración. En Estadística, los datos se recogen en variables. Una variable es por tanto una condición o característica determinada a partir de la cual, obtener información. Dicha información se obtiene en forma de **datos**, el valor de la variable para cada individuo particular.

   En la figura 5.1 resumimos, a modo de esquema, la clasificación de las variables según el tipo de información que almacenan.

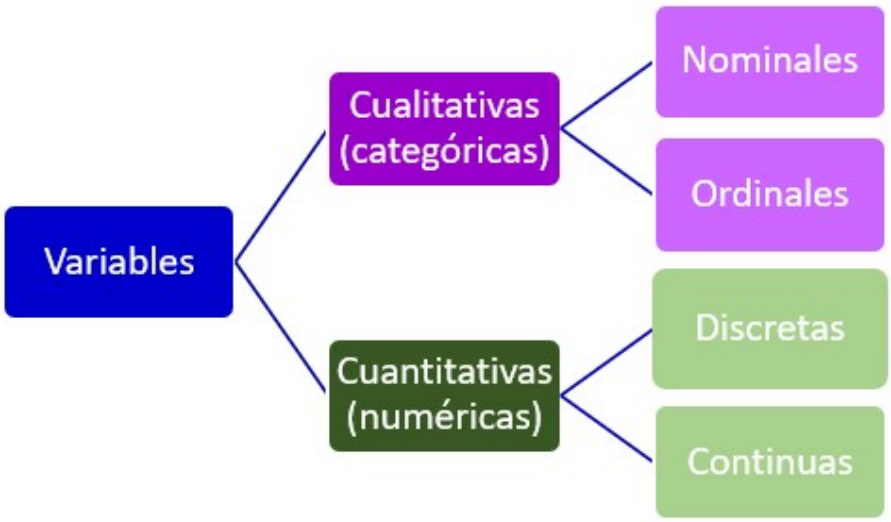


Figura 5.1: Clasificación variables.

Las variables y la información que contienen se agruparán en las denominadas ***bases de datos****. Estas bases de datos, R las recuperará en las estructras que hemos denominado hojas de datos (*data frame*) y a partir de aquí se inicia el proceso de análisis.*

3. **Análisis exploratorio de la información**. Una vez obtenidos los datos, el primer paso es analizar (*explorar*) la información intentanto determinar comportamientos o patrones establecidos en los mismos así como cualquier tipo de error o anomalía que pueda aparecer. Para ello se emplean fundamentalmente técnicas de visualización gráfica. El análisis exploratorio permite acceder a un conocimiento de los datos que determinará las técnicas adecuadas para el análisis inferencial de los mismos. Las técnicas de análisis exploratorio de datos tuvieron su origen en los años 70 y fueron desarrollados por el matemático John Tukey (Estados Unidos).

   Podemos agrupar las técnicas de análisis exploratorio:

   - **Descripción univariente no gráfica**. Análisis individual de las variables con el objetivo de describir la información recogida en ellas y examinar la aparición de patrones.
   - **Analisis gráfico univariante**. Las herramientas gráficas ofrecen una visualización completa de la información de cada variable. Los gráficos univariantes más conocidos son:
     - Diagrama de tallos y hojas
     - Histograma
     - Diagramas de cajas y bigotes
   - **Descripción multivariante no gráfica**. Análisis conjunto de varias variables examinando la posible relación entre ellas.
   - **Análisis gráfico multivariante**. Al igual que en el caso univariante, se aplican técnicas de visualización que permitan la detección de posible patrones de comportamiento. Destacamos:

- Diagrama de barras agrupadas.
- Mapa de calor, representación gráfica de los datos de modo que utiliza el color para representar los valores de los datos y la frecuencia de aparición de los mismos.

4. **Análisis inferencial**. La Inferencia Estadística de los datos permitirá el establecimiento de conclusiones extensibles a la población global.

5. **Conclusiones**. Interpretación de los resultados del análisis.

5.1. Resúmenes estadísticos

Una de las ramas de la Estadística más accesible e intuitiva es la **Descriptiva**. La Estadística Descriptiva analiza la ordenación y el tratamiento de la información para su presentación por medio de tablas y de representaciones gráficas, así como de la obtención de algunos parámetros útiles para la explicación de la información. En esta sección nos ocuparemos de tales parámetros.

El **entorno de computación estadística R** ofrece una gran cantidad de estadísticos y representaciones gráficas. Atendiendo al tipo de variable con que se trabaje, podemos resumir las medidas descriptivas básicas y las representaciones gráficas tal y como indicamos en la siguiente tabla:

Tipo de variable	Estadísticos	Gráficos
Cualitativas	Tendencia central: Moda	Barras
		Sectores
Ordinales	Tendencia central: Moda	Barras
	Posición: Cuantiles	Sectores
	Dispersión: Recorrido	
Cuantitativas	Tendencia central	Histograma
	Posición	Cajas y bigotes
	Dispersión	
	Simetría y curtosis	

5.1.1. Tablas de frecuencias

Los datos cuantitativos discretos se organizan en tablas, llamadas **Tablas de Distribución de Frecuencias**. La primera columna de la tabla contiene cada una de las modalidades de la variable observadas en la población. Tales valores se disponen en la tabla, ordenados de menor a mayor. En las restantes columnas se incluyen los diferentes tipos de frecuencias asociados a una distribución:

- **Frecuencia absoluta**: Indica el número de veces que se repite cada valor de la variable.
- **Frecuencia relativa**: Indica la proporción con que se repite cada valor. Se calcula dividiendo la frecuencia absoluta entre el tamaño de la muestra.

- **Frecuencia absoluta acumulada**: Indica el número de valores que son menores o iguales que el valor observado.
- **Frecuencia relativa acumulada**: Indica la proporción de datos que son menores o iguales que el valor observado.

Veremos a continuación como construir una tabla de frecuencias en R a partir de las herramientas disponibles en su libro base.

Consideremos la base de datos **mtcars** disponible en el libro base de R. Esta base dispone de información relativa a 32 marcas de vehículos. Se mide el consumo en millas por galón (mpg), número de cilindros (cyl), volumen del motor, poder que genera el motor (disp), caballos de fuerza (hp), peso (wt), tiempo en recorrer 1/4 de milla (qsec), tipo de motor (vs), caja de cambios automática o manual (am), número de marchas (gear), número de carburadores (carb).

```
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
```

La función str() describe la información recogida en la hoja de datos: número de registros y variables y el tipo de cada una de tales variables

```
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 ...
```

Aunque todas las variables aparecen como numéricas, la variable *am* debería entenderse como factor (0 = caja de cambios automática y 1 = caja de cambios manual). Igualmente ocurre con la variable *vs* (0 = motor en forma de V, forma en que se disponen los cilindros, 1 = motor recto).

Para crear la tabla de frecuencias asociada a cada variable es preciso *contar cuántas veces se repite cada modalidad* de las variables. La orden `table()`, es la orden implementada en R para construir tablas de frecuencias. Se puede aplicar directamente al *data.frame* de trabajo o a las variables independientemente. Si la aplicamos al total de los datos (table(mtcars)) proporciona un conteo de frecuencias teniendo en cuenta los cruces entre todas las variables de la hoja de datos, por lo que sólo se debe aplicar cuando tengan sentido estos cruces. En el caso de las variables continuas, es preciso primero agrupar sus valores en intervalos.

```
table(mtcars$carb)    #numero de carburadores
```

```
 1  2  3  4  6  8
 7 10  3 10  1  1
```

```
table(mtcars$gear)    #numero de marchas
```

```
 3  4  5
15 12  5
```

La agrupación de datos de una variable continua se puede hacer *categorizando* la variable, usando la función cut() o mediante la función hist()

La sintaxis de la función cut() es la siguiente:

```
cut(variable,                  # Vector de entrada (numérico)
    breaks,                    # Número o vector con los cortes
    labels = NULL,             # Etiquetas para cada grupo
    include.lowest = FALSE,    # Si se incluye el valor más pequeño o no
                               # en el primer intervalo
    right = TRUE,              # Si el intervalo derecho está cerrado
                               # (y el izquierdo abierto) o viceversa
    dig.lab = 3,               # Número de dígitos de los grupos si
                               # labels = NULL
    ordered_result = FALSE,    # Si se debería ordenar el resultado (TRUE)
                               # del factor o no (FALSE)
    ...)                         # Argumentos adicionales
```

```
#agrupamos los datos en 5 intervalos generados por R
cut(mtcars$mpg, breaks=5)
```

```
 [1] (19.8,24.5] (19.8,24.5] (19.8,24.5] (19.8,24.5] (15.1,19.8] (15.1,19.8]
 [7] (10.4,15.1] (19.8,24.5] (19.8,24.5] (15.1,19.8] (15.1,19.8] (15.1,19.8]
[13] (15.1,19.8] (15.1,19.8] (10.4,15.1] (10.4,15.1] (10.4,15.1] (29.2,33.9]
[19] (29.2,33.9] (29.2,33.9] (19.8,24.5] (15.1,19.8] (15.1,19.8] (10.4,15.1]
[25] (15.1,19.8] (24.5,29.2] (24.5,29.2] (29.2,33.9] (15.1,19.8] (15.1,19.8]
[31] (10.4,15.1] (19.8,24.5]
Levels: (10.4,15.1] (15.1,19.8] (19.8,24.5] (24.5,29.2] (29.2,33.9]
```

Se genera una nueva variable en la que cada dato de la variable original se transforma en el intervalo en el que debería ser incluído.

El usuario puede definir intervalos específicos utilizando el argumento `breaks`. Este argumento puede tomar un único valor entero (en cuyo caso indicará el número de intervalos a formar, todos de la misma amplitud) o un vector de valores en el que se indican los extremos de los intervalos a considerar.

```
#agrupamos los datos en intervalos específicos
mpg_interv<-cut(mtcars$mpg,  breaks=c(10, 15, 20, 25, 30, 35) )
mpg_interv
```

```
 [1] (20,25] (20,25] (20,25] (20,25] (15,20] (15,20] (10,15] (20,25] (20,25]
[10] (15,20] (15,20] (15,20] (15,20] (15,20] (10,15] (10,15] (10,15] (30,35]
[19] (30,35] (30,35] (20,25] (15,20] (15,20] (10,15] (15,20] (25,30] (25,30]
[28] (30,35] (15,20] (15,20] (10,15] (20,25]
Levels: (10,15] (15,20] (20,25] (25,30] (30,35]
```

El resultado de la función cut() es una variable tipo factor con niveles, los intervalos generados.

Una vez definida esta variable, podemos construir la tabla de frecuencias asociada a la misma:

```
table(mpg_interv)
```

```
mpg_interv
(10,15] (15,20] (20,25] (25,30] (30,35]
      6      12       8       2       4
```

Y la tabla de doble entrada para las variables `geary` `mpg_interv`:

```
table(mtcars$gear, mpg_interv)
```

```
  mpg_interv
   (10,15] (15,20] (20,25] (25,30] (30,35]
3       5       8       2       0       0
4       0       2       6       1       3
5       1       2       0       1       1
```

Una vez obtenidas las frecuencias absolutas de una variable, será necesario calcular las frecuencias relativas, absolutas acumuladas y relativas acumuladas.

La función prop.table calcula las frecuencias relativas de cada una de las modalidades de una variable. Se aplica sobre una tabla de frecuencias.

```
tf<-table(mtcars$gear)
prop.table(tf)
```

```
      3       4       5
0.46875 0.37500 0.15625
```

La sintaxis de esta función permite trabajar también con tablas de doble entrada, utilizando para ello el argumento `margin` que tomará valores 1 ó 2 indicando si se calculan freceuencias relativas por filas o por columnas.

```
prop.table(tablaFreq, margin=NULL)
```

Para calcular las frecuencias acumuladas trabajaremos con la función cumsum() que, para cada una de las componentes de un vector numérico, proporciona la suma de los valores de las modalidades anteriores.

```
Ni<-cumsum(tf)
Ni
```

```
 3  4  5
15 27 32
```

```
Fi<-cumsum(prop.table(tf))
Fi
```

```
      3       4       5
0.46875 0.84375 1.00000
```

Podemos añadir todos los resultados obtenidos en una única estructura de R

```
tf_2<-as.data.frame(tf)
fr<-as.data.frame(prop.table(tf))
tf_2<-cbind(tf_2, fr[,2], Ni, Fi)
tf_2
```

```
  Var1 Freq fr[, 2] Ni      Fi
3    3   15 0.46875 15 0.46875
4    4   12 0.37500 27 0.84375
5    5    5 0.15625 32 1.00000
```

```
names(tf_2)<-c("x", "n", "f", "N", "F")
tf_2
```

```
  x  n       f  N       F
3 3 15 0.46875 15 0.46875
4 4 12 0.37500 27 0.84375
5 5  5 0.15625 32 1.00000
```

En las **tablas de doble entrada** es posible añadir las distribuciones marginales a las variables fila y columna.

```
t2<-table(mtcars$gear , mpg_interv)
t2
```

```
   mpg_interv
    (10,15] (15,20] (20,25] (25,30] (30,35]
  3       5       8       2       0       0
  4       0       2       6       1       3
  5       1       2       0       1       1
```

```
margin.table(t2)  #total de observaciones
```

```
[1] 32
```

```
margin.table(t2, 1)  # marginal 1 (filas)
```

```
 3  4  5
15 12  5
```

```
margin.table(t2, 2)  # marginal 2 (columnas)
```

```
mpg_interv
(10,15] (15,20] (20,25] (25,30] (30,35]
      6      12       8       2       4
```

```
prop.table(t2)     # frec. rel. según total de observaciones
```

```
   mpg_interv
    (10,15] (15,20] (20,25] (25,30] (30,35]
  3 0.15625 0.25000 0.06250 0.00000 0.00000
  4 0.00000 0.06250 0.18750 0.03125 0.09375
  5 0.03125 0.06250 0.00000 0.03125 0.03125
```

```
prop.table(t2, 1)  # frec. rel. según total por filas
```

```
   mpg_interv
       (10,15]    (15,20]    (20,25]    (25,30]    (30,35]
  3 0.33333333 0.53333333 0.13333333 0.00000000 0.00000000
  4 0.00000000 0.16666667 0.50000000 0.08333333 0.25000000
  5 0.20000000 0.40000000 0.00000000 0.20000000 0.20000000
```

```r
prop.table(t2, 2)  # frec. rel. según total por columnas
```

```
   mpg_interv
      (10,15]   (15,20]   (20,25]   (25,30]   (30,35]
  3 0.8333333 0.6666667 0.2500000 0.0000000 0.0000000
  4 0.0000000 0.1666667 0.7500000 0.5000000 0.7500000
  5 0.1666667 0.1666667 0.0000000 0.5000000 0.2500000
```

Para añadir las distribuciones marginales a una tabla de frecuencias de doble entrada usaremos la función addmargins()

```r
addmargins(t2)
```

```
     mpg_interv
      (10,15] (15,20] (20,25] (25,30] (30,35] Sum
  3         5       8       2       0       0  15
  4         0       2       6       1       3  12
  5         1       2       0       1       1   5
  Sum       6      12       8       2       4  32
```

La función hist() permite construir tablas de frecuencias para variables continuas. La sintaxis de esta función es:

```r
hist(x,
     breaks='Sturges', # intervalos
     prob = TRUE, # densidades de frecuencias
     freq = FALSE, # densidades de frecuencias (NO frecuencias)
     include.lowest=TRUE,  # un valor xi que coincida con un límite de
                           # intervalo será ubicado en el intervalo izqdo
     right=TRUE, # intervalos (liminf,limsup]
     plot=FALSE) # gráfico del histograma
```

```r
hist(mtcars$mpg, plot=FALSE )
```

```
$breaks
[1] 10 15 20 25 30 35

$counts
[1]  6 12  8  2  4

$density
[1] 0.0375 0.0750 0.0500 0.0125 0.0250

$mids
[1] 12.5 17.5 22.5 27.5 32.5
```

```
$xname
[1] "mtcars$mpg"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```

La función hist() da como resultado una lista con las siguientes componentes:

- **$breaks** límites inferior y superior de los intervalos.
- **$counts** frecuencias absolutas de cada uno de los intervalos.
- **$density** densidad de frecuencia relativa (fi/ai)
- **$mids** punto medio de los intervalos
- **$xname** nombre del vector
- **$equidist** indica si todos los intervalos son del mismo tamaño

Por defecto, la función hist() aplica el algoritmo de Sturges para la construcción de los intervalos. En el caso de que queramos especificar intervalos concretos, los extremos de éstos se indicarán en el argumento `breaks`. Por ejemplo para obtener los intervalos (12,18], (18,20] y (20,27], el código será:

```
hist(mtcars$mpg, breaks=c(10, 15, 20, 25, 30, 35), plot=FALSE )
```

```
$breaks
[1] 10 15 20 25 30 35

$counts
[1]  6 12  8  2  4

$density
[1] 0.0375 0.0750 0.0500 0.0125 0.0250

$mids
[1] 12.5 17.5 22.5 27.5 32.5

$xname
[1] "mtcars$mpg"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```

Es posible también usar este argumento **breaks** para indicar el número de intervalos con el que queremos trabajar (en ese caso, R construirá todos los intervalos del mismo tamaño):

```
hist(mtcars$mpg, breaks=5, plot=FALSE )
```

```
$breaks
[1] 10 15 20 25 30 35

$counts
[1]  6 12  8  2  4

$density
[1] 0.0375 0.0750 0.0500 0.0125 0.0250

$mids
[1] 12.5 17.5 22.5 27.5 32.5

$xname
[1] "mtcars$mpg"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```

Además de las tablas de frecuencias, es posible resumir la información mediante estadísticos apropiados. A nivel descriptivo, indicamos a continuación algunos de los más importantes así como las funciones de R que permiten su cálculo:

5.1.2. Medidas de tendencia central

Media

```
mean(variable)
```

Mediana

```
median(variable)
```

Moda

No hay una función estándar para la moda, si bien su cálculo está implementado en distintos libros de R. Uno de ellos es el libro **modeest**, dentro del cual está disponible la función mfv()

```
library(modeest) # carga el libro modeest
mfv(variable) # calcula la moda
```

5.1.3. Medidas de posición

Percentiles

```
quantile(x,  # vector de observaciones
         probs = seq(0, 1, 0.25), # cuantiles a calcular
         ...)
```

El libro **fmsb** dipone de una función alternativa, percentile(), para el cálculo de estos coeficientes:

```
library(fmsb) # carga el libro fmsb
 percentile(variable)
```

5.1.4. Medidas de dispersión

Rango

```
range(variable)
```

Varianza

```
var(variable)
```

Desviación típica

```
sd(variable)
```

Rango intercuartílico

```
 IQR(variable)
```

5.1.5. Medidas de forma

Para las medidas de forma, necesitamos recurrir a la utilización de un libro específico: **moments**, de modo que lo primero es cargarlo con la orden library():

```
library(moments)
```

Coeficiente de simetría

```
skewness(variable)
```

- Coeficiente de asimetría nulo: distribución simétrica.
- Coeficiente de asimetría positivo: distribución asimétrica a la derecha.
- Coeficiente de asimetría negativo: distribución asimétrica a la izquierda.

Coeficiente de curtosis

```
kurtosis(variable)
```

- Coeficiente de curtosis nulo: Distribución igual de apuntada que la normal
- Coeficiente de curtosis positivo: Distribución platicúrtica (más apuntada que la normal).
- Coeficiente de curtosis negativo: Distribución leptocúrtica (menos apuntada que la normal).

Momentos centrados y no centrados

```
moment(variable, order=valor numérico, central = TRUE) #momento centrado
```

```
moment(variable, order=valor numérico, central = FALSE) #momento no centrado
```

5.2. Representaciones gráficas

Los gráficos estadísticos constituyen una potente herramienta para la visualización de los datos, facilitando el acceso a la información así como el establecimento de análisis apropiados.

La presentación de resultados estadísticos utilizando representaciones gráficas se considera un aspecto muy importante en el proceso de transmisión de información y de análisis de una base de datos. A través de los gráficos podemos presentar razonamientos complejos de una manera clara, precisa y fácilmente entendible. Las representaciones gráficas de los datos constituyen un importante paso en el análisis de los mismos ya que van a proporcionar una primera *pista* sobre su comportamiento.

Cuando se crea un gráfico estadístico, este ha de tener una serie de características que garanticen la veracidad de la información que transmite. Entre las más importantes podemos destacar:

- Descripción de los datos con que se está trabajando.
- Empleo correcto de las escalas de representación así como de las leyendas. La gráfica obtenida ha de trasladar correctamente la información almacenada en los datos.
- Capacidad para resumir coherentemente grandes volúmenes de información.
- Posibilidad del establecemiento de comparaciones entre grupos siempre que esto sea posible.

El entorno de computación estadística R tiene una gran disponibilidad de recursos gráficos para la representación de datos y funciones.

Las funciones para la realización de gráficos en R se dividen en dos grandes grupos:

- **Funciones gráficas de alto nivel**; encargadas de la representación de gráficos "completos"
- **Funciones gráficas de bajo nivel**; funciones que permiten la adición de elementos a un gráfico ya existente (generado por una función de alto nivel).

Son muchos los libros desarrollados en R para la generación de representaciones gráficas, podemos destacar:

- **graphics**. Disponible en la instalación básica de R.
- **plotrix**. Libro de gráficos con funciones que permiten colocar leyendas con gran precisión, agregar cuadrículas, invertir gráficos,....)
- **scatterplot3D**. Libro para la creación de gráficos en 3D (lattice también crea este tipo de gráficos)
- **rgl**. Permite también gráficos 3D pero en este caso interactivos.
- **lattice**. Proporciona una amplia gama de representaciones gráficas así como diferentes posibilidades de presentación de la información y personalización de las representaciones.
- **ggplot2**. Basado en la GRAMÁTICA DE GRÁFICOS: *cualquier gráfico puede expresarse a partir de la combinación de diversos componentes: un conjunto de* ***datos***, *un sistema de* ***coordenadas*** *y un conjunto de geometrías,* ***geoms***, (*determinan la representación visual de los datos*).

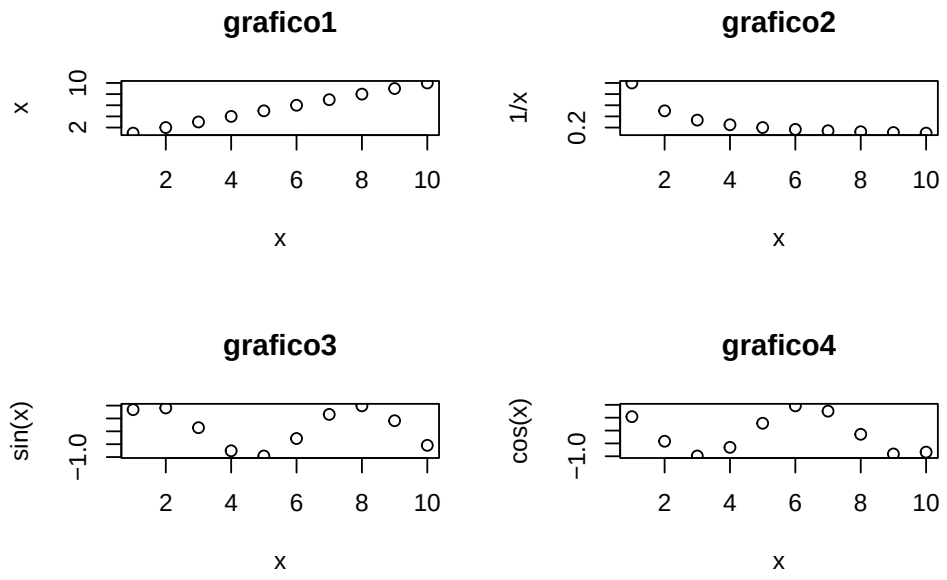
5.2.1. Funciones generales en R para la realización de gráficos

Sea cual sea el programa empleado para obtener una representación gráfica, este ha de disponer de herramientas que permitan trabajar con ejes y escalas. En el caso de trabaja con el entorno R, cada uno de los libros (*packages*) que permiten la obtención de representaciones gráficas, llevan también asociadas un conjunto de funciones que permiten tanto la modificación de los gráficos en sí mismos, como la modificación del *entorno* a través del ajuste de ejes, inclusión de nueva información en el gráfico, adición de notas, títulos, ...

En referencia a las "modificaciones" citadas anteriormente, pueden hacerse a través de los argumentos de las funciones gráficas y a través de funciones específicas, algunas de las cuales pasamos a describir brevemente:

1. Función par(). Permite la modificación directa de las características físicas (color, tamaño, posición,...) del gráfico que se va a realizar. Puede ser ejecutada antes de abrir cualquier ventana gráfica y ser válida para todos los gráficos que se realicen. Del mismo modo, en algunos casos, es posible utilizar los argumentos de la función par() como argumentos propios de la función de representación.

```r
x<-1:10
par(mfrow=c(2,2),    # 4 ventanas gráficas: dos filas y dos columnas
    oma=c(1,2,4,3)) # nuevos márgenes

# mfrow=c (nrow, ncol), número filas y columnas de la ventana gráfica
# oma=c(bottom, left, top, right) márgenes
# permiten incluir texto en la ventana gráfica

plot(x, x, main = "grafico1")
plot(x, 1/x, main = "grafico2")
plot(x, sin(x), main = "grafico3")
plot(x, cos(x), main = "grafico4")
```



La orden help(par) proporciona información sobre todos los argumentos de la función par()). Entre los más utilizados, podemos señalar:

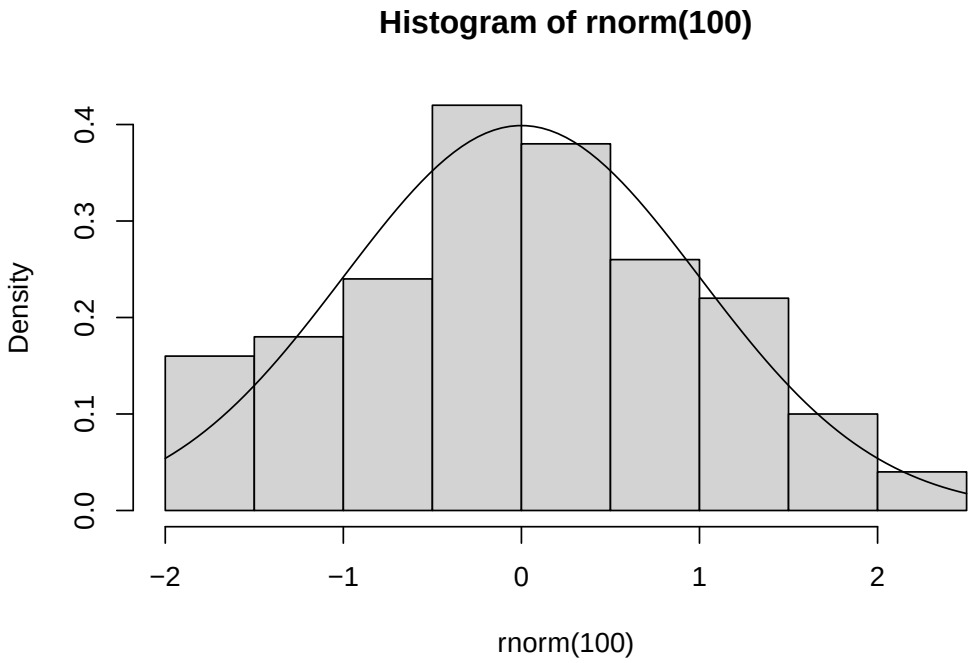
- `font.axis`, especifica el tipo de fuente en los ejes (toma valores numéricos)
- `las`, determina la orientación de las etiquetas de los ejes. Toma valores 0,1, 2, 3 siendo: 0 orientación paralela a los ejes, 1 etiquetas siempre horizontales, 2 etiquetas perpendiculares a los ejes, 3 etiquetas siempre verticales
- `mfrow`, `mfcol`, disposición de filas y columnas en la ventana gráfica
- `bty`, tipo de caja (bordes)
- `pch`, tipo de símbolo en la represención
- `oma`, vector de la forma `c(abajo, izquierda, arriba, derecha)` para establecer los márgenes de la ventana gráfica

2. **Argumentos de las funciones gráficas**. Aunque cada función gráfica tiene unos argumentos específicos para su ejecución, existen algunos que son comunes a gran parte de las representaciones y que pasamos a enumerar:

```
log = "<x | y |xy>"  # Ejes logarítmicos
main = "Título"      # Título del gráfico
new = <logical>      # Adiciona sobre el gráfico actual
sub = "Subtítulo"    # Anotación bajo el gráfico
type = < l | p | b | n>  #líneas, puntos, ambos, nada
lty = n              # Tipo de línea (continua, a trazos...)
lwd = n              # Grosor
pch='.'     # Carácter de dibujo.
             # Forma de los puntos (círculo, cuadrado, estrella, etc).
help(points)   # listado de valores y formas disponibles de los puntos
xlab = "Nombre del eje x"
ylab = "Nombre del eje y"
xlim = c(xminimo, xmáximo)  #rango eje OX
ylim = c(yminimo, ymáximo)  #rango eje OY
col= valor # Color  para el gráfico (ya sea para puntos, líneas...).
colors()  # lista de los colores disponibles en R
help(colors) # ayuda para obtener aún más colores
font = valor # Fuente a usar en el texto
las = 0,1, 2, 3 # Cambia el estilo de las etiquetas de los ejes
                # 0 paralelo a los ejes,
                # 1 siempre horizontales,
                # 2, perpendiculares a los ejes,
                # 3 siempre verticales
```

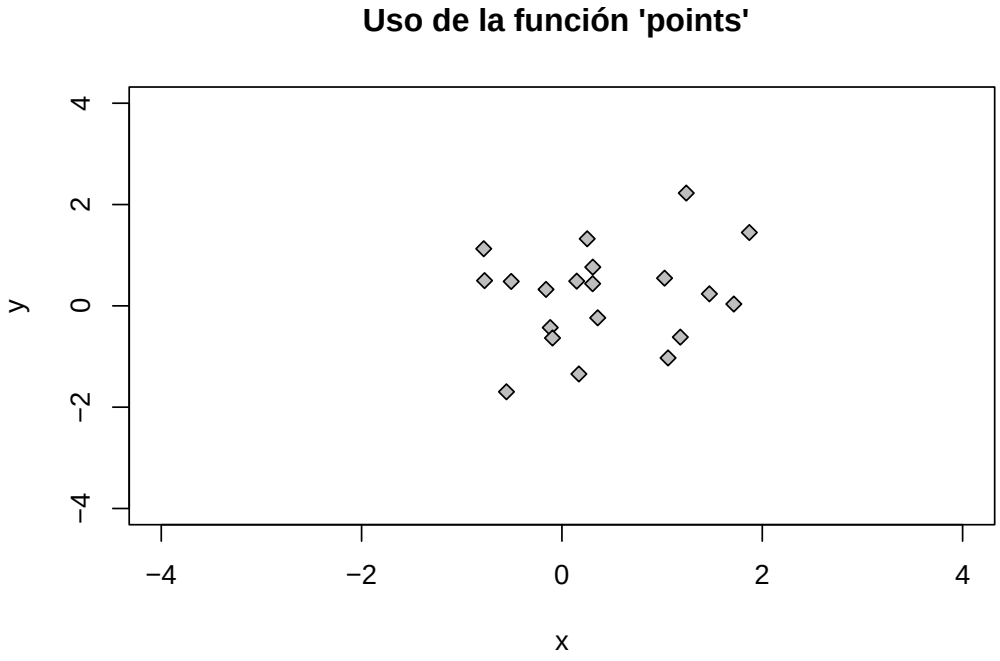
3. Función curve(), permite superponer una curva a un gráfico previamente existente

```
hist(rnorm(100), freq=FALSE) #histograma 100 valores aleatorios N(0,1)
curve(dnorm(x), add = TRUE) #función de densidad N(0,1)
```



4. Función points(), permite añadir puntos al gráfico

```
plot(-4:4, -4:4,
     type = "n",
     xlab="x",
     ylab="y",
     main="Uso de la función 'points' ")

points(rnorm(20), rnorm(20), pch=23,bg = "gray")
```




```
# pch=19: círculo sólido
# pch=20: círculo más pequeño
# pch=21: círculo
# pch=22: Cuadrado
# pch=23: Diamante
# pch=24: Triángulo punta hacia arriba
# pch=25: Triángulo punta hacia abajo
```

El argumento `pch` permite elegir el tipo de símbolo para la representación.

El argumento `bg` indica el color de dicho símbolo

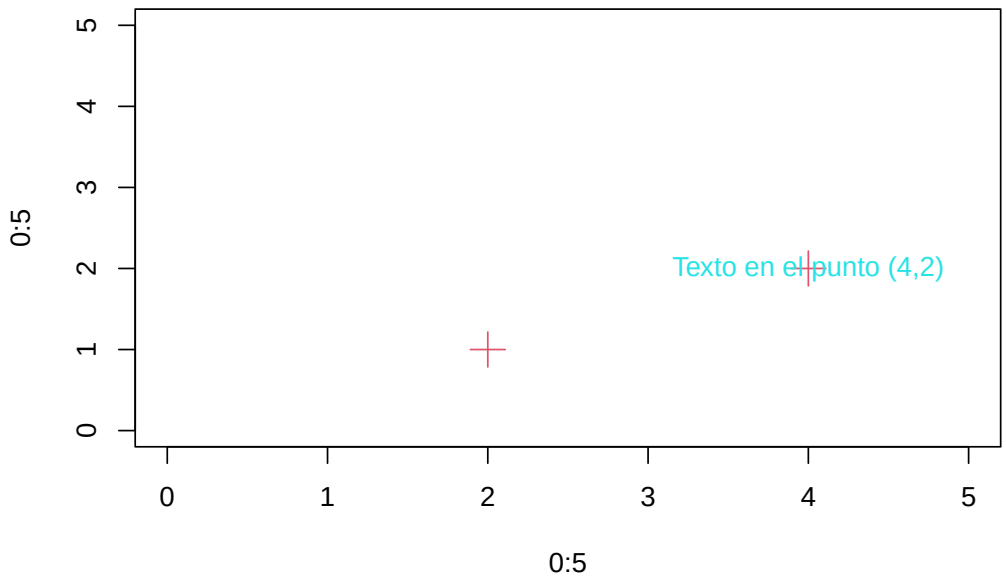
5. Función lines(), representa líneas rectas en un gráfico previo
6. Función mtext(), permite escribir texto en alguno de los cuatro márgenes del gráfico o en los márgenes exteriores de la ventana gráfica

```
mtext(text,      #    Texto
   side = 3, # Lado del gráfico sobre el que situar el texto
             # 1=abajo, 2=izquierda, 3=arriba, 4=derecha
   line = 0, # línea de margen en la que escribir
             # empezando en 0 y contando hacia fuera
    outer = FALSE, # Para usar márgenes exteriores
                   # cuando han sido definidos (con par(mar=(..))
    at = NULL,  # ubicación en coordenadas de usuario.
                # Si 'length(at)==0' (por defecto),
                # ubicación determinada por 'adj'.
    adj = NA, ...)
```

7. Función text(), permite añadir texto a un gráfico

```
text(x, y = NULL # posición del texto
   labels, # texto a escribir
           # "\n" permite dividir el texto en varias líneas
   adj = NULL, # uno o dos valores en [0, 1]: justificación
               # 0 left/bottom, 1 right/top, 0.5  centered.
   poss = NULL, # especifica posición del texto
                # Valores  1, 2, 3 y 4, que significan:
                # abajo, izquierda, encima, derecha de (x,y)
                # de las coordenadas iniciales
   offset = 0.5,  # si poss está especificado,
                  # controla la distancia a la etiqueta del texo
   cex = 1,     # tamaño de la fuente
   col = NULL,  # color de la fuente
   font = NULL, # estilo de fuente
   srt = n, # rotación del texto
   ...)
```

```
plot(0:5, 0:5, type = "n")  # establecemos los ejes

points(c(4,2), c(2,1),
    pch = 3, # tipo de símbolo
    cex = 2, # tamaño del símbolo
    col = 2) # color del símobolo

 text(4, 2, "Texto en el punto (4,2)", cex=1, col=5)
```



5.2.2. Funciones gráficas de alto nivel

Las funciones gráficas de R se subdividen en dos grandes grupos según que sean capaces de generar un gráfico directamente o que requieran que haya uno previo (funciones gráficas de **alto nivel** y **bajo nivel** respectivamente).

Las **funciones Gráficas de Alto Nivel** son las que *generan gráficos completos*. En el libro base de R, entre las más utilizadas podemos citar:

- plot(), gráficos de nubes de puntos.
- hist(), histogramas.
- barplot(), diagramas de barras.
- boxplot(), diagramas de cajas y bigotes.
- pie(), diagrama de sectores.
- pers(), superficies en 3D...

Todas estas funciones disponen de multitud de argumentos (tal y como hemos indicado en el apartado anterior) que permiten controlar las etiquetas de los ejes, sus límites, títulos, tamaño, colores, etc.:

- **xlim, ylim**: controlan, respectivamente, las dimensiones de los ejes X e Y.

  Así, xlim=c(0,10) indica que el eje X se extiende de 0 a 10; ylim=c(-5,5) indica que el eje Y va de -5 a 5.

  Si no se incluyen estos valores, R los ajusta por defecto de modo que se incluyan todos los valores disponibles en la hoja de datos.

- **xlab, ylab**: especifican las etiquetas para los ejes X e Y.

- **main**: indica el título del gráfico.

- **sub**: permite añadir un subtítulo.

Dos argumentos importantes que son comunes a la mayoría de gráficos de alto nivel son los siguientes:

- **add = TRUE**: fuerza a la función a actuar como si fuese de bajo nivel (superpone la figura que genera a un gráfico ya existente). Esta opción no está disponible para todas las funciones.
- **type**: indica el tipo de gráfico a realizar. En concreto:
  - type = "p" representa puntos (opción por defecto)
  - type = "l" representa líneas
  - type = "b" (both, ambos) representa puntos unidos por líneas.
  - type = "n" No dibuja nada.

Dependiendo del tipo de datos con el que se trabaje se optará por una u otra representación. Entre las representaciones gráficas más conocidas podemos destacar el diagrama de barras, el diagrama de sectores y el histograma, pero son muchos más los gráficos de los que se dispone en el análisis estadístico. A continuación pasamos a describir cómo trabaja R con estos gráficos en el libro base. En los próximos capítulos introduciremos el manejo de otros paquetes gráficos.

5.2.2.1. Gráfico de dispersión: Nube de puntos

La función plot() es una función genérica para la representación gráfica de objetos en R.

Los gráficos más sencillos que permite generar esta función son **nubes de puntos** (x,y).

Para ver un ejemplo[1] utilizaremos la hoja de datos `trees`, disponible en el libro base de R `datasets`. Esta hoja de datos contiene información de 31 cerezos, de los que se toman las siguientes medidas:

- *Girth* (diámetro del tronco en pulgadas),
- *Height* (altura del árbol en pies) y
- *Volume* (volumen de madera en el árbol).

```
head(trees)
```

```
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
```

```
nrow(trees)
```

```
[1] 31
```

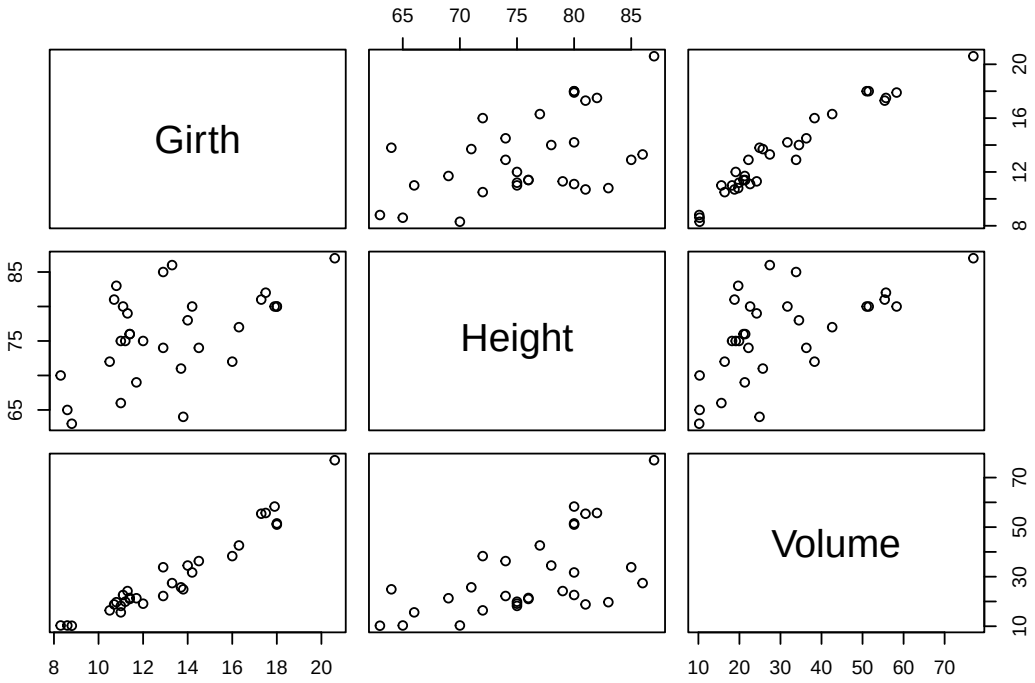
```
str(trees)
```

```
'data.frame':   31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

La orden plot() se puede ejecutar directamente sobre un objeto tipo *data frame* en cuyo caso mostrará las nubes de puntos asociadas a todos las combinaciones de variables de la hoja de datos:

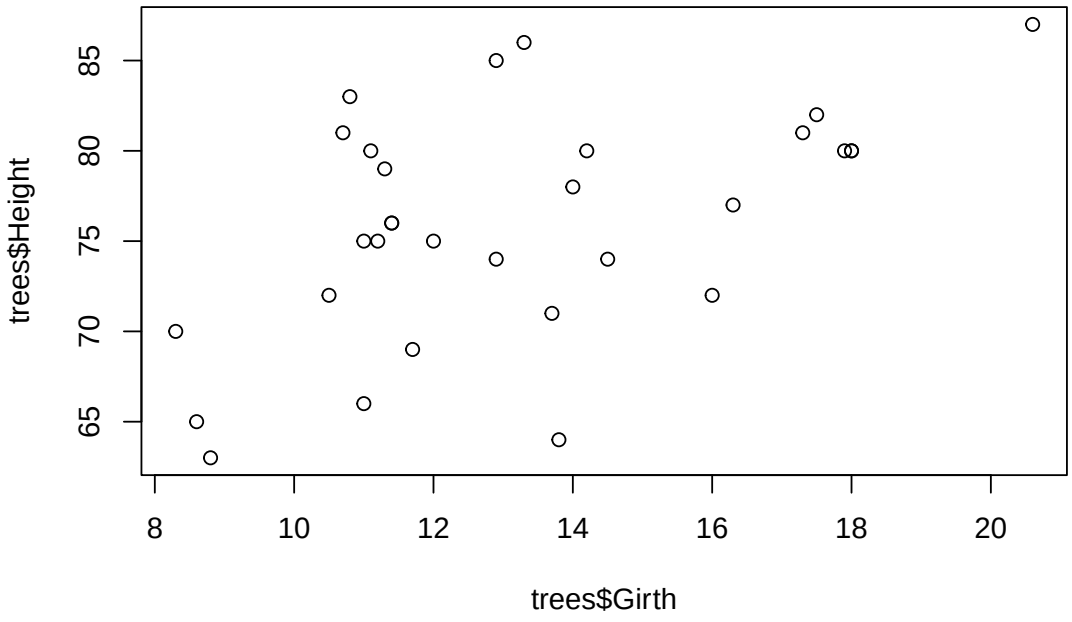
```
plot(trees)
```

[1] http://www.dma.ulpgc.es/profesores/personal/stat/cursoR4ULPGC/9d-grafPlot.html

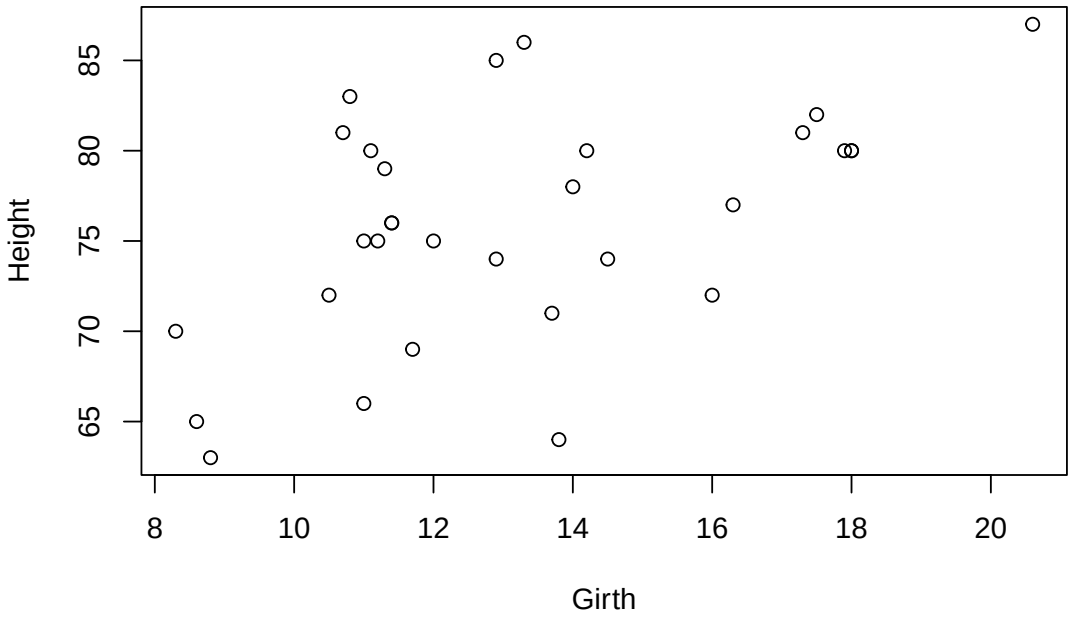


En el caso de querer especificar dos variables concretas utilizaremos el nombre de las variables o haremos uso de la orden with():

```
plot(trees$Girth, trees$Height)
```



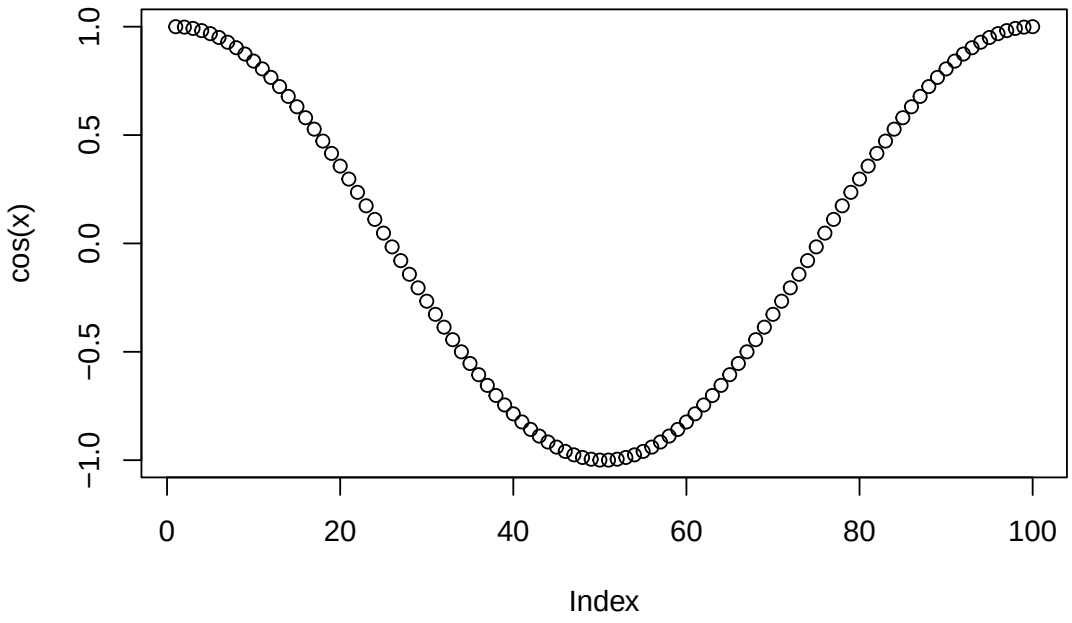
```
with(trees, plot(Girth, Height)) # Lee las variables de la hoja trees
```



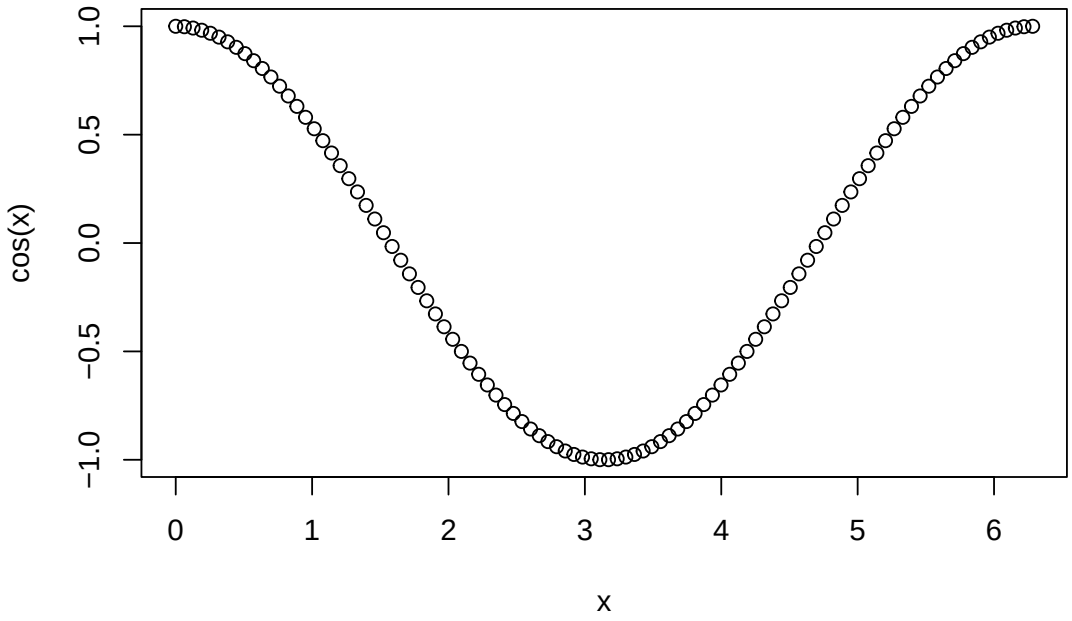
La función plot() también posibilita la representación de **funciones matemáticas**:

Ejercicio: Representar las función coseno en el intervalo $[0, 2\pi]$

```
x <- seq(0, 2*pi, length = 100) # se generan valores x,
                                # soporte de la función
                                # 100 valores equiespaciados
plot(cos(x))  # representa la función coseno evaluada en x
```

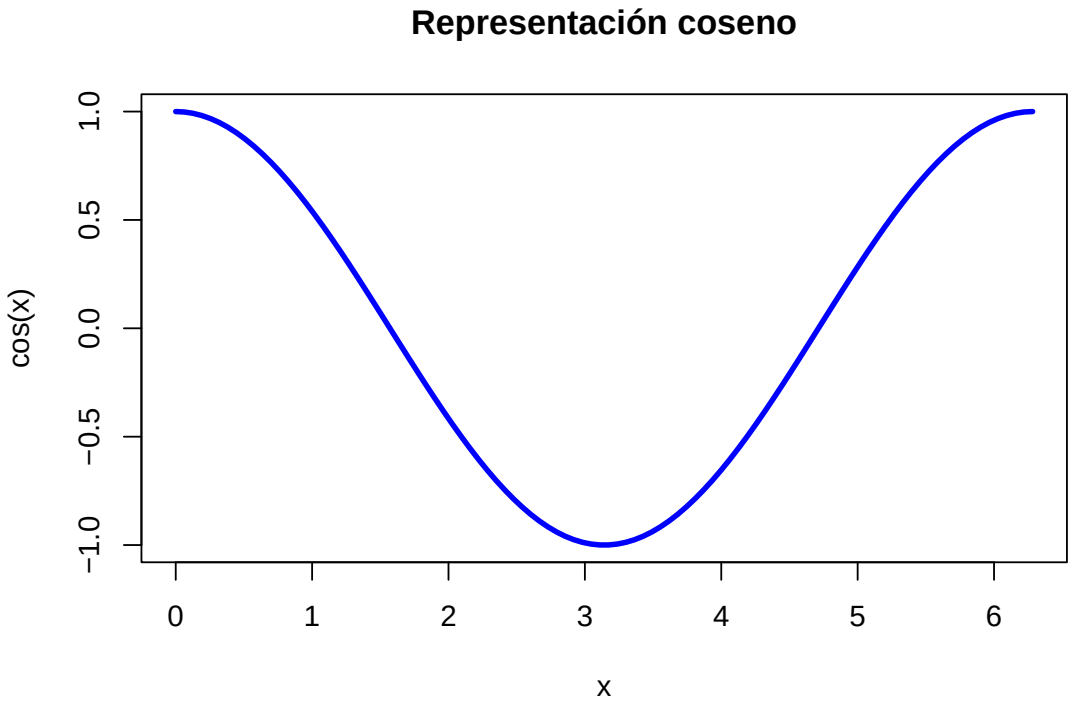



```
plot(x, cos(x))  # orden equivalente a la anterior
```



La utilización de argumentos de la función plot() mejora el aspecto de la representación:

```
plot(x, cos(x),
     type = "l",    # representación tipo línea
     col = "blue",       # color azul
     lwd = 3,       # grosor de la línea
     main = "Representación coseno" # título del gráfico
     )
```

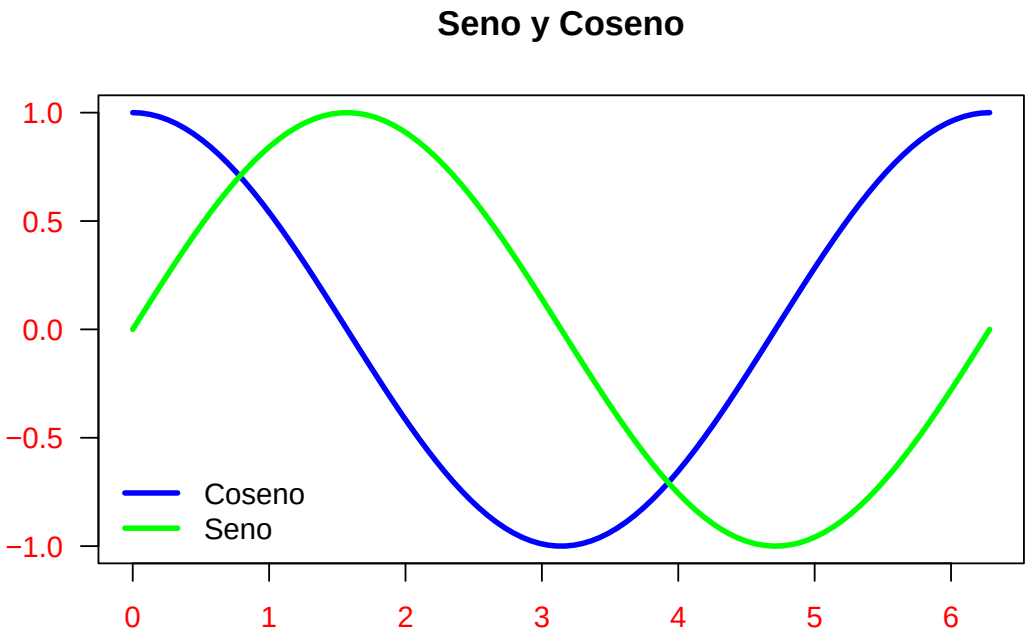


Representación simultánea de varias funciones

Supongamos ahora que nuestro objetivo es construir un gráfico en el que se representan simultáneamente las funciones `sen(x)` y `cos(x)` en $[0, 2\pi]$. Utilizaremos para ello la función lines()

```
plot(x, cos(x),
     type="l",
     col="blue",
     lwd=3,
     main="Seno y Coseno",
     xlab="",  # etiqueta eje OX
     ylab="", # etiqueta eje OY
     las=1, # orientación valores ejes
     col.axis="red"  # color valores ejes
     )

lines(x, sin(x),  # añade la representación del seno(x)
      col="green",
```

```
      lwd=3)

legend("bottomleft",   #añade leyenda parte inferior izquierda
       col=c("blue","green"),
       legend =c("Coseno","Seno"),
       lwd=3,
       bty = "n"  #elimina el recuadro de la leyenda
       )
```



El uso de leyendas en un gráfico permite identificar elementos de la representación

La función legend() permite generar tales leyendas. La función legend() se ejecuta tras la generación de una representación gráfica. Su sintaxis responde a la siguiente estructura:

```
legend(x, y,    # Coordenadas de situación
               # Se puede hacer una ubicación automática con los valores:
               # "bottomright", "bottom", "bottomleft", "left",
               # "topleft", "top", "topright", "right", "center"
       tittle, # título de la leyenda
       legend, # Vector de valores representados en el gráfico
       fill,   # Rectángulos con los colores del gráfico
               # para cada valor de legend
       col, # Color de las líneas o símbolos
       border = "black", # Color del borde de los rectángulos
       lty, lwd,         # Tipo y ancho de línea
       pch,              # Añade símbolos pch a las líneas o rectángulos
```

```
       bty = "o",          # Tipo de caja de la leyenda
                           # bty = "n" elimina el borde
       bg = par("bg")      # Color de fondo de la leyenda
       box.lwd = par("lwd"), # Ancho de línea de la leyenda
       box.lty = par("lty"), # Tipo de línea de la leyenda
       box.col = par("fg"),  # Color de línea de la leyenda
       cex = 1,            # Tamaño de la leyenda
       horiz = FALSE       # Leyenda horizontal (TRUE) o vertical (FALSE)
       title = NULL        # Título de la leyenda
)
```

La función lines() es una de las funciones que hemos denominado de **bajo nivel**. Para su ejecución, requiere que exista un gráfico preexistente.

5.2.2.2. Diagrama de tallos y hojas

El diagrama de tallos y hojas fue propuesto por Tukey (1977) y permite ver la distribución de los datos. A partir de él es muy fácil determinar:

- Mediana de los datos
- Simetría de la distribución (en función de como se agrupen los datos antes y después de la mediana.
- Outliers. Observaciones individuales que caen muy por fuera del patrón general de los datos
- *Huecos* en la distribución

```
x <- rnorm(25)  # se generan 25 valores de una N(0,1)
stem(x)
```

```
  The decimal point is at the |

  -1 | 41
  -0 | 976552221
   0 | 00227789
   1 | 03458
   2 | 0
```

5.2.2.3. Diagrama de barras

El diagrama de barras se utiliza para variables cualitativas o cuantitativas discretas. Sobre el eje de ordenadas se representan las modalidades de la variable y sobre cada modalidad se levanta una barra de altura la frecuencia de dicha modalidad.

```
barplot(height,
        width = 1,
        space = NULL,
        names.arg = NULL,
        legend.text = NULL,
        beside = FALSE,
        horiz = FALSE, ...)
```

Argumentos:

- *height*: vector o matriz de valores con las frecuencias de cada modalidad. que describen las barras.
  - Si es un vector, entonces la orden barplot() genera una secuencia de barras rectangulares cuyas alturas corresponden a los valores del vector.
  - Si es una matriz y 'beside=FALSE´, entonces cada barra del gráfico corresponde a una columna de la matriz. La representacion proporciona barras apiladas.
  - Si es una matriz y 'beside=TRUE´,las barras aparecen yuxtapuestas.
- *width*: especifica mediante un vector el ancho de las barras.
- *space*: determina el espacio entre barras. Puede especificarse con un único valor o un valor para cada barra.
- *names.arg*: Un vector categórico con la identificación de las barras. Si se omite, entonces los nombres son tomados de los atributos de nombres que estén contenidos en el objeto especificado en ´height'.

Trabajaremos con el fichero de datos *airquality* disponible en el libro base de R. En el se recoge información de la calidad del aire de Nueva York entre mayo y septiembre de 1973. Las variables que se analizan son :

- ***Ozone***. Nivel de ozono (partes por millón)
- ***Solar.R***. Nivel de radiación solar (Amgstrom)
- ***Wind***. Velocidad del viento (millas por hora)
- ***Temp***. Temperatura (grados Fahrenheit)
- ***Month***. Mes (codificado de 1 a 12)
- ***Day***. Día del mes (1 a 31)

```
head(airquality) # cabecera de la hoja de datos
```

```
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
```

```
str(airquality)  #información variables hoja de datos
```

```
'data.frame':   153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

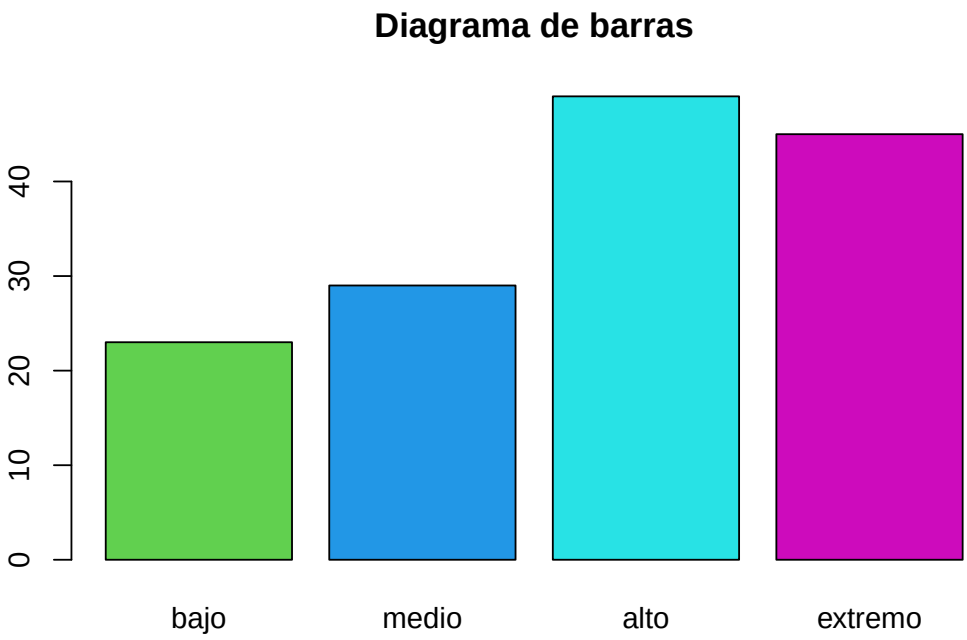
A partir de la información disponible, generaremos una nueva variable, tipo factor, que recopile los niveles de radiación en cuatro categorías: bajo, medio, alto y extremo. Denominaremos dicha variable *nivel.radiacion*

```
attach(airquality)  # permite el acceso directo
                    #  a las variables de la base de datos

#generamos una variable factor con los niveles radiación

nivel.radiacion = cut(Solar.R , breaks = c(0,75,150,250,350),  right=T)

nivel.radiacion<-factor(nivel.radiacion,
                        labels=c("bajo", "medio", "alto", "extremo"))

table(nivel.radiacion)
```

```
nivel.radiacion
   bajo   medio    alto extremo
     23      29      49      45
```

El diagrama de barras de la variable *nivel.radiacion* quedaría:

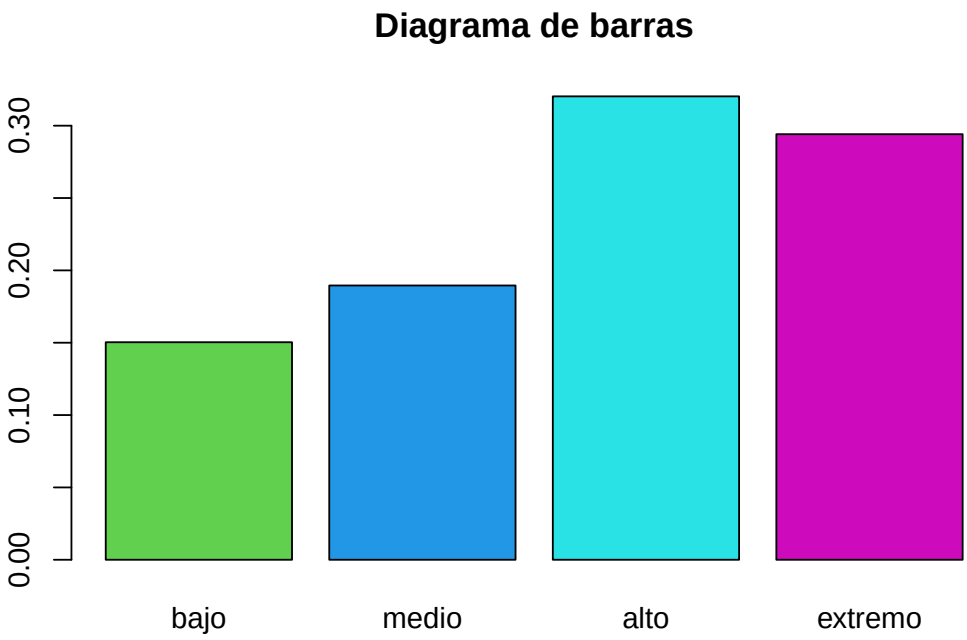
```
barplot(table(nivel.radiacion), col=3:6,
        main="Diagrama de barras")
```



NOTA: La función barplot() requiere como argumento las **FRECUENCIAS** de cada una de las modalidades

Para poder establecer comparaciones con diagramas de barras es preciso trabajar con frecuencias relativas

```
barplot(table(nivel.radiacion)/length(nivel.radiacion), col=3:6,
        main="Diagrama de barras")
```



5.2.2.4. Diagrama de sectores

El diagrama de sectores es un gráfico con estructura circular en el que se representan todas las modalidades de la variable (cualitativa o cuantitativa discreta) mediante sectores de amplitud proporcional a la frecuencia de aparición de la modalidad correspondiente.

El diagrama de sectores en variables cualitativas es uno de los recursos estadísticos más utilizados. Es especialmente útil en los casos en que son pocos los valores que toma la variable.

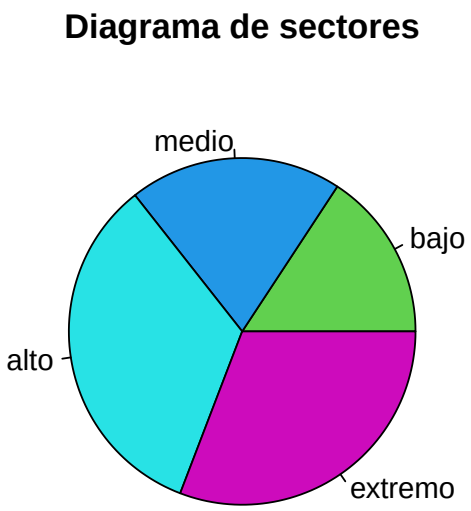
```
pie(x,
    labels=names(x),
    edges=200,
    radius=0.8,
    col=NULL,...)
```

Argumentos:

- *x*: Vector de cantidades positivas, representan las áreas de cada sector del gráfico
- *labels*: Vector de caracteres "strings" que dan nombres a las áreas..
- *edges*: Aproxima la linea exterior circular mediante un polígono con el número de lados especificado, que por defecto es 200.
- *radius*: Radio del diagrama.

Al igual que con el diagrama de barras, la función pie() del diagrama de sectores toma como argumento la distribución de frecuencias de cada una de las modalidades de la variable analizada

```
pie(table(nivel.radiacion),
    col=3:6,
    main="Diagrama de sectores")
```

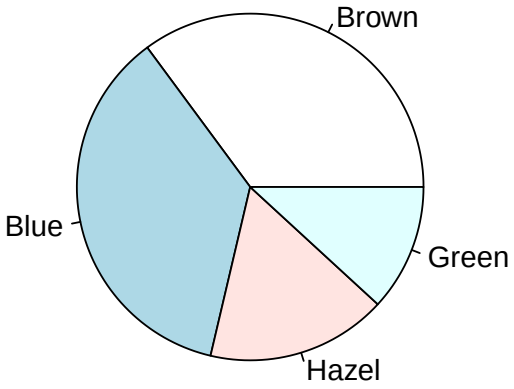


Ejercicio: Representar mediante una diagrama de sectores la variable *color de ojos* de la hojas de datos HairEyeColor

```
head(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
```

```
pie(colSums(HairEyeColor[,,Sex="Male"]))
```



5.2.2.5. Histograma

El histograma es una representación gráfica para variables cuantitativas continuas. Permite representar la forma en que los datos se distribuyen. Para ello, los datos se agrupan en intervalos y se levantan barras con base dichos intervalos de modo que el área de la barra representada sea proporcional a la frecuencia de observaciones en el intervalo

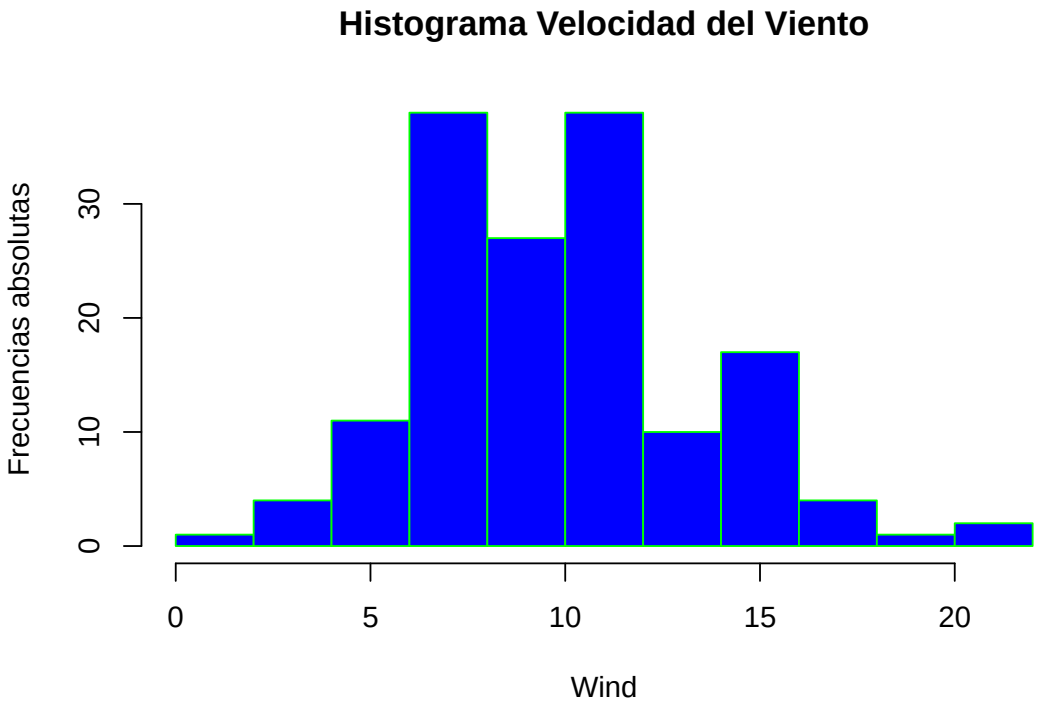
```
hist(x,
     breaks,
     freq,
     right, ...)
```

Argumentos:

- *x*: Vector de valores para el que se construye el histograma.
- *breaks*: Puede ser un valor con el cual se indica el número aproximado de clases o un vector cuyos elementos indican los puntos límites entre las clases o intervalos.
- *freq*: argumento lógico; si se especifica como `TRUE`, el histograma representa las frecuencias absolutas o conteo de datos en cada clase; si se especifica como `FALSE`, reprenta la densidad de probabilidad del intervalo. Se calcula como la *frecuencia relativa del intervalo, dividida por la amplitud del mismo*. Por defecto, este argumento toma el valor de `TRUE` siempre y cuando los intervalos sean de igual amplitud.

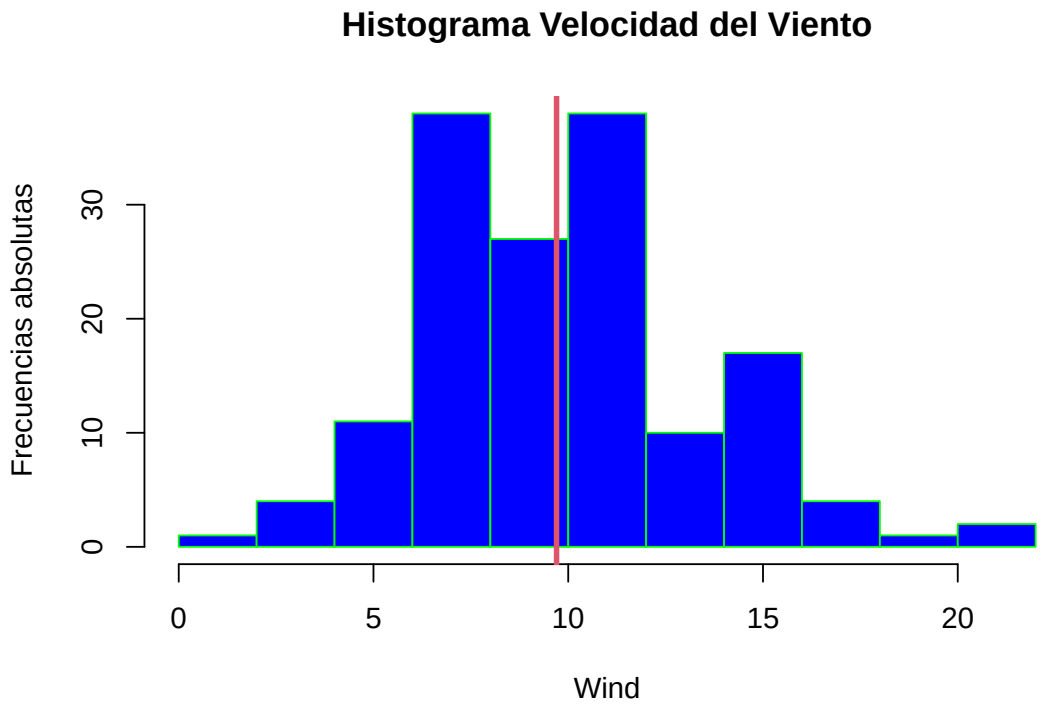
- *right*: Argumento lógico; si es `TRUE`, los intervalos son abiertos la izquierda - cerrados a la derecha (a,b]. Para la primera clase o intervalo si `include.lowest = TRUE`, el valor más pequeño de los datos será incluido en éste. Si es `FALSE` los intervalos serán de la forma [a,b) y el argumento `include.lowest=TRUE` tendrá el significado de incluir el "más elevado".
- *col*: Para definir el color de las barras. Por defecto, "NULL" produce barras sin color.
- *border*: Para definir el color de los bordes de las barras.
- *plot*: Argumento lógico. Por defecto es `TRUE`, y el resultado es el gráfico del histograma; si se especifica como `FALSE` el resultado es una lista con los valores estimados para el cálculo del histograma.
- *labels*: Argumento lógico o carácter. Si se especifica como `TRUE` coloca etiquetas encima de cada barra.

```
hist(airquality$Wind,
     col = "blue",
     border = "green",
     main = "Histograma Velocidad del Viento",
     xlab = "Wind",
     ylab = "Frecuencias absolutas"
      )
```



El histograma permite determinar la forma de la distribución. Informa de la posición de los valores centrales así como de la simetría de las observaciones.

```
hist(airquality$Wind,
     col = "blue",
     border = "green",
     main = "Histograma Velocidad del Viento",
     xlab = "Wind",
     ylab = "Frecuencias absolutas"
      )

abline(v=median(airquality$Wind), col=2, lwd=3)
```



Para acceder a la información numérica que también genera esta función utilizaremos el código:

```
histograma_wind<-hist(airquality$Wind,
     col = "blue",
     border = "green",
     main = "Histograma Velocidad del Viento",
     xlab = "Wind",
     ylab = "Frecuencias absolutas",
     plot = FALSE
      )

histograma_wind[]
```

```
$breaks
```

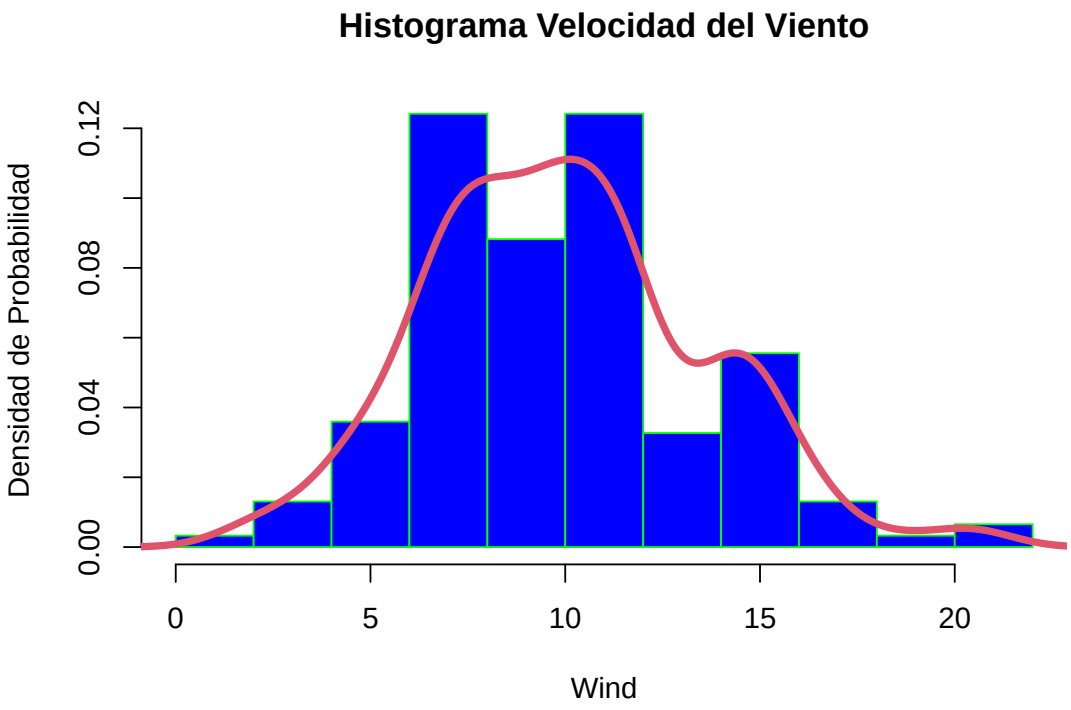
```
 [1]  0  2  4  6  8 10 12 14 16 18 20 22

$counts
 [1]  1  4 11 38 27 38 10 17  4  1  2

$density
 [1] 0.003267974 0.013071895 0.035947712 0.124183007 0.088235294 0.124183007
 [7] 0.032679739 0.055555556 0.013071895 0.003267974 0.006535948

$mids
 [1]  1  3  5  7  9 11 13 15 17 19 21

$xname
[1] "airquality$Wind"

$equidist
[1] TRUE
```

5.2.2.6. Función de densidad de una variable mediante un estimador tipo núcleo

La función de densidad de probabilidad empírica es una versión suavizada del histograma. Esta suavización también se conoce como estimador de Parzen-Rosenblatt o estimador kernel (tipo núcleo). La función de densidad de una variable se calcula con la función density()

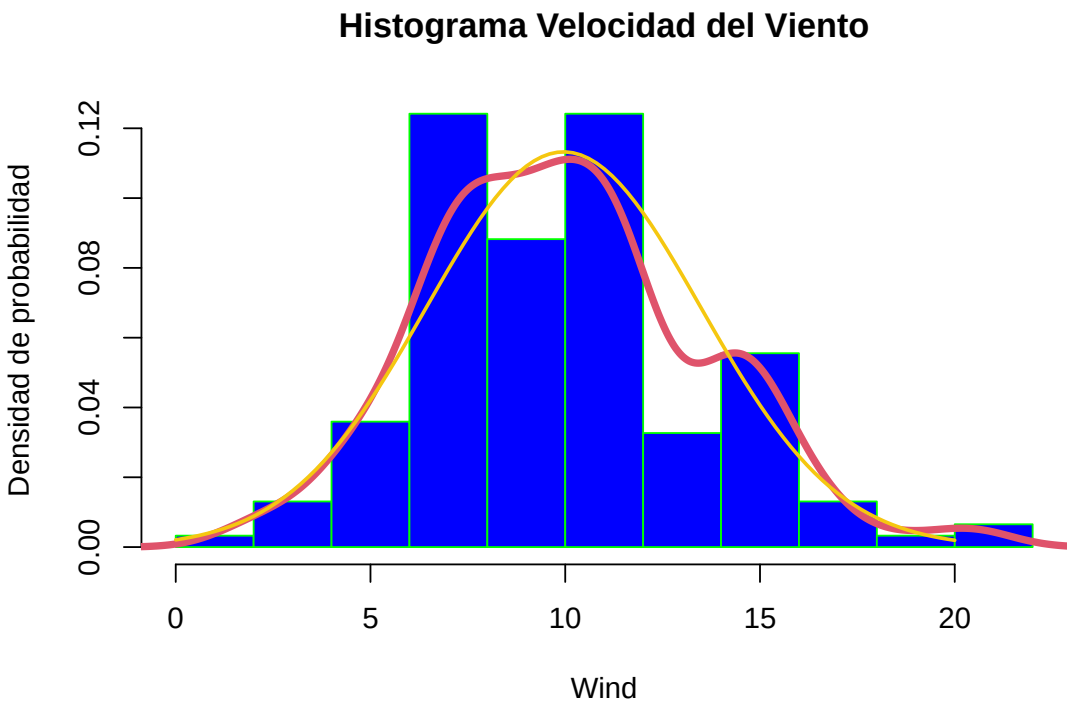
```
density(x)
```

```
hist(airquality$Wind,
     col = "blue",
     border = "green",
     freq = FALSE , # se representan las Densidades de Probabilidad
     main = "Histograma Velocidad del Viento",
     xlab = "Wind",
     ylab = "Densidad de Probabilidad"
      )

lines(density(airquality$Wind),
      lwd = 4,
      col=2)
```



Para comparar el comportamiento de nuestros datos con el de la verdadera distribución normal (con media y varianza la de los datos), representamentos también la **distribución normal teórica**:

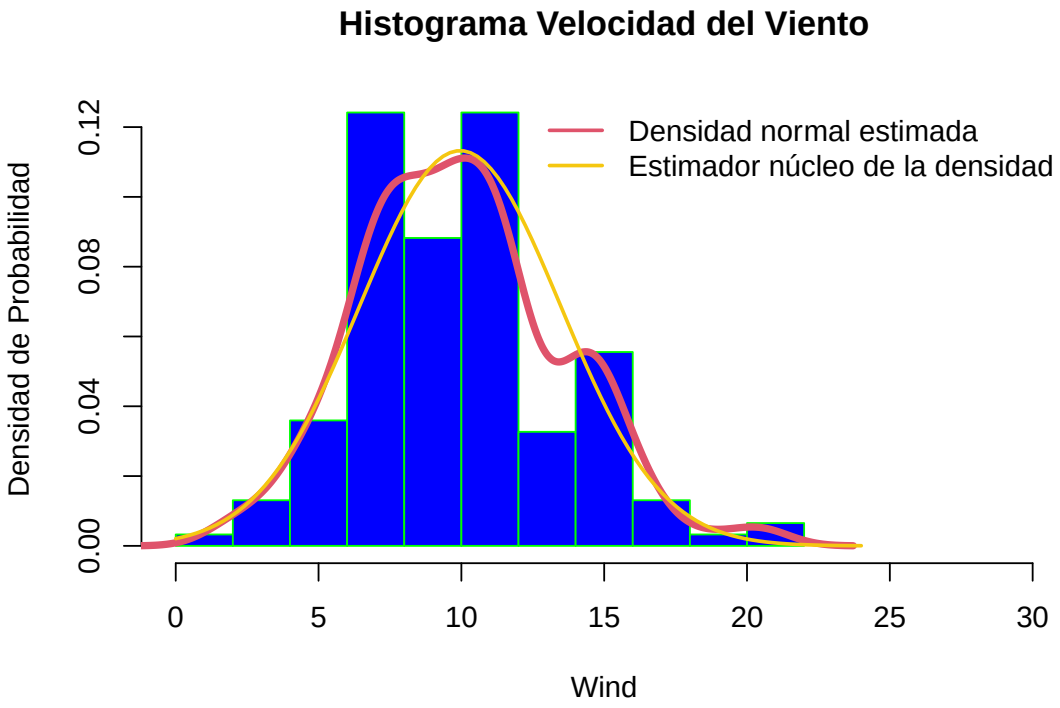
```
hist(airquality$Wind,
     col = "blue",
     border = "green",
     freq = FALSE , # se trabaja con DENSIDADES DE FRECUENCIAS
     main = "Histograma Velocidad del Viento",
     xlab = "Wind",
     ylab = "Densidad de probabilidad"
      )

lines(density(airquality$Wind),
      lwd = 4,
      col=2)

curve(dnorm(x,mean=mean(airquality$Wind),sd=sd(airquality$Wind)),
      from = 0,
      to = 20,
      add = TRUE,
      col = 7,
      lwd = 2)
```



LEYENDA: Por último añadimos una leyenda explicando qué representa cada curva:

```
hist(airquality$Wind,
     col = "blue",
     border = "green",
     freq = FALSE , # Se representan las densidades de probabilidad
     main = "Histograma Velocidad del Viento",
     xlab = "Wind",
     ylab = "Densidad de Probabilidad",
     xlim = c(0,30)
      )

lines(density(airquality$Wind),
      lwd = 4,
      col=2)

curve(dnorm(x,mean=mean(airquality$Wind),sd=sd(airquality$Wind)),
      from = 0,
      to = 24,
      add = TRUE,
      col = 7,
      lwd = 2)

legend("topright",
       col=c(2, 7),
       legend =c("Densidad normal estimada",
```

```
                "Estimador núcleo de la densidad"),
       lwd=2,
       bty = "n"
       )
```



5.2.2.7. Diagrama de cajas y bigotes: *Box-Plot*

El Box-Plot o diagrama de cajas y bigotes debe su origen a Tukey (1977) y es ampliamente utilizado por la gran cantidad de información que aporta. Utiliza para ello cinco medidas numéricas: mediana, cuartiles, intervalos determinación valores extremos.

Entre sus principales características destacamos:

1. Describe de forma clara la distribución de los datos y sus principales características referentes a la localización y dispersión a la vez que informa sobre la simetría y la distribución de las colas
2. Permite comparar diversas poblaciones simultáneamente.

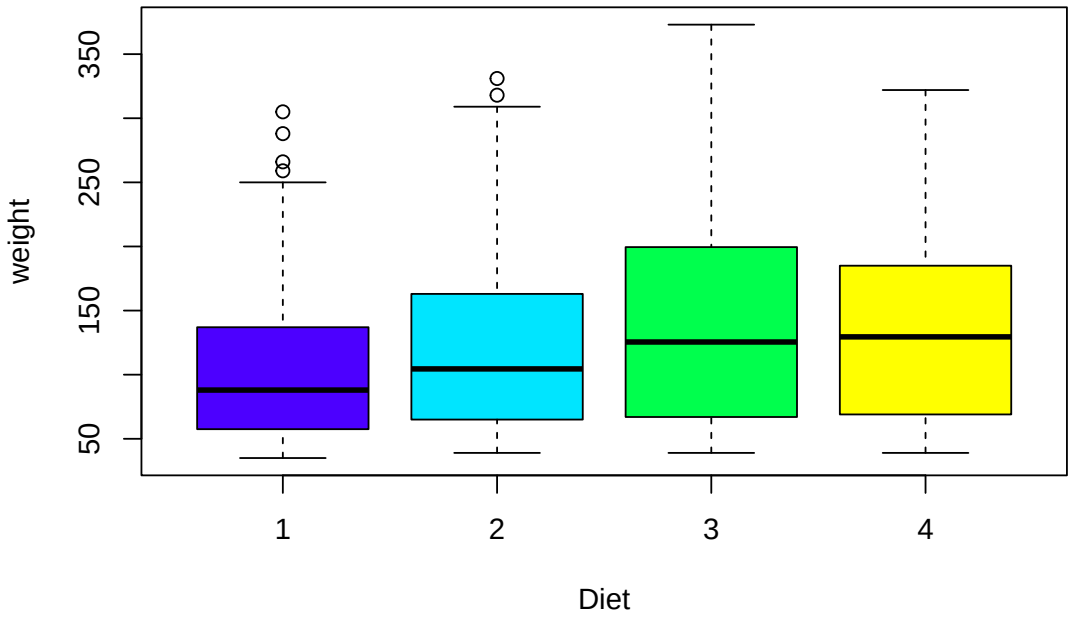
La función de R para obtener un diagrama de cajas y bigotes en R es:

```
boxplot(x,
        data,
        range,
        width,
```

```
        varwidth,
        plot,
        horizontal, ...)
```

Argumentos:

- *x*, ...: Datos que van a ser graficados con el boxplot. Puede ser un conjunto de vectores separados, cada uno correspondiendo a un boxplot componente o una lista que contiene a tales vectores. Alternativamente una especificación de la forma x~g puede ser dada para indicar que las observaciones en el vector x van a ser agrupadas de acuerdo a los niveles del factor g.
- *data*:indica el nombre del data.frame en el que se encuentran las variables que se quieren graficar.
- *range*: Determina la extensión de los bigotes de la caja. Si es positivo, se extienden hasta el dato más extremo.
- *width*: Vector que da los anchos relativos de las cajas.
- *varwidth*: Si es "TRUE", las cajas son dibujadas con anchos proporcionales a las raíces cuadradas del número de observaciones en los grupos.
- *plot*: Si es "TRUE" (por defecto) entonces se produce el gráfico. De lo contrario, se producen los resúmenes de los boxplots.
- *horizontal*: Si es "FALSE" (por defecto) los diagramas son dibujados verticalmente.



5.2.3. Funciones gráficas de bajo nivel

Permiten añadir líneas, puntos, etiquetas... a un gráfico ya existente. Son de gran utilidad para completar un gráfico. Entre estas funciones cabe destacar:

- lines(): Permite añadir lineas (uniendo puntos concretos) a una gráfica ya existente.
- abline(): Añade lineas horizontales, verticales u oblicuas, indicando pendiente y ordenada.
- points(): Permite añadir puntos.
- legend(): Permite añadir una leyenda.
- text(): Añade texto en las posiciones que se indiquen.
- grid(): Añade una malla de fondo.
- title(): permite añadir un título o subtítulo.

Otras funciones permiten identificar puntos en un gráfico:

- Función locator(): al situar el cursor sobre la ventana de gráficos, cada vez que pulsemos el botón izquierdo del ratón, se almacenan en memoria las coordenadas del punto que marquemos. Al pulsar la tecla < ESC >, R nos muestra dichas coordenadas en la consola (si hemos ejecutado simplemente locator()) o las guarda en el objeto al que hayamos asignado la salida dicha función.

  Por ejemplo, si hemos ejecutado `posiciones=locator()`, al pulsar < ESC > las coordenadas de los puntos que hayamos marcado en el gráfico se guardan en el objeto `posiciones`.

- Función identify(). Permite identificar con el ratón a qué posiciones dentro del conjunto de datos corresponden los puntos que señalemos en un gráfico.

  Si, por ejemplo, ejecutamos `plot(var1, var2)` para mostrar la nube de puntos de var1 frente a var2, y a continuación ejecutamos `identify(var1, var2)` y picamos en algunos puntos del gráfico con el ratón, al pulsar < ESC >, R nos devuelve en la consola (y representa en el gráfico) los números de orden dentro del data-frame a los que corresponden los puntos que hemos marcado.