FEIR 60: Taller de Gráficos

Apuntes del curso FEIR3, curso 2014/15 actualizados. `Última actualización:` jueves 04 abril 2019, 09:27:50

María Elvira Ferre Jaén

Índice

1. Introducción

Para el desarrollo de este capítulo hemos utilizado como referencia bibliográfica básica el libro Maurandi-López, Balsalobre-Rodríguez, & del-Río-Alonso (2013) y la web Kabacoff (2014).

Los gráficos son una forma muy útil de estudiar tus datos antes de analizarlos. Si no inspeccionamos los datos mediante un gráfico, realmente no sabremos cómo interpretar los resultados.

Si utilizamos el comando `demo(graphics)` podemos ver muchas de las posibilidades que nos ofrece `R` en lo que se refiere a la generación de gráficos.

Al realizar un gráfico en Reste es enviado a un dispositivo gráfico, que no es más que una ventana gráfica o un archivo.

Por otro lado existen diferentes sistemas gráficos que podemos utilizar: *base*, *lattice* o *ggplot2* y generalmente no podremos mezclar.

Además podemos diferenciar entre dos tipos de funciones gráficas

- funciones de alto nivel que lanzan un nuevo *device gráfico*
- funciones de bajo nivel que añaden elementos a una gráfica ya existente.

2. Sistemas gráficos en R

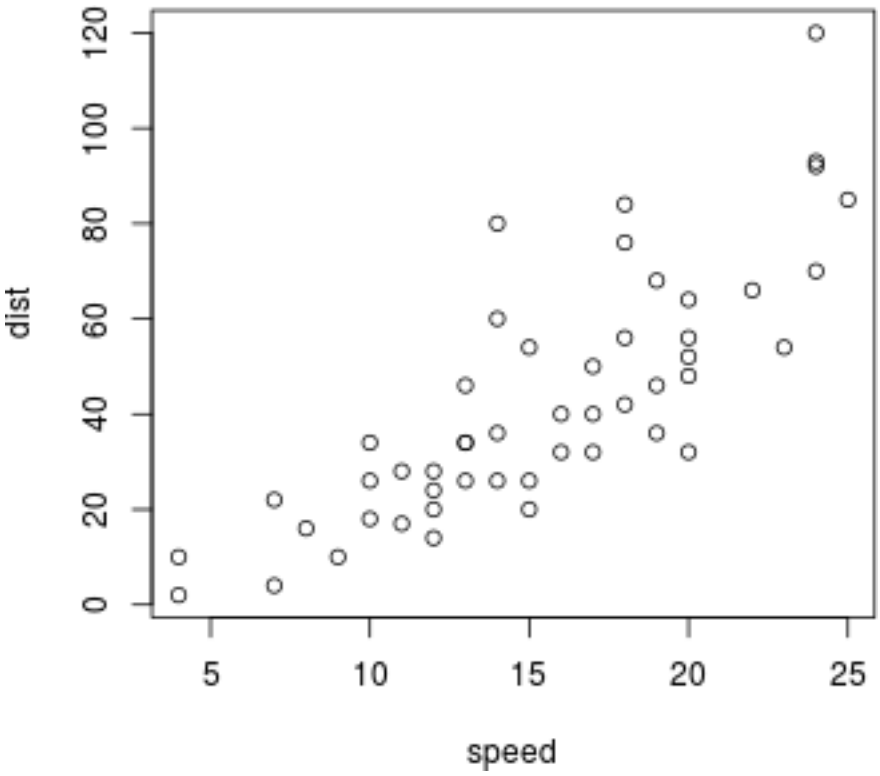
2.1. *Base*

Los gráficos se construyen poco a poco, tratando cada aspecto del gráfico separadamente a través de una serie de funciones; es más simple conceptualmente hablando y permite construir reflejando el proceso de pensamiento.

Los inconvenientes de este sistema son

- No puedes volver atrás una vez que has empezado (e.d. para ajustar márgenes); necesitamos pensarlo de antemano.
- Es difícil "traducirle" el gráfico a otros una vez hemos creado el gráfico
- El gráfico es sólo una serie de expresiones de R

```
library( datasets )
data( cars )
with( cars, plot( speed, dist ) )
```



2.2. *Lattice*

Los gráficos se crean normalmente mediante una única función, por lo que todos los parámetros del gráficos se tienen que especificar a la vez, lo que permite a R calcular automáticamente los espaciados y los tamaños de letra.

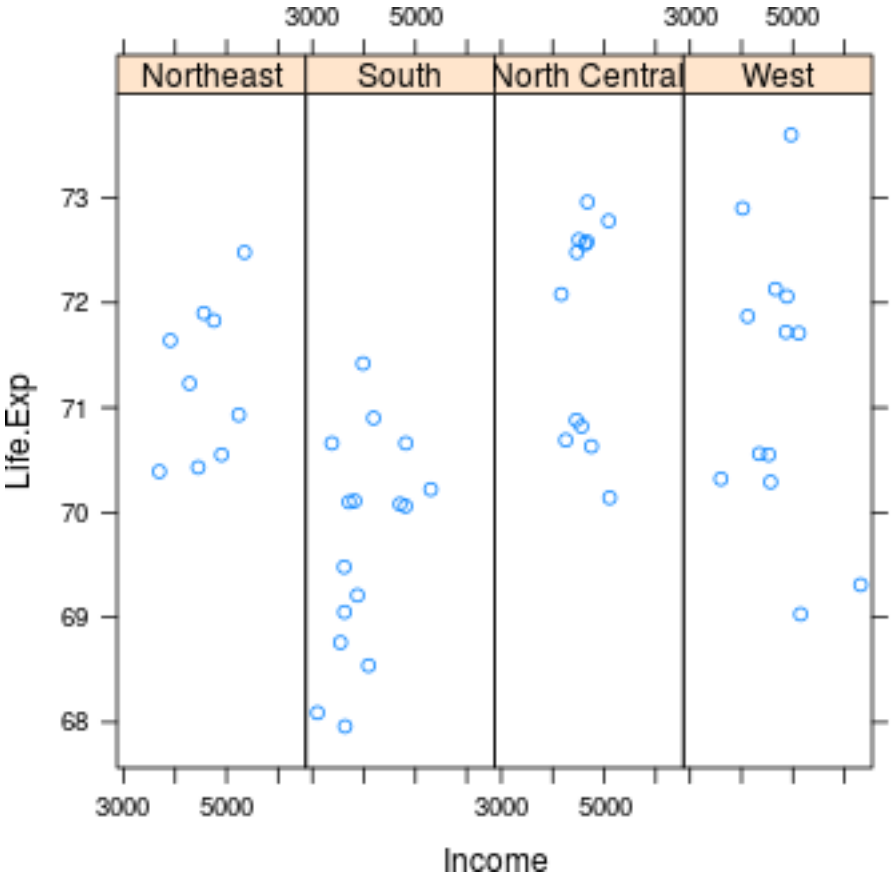
Muy útiles para gráficos condicionados: analizando cómo cambia y a través de los distintos niveles de z. Además es un buen sistema para poner varios gráficos en una mismo cuadro.

Sus inconvenientes son

- A veces es complicado especificar un gráfico completo en una sola llamada a una función

- Los títulos, etiquetas, etc no son intuitivos
- Dificultad de uso y se necesita preparación intensa
- No se puede añadir elementos al gráfico una vez está creado

```
library( lattice )
df <- data.frame( state.x77, region = state.region )
xyplot( Life.Exp ~ Income | region, data = df, layout = c( 4, 1 ) )
```



2.3. *ggplot2*

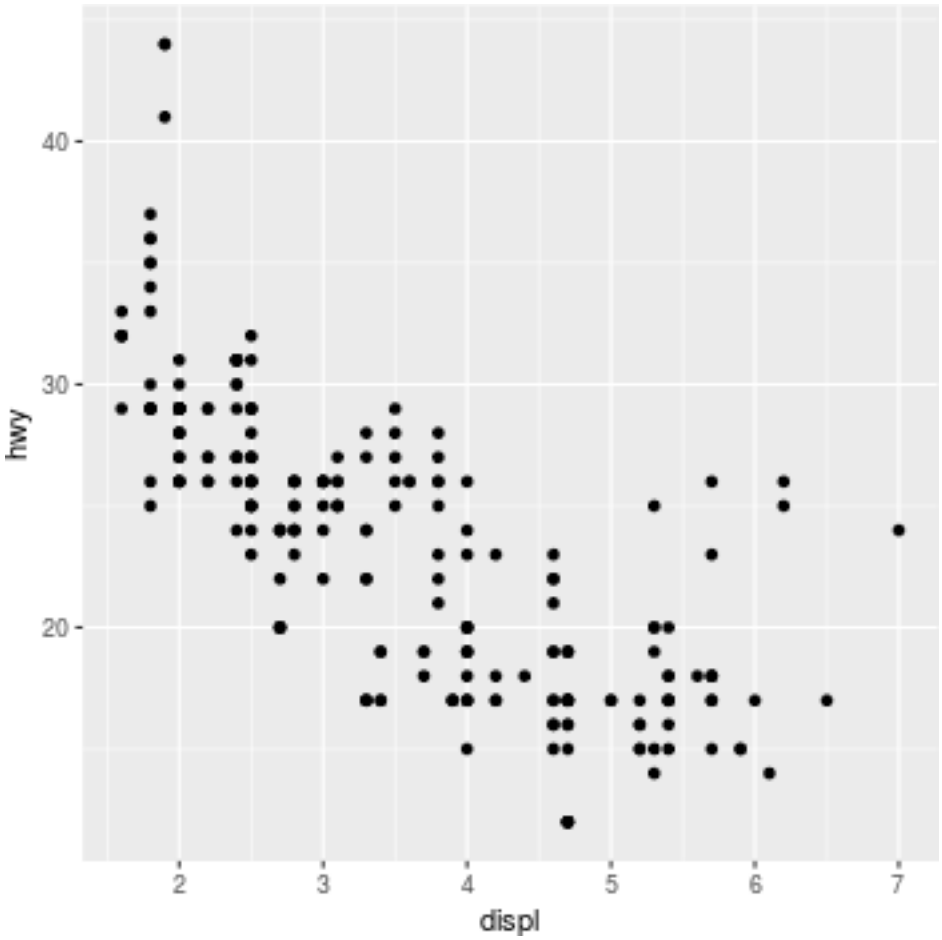
Combina conceptos de los dos sistemas *base* y del *lattice*, pero usa una implementación diferente.

Se ocupa automáticamente con espaciados, texto, títulos pero además nos permite añadir elementos al gráfico.

Superficialmente es similar al sistema *lattice* pero generalmente es más fácil/intuitivo de usar

El modo por defecto hace muchas elecciones por ti, aunque todavía podemos personalizarlo a nuestros deseos.

```
library( ggplot2 )
data( mpg )
qplot( displ, hwy, data = mpg)
```



Como vemos los sistemas gráficos son muy diferentes entre sí y normalmente no podemos mezclarlos. Nosotros nos vamos centrar en el *sistema base* .

Podemos encontrar más información sobre los sistemas gráficos en Roger D. Peng & Caffo (2014).

3. Sistema base

El sistema gráfico básico es el más utilizado usualmente y es un sistema muy poderoso para la creación de gráficos de dos dimensiones. Está implementado en los siguientes paquetes Kabacoff (2014)

- `graphics`: contiene las funciones del sistema de gráfico básico, incluyendo `plot`, `hist`, `boxplot` y otros muchos.
- `grDevices`: contiene los dispositivos gráficos como X11, PDF, PostScript, PNG, etc.

3.1. Dispositivos gráficos (device)

Un dispositivo gráfico es el "sitio" donde hacemos que aparezca el gráfico

- Una ventana del ordenador
  - En Unix/Linux se lanza con `x11()`
  - En Windows se lanza con `windows()`
  - En Mac se lanza con `quartz()`

- Un archivo PDF
- Un archivo PNG o JPEG
- etc

Cuando hacemos un gráfico, debemos considerar para qué usaremos el gráfico para determinar a qué dispositivo gráfico debemos enviarlo. El lugar más común para enviar un gráfico es a una ventana y este es al que por defecto los manda `R` si no indicamos lo contrario.

La lista de dispositivos gráficos disponible en `R` se obtiene ejecutando el comando `?device`. Si abrimos más de un dispositivo gráfico, el último dispositivo que hayamos ejecutado se convierte en el dispositivo activo, sobre el cual se van a dibujar las gráficas que generemos. Con la función `dev.list()` se muestra una lista de todos los dispositivos abiertos.

3.1.1. Manejo de devices

Para saber cual es el dispositivo activo, cambiarlo o bien cerrarlo, utilizamos los siguientes comandos

```
dev.cur( ) # nombre y número del device activo
dev.set( n ) # cambiarimos el device activo al n
dev.off( ) # cerrar el device activo
dev.off( n ) # cerrar el device n
```

3.1.2. Exportar un gráfico

```
library( datasets )
pdf( file = "migrafico.pdf" ) # abre un device PDF y crea "migrafico.pdf" en el wd
# Crea el pdf y lo manda a un archivo, no aparece en el pantalla
plot( faithful$eruptions, faithful$waiting )
dev.off( ) # cierra el device PDF
# Cualquier gráfico que dibujemos ahora aparecerá en la pantalla
```

Podemos guardarlos gráficos como imágenes (.png y .jpeg) con las funciones `png( "migrafico.png" )` y `jpeg( "migrafico.jpg" )`.

3.2. Creación de gráficos en R

Hay dos fases en la creación de un gráfico en este sistema (*base*)

- inicializar un gráfico (funciones de alto nivel)
- añadir elementos a un gráfico existente (funciones de bajo nivel)

Funciones como `plot( )`, `hist( )` lanzan un *device* gráfico (si no está ya creado) y representan un nuevo gráfico en el dispositivo gráfico.

Estas funciones tienen muchos *argumentos* como poner un título, etiquetas a los ejes, etc. Para ver los argumentos utilizamos la ayuda de la función, `?plot`.

Además el sistema gráfico básico tiene muchos *parámetros* que podemos establecer y ajustar. Todos estos parámetros están documentados en `?par`, los veremos más adelante.

3.3. Funciones gráficas de alto nivel

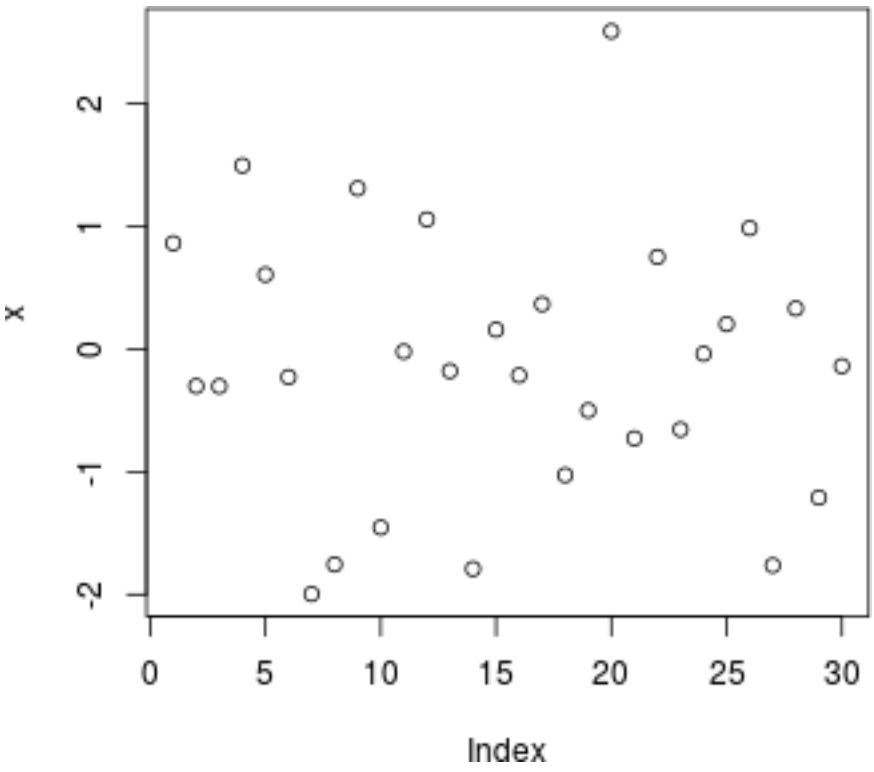
Veamos a continuación algunas de las funciones gráficas más utilizadas. Todas ellas admiten los siguientes **argumentos generales**

- `add = FALSE` (por defecto): si es TRUE superpone al gráfico existente
- `axes = TRUE` (por defecto): si es FALSE no dibuja los ejes ni la caja del gráfico
- `type =`: especifica el tipo de gráfico, p = puntos (por defecto), l = lineas, b = puntos conectados por lineas, h = lineas verticales
- `xlim =`, `ylim =`: especifica los límites inferiores y superiores de los ejes
- `xlab =`, `ylab =`: añade etiquetas a los ejes
- `main =`: añade el título principal
- `sub =`: añade un subtitulo (letra más pequeña).

3.3.1. Gráficos de dispersión

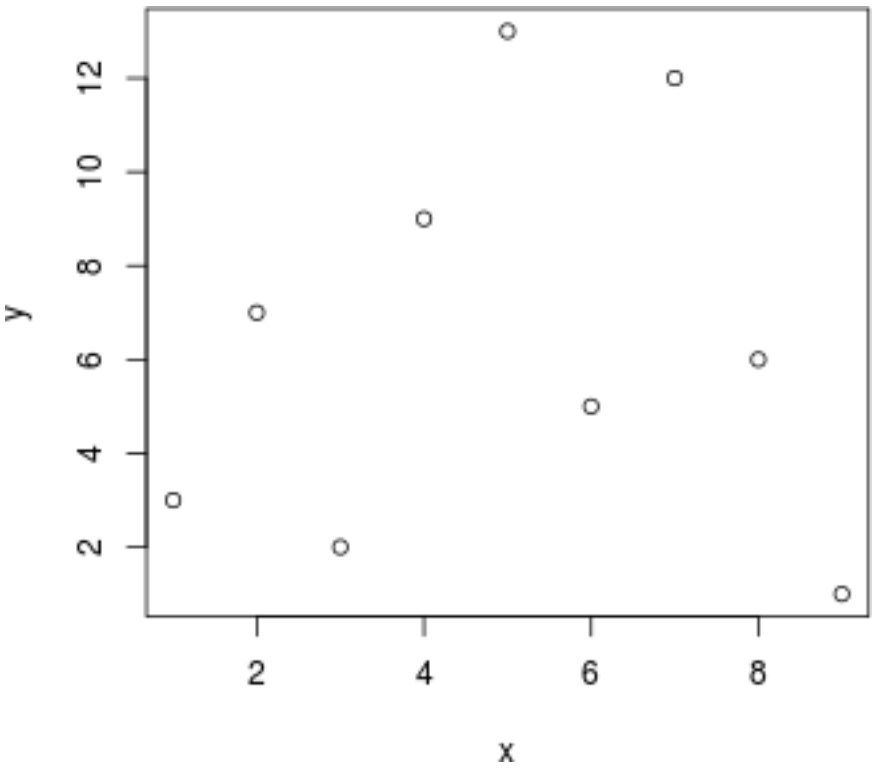
- `plot( x )`: dibuja los valores de x (en el eje y) ordenados en el eje x.

```
set.seed( 13614 )
x <- rnorm( 30 )
plot( x )
```

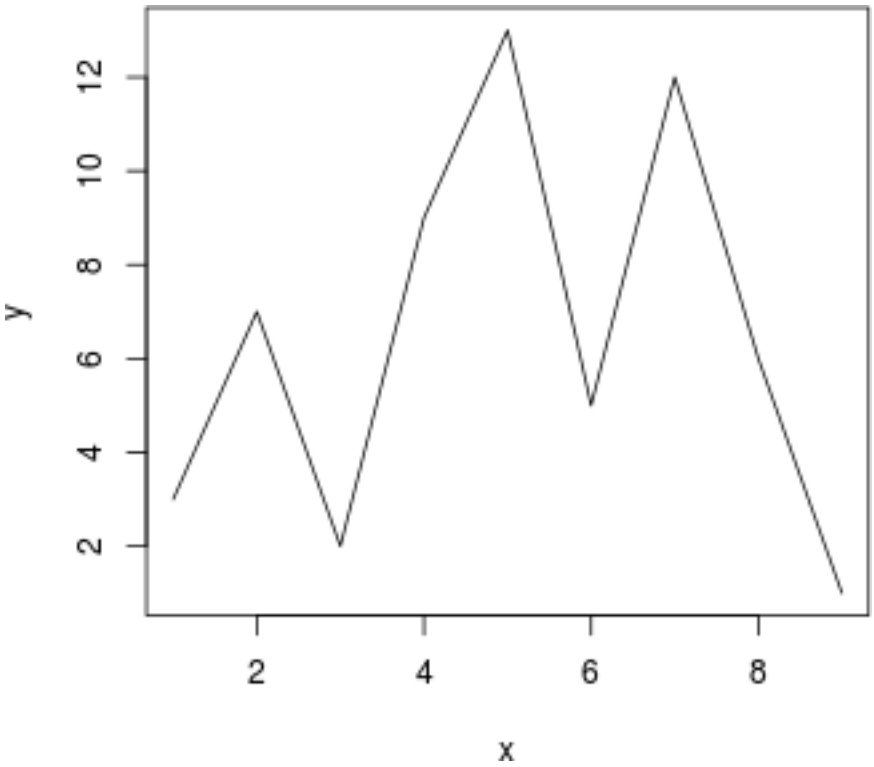


- `plot( x, y )`: gráfico bivariado para los valores de x e y.

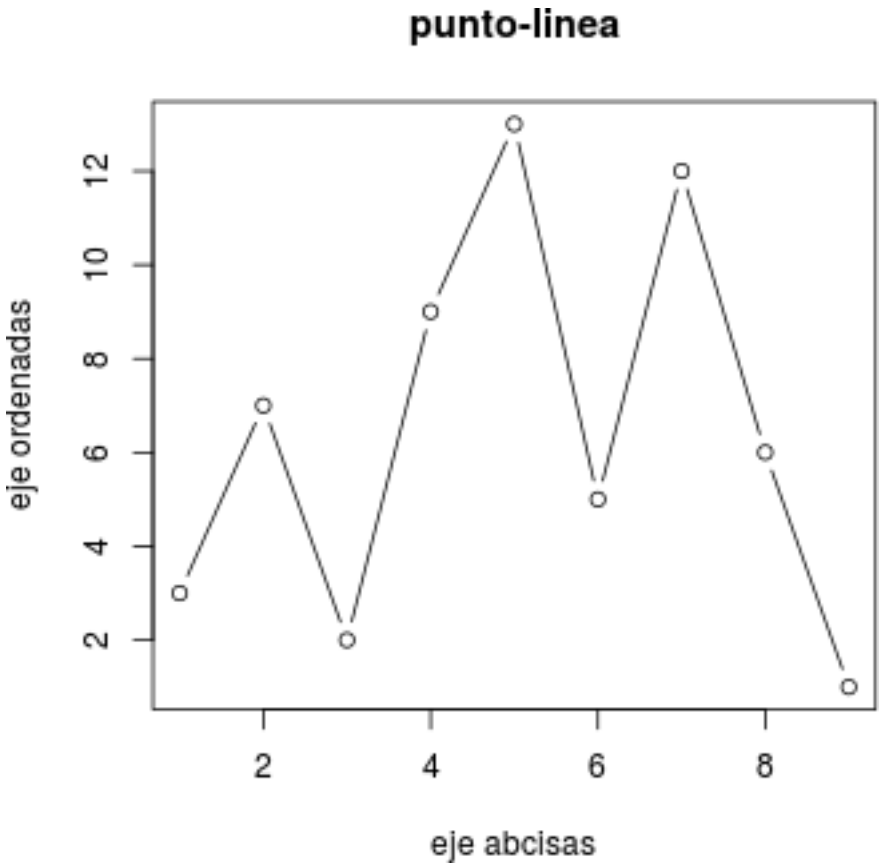
```
x <- 1:9; y <- c( 3, 7, 2, 9, 13, 5, 12, 6, 1 ) # create some data
plot( x, y )
```



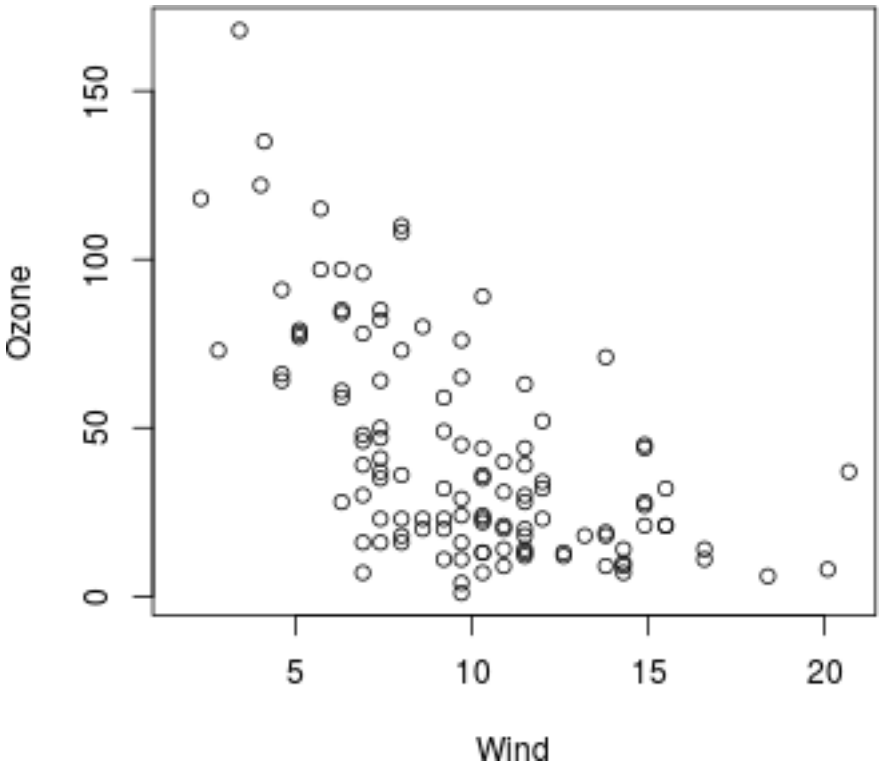

```
plot( x, y, type = "l" )
```




```
plot( x, y, type = "b", main = "punto-linea",
      xlab = "eje abcisas", ylab = "eje ordenadas" )
```

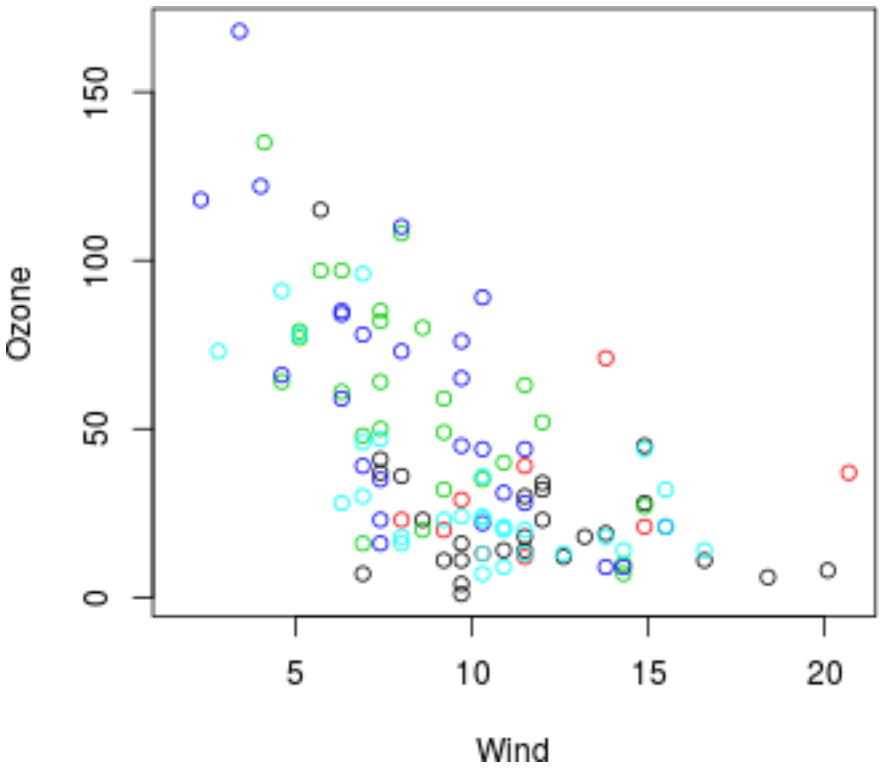



```
library( datasets )
with( airquality, plot( Wind, Ozone ) )
```



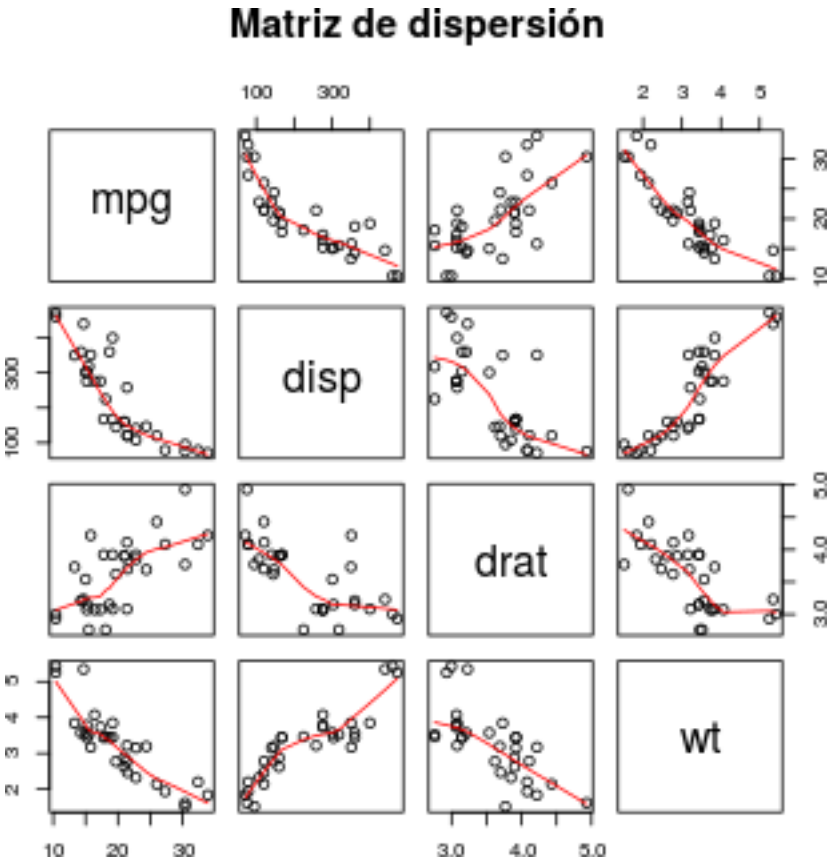
Por colores

```
colores <- factor( airquality$Month )
with( airquality, plot( Wind, Ozone , col = colores ) )
```



- `pairs(x)`: si x es una matriz o un *data.frame*, esta función dibuja todas las posibles gráficas bivariadas entre las columnas de x.

```
df <- mtcars[ , c( 1, 3, 5:6 ) ]
pairs( df, panel = panel.smooth, main = "Matriz de dispersión" )
```



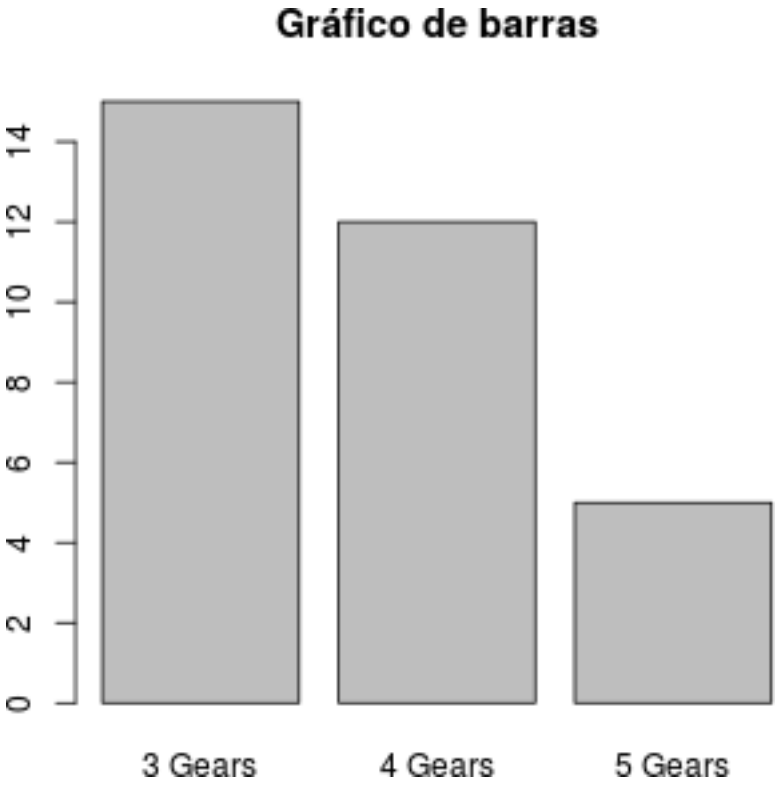
3.3.2. Gráficos de barras

Creamos gráficos de barras con la función `barplot( x )`, donde `x` es un vector o una matriz. Si `x` es un vector, los valores determinan las alturas de las barras y si es una matriz y `beside = FALSE` (por defecto) entonces cada barra corresponde a una columna de altura, con los valores de la columna dando las alturas de "sub-barras" apiladas. Si `x` es una matriz y `beside = TRUE`, entonces los valores de cada columna se yuxtaponen en lugar de apilarse.

Veamos un ejemplo de cada uno de estos casos (Maurandi-López et al., 2013).

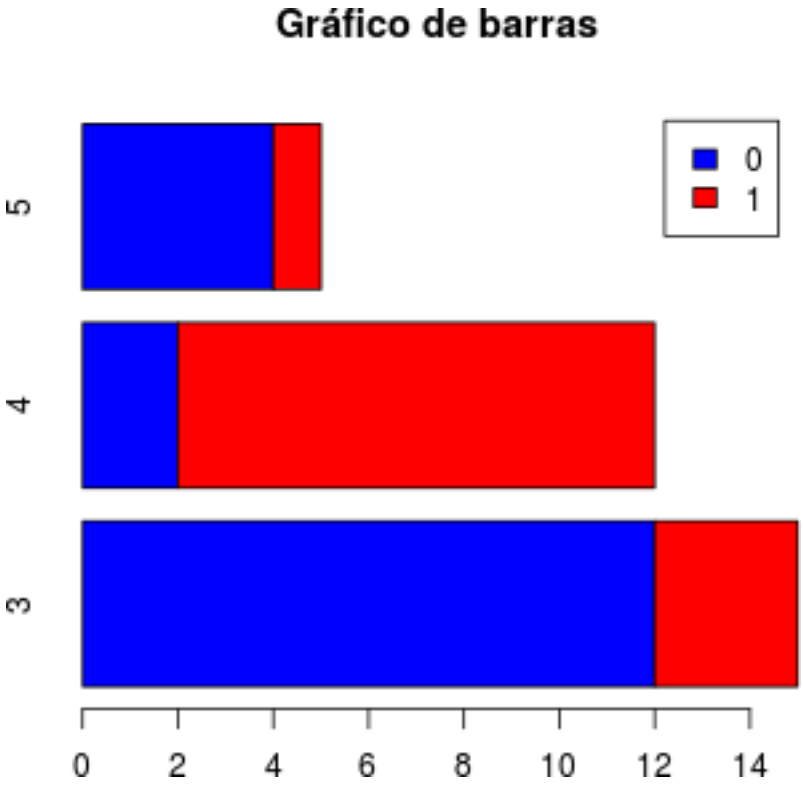
- **Gráfico de barras simple**

```
x <- table( mtcars$gear )
barplot( x, main = "Gráfico de barras",
         names.arg = c("3 Gears", "4 Gears", "5 Gears") )
```



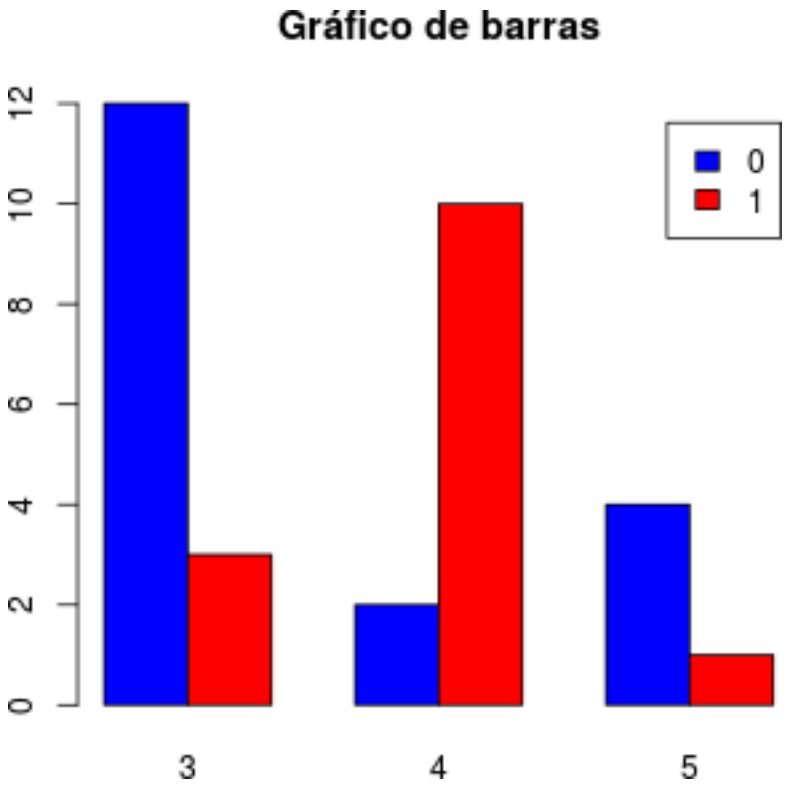
- **Gráfico de barras apilado**

```
x <- table( mtcars$vs, mtcars$gear )
barplot( x, main = "Gráfico de barras", horiz = TRUE,
         col = c("blue","red" ), legend = rownames( x ) )
```



- **Gráfico de barras agrupado**

```
barplot( x, main = "Gráfico de barras", beside = TRUE,
         col = c("blue","red" ), legend = rownames( x ) )
```



3.3.3. Boxplot

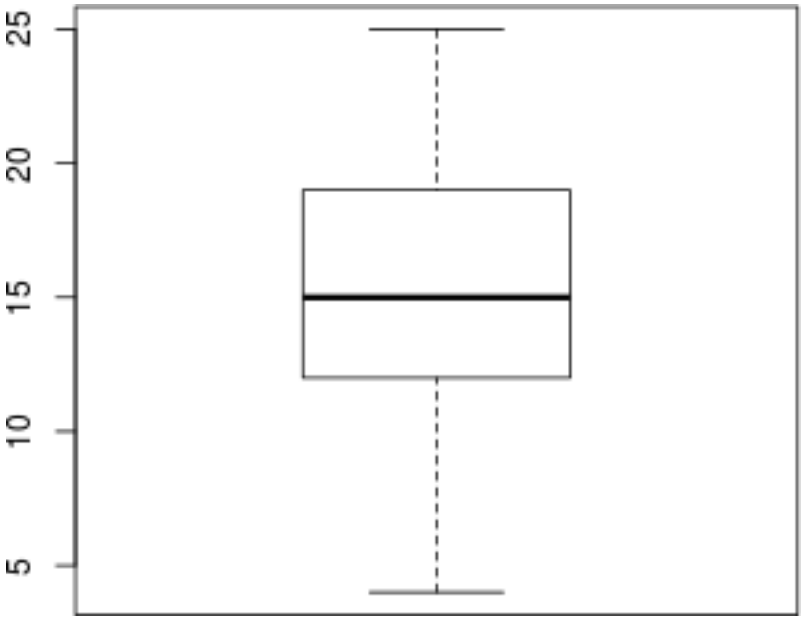
Diagramas de caja-bigote se pueden crear para variables individuales o para variables por grupo. El formato de la función es `boxplot( x, data = )`, donde `x` es una fórmula y `data` es el *data.frame* que contiene los datos.

Un ejemplo de una fórmula es `y ~ grupo` en el que se genera un diagrama de cajas separado para la variable numérica por cada valor de grupo.

La opción `varwidth = TRUE` hace la anchura del diagrama proporcional a los tamaños muestrales y `horizontal = TRUE` invierte la orientación del eje.

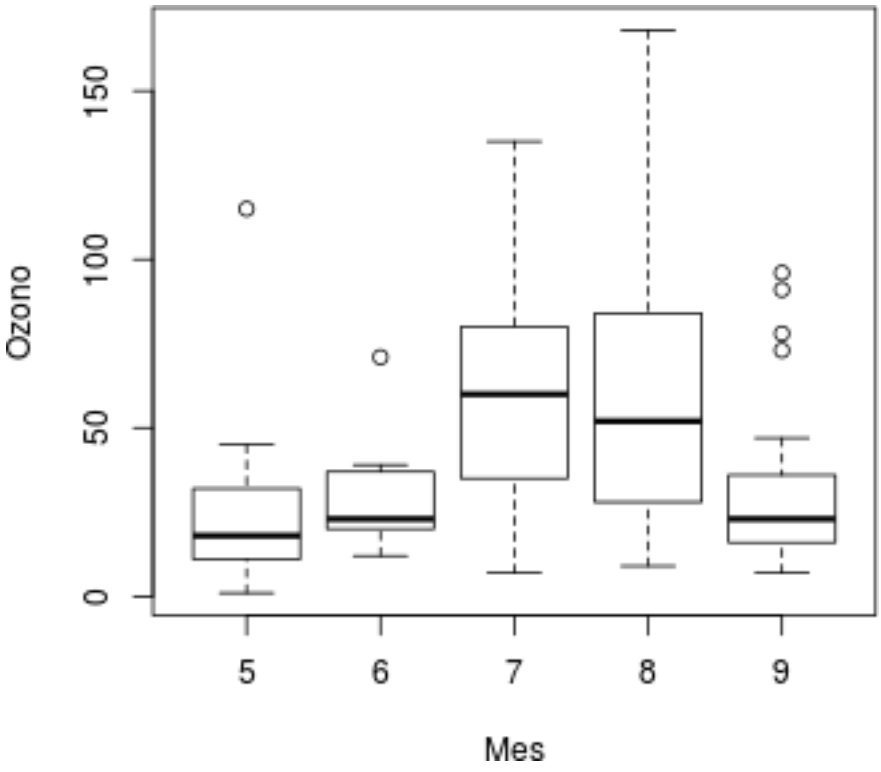
- **Boxplot simple**

```
with( cars, boxplot( speed ) )
```



- **Boxplot múltiple**

```
airquality$Month <- factor( airquality$Month )
boxplot( Ozone ~ Month, data = airquality, xlab = "Mes", ylab = "Ozono" )
```

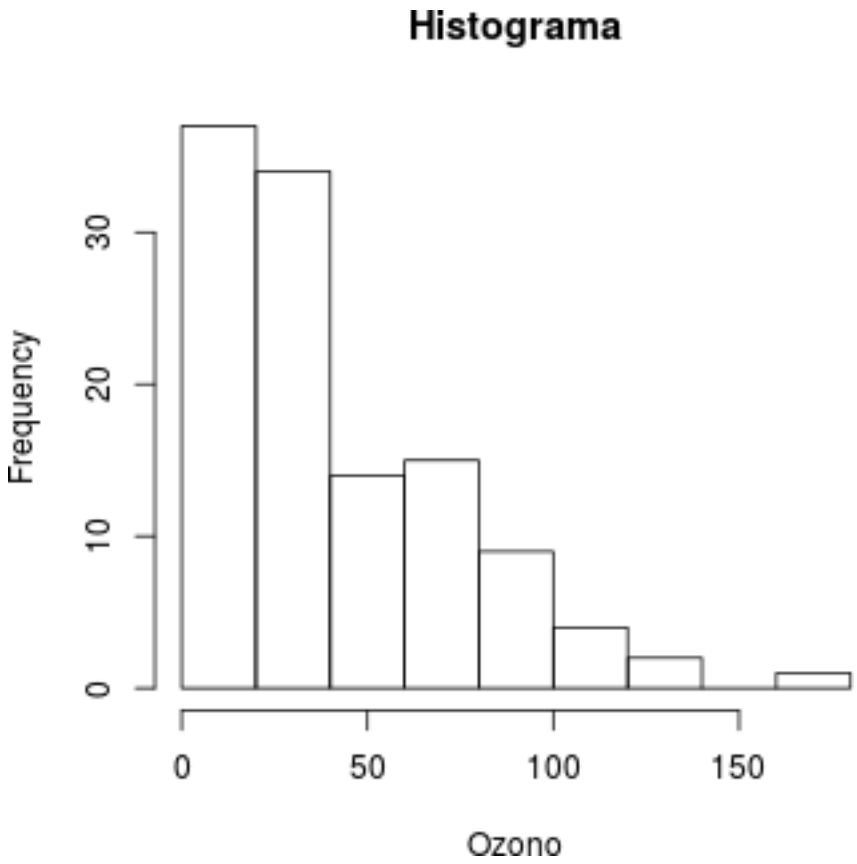


3.3.4. Histogramas

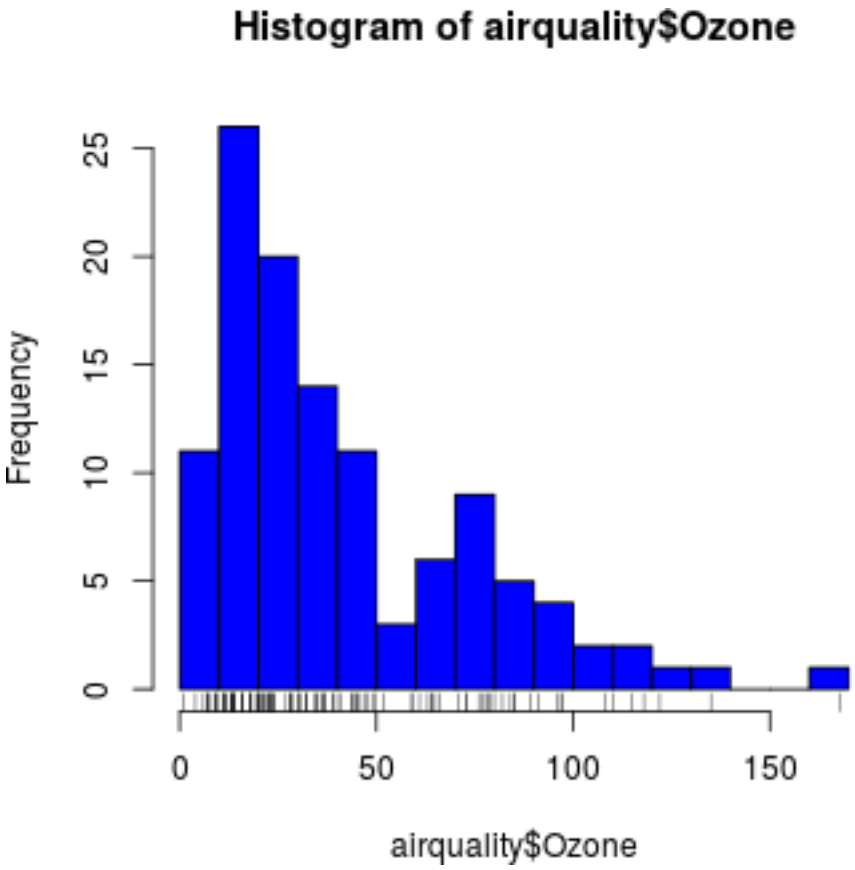
Podemos crear histogramas con la función `hist( x )`, donde `x` es un vector numérico. Con la opción `frec = FALSE` se representan las densidades en lugar de las frecuencias. La opción `breaks = n` controla el número de

barras.

```
hist( airquality$Ozone, main = "Histograma", xlab = "Ozono" )
```



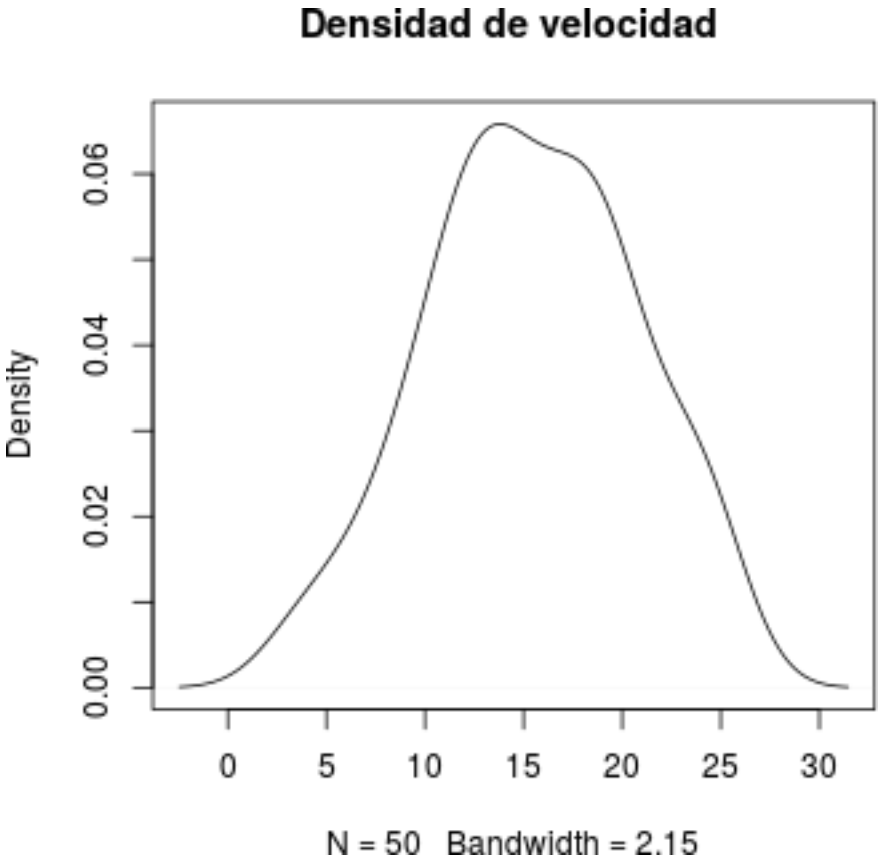

```
hist( airquality$Ozone, breaks = 13, col= "blue" )
rug( airquality$Ozone )
```



3.3.5. Densidades

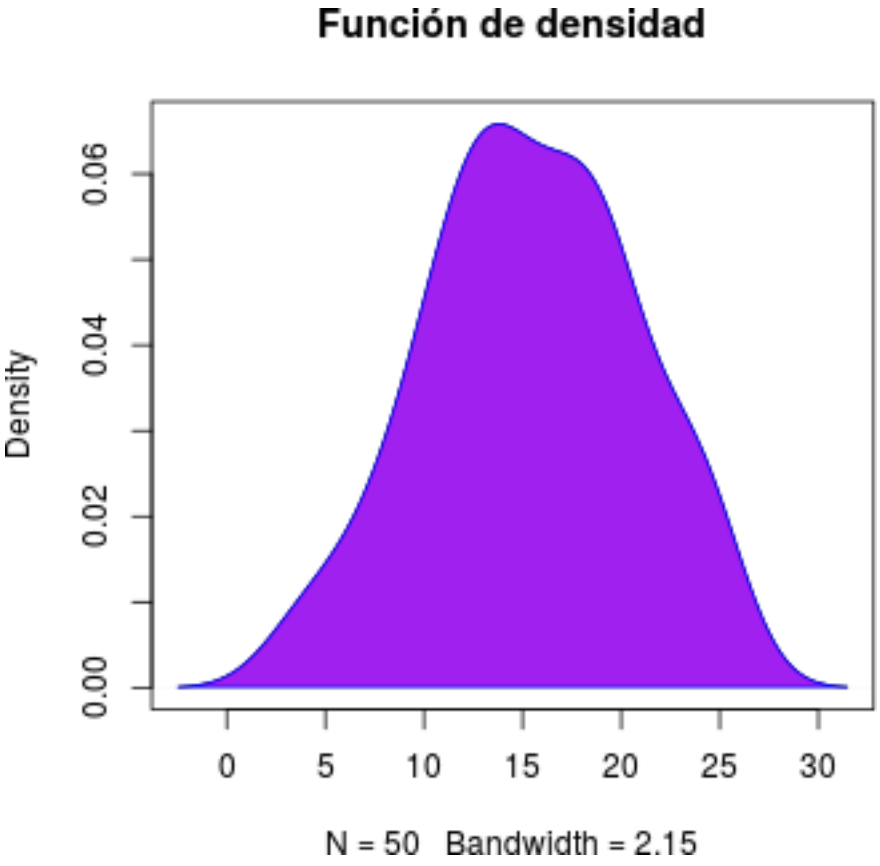
Tenemos que calcular la función de densidad mediante `density( x )` y después representarla con `plot( )`.

```
dens <- density( cars$speed )
plot( dens, main = "Densidad de velocidad " )
```



Podemos rellenar el área que hay bajo la curva de densidad

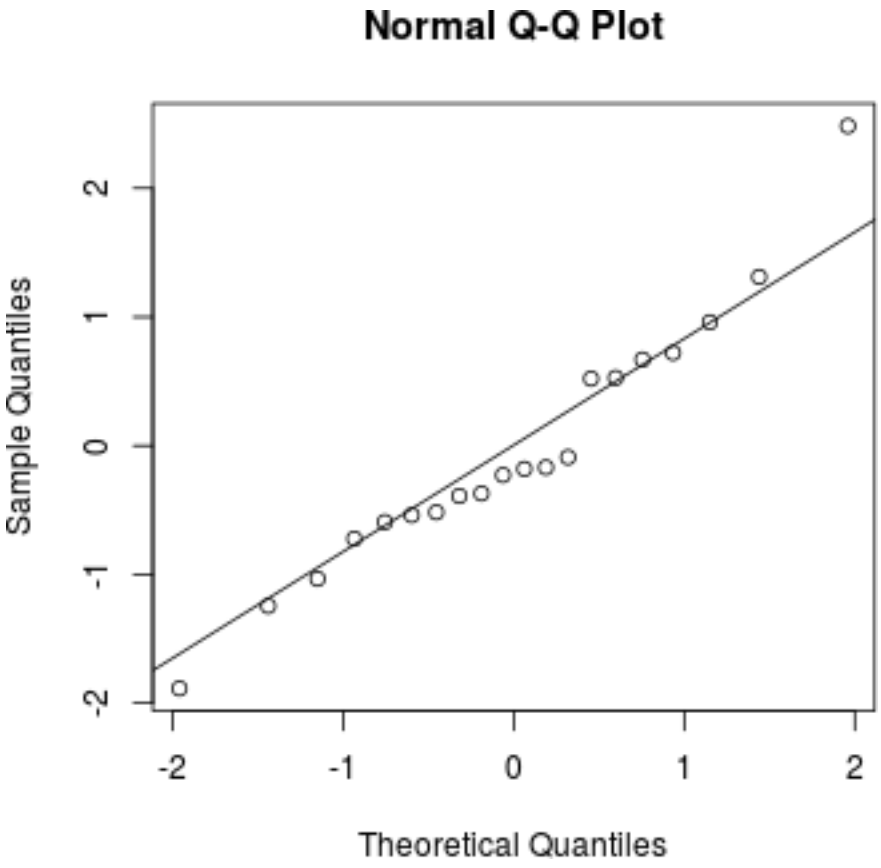
```
plot( dens, main = "Función de densidad" )
polygon( dens, col = "purple", border = "blue" )
```



3.3.6. Gráficos Q-Q plot (normalidad)

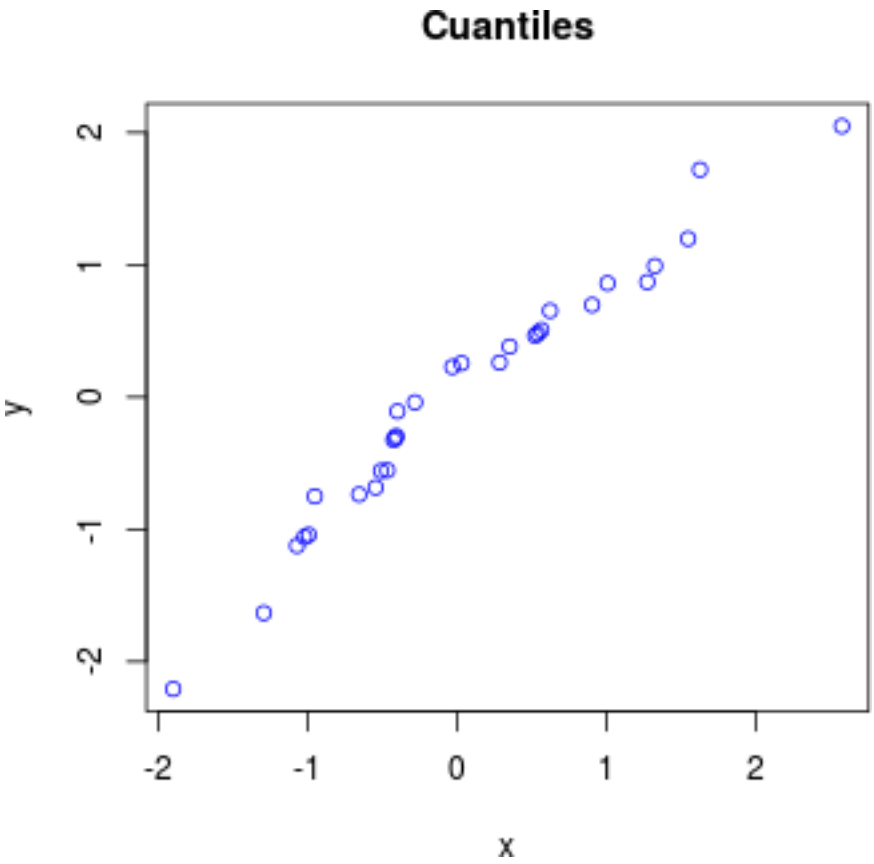
- `qqnorm(x)`: cuantiles de x con respecto a lo esperado bajo una distribución normal.

```
x <- rnorm( 20 )
qqnorm( x )
qqline( x )
```



- `qqplot(x, y)`: cuantiles de y con respecto a los de x

```
x <- rnorm( 30 )
y <- rnorm( 30 )
qqplot( x, y , col = "blue", main = "Cuantiles" )
```



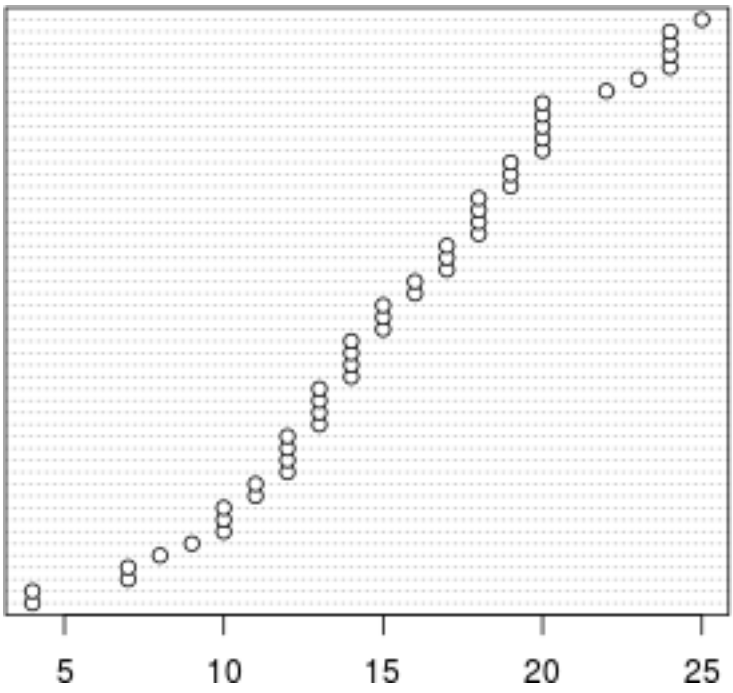
3.3.7. Otras funciones

- `pie()`: gráfico de sectores.

```
precios <- c( 0.23, 0.35, 0.14, 0.2, 0.23 )
names( precios ) <- c( "manzana", "uva", "pera", "naranja", "cereza" )
pie( precios, col = rainbow( length( names( precios) ) ),
     main = "Precios frutas" )
```

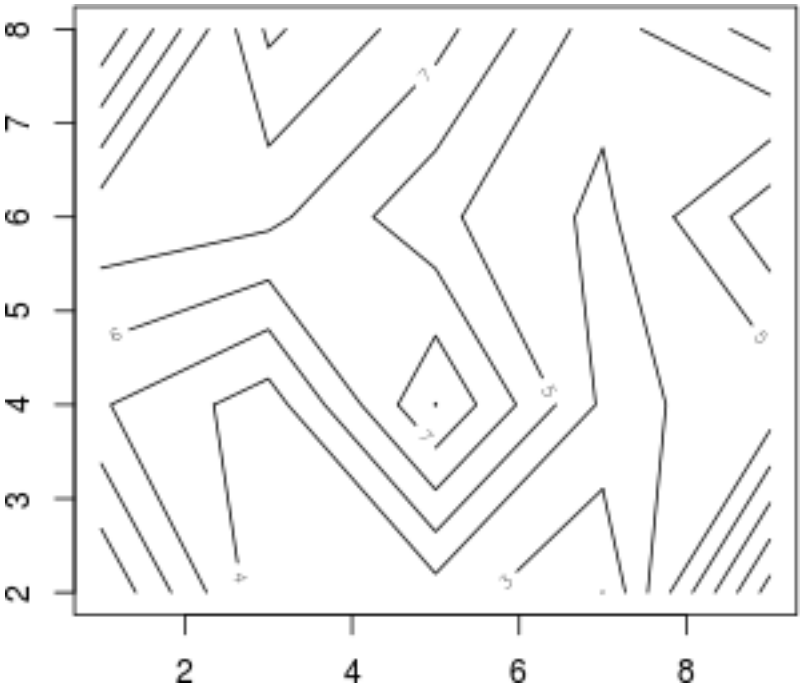
- `dotchart(x)`: si x es un *data.frame*, realiza gráficos apilados fila por fila y columna por columna.

```
data( cars )
with( cars, dotchart( speed ) )
```



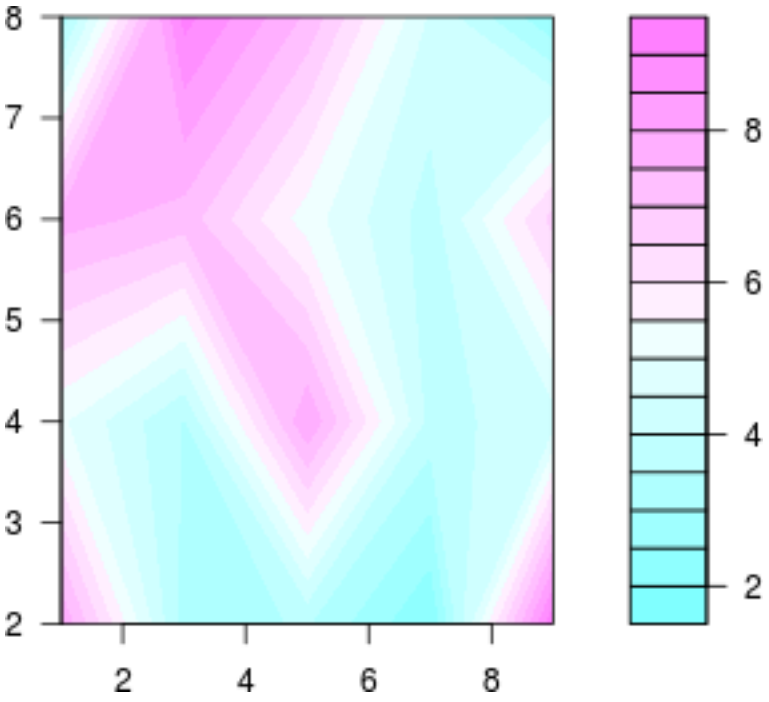
- `contour(x, y, z)`: gráfico de contornos (los datos son interpolados para dibujar las curvas), `x` e `y` deben ser vectores, `z` debe ser una matriz tal que `dim(z)=c(length(x), length(y))`.

```
x <- c( 1, 3, 5, 7, 9 )
y <- c( 2, 4, 6, 8 )
z <- matrix( runif( 20, 1, 10 ), 5, 4 )
contour( x, y, z )
```



- `filled.contour( x, y, z )`: igual que el anterior, pero las áreas entre contornos están coloreadas, y se dibuja una leyenda de colores.

```
filled.contour( x, y, z )
```

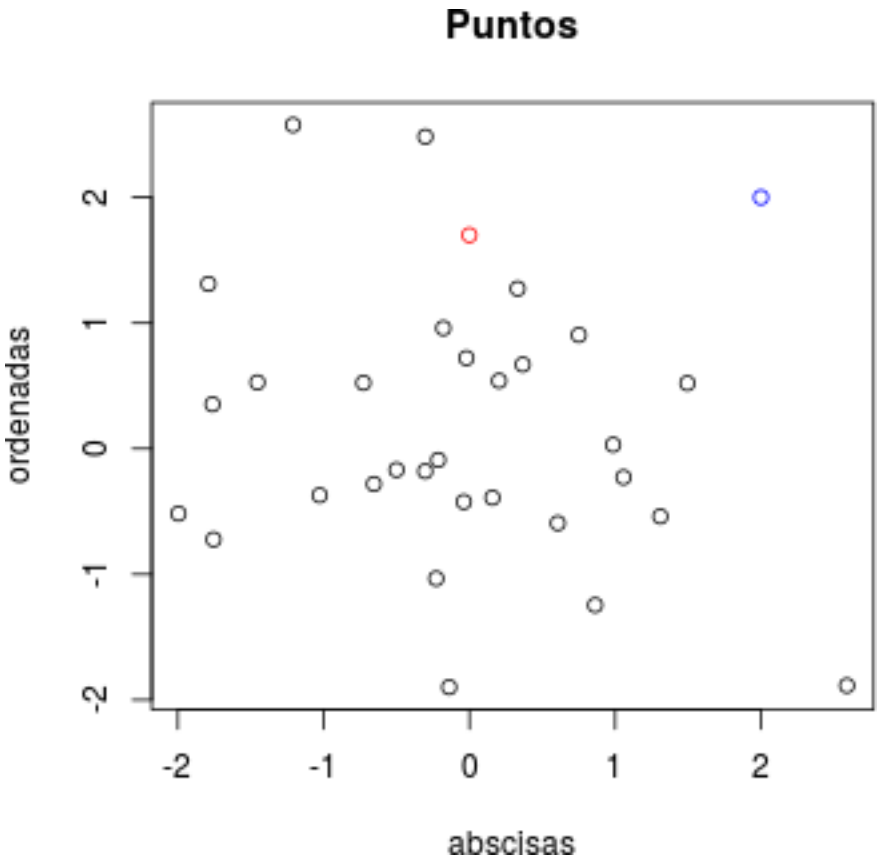


3.4. Funciones de bajo nivel

Cuando tenemos ya lanzado un *device*, podemos añadirle una serie de "objetos" utilizando un conjunto de expresiones. Veamos los más importantes:

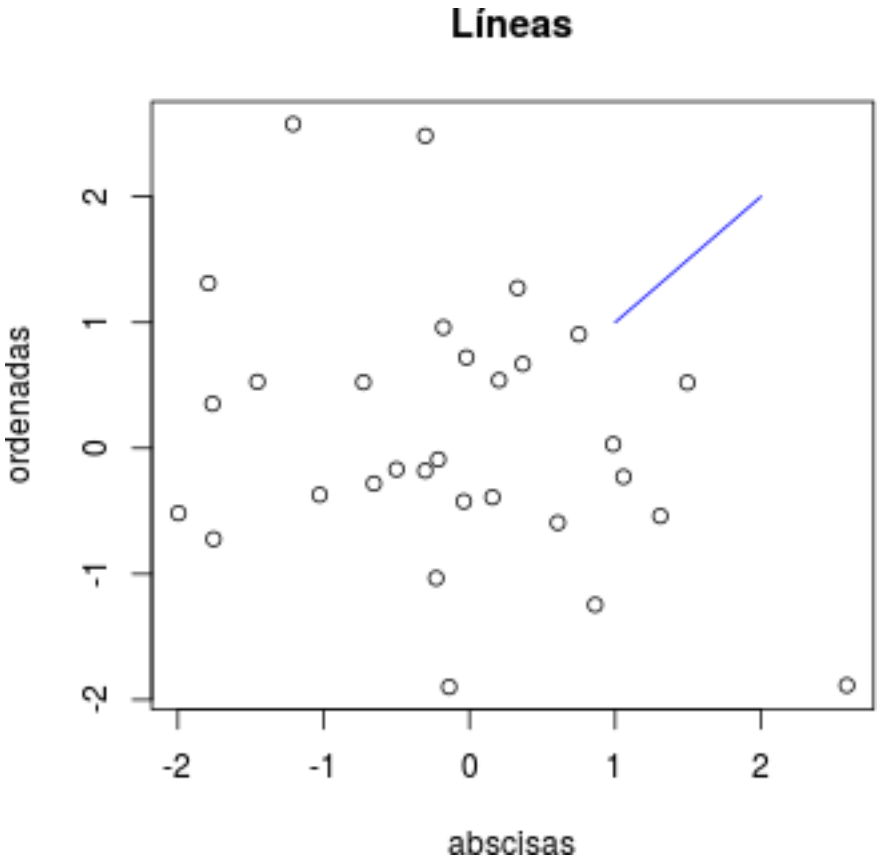
- `points( x, y )`: agrega puntos con coordenadas (x,y)

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
      main = "Puntos" )
points( 0, 1.7 , col = "red" )
points( 2, 2 , col = "blue" )
```



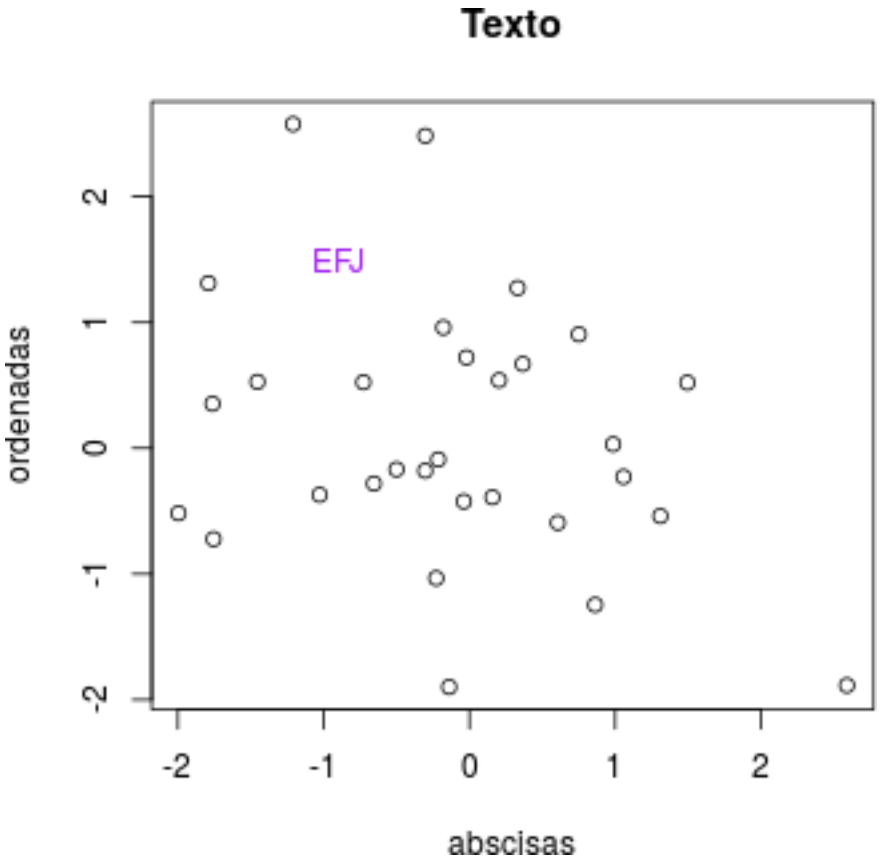
- `lines( x, y )`: añade lineas en el vector de coordenadas `( x, y )`

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Líneas" )
lines( c( 1, 2 ) , col = "blue" )
```



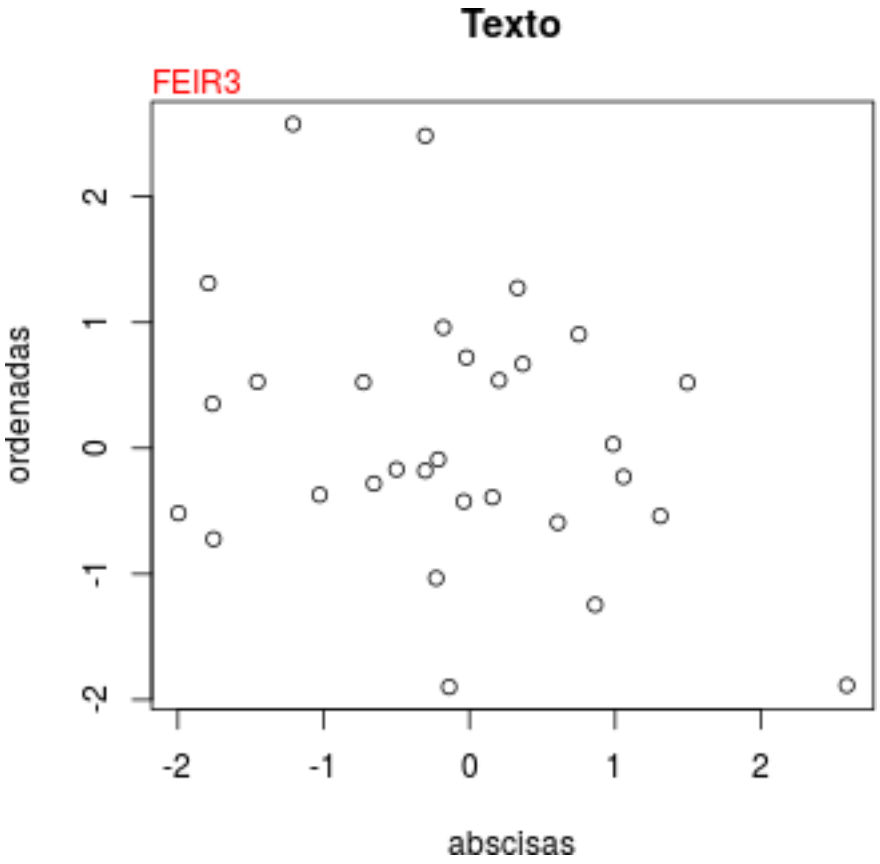
- `text( x, y, labels, ...)`: agrega texto dado en labels (etiquetas) en las coordenadas `( x, y )`

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Texto" )
text( -0.9, 1.5, "EFJ", col = "purple" )
```



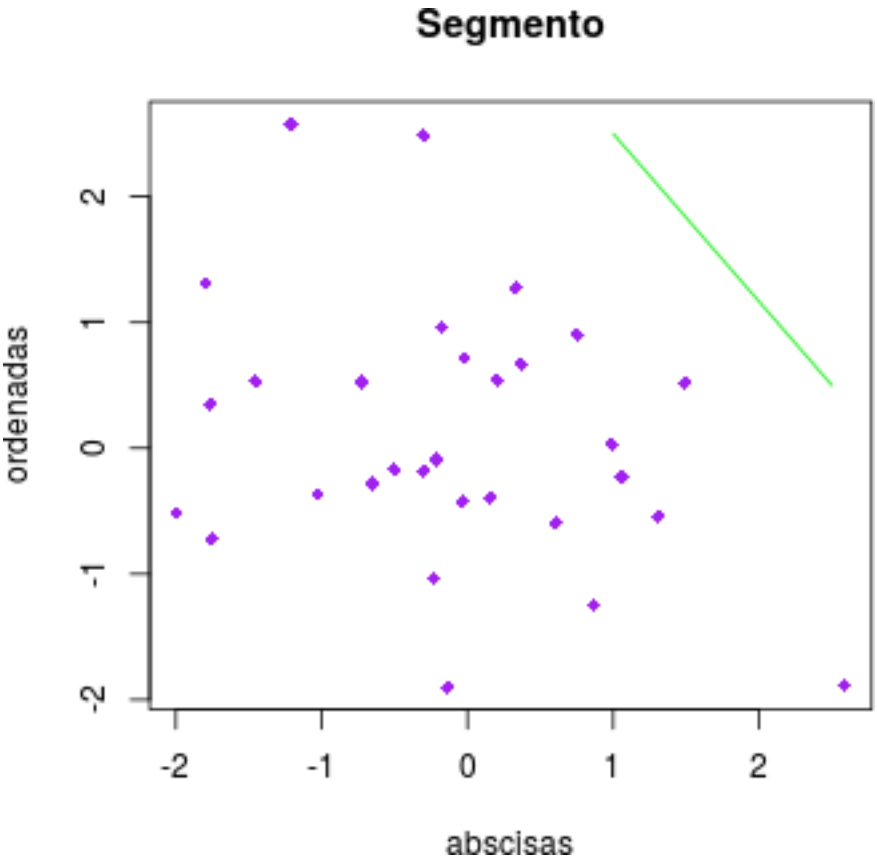
- `mtext( text, side, line, col, adj, ... )`
  - `side`: lado de la gráfica (1 = abajo, 2 = izquierda, 3 = arriba, 4 = derecha)
  - `line`: línea de margen, empezando a contar de 0 hacia fuera
  - `adj` : alinear el texto (0 = izquierda/abajo , 1 = arriba/derecha)

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Texto" )
mtext( "FEIR3", side = 3, col = "red", adj = 0 )
```



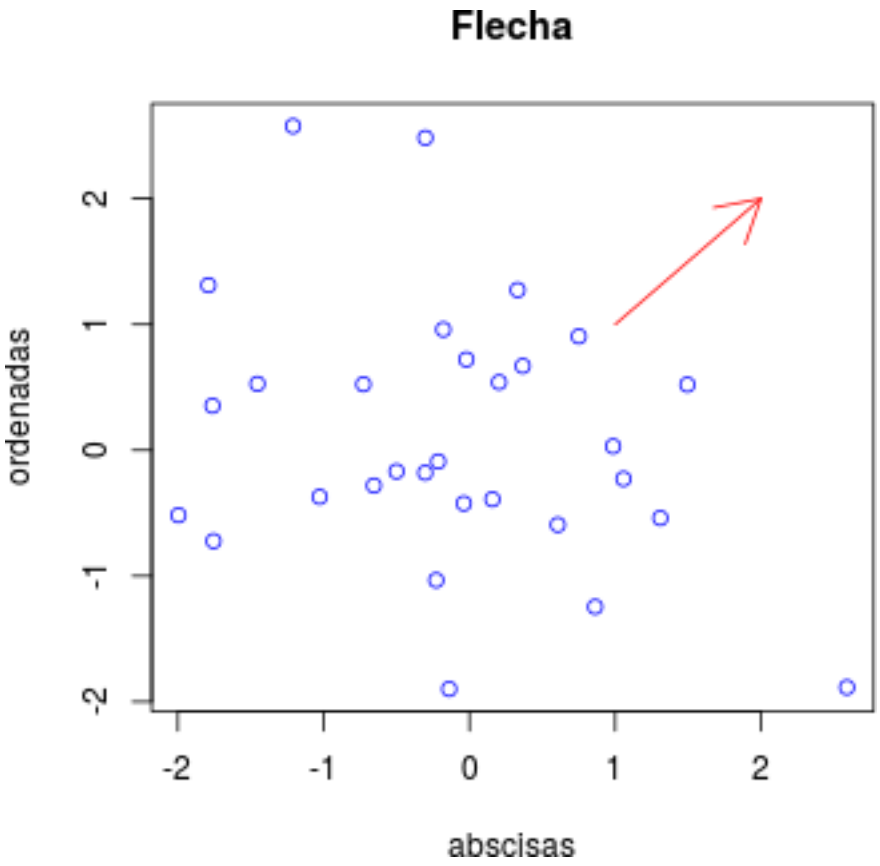
- `segments( x0, y0, x1, y1 )`: dibuja una linea desde el punto `( x0, y0 )` a `( x1, y1 )`

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Segmento", col = "purple", pch = 18 )
segments( 1, 2.5, 2.5, 0.5, col = "green2" )
```



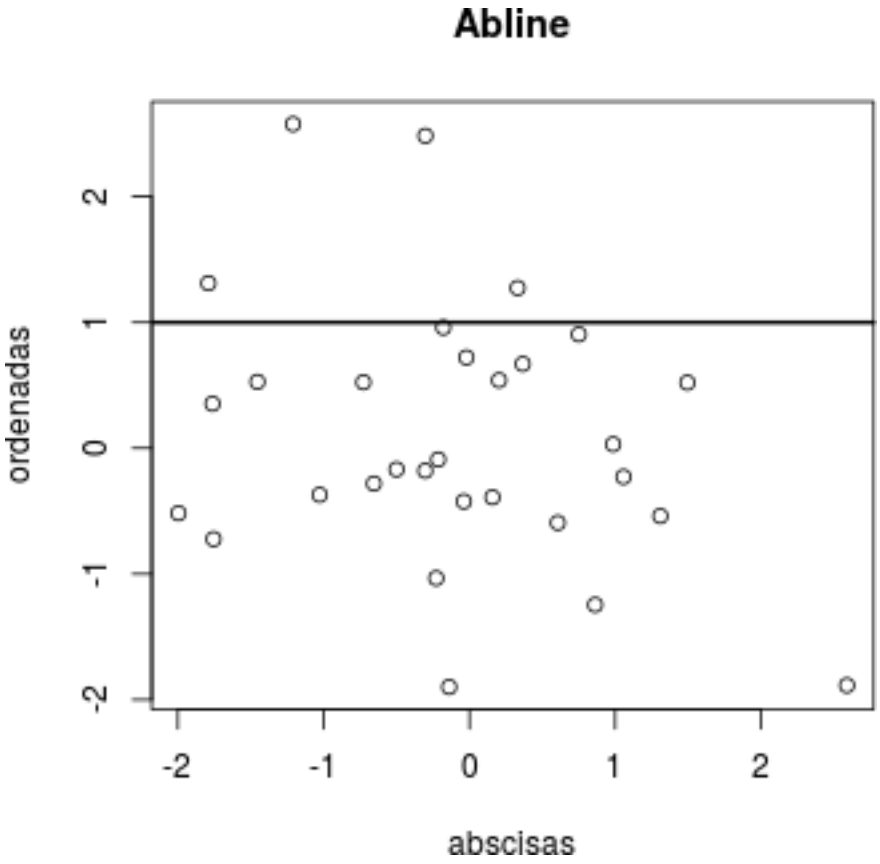
- `arrows( x0, y0, x1, y1 )`: igual que el anterior pero con flechas.

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Flecha", col = "blue" )
arrows( 1, 1, 2, 2, col = "red" )
```



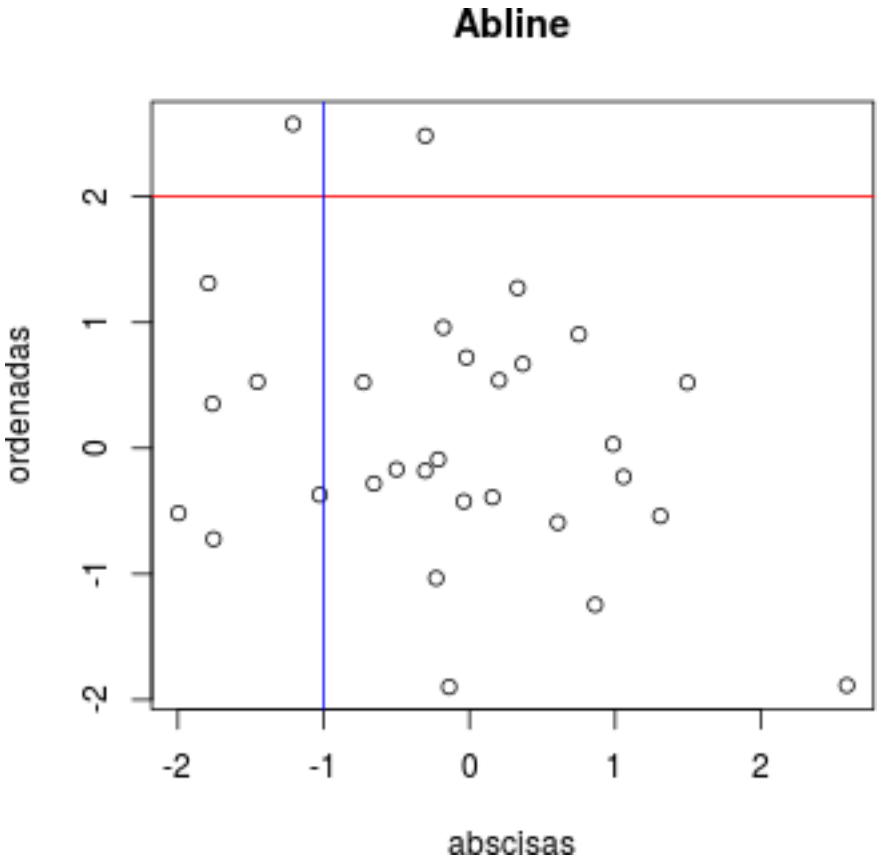
- `abline( a, b )`: dibuja una linea con pendiente b e intercepto a

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Abline" )
abline( 1, 0, lwd = 2 )
```



- `abline( h = y )`: dibuja una linea horizontal en la ordenada y
- `abline( v = x )`: dibuja una linea vertical en la abscisa x

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Abline" )
abline( h = 2, col = "red" )
abline( v = -1, col = "blue" )
```

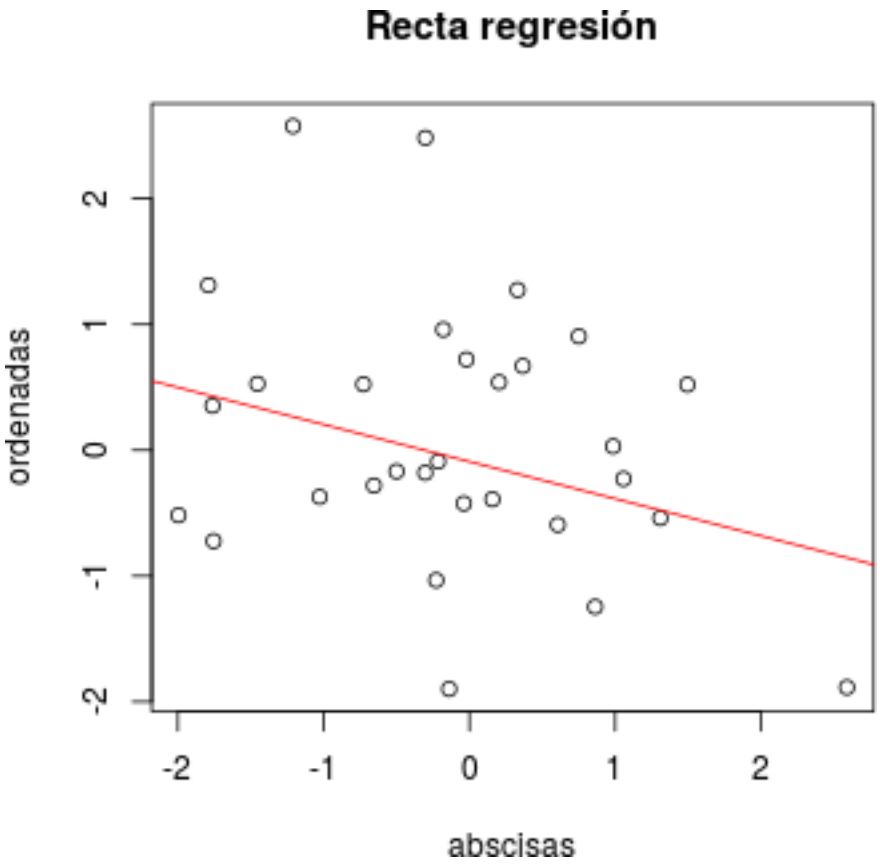


- `abline( modelo )`: dibuja la linea de regresión dada por `modelo`

```
modelo <- lm( x ~ y )
modelo
```

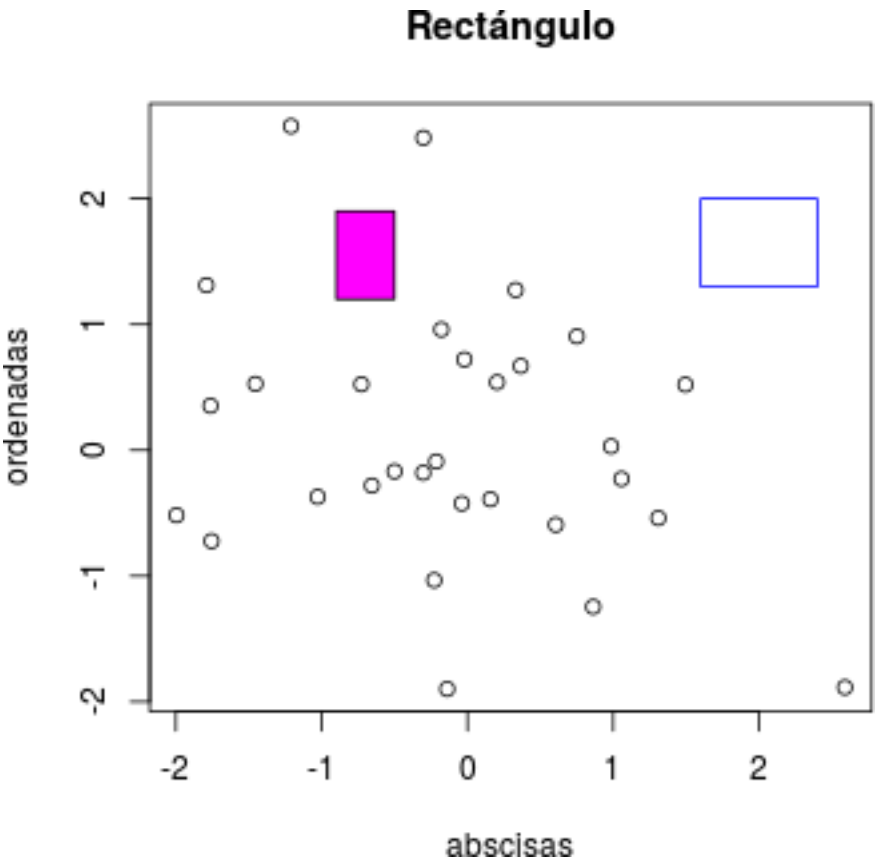
```
##
##  Call:
##  lm(formula = x ~ y)
##
##  Coefficients:
##  (Intercept)            y
##     -0.09172     -0.29505
```

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
      main = "Recta regresión" )
abline( modelo, col = "red" )
```



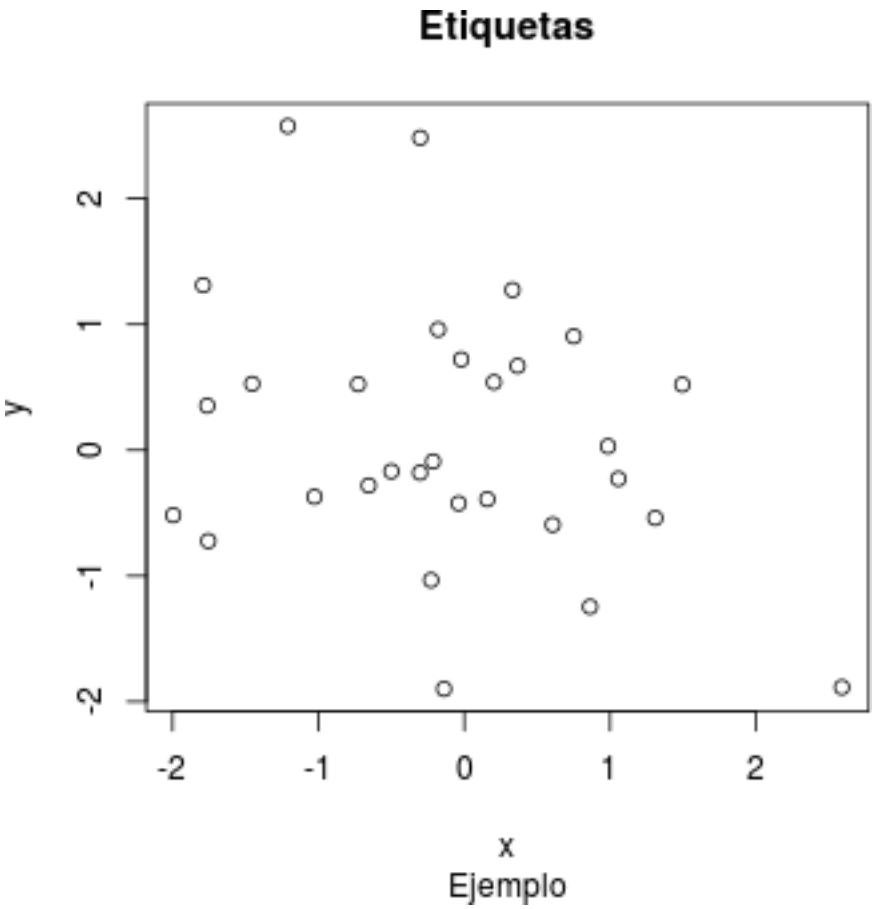
- `rect( x1, y1, x2, y2 )`: dibuja un rectángulo donde las esquinas son x1 = izquierda, y1 = inferior, x2 = derecha, y2 = superior.

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas",
     main = "Rectángulo" )
rect( 1.6, 2, 2.4 ,1.3, border = "blue" )
rect( -0.9, 1.9, -0.5 ,1.2, col = "magenta" )
```



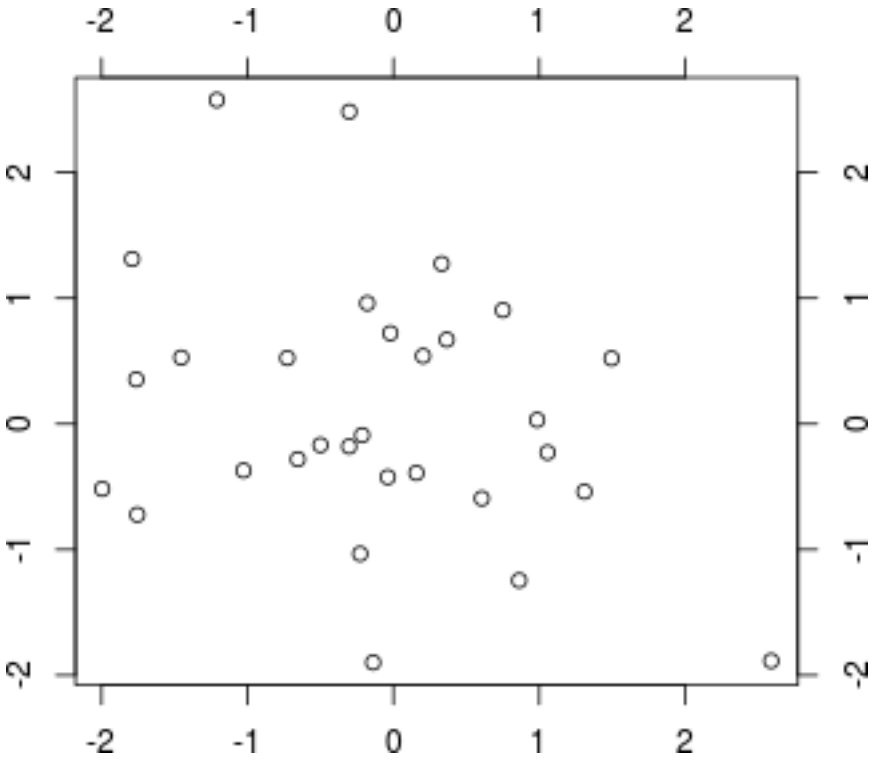
- `title( )`: agrega todas las etiquetas al gráfico
  - main = "título principal"
  - sub = "sub-título"
  - xlab = "etiqueta eje x"
  - ylab = "etiqueta eje y"

```
plot( x, y )
title( main = "Etiquetas", sub =  "Ejemplo" )
```



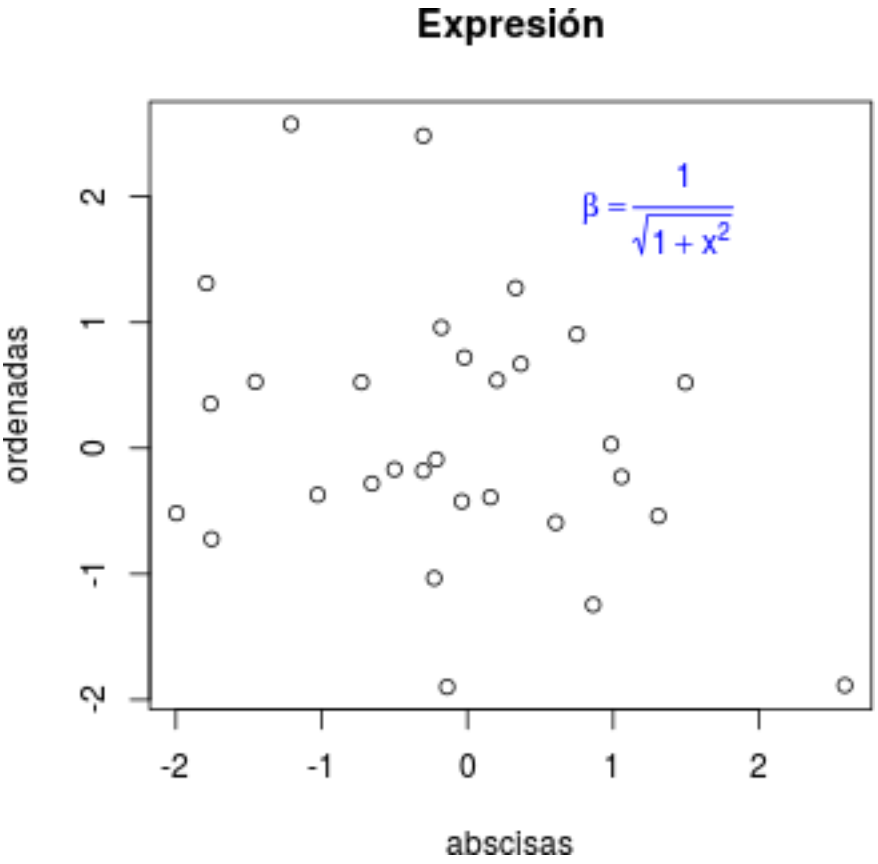
- `axis( side, at=, labels=, pos=, lty=, col=, las=, tck=, ...)`
  - `side`: lugar del gráfico donde dibujar el eje (1=abajo, 2=izquierda, 3=arriba, 4=derecha)
  - `at`: vector numérico indicando dónde se deben dibujar las marcas
  - `labels` vector de caracteres con las etiquetas
  - `pos` las coordenadas donde se dibujará la linea del eje

```
plot( x, y, xlab = "", ylab = "", main = "" )
axis( side = 3 )
axis( side = 4 )
```



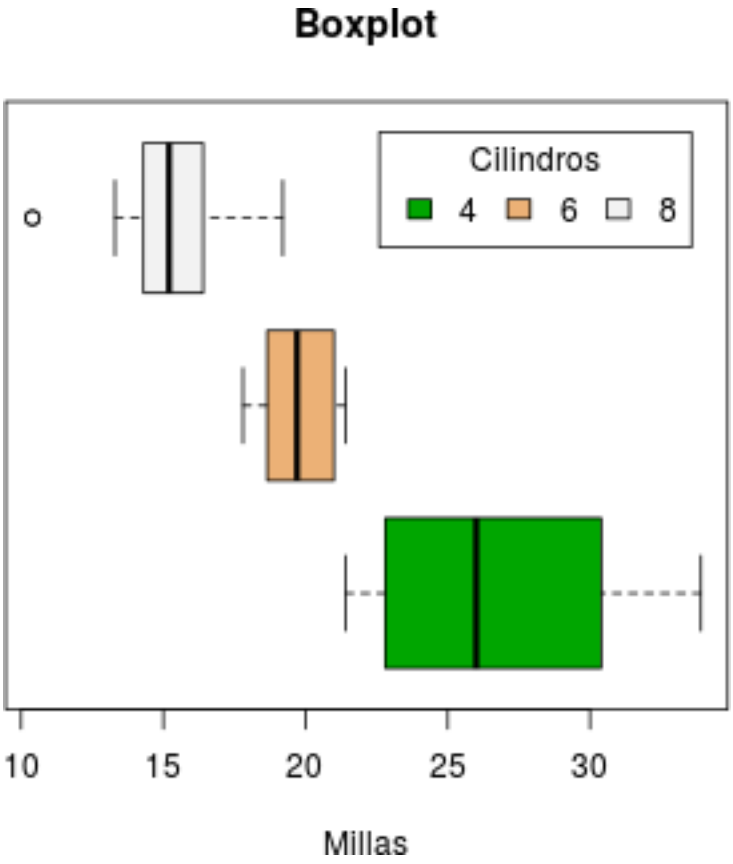
También podemos añadir expresiones matemáticas a una gráfica con el comando `text( x, y, expression( ) )`, donde la función expresión transforma su argumento en una ecuación matemática.

```
plot( x, y, xlab = "abscisas", ylab = "ordenadas", main = "Expresión" )
eq <- expression( beta== over( 1, sqrt( 1 + x ^ 2 ) ) )
text( 1.3, 1.9, labels = eq, col = "blue" )
```



- `legend( posicion, titulo, legenda, ... )` Otras opciones incluyen la leyenda son `bty` para el tipo caja, `bg` para color de fondo, `cex` para el tamaño, y `col` para el color del texto. Con la opción `horiz = TRUE` la leyenda horizontalmente en lugar de verticalmente.

```
data( mtcars )
df<- mtcars
boxplot( df$mpg ~ df$cyl, main = "Boxplot", yaxt = "n",
         xlab = "Millas", horizontal = TRUE, col = terrain.colors( 3 ) )
legend("topright", inset=.05, title="Cilindros",
   c("4","6","8"), fill=terrain.colors(3), horiz=TRUE)
```



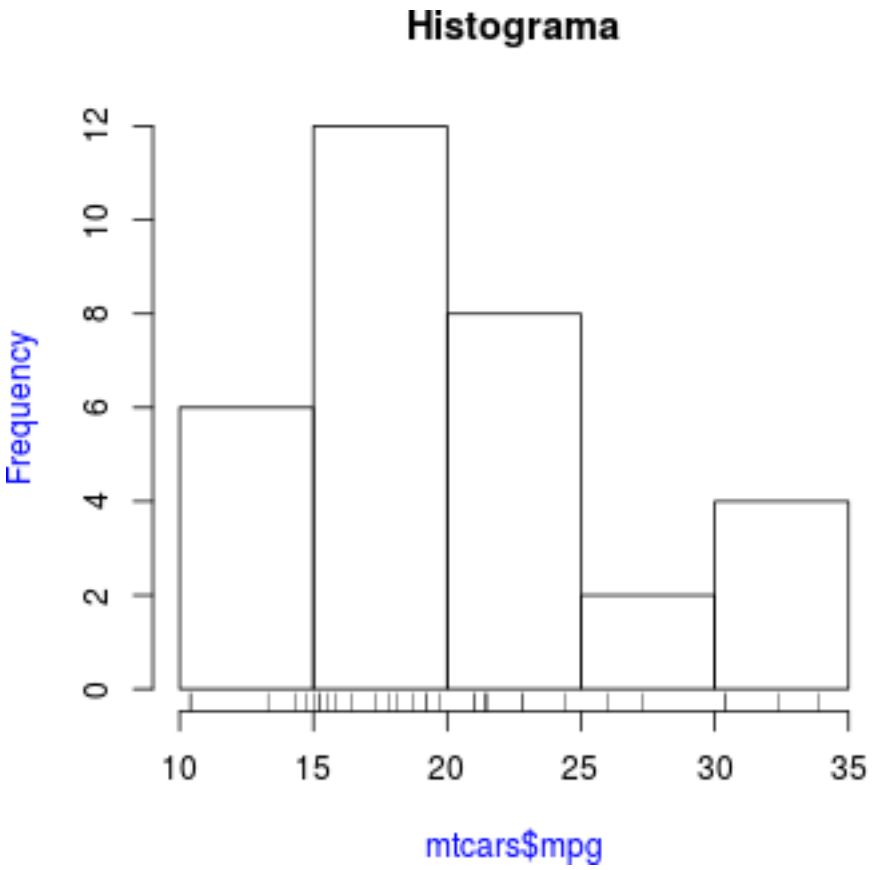
3.5. Parámetros gráficos

Podemos personalizar muchas características de nuestros gráficos (colores, ejes, títulos, fuentes de letra,...) a través de parámetros gráficos adicionales.

Una forma de especificar estas opciones es a través de la función `par()`. Si establecemos valores para los parámetros aquí, los cambios estarán en vigor durante el resto de la sesión o hasta que vuelvan a cambiar.

La función toma la forma `par(nombreOpcion = valor, nombreOpcion = valor, ...)`.

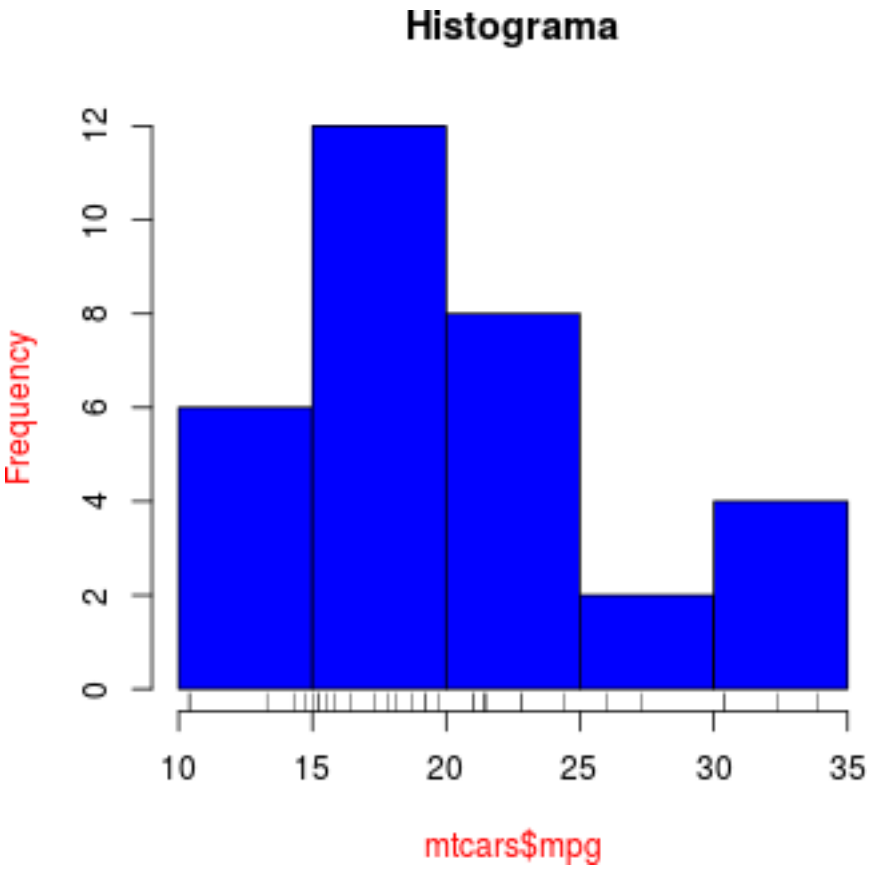
```
par( ) # vemos la configuración actual

opar <- par( ) # hacemos una copia de la configuración actual
par( col.lab = "blue" ) # etiquetas de x e y rojas
hist( mtcars$mpg, main = "Histograma") # crear un histograma con las nuevas opciones
rug( mtcars$mpg )
```




```r
par( opar ) # restaura las opciones originales
```

Una segunda opción para especificar parámetros gráficos es añadiendo los parámetros `OptionName = valor` directamente a los argumentos de una función de alto nivel. En este caso, las opciones sólo afectan a ese gráfico específico.

```r
# Establecer un parámetro dentro de la función gráfica
hist( mtcars$mpg, col.lab = "red", col = "blue", main = "Histograma" )
rug ( mtcars$mpg )
```



Vemos a continuación las principales posibilidades que nos ofrece esta función.

3.5.1. Tamaño del texto y los símbolos

Podemos utilizar las siguientes opciones para controlar el tamaño del texto y de los símbolos en los gráficos.

- `cex`: valor que escala el tamaño del texto y de los símbolos con respecto al valor por defecto. El valor por defecto es 1, un 1.5 significa un 50 % más grande y 0.5 es un 50 % más pequeño.
- `cex.axis`: aumento de los ejes con respecto al `cex`
- `cex.lab`: incremento de las etiquetas de x e y en relación a `cex`.
- `cex.main`: magnificación de los títulos relativos al `cex`.
- `cex.sub`: ampliación del subtitulo con respecto al `cex`.

3.5.2. Símbolos

Podemos personalizar las opciones de los puntos.

- `pch`: controlar el tipo de símbolo para trazar los puntos con un entero entre 1 y 25.
- `col`: determinar el color del borde
- `bg`: modificar el color de relleno

```
# Establecer un parámetro dentro de la función gráfica
plot( x, y, pch = 15, bg = 7, col = "blue",
      main = "Símbolos" )
```

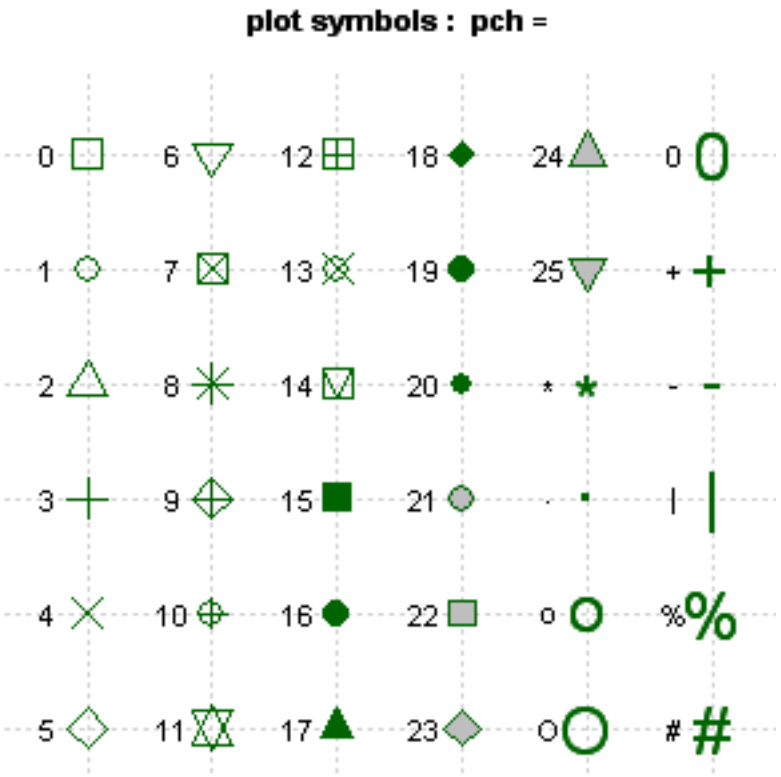
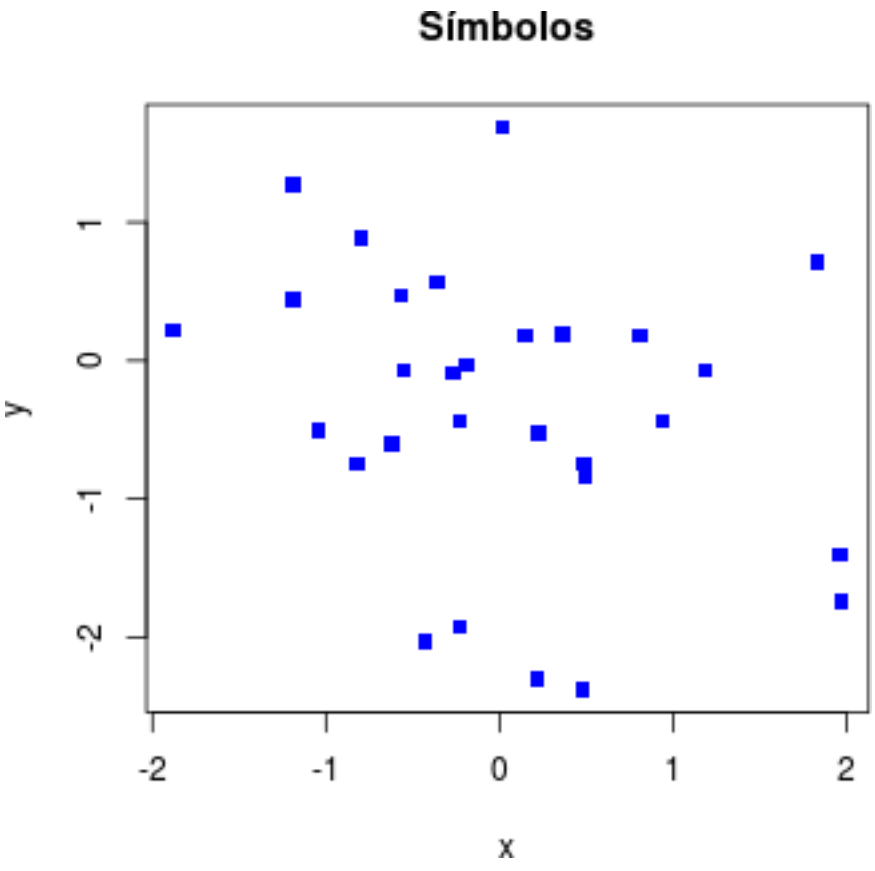


Figura 1: supuest

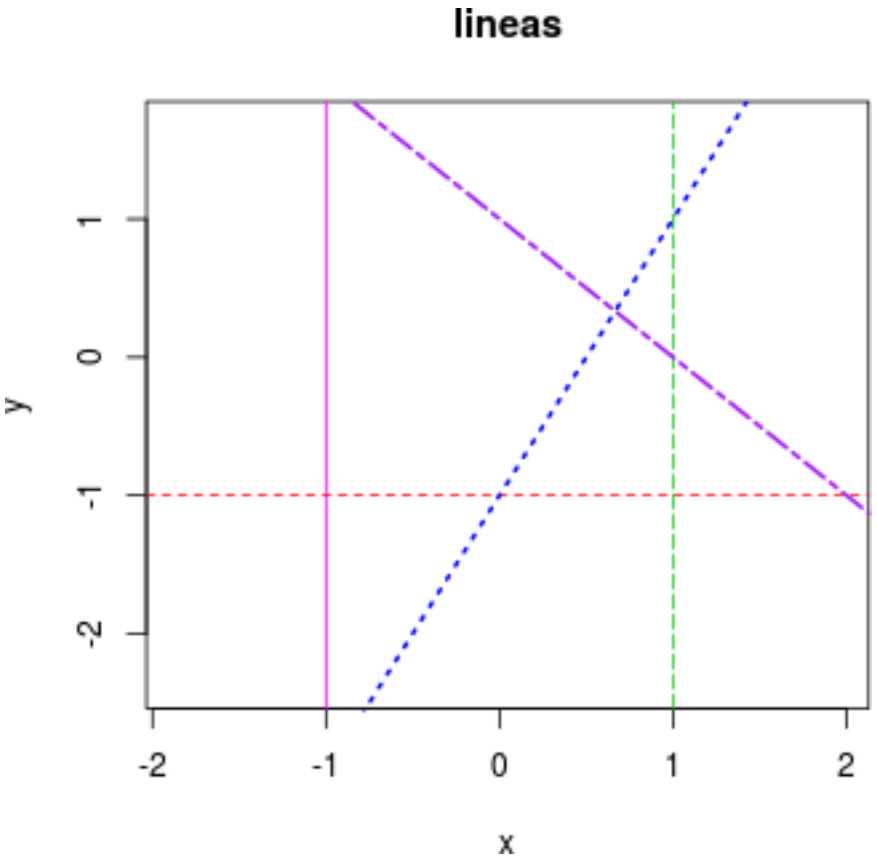


3.5.3. Líneas

Podemos cambiar el trazado de las lineas mediante las siguientes opciones.

- `lty`: controla el tipo de las lineas
  - 1: solida
  - 2: quebrada
  - 3: punteada
  - 4: punto-linea
  - 5: linea larga-corta
  - 6: dos lineas cortas
- `lwd`: determinar el ancho de linea en relación con el valor por defecto (defecto=1)

```
plot( x, y, main = "lineas", type = "n" )
abline( v = -1, lty = 1, col = "magenta" )
abline( h = -1, lty = 2, col = "red" )
abline( -1, 2, lty = 3, col = "blue", lwd = 2 )
abline( v = 1, lty = 5,  col = "green3")
abline( 1, -1, lty = 6,  col = "purple", lwd = 2 )
```



3.5.4. Colores

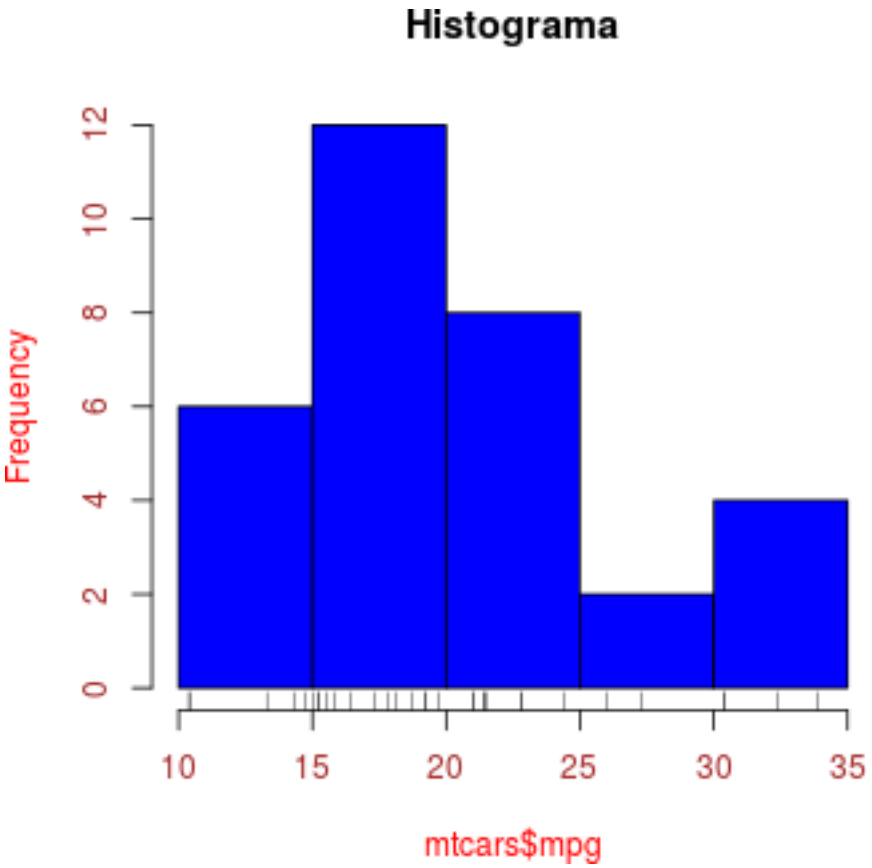
- `col`: color por defecto del trazado
- `col.axis`: color para los ejes
- `col.lab`: color de las etiqueta de los ejes
- `col.main`: color del título principal
- `col.sub`: color para los subtitulos

- `fg`: color para el primer plano del gráfico (ejes, cajas) *`bg`: color de fondo

En `R` podemos especificar los colores mediante el índice, el nombre, código hexadecimal o RGB. Por ejemplo `col = 1`, `col = "white"`, y `col = "# FFFFFF"` son equivalentes.

Podemos ver toda la carta de colores con la función `colors()`, que devuelve el nombre de todos los colores disponibles.

```
# Establecer un parámetro dentro de la función gráfica
hist( mtcars$mpg, col.lab = "red", col = "blue",
      col.axis = "brown", main = "Histograma" )
rug( mtcars$mpg )
```



También podemos crear un vector de `n` colores contiguos utilizando las funciones `rainbow( n )`, `heat.colors( n )`, `terrain.colors( n )`, `topo.colors( n )`, and `cm.colors( n )`.

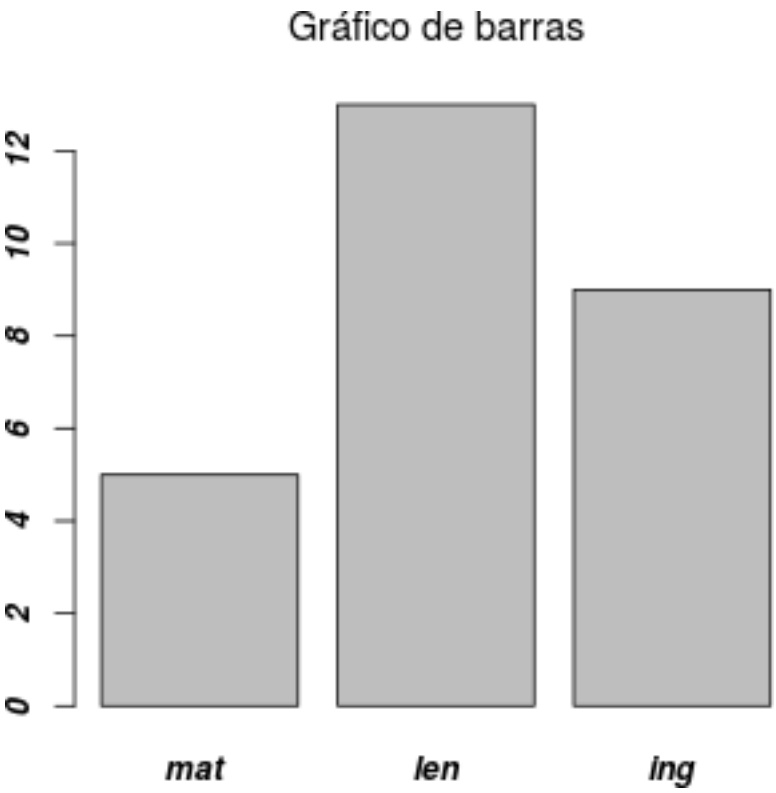
3.5.5. Fuentes

Podemos configurar fácilmente el tamaño y el estilo de la fuente de la letra

- `font`: especifica la fuente para el texto
  - 1: texto plano
  - 2: negrita
  - 3: cursiva
  - 4 cursiva-negrita
- `font.axis`: fuente para los ejes
- `font.lab`: fuente para las etiquetas
- `font.main`: fuente para los títulos

- `font.sub`: fuente para los subtitulos

```
x <- c( 5, 13, 9 )
barplot( x, font = 4, font.main = 1,  main = "Gráfico de barras",
  ,names.arg = c( "mat ", "len", "ing" ) )
```



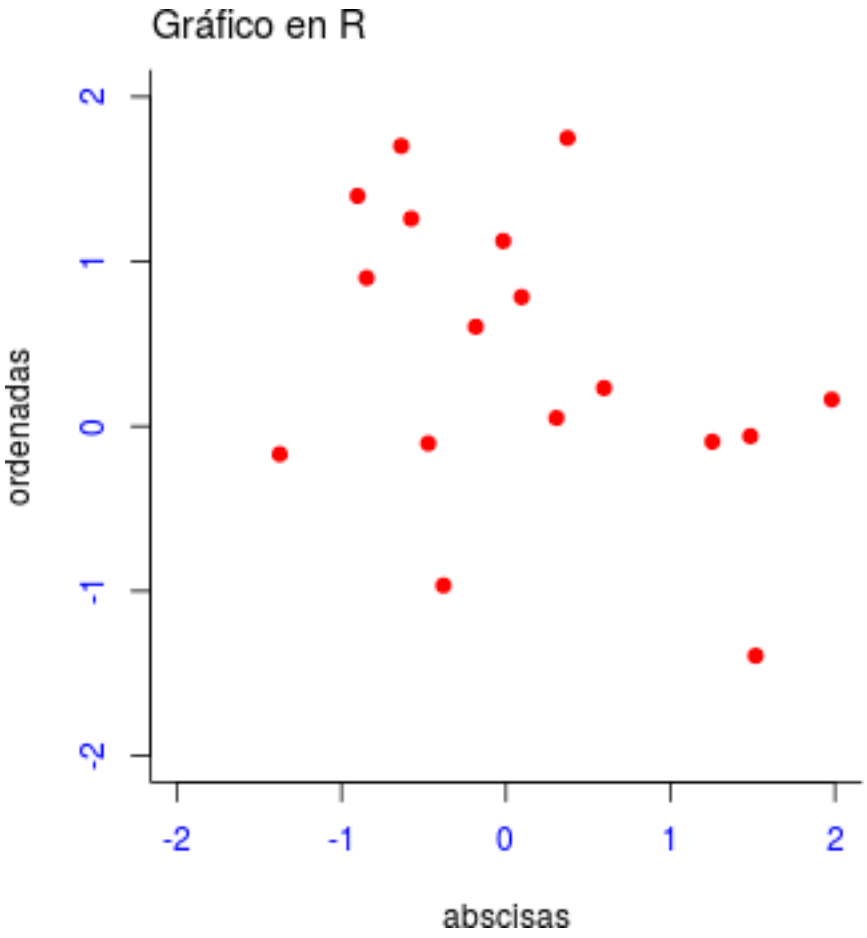
3.5.6. Otras opciones

- `adj`: posición del texto
  - `adj=0` ajustado a la izquierda
  - `adj=0.5` texto centrado
  - `adj=1` ajustado a la derecha.
- `bty`: tipo de caja que rodea los gráficos. Las opciones son *"o"*, *"l"*, *"7"*, *"c"*, *"u"*, *"]"* y *"n"* (suprime la caja).
- `las`: especifica la posición de las etiquetas de los ejes
  - 0: paralelo al eje (por defecto)
  - 1: horizontal
  - 2: perpendicular al eje
  - 3: vertical.
- `mar`: vector numérico que controla el espacio entre los ejes y el borde de la gráfica. Tiene la forma `c( inferior, izquierda, superior, derecha )` y los valores por defecto son `c( 5, 4, 4, 2 ) + 0.1`.
- `mfcol`: vector del tipo `c( nf, nc )` que divide la ventana gráfica en una matriz cn nf filas y nc columnas. Las gráficas se van dibujando sucesivamente por columnas.
- `mfrow`: igual que el anterior pero las gráficas se dibujan por filas.

- `xaxt`: si `xaxt = "n"` el eje x se coloca pero no se muestra.
- `yaxt`: si `yaxt= "n"` el eje y se coloca pero no se muestra.

Veamos un ejemplo con la función `par( )`

```
x <- rnorm( 20 )
y <- rnorm( 20 )
par( col.axis = "blue", mar = c( 4, 4, 2.5, 2.5 ), font = 2 )
plot( x, y, xlab = "abscisas", ylab = "ordenadas", xlim = c( -2, 2 ),
     ylim = c(-2, 2 ), pch = 20, col = "red", bty = "l", cex = 1.5 )
title( "Gráfico en R", font.main = 1, adj = 0 )
```



3.6. Combinación de gráficos

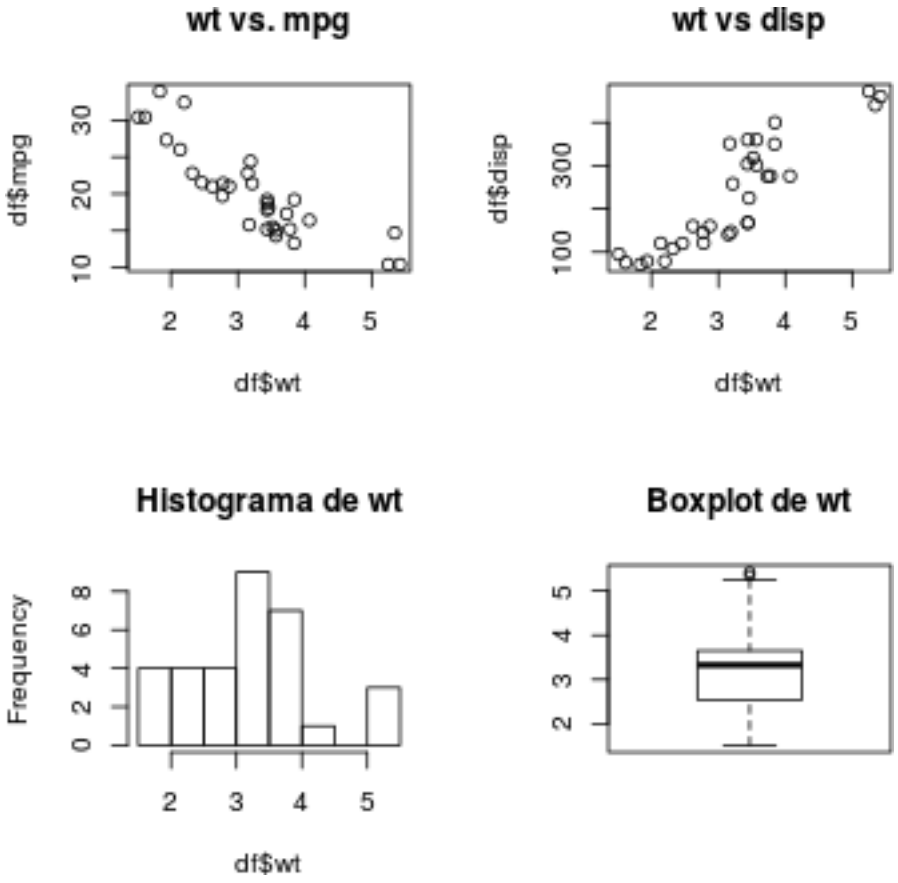
Podemos combinar varias parcelas en una gráfica global, utilizando las funciones `par( )` o `layout( )`.

Con la función `par( )` podemos incluir la opción `mfrow = c( nfilas, ncolumnas )` para crear una matriz de nfilas x ncolumas gráficos que se introducen por filas. `mfcol = c( nfilas, ncolumas )` rellena la matriz por columnas.

Vemos primero un ejemplo con `mfrow = c( nfilas, ncolumnas )`

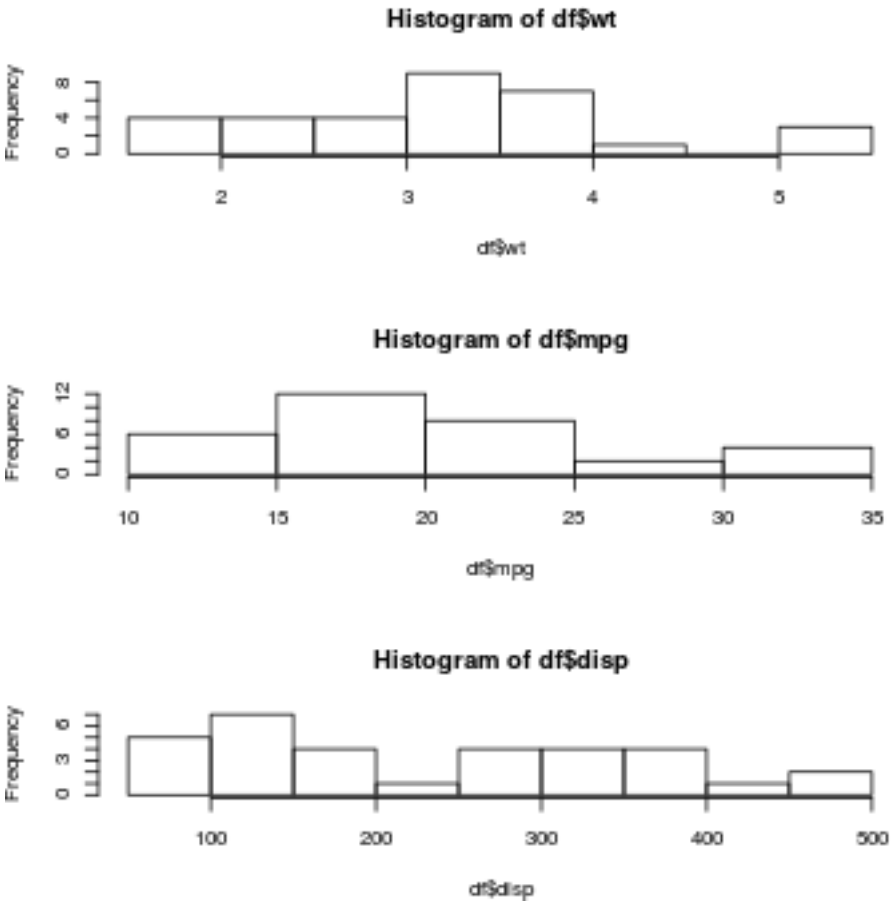
```
# 4 imagens ordenadas en 2 filas y 2 columnas
df <- mtcars
par( mfrow = c( 2, 2 ) )
```

```
plot( df$wt, df$mpg, main = "wt vs. mpg" )
plot( df$wt, df$disp, main = "wt vs disp" )
hist( df$wt, main = "Histograma de wt" )
boxplot( df$wt, main = "Boxplot de wt" )
```



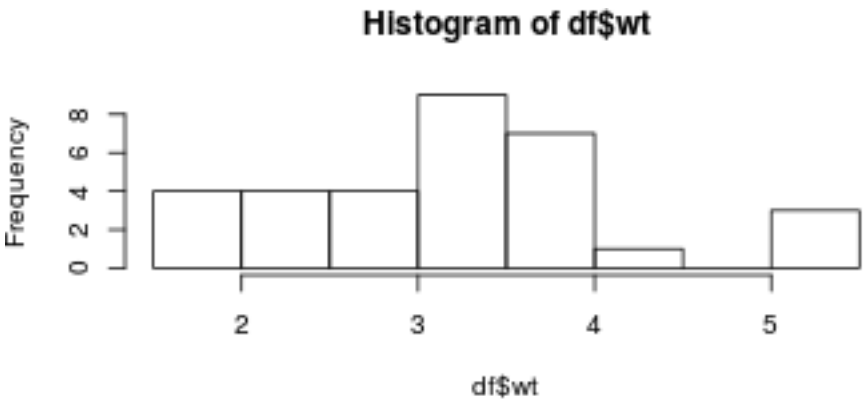
y ahora otro con `mfcol = c(nfilas, ncolumas)`

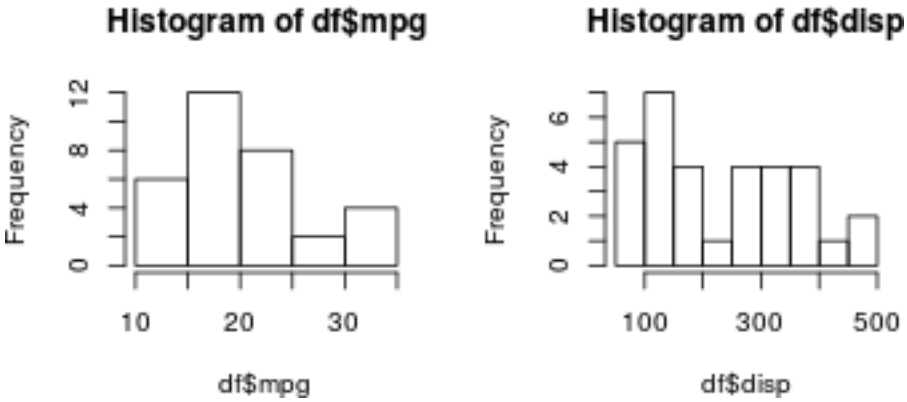
```
# 3 figures arranged in 3 rows and 1 column
par( mfrow = c( 3, 1 ) )
hist( df$wt )
hist( df$mpg )
hist( df$disp )
```



La función `layout()` divide el dispositivo activo en varias partes donde se colocarán las gráficas de manera sucesiva. Esta función tiene como argumento principal una matriz de enteros indicando el número de figuras del gráfico.

```
# Una figura en la fila 1 y dos figuras en la fila 2
layout( matrix( c( 1, 1, 2, 3 ), 2, 2, byrow = TRUE ) )
hist( df$wt )
hist( df$mpg )
hist( df$disp )
```

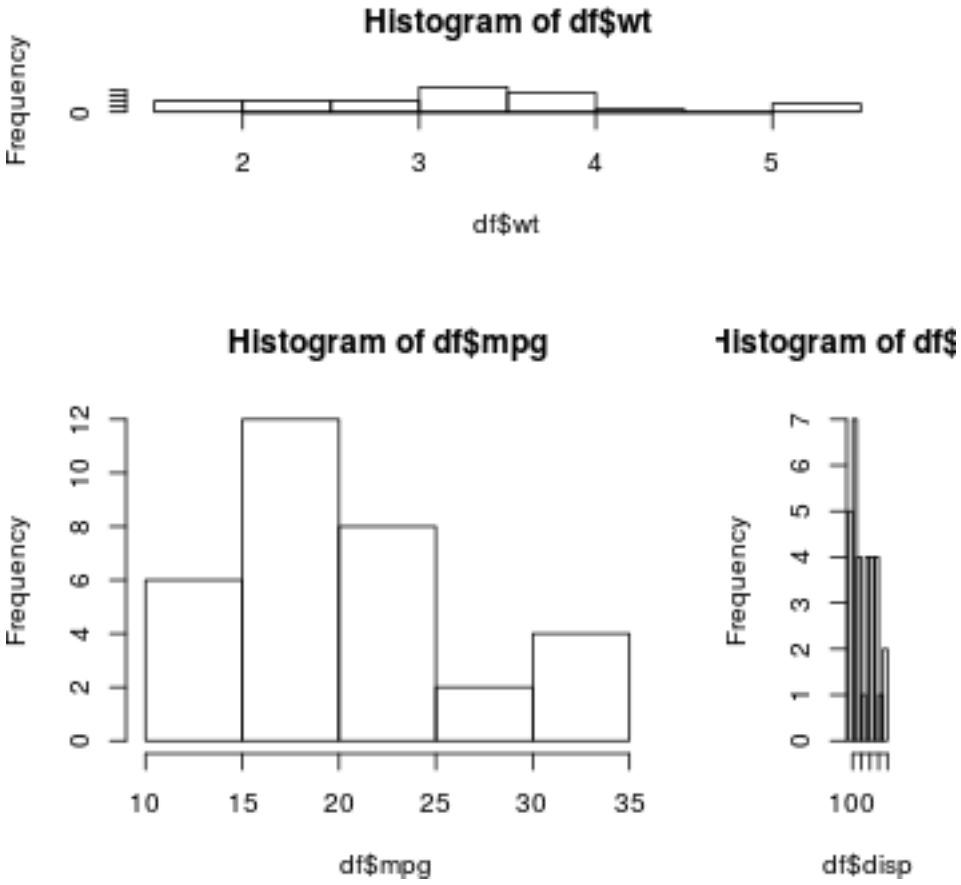




Por defecto `layout()` divide el dispositivo en dimensiones regulares, pero podemos incluir las opciones anchura y altura en la función de `layout()` para controlar el tamaño de cada figura con mayor precisión. Estas opciones tienen la forma

- `widths` = un vector de valores de las anchuras de las columnas
- `heights` = un vector de valores de las alturas de las filas.

```
# Una figura en la fila 1 y dos figuras en la fila 2
# La fila 1 tiene 1/3 de la altura de 2
# La columna 2 tiene 1/4 de la anchura de la 1
layout( matrix( c( 1, 1, 2, 3 ), 2, 2, byrow = TRUE ),
        widths = c( 3, 1 ), heights = c( 1, 2 ) )
hist( df$wt )
hist( df$mpg )
hist( df$disp )
```

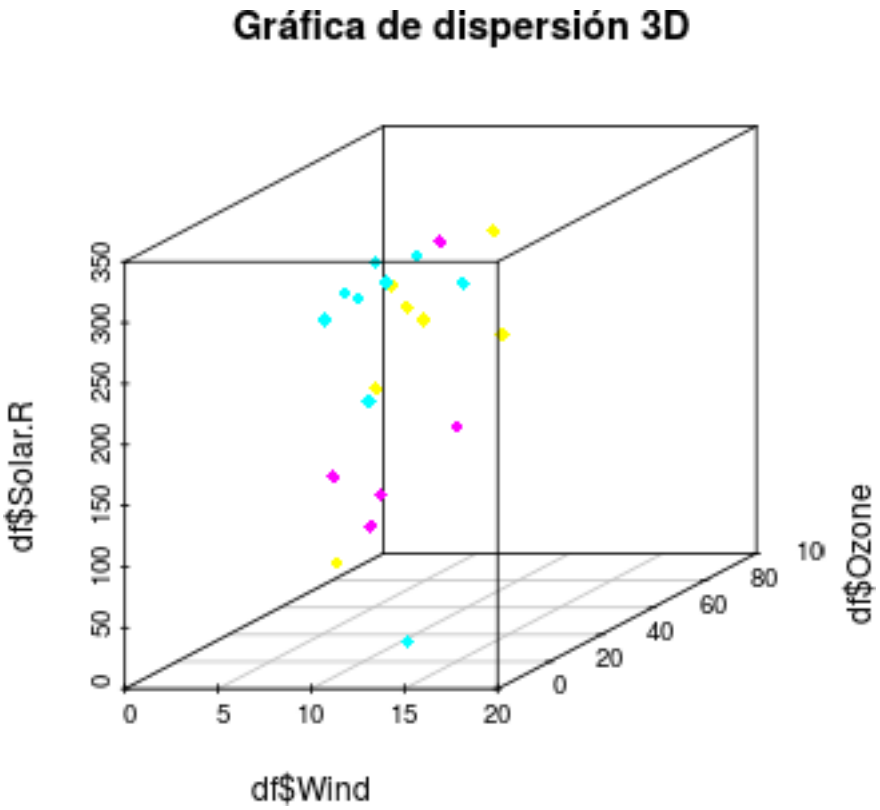


3.7. Gráficos más complejos

En este apartado hemos hecho uso tanto de Kabacoff (2014) como de Peterson (2013) para el desarrollo del mismo.

3.7.1. Gráfico de dispersión 3D

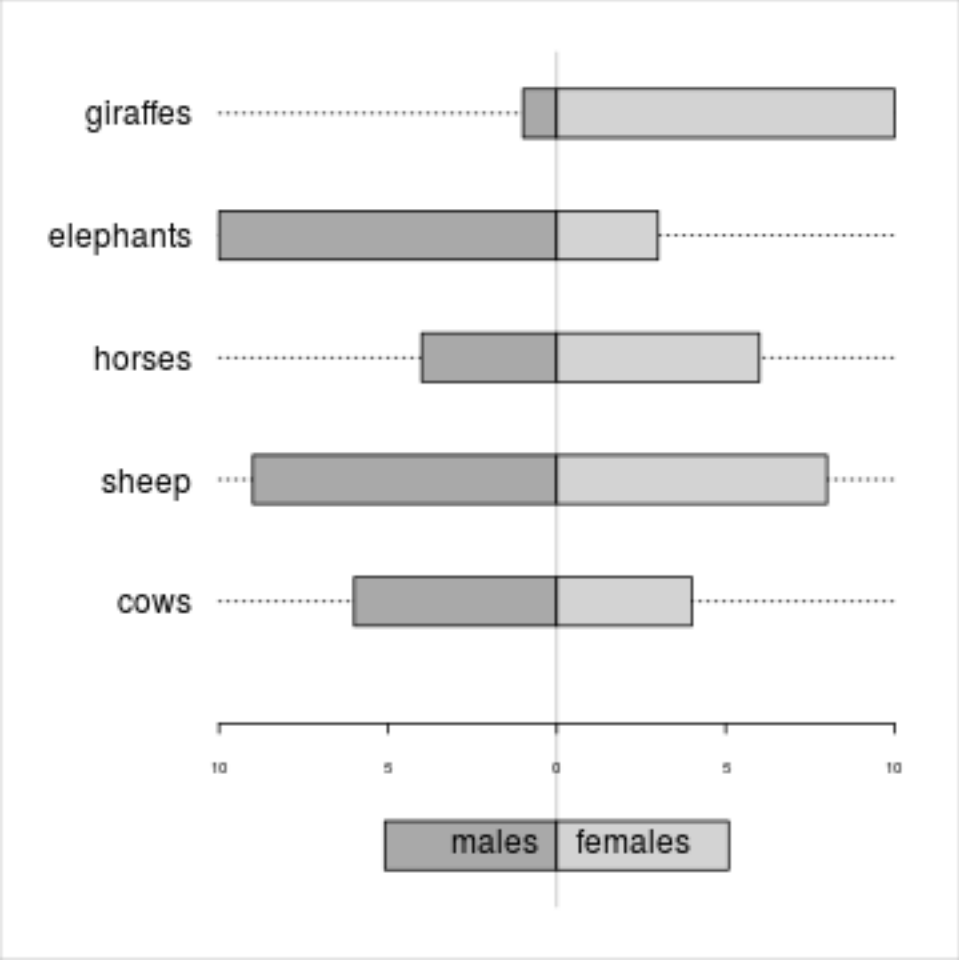
```
library( datasets )
data( airquality )
df <- airquality[ airquality$Month == c(5,6,7), ]

library( scatterplot3d )
colores <- factor( df$Month )
scatterplot3d(df$Wind, df$Ozone, df$Solar.R,
              color = colores, pch = 18,
              main = "Gráfica de dispersión 3D" )
```



3.7.2. Gráfico de barras horizontales

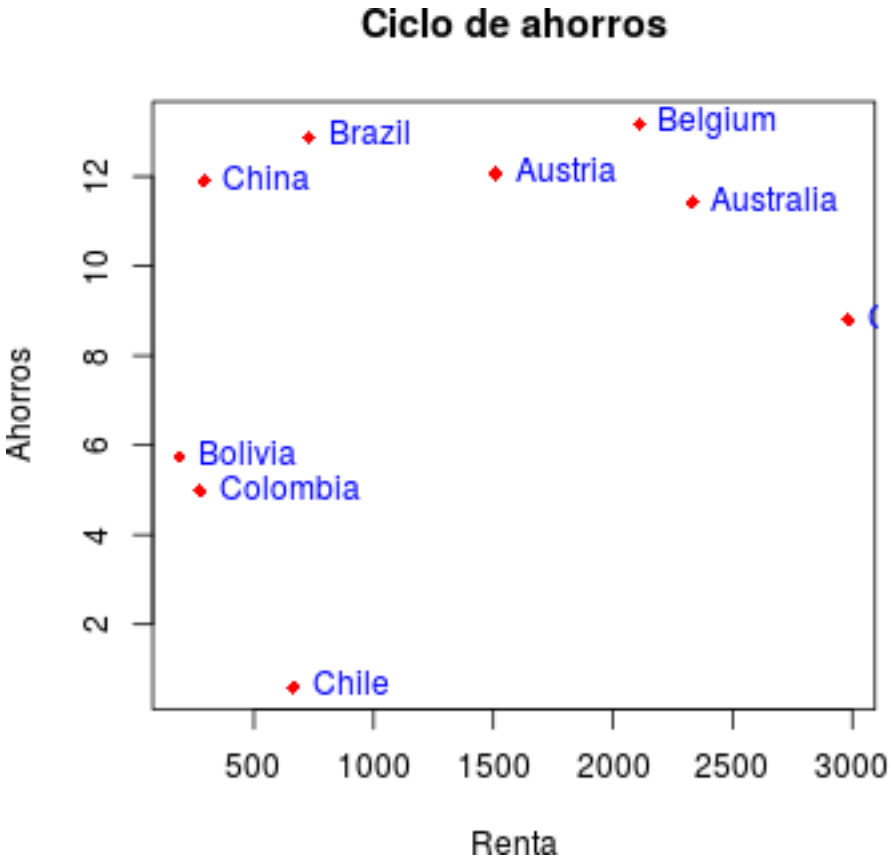
```
groups <- c( "cows", "sheep", "horses", "elephants", "giraffes" )
males <- sample( 1:10, 5 )
females <- sample( 1:10, 5 ) # muestras aleatorias

par( mar = c( 0.5, 5, 0.5, 1 ) ) # margenes

plot.new( )
plot.window( xlim = c( -10, 10 ), ylim = c( -1.5, 5.5 ) )

ticks <- seq( -10, 10, 5 )
y <- 1:5
h <- 0.2

lines( rep( 0, 2 ), c( -1.5, 5.5 ), col = "grey" )
segments( -10, y, 10, y, lty = "dotted" )
rect( -males, y-h, 0, y+h, col = "dark grey" )
rect( 0, y-h, females, y+h, col = "light grey" )
mtext( groups, at = y, adj = 1, side = 2, las = 2 )
par( cex.axis = 0.5, mex = 0.5 )
axis( 1, at = ticks, labels = abs( ticks ), pos = 0 )

tw <- 1.5*strwidth( "females" )
rect( -tw, -1-h, 0, -1+h, col = "dark grey" )
rect( 0, -1-h, tw, -1+h, col = "light grey" )
text( 0, -1, "males", pos = 2 )
```

```
text( 0, -1, "females", pos = 4)

box( "inner", col = "grey" )
```



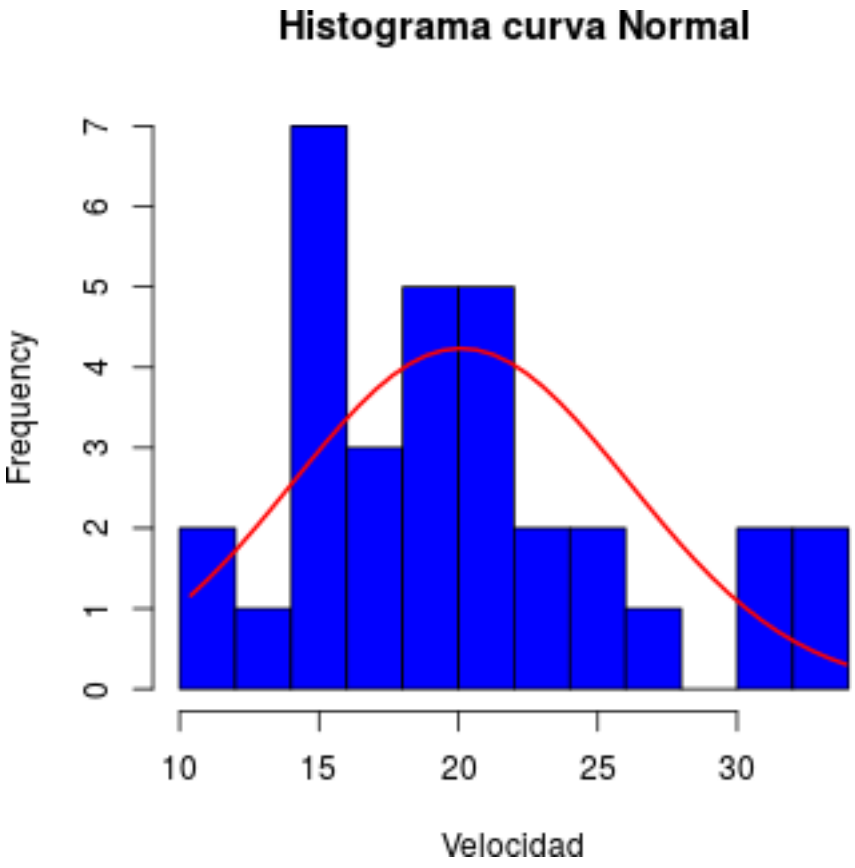
3.7.3. Etiquetar casos

```
df <- LifeCycleSavings[ 1:9, ]
plot( df$sr ~ df$dpi, xlim = range( df$dpi ), col = "red", pch = 18, xlab = "Renta",
     ylab = 'Ahorros', main = 'Ciclo de ahorros', data = df )

text( df$sr ~ df$dpi, labels = row.names( df ), pos = 4, col = "blue" )
```



3.7.4. Histograma con normal

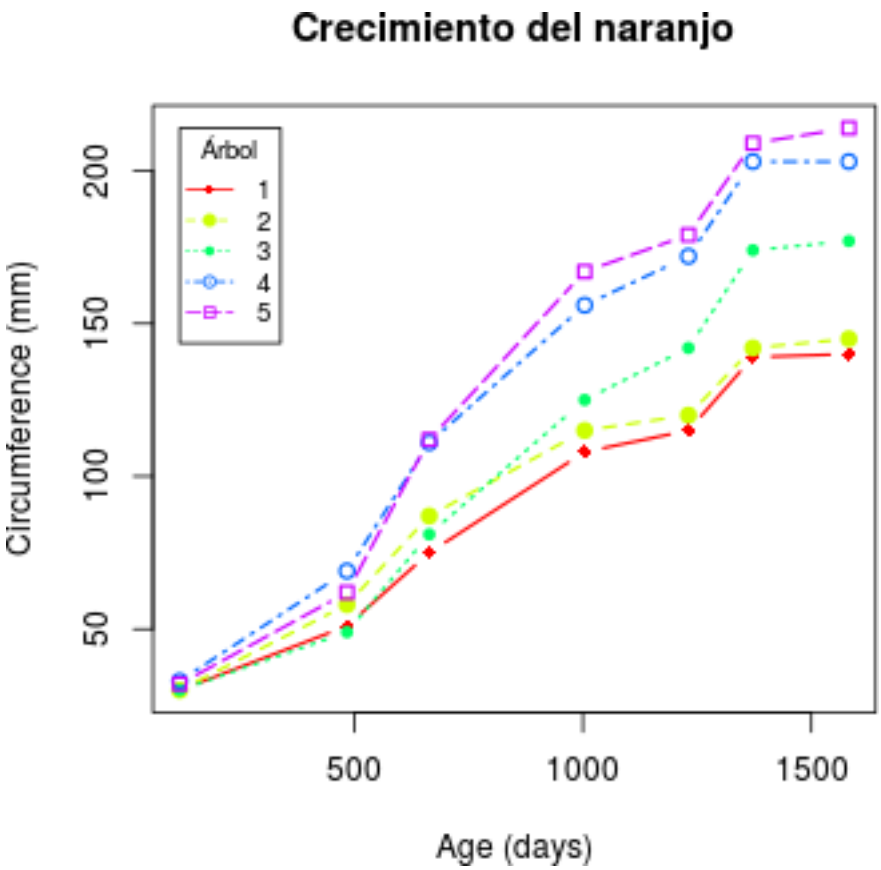
```
# Add a Normal Curve (Thanks to Peter Dalgaard)
x <- mtcars$mpg
h <- hist( x, breaks = 10, col = "blue", xlab = "Velocidad",
        main = "Histograma curva Normal" )
xfit <- seq( min( x ), max( x ), length = 40 )
yfit <- dnorm( xfit, mean = mean( x ), sd = sd( x ) )
yfit <- yfit * diff( h$mids[ 1:2 ] ) * length( x )
lines( xfit, yfit, col = "red", lwd = 2 )
```



3.7.5. Gráfico de lineas

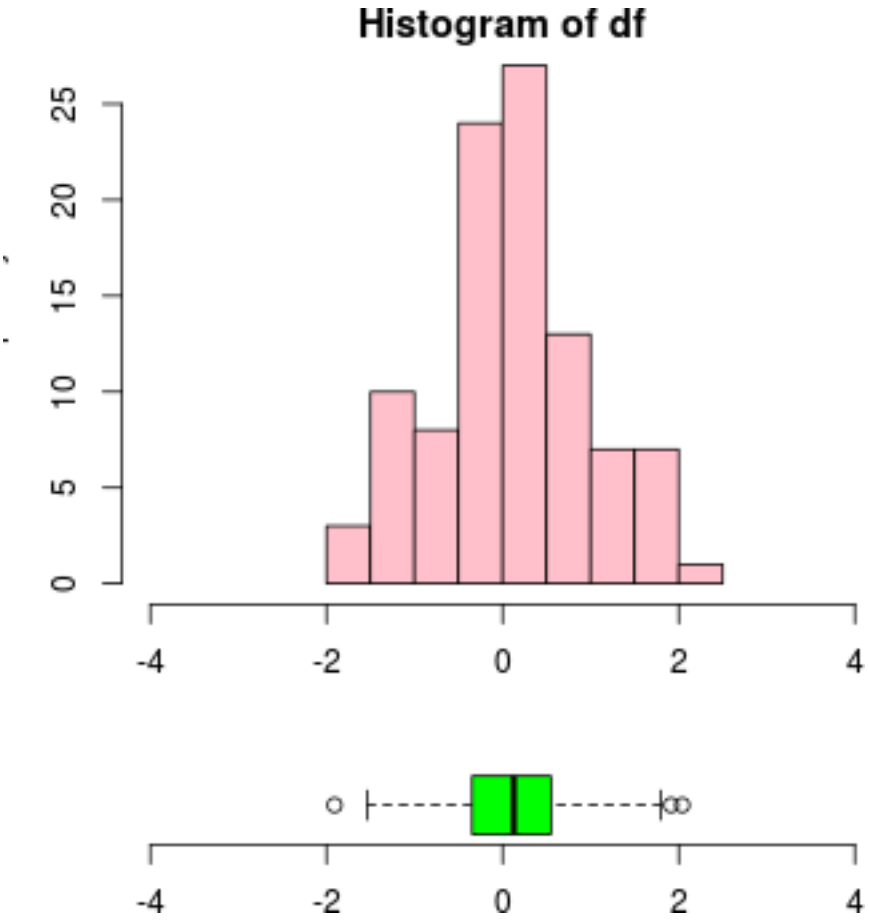
```
# Convertir un factor a un vector numérico
Orange$Tree <- as.numeric( Orange$Tree )
ntrees <- max( Orange$Tree ) # máx árboles

# Rango de x e y
xrange <- range( Orange$age )
yrange <- range( Orange$circumference )

# Crear el gráfico con sus dimensiones
plot( xrange, yrange, type = "n", xlab = "Age (days)",
ylab = "Circumference (mm)" )

# Colores, lineas y caracteres
colors <- rainbow( ntrees )
ltipo <- c( 1:ntrees )
char <- seq( 18, 18 + ntrees, 1 )

# Añadir las lineas al gráfico
for (i in 1:ntrees ) {
tree <- subset( Orange, Tree == i )
lines( tree$age, tree$circumference, type = "b", lwd = 1.5,
lty = ltipo[ i ], col = colors[ i ], pch = char[ i ] )
}
```

```
# Añadir un título and sutitulo
title( "Crecimiento del naranjo" )

# Añadir la leyenda
legend( xrange[ 1 ], yrange[ 2 ], 1:ntrees, cex = 0.8,
        col = colors, pch = char, lty = ltipo, title = "Árbol" )
```



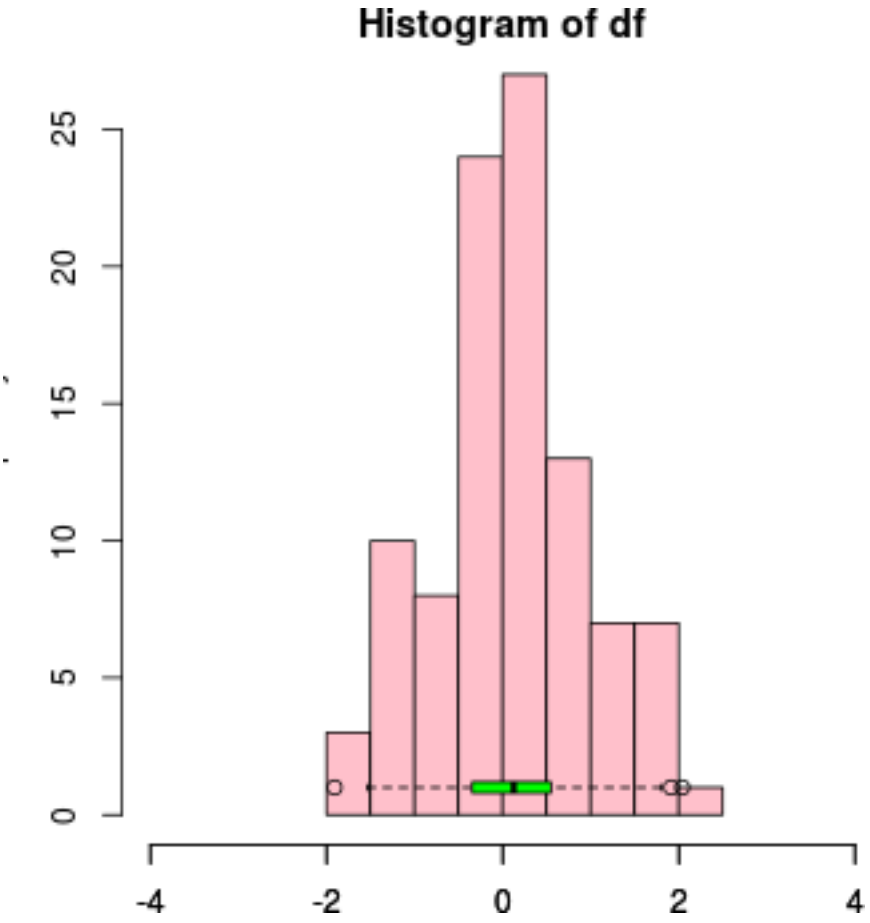
3.7.6. Histograma con boxplot

```
set.seed( 4566 )
df <- rnorm( 100 )
layout( mat = matrix( c( 1, 2 ), 2, 1, byrow = TRUE ),  height = c( 3, 1 ) )
par( mar = c( 3.1, 3.1, 1.1, 2.1 ) )
hist( df, xlim = c( -4, 4 ), col = "pink" )
boxplot( df, horizontal = TRUE,  outline = TRUE, ylim = c( -4, 4 ),
         frame = FALSE, col = "green1", width = 10 )
```



Ponemos el *boxplot* dentro del gráfico del histograma

```
par( mar = c( 3.1, 3.1, 1.1, 2.1 ) )
hist( df, xlim = c( -4,4 ), col = "pink" )
boxplot( df, horizontal = TRUE,  outline = TRUE, ylim = c( -4, 4 ),
         frame = FALSE, col = "green1", add = TRUE )
```



Otra

```
# Add boxplots to a scatterplot
par( fig = c( 0, 0.8, 0, 0.8 ), new = TRUE )
plot( mtcars$wt, mtcars$mpg, xlab="Peso", ylab = "Velocidad" )
par( fig = c( 0, 0.8, 0.55, 1 ), new = TRUE )
boxplot( mtcars$wt, horizontal = TRUE, axes = FALSE )
par( fig = c( 0.65, 1, 0, 0.8), new = TRUE )
boxplot( mtcars$mpg, axes = FALSE )

mtext("Dispersión con boxplot", side = 3, outer = TRUE, line = -3 )
```

Volver al índice del curso

Referencias y bibliografía

Kabacoff, R. (2014). Quick-r. basic graphs. Retrieved November 13, 2014, from http://www.statmethods.net/graphs/index.html

Maurandi-López, A., Balsalobre-Rodríguez, C., & del-Río-Alonso, L. (2013). *Fundamentos estadísticos para investigación. introducción a r*. BUBOK Publishing S.L. Retrieved from http://www.bubok.es/libros/223207/Fundamentos-estadisticos-para-investigacionIntroduccion-a-R

Peterson, J. (2013). Graph gallery: A collection. Retrieved November 13, 2014, from http://rgraphgallery.blogspot.ca/

Roger D. Peng, J. L., & Caffo, B. (2014). Quick-r. basic graphs. Johns Hopkins University. Retrieved November 13, 2014, from http://www.statmethods.net/graphs/index.html