Índice general

Tema 4

Metodología del análisis estadístico con R

4.1. Gráficos

Los gráficos son una herramienta fundamental no sólo para presentar resultados sino para visualizar e intuir estructuras en los datos

> *The greatest value of a picture is when it forces us to notice what we never expected to see.* (John Tukey).

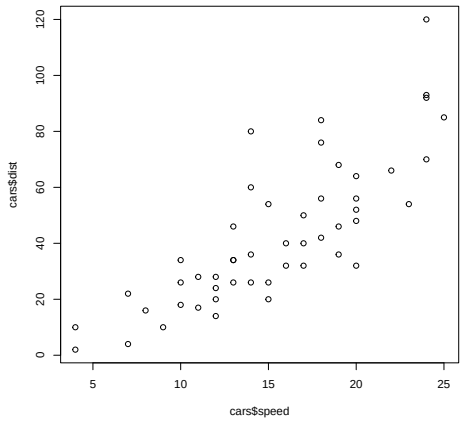
Si bien los gráficos han estado unidos a la Estadística desde sus inicios, hoy en día han cobrado un papel fundamental constituyendo un ámbito de investigación en rápido desarrollo que comparten las áreas de Estadística, Computación y la popular *Data Science*. En este tema nos ocuparemos fundamentalmente de describir las opciones básicas disponibles en el sistema base. Recuerda que puedes ver un buen conjunto de ejemplos escribiendo `demo(graphics)`. No obstante, en la actualidad existen diversos paquetes específicos para la creación de gráficos avanzados, de estos haremos tan sólo breves menciones al finalizar este apartado del tema, señalando algunas referencias para permitir si se desea hacer un estudio más profundo del tema.

4.1.1. La función `plot`

La función básica para la creación de gráficos es `plot`. Se trata de una función genérica que se puede aplicar a distintos tipos de objetos. Su aplicación más básica (`plot.default`) opera sobre vectores y produce diagramas de dispersión.

La sintaxis básica de la función es sencilla y la podemos ilustrar en el siguiente ejemplo:

```
> plot(cars$speed,cars$dist)
```



En este caso la función se ha evaluado con dos argumentos, ambos vectores numéricos y el resultado es un diagrama de dispersión del primero (en el eje de abcisas) frente al segundo (eje de ordenadas). Es posible obtener el mismo resultado escribiendo `plot(cars)`, en cuyo caso se estaría indicando como único argumento el data frame `cars`[1] que contiene dos columnas.

La apariencia del gráfico anterior es la que la función tiene programada por defecto (títulos de los ejes, tipo de símbolos y color de los puntos, rango de los ejes etc.). Consultando la ayuda de la función (`help(plot)` o más específicamente `help(plot.default)`) podemos ver que esta se puede modificar cambiando el valor por defecto de sus argumentos (los más relevantes los hemos resumido en la Tabla 4.1). Así por ejemplo, el gráfico anterior representa por defecto puntos asociados a cada par de coordenadas (x, y) indicadas en los dos primeros argumentos. Esto corresponde al valor del argumento `type='p'` (por defecto). Otras posibilidades incluyen `type='l'` (cada par (x, y) se unen por líneas), o `type='b'` (puntos y lineas), así como otras mostradas en la Tabla 4.2. Por otro lado el argumento `pch` indica el tipo de símbolo que será empleado para los puntos. Su valor puede ser numérico, del 0 al 25, o caracteres. En particular, los símbolos del 21 al 25 son figuras bicolores, que toman el valor del parámetro `col` para su contorno y del parámetro `bg`, para el relleno. Por defecto (si no se especifica como hicimos en el primer gráfico anterior) se considera el valor `pch=1` que consiste en un círculo pequeño sin relleno y con borde en color negro. Respecto a los colores es posible especificarlos de varias formas[2]. En este ejemplo lo hemos hecho mediante nombres (`cyan`, `blue`, `red`, etc., puedes ver los disponibles escribiendo `colors()`). Otra forma más sencilla, aunque más limitada, es mediante

[1] Comprueba que en efecto se obtiene el mismo resultado. Si el data frame tuviera más de dos columnas entonces se obtendría una matriz de diagramas de dispersión considerando las columnas dos a dos. Prueba como ejemplo a representar el data frame `mtcars` escribiendo `plot(mtcars)`.

[2] El esquema más general para la especificación de un color es por medio de la especificación de sus contenidos de los colores primarios: rojo, verde y azul, RGB, por sus nombres en inglés. En este caso, hay por lo menos dos maneras de especificar el color: por medio de su código hexadecimal y por medio de la especificación de sus contenidos porcentuales de color usando la función `rgb`. Además en el lenguaje se han desarrollado diversas paletas de colores con fines específicos, y que están asociadas a funciones tales como: `heat.colors`, `terrain.colors`, `rainbow`, etc.

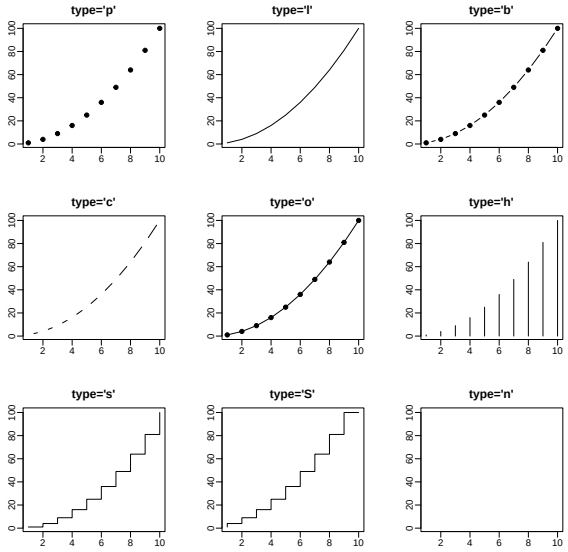
un número entero. Este número hace referencia a la posición del color en la paleta activa (consultar escribiendo `palette()`). Por defecto esta dispone de ocho colores diferentes; esto es, del 1 al 8, en orden: negro, rojo, verde, azul, turquesa, magenta, amarillo y gris. A partir del color 9, la secuencia se repite.

`Parámetro`	Descripción
`type`	Tipo de gráfico
`main`	Título del gráfico
`sub`	Subtítulo
`xlab`, `ylab`	Etiquetas eje abcisas y ordenadas
`xlim`,`ylim`	Rango de valores en cada eje
`col`	Color
`pch`	Símbolo para puntos
`bg`	Color de fondo
`cex`	Escalado del tamaño de los símbolos
`lty`	Tipo de líneas
`lwd`	Grosor de línea

Tabla 4.1: Algunos argumentos opcionales de la función `plot`

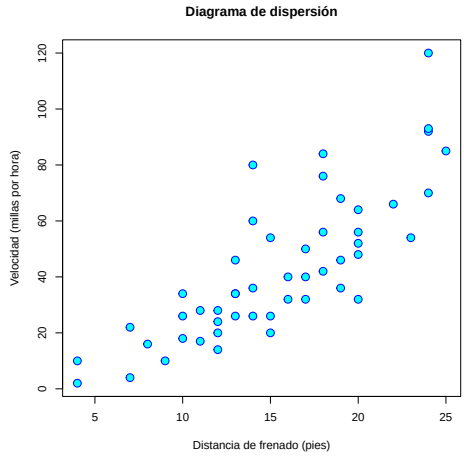
`type=`	Tipo de gráfico
'p'	puntos (por defecto, gráfico dispersión)
'l'	líneas
'b'	puntos y líneas
'c'	solo las líneas de `type='b'`
'o'	puntos y líneas superpuestos
'h'	barras (tipo histograma)
's'	función en escalera (también `'S'`)
'n'	solo se representan los ejes

Tabla 4.2: Tipo de gráficos que ofrece la función `plot`



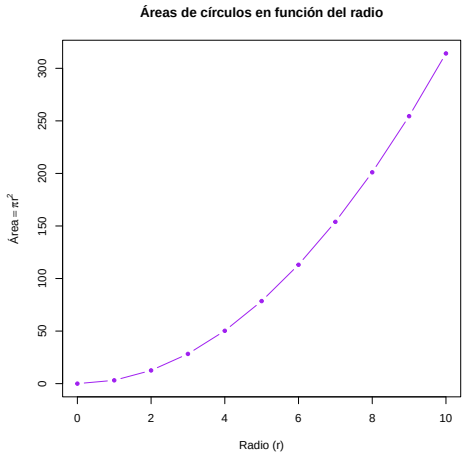
A continuación probamos algunas modificaciones de los mismos sobre el gráfico que construíamos antes (te proponemos además que pruebes otras modificaciones practicando con los distintos argumentos).

```
> plot(cars$speed,cars$dist,main='Diagrama de dispersión',
+      xlab='Distancia de frenado (pies)', ylab='Velocidad (millas por hora)',
+      pch=21, col='blue', bg='cyan', cex=1.5)
```



El siguiente ejemplo construye un gráfico que muestra las áreas de los círculos en función de sus radios:

```
> radio <- 0:10
> area <- pi*radio^2
> plot(radio, area, type='b', main='Áreas de círculos en función del radio',
+ xlab="Radio (r)", ylab=expression(Área == pi*r^2), col="purple", pch=20)
```

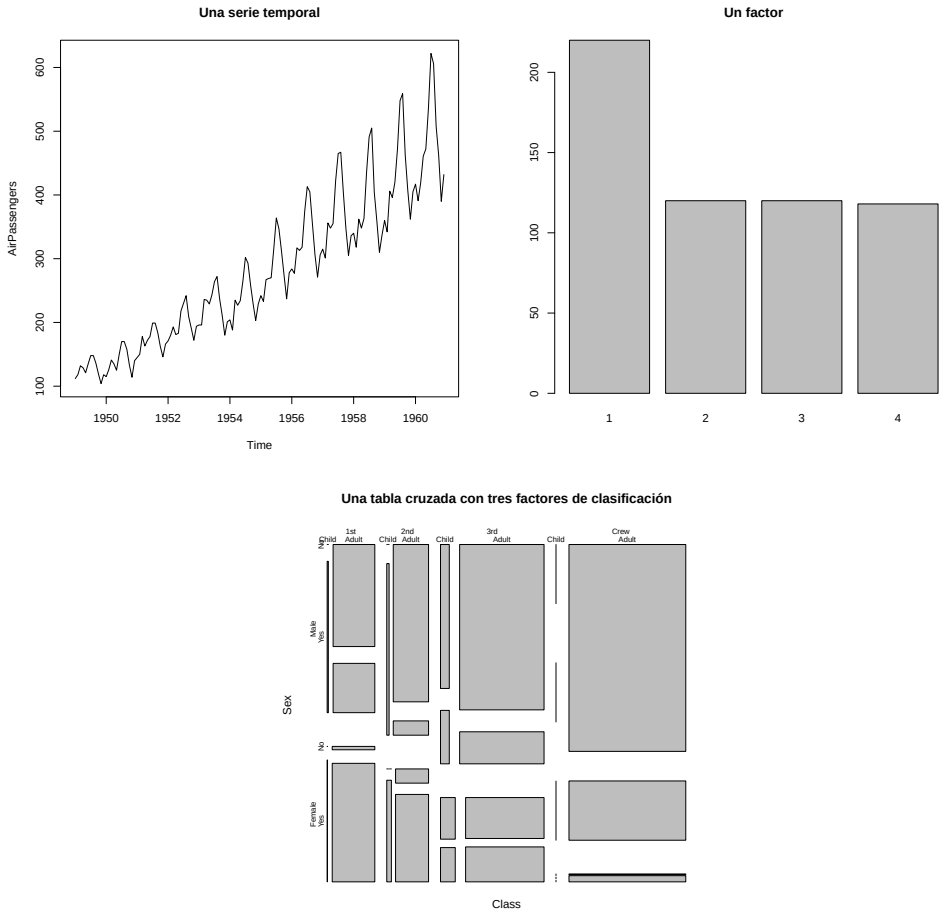


En este gráfico se ha indicado como etiqueta del eje de ordenadas una expresión matemática. Para ello se ha usado la función `expression` a través de la cual el lenguaje se encarga de traducir la expresión indicada a su representación matemática simbólica[3].

En los ejemplos anteriores hemos evaluado `plot` en vectores numéricos, obteniendo en todos los casos gráficos del tipo diagrama de dispersión. Como comentábamos antes, se trata de una función genérica aplicable a varios tipos de objetos y cuyo resultado dependerá del mismo. Observa a continuación algunos ejemplos donde la aplicamos a otro tipo de objetos:

```
> class(AirPassengers)

[1] "ts"

> plot(AirPassengers, main='Una serie temporal')
> class(ChickWeight$Diet)

[1] "factor"

> plot(ChickWeight$Diet,main='Un factor')
> class(Titanic)

[1] "table"

> plot(Titanic, main='Una tabla cruzada con tres factores de clasificación')
```

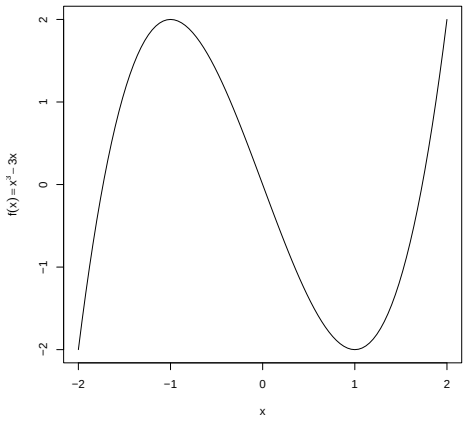
[3]Puedes consultar la notación matemática disponible en la ayuda, `help(plotmath)`, y ver algunos ejemplos con `example(plotmath)` o `demo(plotmath)`.



4.1.2. Gráficos de funciones matemáticas: función `curve`

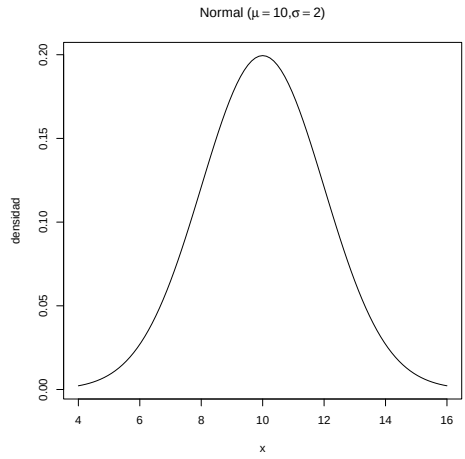
La función `curve` permite representar una función $f(x)$ en un intervalo de x dado. Como ejemplo representamos la función $f(x) = x^3 - 3x$ en el intervalo $(-2, 2)$:

```
> curve(x^3-3*x, -2, 2, ylab=expression(f(x)==x^3-3*x))
```



En este ejemplo el primer argumento es la expresión de la función dependiendo de `x`. También es posible proporcionar un objeto de tipo función. El siguiente ejemplo representa la función de densidad de una distribución Normal con media $\mu = 10$ y desviación típica $\sigma = 2$:

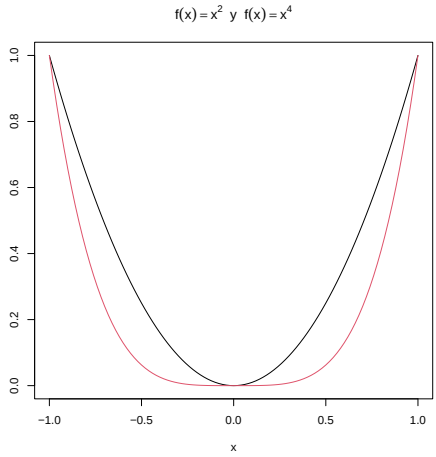
```
> curve(dnorm(x,mean=10,sd=2), 4, 16, ylab='densidad',
+       main=expression(paste('Normal (' , mu==10, ',', sigma==2 , ')')))
```



En este ejemplo el primer argumento es la función `dnorm(x,mean,sd)` que evalúa la función de densidad de la Normal (con la media y la desviación típica especificada) en un vector `x`.

Finalmente es posible superponer dos funciones en un mismo gráfico usando el argumento `add` de la función `curve` como muestra el siguiente ejemplo:

```
> curve(x^2,-1, 1, ylab='',main=expression(paste(f(x)==x^2,
+     '  y  ', f(x)==x^4)))
> curve(x^4, -1, 1, col=2,add=TRUE)
```



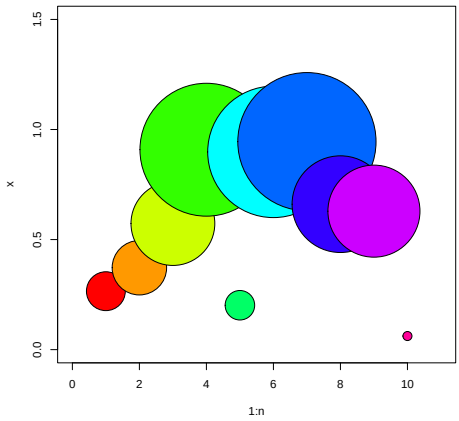
Ejercicio 1. Representa en un mismo gráfico las funciones de densidad de tres distribuciones normales con parámetros: $(\mu, \sigma) = \{(0,1); (0,0.5); (3,1)\}$. Utiliza distintos colores y tipos de línea para distinguirlas. Además deberás definir los límites de los ejes x e y de modo que se visualicen correctamente.

4.1.3. Gráficos de símbolos: función `symbols`

La función `symbols` permite dibujar varias formas (círculos, rectángulos, estrellas, etc.) en una posición determinada de un gráfico. El tipo de forma, la posición, así como su tamaño se indican como argumentos.

Como ejemplo generamos una muestra de 10 valores de una uniforme entre 0 y 1, y asociado a cada valor representamos un círculo de distinto color.

```
> n<-10
> set.seed(1)
> x<-runif(n)
> colores<-rainbow(n)
> symbols(1:n,x,circles=x,bg=colores,ylim=c(0,1.5),xlim=c(0,11))
```



4.1.4. Añadir elementos a un gráfico

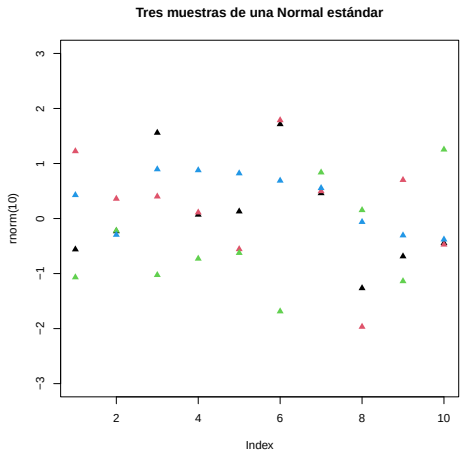
Una característica muy conveniente de los gráficos en R es lo fácil que resulta añadir "cosas" a un gráfico ya creado. Estas pueden ser líneas, puntos, símbolos, texto, leyendas, etc. para lo que disponemos de varias funciones[4]. A continuación describimos algunas de las más habituales.

Función `points`

Esta función genérica que permite añadir una secuencia de puntos a un gráfico ya creado en las coordenadas indicadas. Su sintaxis básica sería `points(x,y)`, si bien admite argumentos `pch`, `col`, etc. que permiten modificar la forma, color etc. de los puntos. Veamos su uso con algún ejemplo:

```
> set.seed(123)
> plot(rnorm(10),main='Tres muestras de una Normal estándar',
+      pch=17,ylim=c(-3,3))
> points(rnorm(10),pch=17,col=2)
> points(rnorm(10),pch=17,col=3)
> points(rnorm(10),pch=17,col=4)
```

[4]Es lo que el lenguaje denomina *low-level plotting functions* en contraposición a *high-level plotting functions* como la función `plot`.



Observa que la función `plot` se evalúa en primera lugar lo que genera el gráfico y a continuación ya podemos añadir los puntos con la función `points`. Sin un gráfico de base esta última función nos daría un error. Por otro lado, al evaluar ambas funciones sólo estamos indicando un vector de datos (`rnorm(10)`), esto da lugar a lo que denomina *index plot*. Los elementos del vector (datos) se representan en el eje de ordenadas frente al índice correspondiente en el eje de abcisas.

Funciones `lines` y `abline`

La función `lines` es similar a `points` sólo que dibuja líneas uniendo las coordenadas dadas en los argumentos. La función `abline` permite añadir líneas que pueden ser especificadas de varias formas:

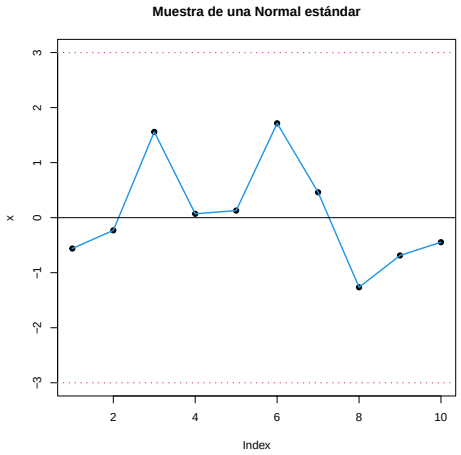
- `abline(a,b)`, añade una recta con pendiente `b` y ordenada en el origen `a`;
- `abline(h=y)`, añade una recta horizontal a lo largo del gráfico a la altura `y`;
- `abline(v=x)`, añade una recta vertical en la abcisa `x`,
- `abline(lm.obj)`, superpone una recta de regresión calculada y almacenada en un objeto de tipo `lm`[5].

Vemos estas opciones en algunos ejemplos:

```
> set.seed(123)
> x<-rnorm(10)
> plot(x,main='Muestra de una Normal estándar',
+      ylim=c(-3,3),pch=19)
> lines(x,col=4,lwd=2)
> abline(h=0,col=1)
> abline(h=-3,col=2,lty=3,lwd=2)
```

[5]Este tipo de objeto lo estudiaremos con más detalle más adelante en este tema.

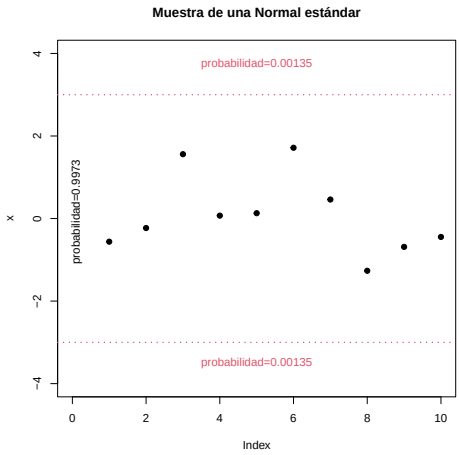
```
> abline(h=3,col=2,lty=3,lwd=2)
```



Función `text`

Permite añadir texto a un gráfico indicando su posición en coordenadas. Sus argumentos principales son: `x` e `y` (vectores numéricos especificando las coordenadas en las que se escribirá el texto); `labels` (vector de tipo carácter especificando el texto a escribir, con igual longitud a `x`); `adj` (uno o dos valores entre 0 y 1 que indican el ajuste del texto en `x` e `y`, respectivamente, por defecto `adj=c(0.5,0.5)` lo que supone centrarlo en ambas coordenadas); `pos` (uno de los valores `1, 2, 3` o `4`, indicando que el texto debe estar debajo, a la izquierda, encima, o a la derecha de las coordenadas especificadas, respectivamente); `cex`, `col` y `font` (indicando tamaño, color y fuente del texto); `srt` (ángulo de rotación del texto respecto al valor `adj`).

```
> plot(x,main='Muestra de una Normal estándar',
+      ylim=c(-4,4),xlim=c(0,10),pch=19)
> abline(h=-3,col=2,lty=3,lwd=2)
> abline(h=3,col=2,lty=3,lwd=2)
> pr3<-round(pnorm(-3),5) # P[X<-3] para X->N(0,1)
> text(5,3.5,paste0('probabilidad=',pr3),col=2,pos=3)
> text(0.2,0,paste0('probabilidad=',1-2*pr3),pos=3,srt=90)
> text(5,-3.5,paste0('probabilidad=',pr3),col=2)
```

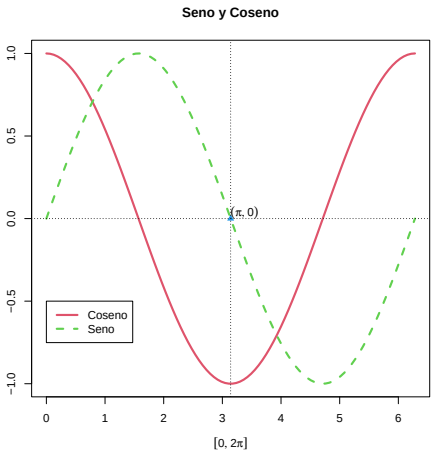


Función `legend`

Permite añadir una leyenda al gráfico actual en la posición especificada. Los elementos en el gráfico (puntos, líneas) se identifican con las etiquetas proporcionadas en el argumento `legend` de la función. Consultando la ayuda de la función se pueden ver varios parámetros que permiten adaptar el aspecto de la leyenda al deseado. Como ilustración de uso y de las posibilidades de la función `legend` mostramos a continuación algunos ejemplos.

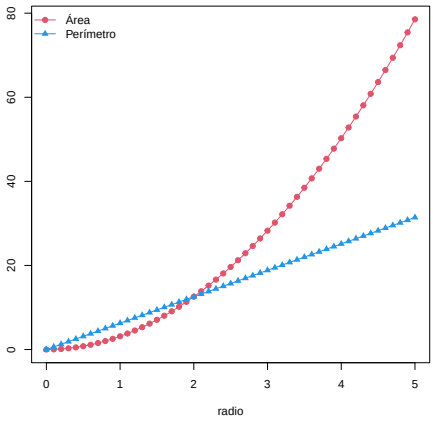
El primer ejemplo ofrece una gráfica que representa las funciones seno y coseno en el intervalo $[0, 2\pi]$. Para superponer las funciones hacemos uso de la función `lines`. Y para distinguir una de la otra añadimos una leyenda usando la función `legend`.

```
> x <- seq(0, 2 * pi, length = 100)
> y1 <- cos(x)
> y2 <- sin(x)
> plot(x, y1, type = "l", col = 2, lwd = 3,
+      xlab = expression(group("[",list(0, 2*pi),"]")),
+      ylab = "", main = "Seno y Coseno")
> lines(x, y2, col = 3, lwd = 3, lty = 2)
> points(pi, 0, pch = 17, col = 4)
> legend(0, -0.5, c("Coseno", "Seno"), col = 2:3, lty = 1:2, lwd = 3)
> abline(v = pi, lty = 3)
> abline(h = 0, lty = 3)
> text(pi, 0, expression(group("(",list(pi,0),")")),
+      adj = c(0,0))
```



El segundo ejemplo consiste en representar una gráfica que contiene dos curvas: el área y el perímetro de un círculo como función de su radio.

```
> radio <- seq(0,5,by=0.1)
> area <- pi*radio^2
> perimetro <- 2*pi*radio
> plot(radio, area,  type='o', ylab='', pch=19, col=2)
> lines(radio, perimetro, type='o', pch=17, col=4)
> legend('topleft', legend = c('Área','Perímetro'),
+ lty = 1, pch = c(19,17), col = c(2,4), bt='n')
```



Ejercicio 2. Añade una leyenda al gráfico que creaste como solución del Ejercicio 1.

Otras funciones

A continuación describimos otras funciones que permiten añadir distintos elementos a un gráfico que ya existe.

`polygon`: Permite añadir un polígono, definido por sus vértices ordenados, a un gráfico existente.

`segments`: Añade segmentos lineales.

`arrows`: Añade flechas.

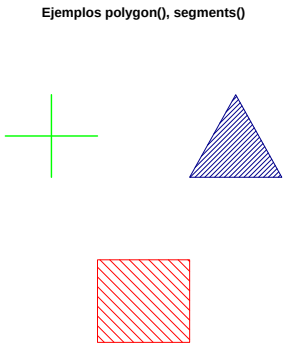
`grid`: Añade líneas de referencia (verticales, horizontales o una cuadrícula).

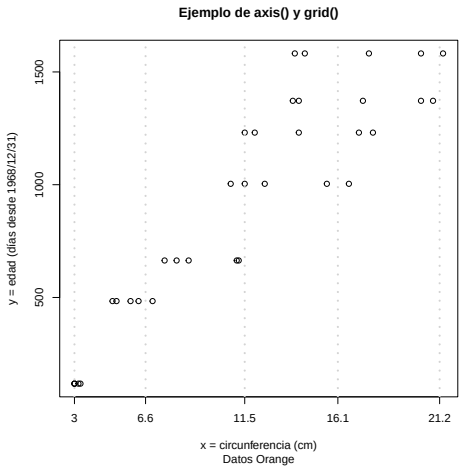
`title`: Añade un título y/o un subtítulo.

`axis`: Dibuja ejes horizontales o verticales. Su utilidad está asociada al caso en que se especifica el argumento `axes=FALSE` (también `xaxt='n'` y `yaxt='n'`) en una función gráfica de primer nivel como `plot`.

El siguiente ejemplo ilustra algunas de estas funciones:

```
> # Ejemplo con polygon() y segments()
> plot(1:9,type='n',xlab='',ylab='', axes=FALSE,
+      main='Ejemplos polygon(), segments()')
> polygon(c(4,6,6,4,NA,6,8,7),c(2,2,4,4,NA,6,6,8),
+         density=c(10,20),angle=c(-45,45),col=c('red','darkblue'))
> segments(x0=c(3,2),y0=c(6,7),x1=c(3,4),y1=c(8,7),
+          lwd=2,col='green')
> # Ejemplo con axis() y title()
> plot(Orange$circumference,Orange$age,xaxt='n',
+      xlab='',ylab='')
> x<-quantile(Orange$circumference,probs=c(0.01,0.25,0.5,0.75,0.99))
> xcm<-x/10 ## x está en milímetros, xcm en centímetros
> axis(1,at=x,labels=round(xcm,1))
> abline(v=x,lty=3,lwd=3,col='lightgray')
> title(main='Ejemplo de axis() y grid()',
+       sub='Datos Orange',
+       xlab='x = circunferencia (cm)',
+       ylab='y = edad (días desde 1968/12/31)')
```





4.1.5. Opciones gráficas: función `par`

Las funciones gráficas que hemos visto permiten, a través de sus argumentos, adaptar el resultado a nuestros objetivos. Esto supone alteraciones temporales de los parámetros gráficos de R. Sin embargo es posible realizar alteraciones permanentes usando la función `par`.

La función `par` permite acceder y modificar diversos parámetros gráficos para cada dispositivo gráfico[6]. Si escribimos `par()` obtenemos una lista con los valores que por defecto R ha establecido en las opciones gráficas. Consultando la ayuda de la función `par` podemos ver todos los parámetros disponibles (alrededor de 72) y su definición. Se trata de una larga lista de la que recogemos algunos de los que creemos más relevantes en la Tabla 4.3.

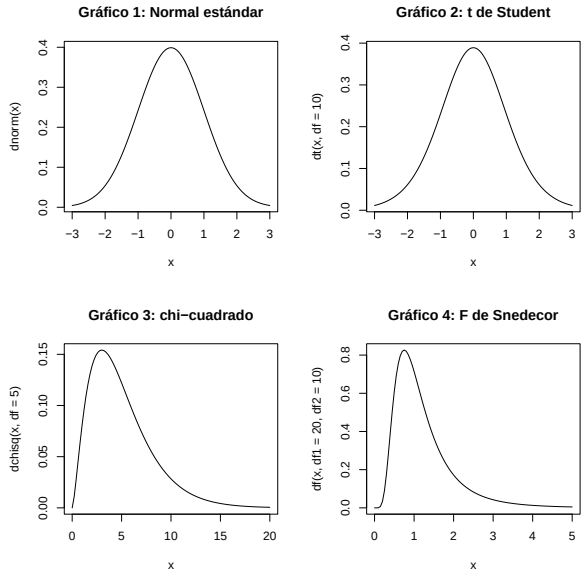
En el siguiente ejemplo ilustramos algunas de estas opciones. En primer lugar utilizamos el argumento `mfrow` para crear un panel con cuatro gráficos (dos filas y dos columnas), que corresponden a las funciones de densidad de cuatro distribuciones de probabilidad continuas:

```
> par(mfrow=c(2,2)) # matriz de 2x2 gráficos (se rellenan por filas)
> curve(dnorm(x),-3,3,main='Gráfico 1: Normal estándar')
> curve(dt(x,df=10),-3,3,main='Gráfico 2: t de Student')
> curve(dchisq(x,df=5),0,20,main='Gráfico 3: chi-cuadrado')
> curve(df(x,df1=20,df2=10),0,5,main='Gráfico 4: F de Snedecor')
```

[6]Los dipositivos gráficos los trataremos en la siguiente sección.

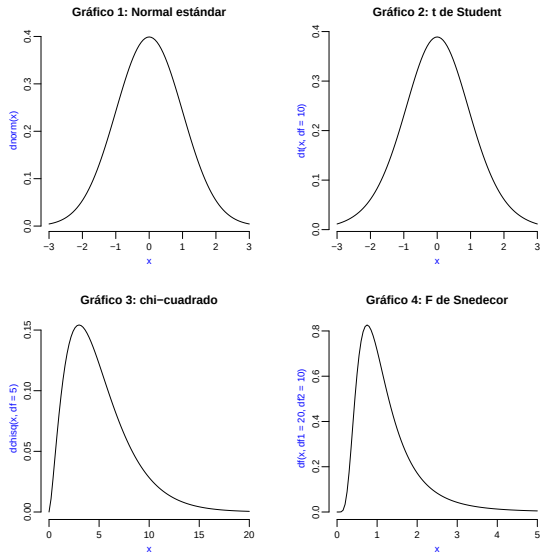
Parámetro	Descripción
	Elementos gráficos
pch,lty,lwd	Tipo de punto, línea, color
col,col.axis,col.lab,col.main	Colores (puntos/líneas, ejes, etiquetas, etc.)
cex,cex.axis,cex.lab,cex.main	Factor que define el tamaño
font,font.axis,font.lab,font.main	Fuente/estilo (2:cursiva, 3:negrita, 4:ambos)
adj	Justificación del texto relativo a su posición
	Ejes y marcas
lab=c(5,7,12)	Número de marcas en los ejes (x,y) y longitud etiqueta (caracteres).
las	Orientación de las etiquetas de los ejes (0=paralela, 1=horizontal, 2=perpendicular)
mgp=c(3,1,0)	Posición de las componentes de los ejes
tck=0.01	Longitud de las marcas en los ejes
bty='o'	Tipo de cuadro que contiene el gráfico ('o','l', '7', 'c','u', ']' ó 'n')
	Márgenes
mai=c(1,0.5,0.5,0)	Margen inferior, izquierdo, arriba y derecho, respectivamente, en pulgadas
mar=c(4,2,2,1)	Como mai, número líneas en lugar de pulgadas
	Paneles con múltiples gráficos
mfcol=c(1,1), mfrow=c(1,1)	Número de filas y columnas gráficos por columnas o filas, respectivamente
mfg=c(2,2,3,2)	Posición del gráfico actual en la matriz
oma=c(2,0,3,0), omi=c(0,0,0.8,0)	Como mar y mai

Tabla 4.3: Opciones gráficas: argumentos de la función par



Observa que el panel gráfico resultante admite varias mejoras, sobre todo en lo relativo a los márgenes. Estos y algunos o otros aspectos se pueden mejorar modificando algunos de los parámetros gráficos, por ejemplo lo hacemos con `mar`, `oma`, `mgp`, `cex.axis`, `cex.lab`, `cex.main`, `col.lab` y , `bty` a continuación:

```
> par(mar=c(3.5,3.5,2.5,1.5),oma=c(1,1,1,1),mgp=c(1.5,0.5,0),
+     cex.axis=0.8,cex.lab=0.8,cex.main=1,col.lab='blue',bty='n')
> par(mfrow=c(2,2)) # matriz de 2x2 gráficos (se rellenan por filas)
> curve(dnorm(x),-3,3,main='Gráfico 1: Normal estándar')
> curve(dt(x,df=10),-3,3,main='Gráfico 2: t de Student')
> curve(dchisq(x,df=5),0,20,main='Gráfico 3: chi-cuadrado')
> curve(df(x,df1=20,df2=10),0,5,main='Gráfico 4: F de Snedecor')
```



4.1.6. Dispositivos gráficos

En los ejemplos anteriores hemos visto que R automáticamente abre un dispositivo gráfico (*graphic device*) donde se muestra cada gráfico[7]. No obstante el sistema proporciona medios para controlar este aspecto, permitiendo por ejemplo iniciar (crear) de forma manual estos dispositivos, interactuar con ellos, o incluso dirigir el resultado del gráfico a un fichero.

Los dispositivos gráficos pueden iniciarse utilizando distintas funciones[8]. Dependiendo del sistema operativo podremos iniciar un dispositivo gráfico escribiendo:

Bajo Windows: `windows()`.

[7] En RGui una ventana. En RStudio los gráficos se muestran por defecto en la pestaña "Plots".

[8] Podemos ver el listado completo de estas funciones en la ayuda escribiendo `help(Devices)`.

En Unix o Linux: `X11()`.

En Mac OS X de Apple: `quartz()`.

El tamaño de la ventana gráfica que inician las funciones anteriores es por defecto 7 por 7 pulgadas. No obstante es posible cambiarlo indicando el tamaño deseado en los argumentos `width` y `height`, o bien de forma interactiva con ayuda del ratón.

En lugar de dirigir el gráfico a una ventana dentro de R, podemos almacenarlo (permanentemente) en un fichero. Para los formatos más habituales disponemos de las funciones `pdf()`, `postcript()`, `jpeg()`,`png()`, `bmp()`. En general para abrir este tipo de dispositivos debemos proporcionar un nombre de fichero, si no lo hacemos el sistema asignará uno por defecto. Además es posible especificar varios argumentos para definir el ancho y alto del gráfico, el color de fondo, etc.

Durante una sesión de R podemos tener uno o varios dispositivos gráficos iniciados. A excepción del dispositivo por omisión (dispositivo 1), R administra los dispositivos mediante una lista circular, enumerados a partir del número 2 (se puede consultar escribiendo `dev.list()`). Cuando evaluamos una función que produce un gráfico, este se mostrará en el dispositivo actual (*current*) o activo (su número puede consultarse escribiendo `dev.cur()`). En cualquier momento podemos establecer cuál queremos que sea el dispositivo actual con la función `dev.set()`. Además tenemos que tener en cuenta que cuando se inicia un nuevo dispositivo por defecto se convierte en el dispositivo actual.

A continuación resumimos las funciones más relevantes para interactuar con los dispositivos gráficos en R:

`dev.cur()`: Muestra el dispositivo en uso actualmente (activo).

`dev.list()`: Muestra una lista con todos los dispositivos abiertos.

`dev.next()`: Cambia del dispositivo actual al siguiente.

`dev.prev()`: Cambia del dispositivo actual al anterior.

`dev.set(which=d)`: Establece el dispositivo `d` como el actual.

`dev.copy()`: Copia el contenido del dispositivo actual en otro dispositivo, el cual pasa a ser el nuevo dispositivo actual.

`dev.off()`: Termina (cierra) el dispositivo actual.

`graphics.off()`: Cierra todos los dispositivos abiertos[9].

Para ilustrar el proceso de iniciación de dispositivos gráficos, vamos a realizar un ejemplo en el que se abrirán varios dispositivos en pantalla (con la función `windows()` o la correspondiente para el sistema operativo bajo el que estemos trabajando) así como en un fichero pdf, y en distintos momentos se enviarán gráficos a estos dispositivos. .

[9]Esto se produce de forma automática al finalizar una sesión con R.

```
# Bloque 1: Generación de datos para los gráficos
x<-seq(0,2*pi,length.out=20)
set.seed(2)
y1<-sin(x)+0.3*rnorm(20)
y2<-sin(x)+0.15*rnorm(20)

# Bloque 2: Iniciar un dispositivo gráfico (número 2)
#           y representar (x,y1)
windows()
plot(x,y1,type='o',pch=15,col=2)

# Bloque 3: Iniciar otro dispositivo gráfico (número 3)
#           y representar (x,y2)
windows()
plot(x,y2,type='o',pch=16,col=4)

## Comprueba la lista de dispositivos y el dispositivo actual
dev.list()
dev.cur()

# Bloque 4: Añadir la curva $f(x)=sen(x)$ a ambos gráficos
#           primero en rojo en el dispositivo 2
dev.set(2)
curve(sin(x),add=TRUE,col=2,lty=3)
#           después en azul el dispositivo 3
dev.set(3)  # esto no sería necesario (3 es el dispositivo actual)
curve(sin(x),add=TRUE,col=4,lty=3)

# Bloque 5: Comparar las dos gráficas en un nuevo dispositivo (4)
#           en este caso será un fichero con nombre 'graf.pdf'
pdf('graf.pdf')
plot(x,y1,pch=15,col=2,ylab='')
points(x,y2,pch=16,col=4)
# añadimos segmentos entre los puntos
segments(x,y1,x,y2,col='purple')
# y la curva f(x)=sen(x) de referencia
curve(sin(x),add=TRUE,lty=3)
# cerramos el dispositivo actual (el pdf)
dev.off()
```

Como resultado debes obtener los tres dispositivos gráficos mostrados en la Figura 4.1.

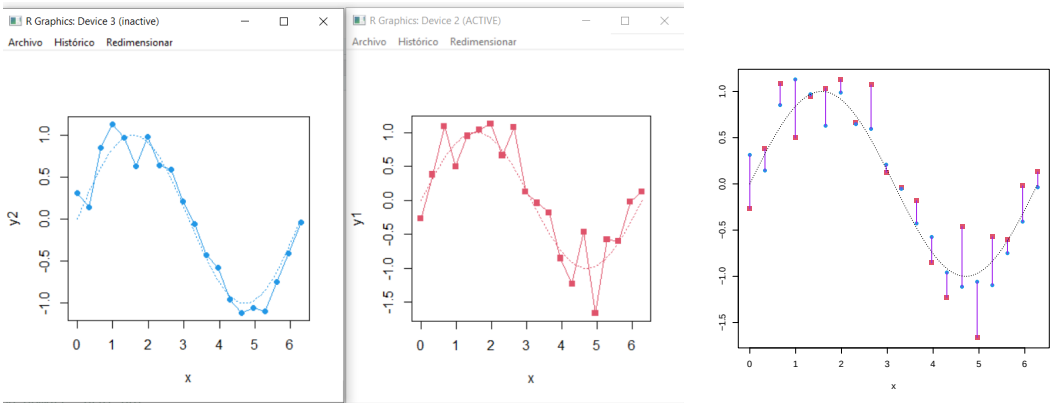


Figura 4.1: Ejemplo: Dispositivos gráficos. De izquierda a derecha: dos creados con `windows()` y el último corresponde al fichero pdf creado con `pdf()`.

4.1.7. Gráficos de variables estadísticas

La clase de gráfico que podemos produce depende en gran medida de la naturaleza de las variables consideradas, además por supuesto de lo que se quiera visualizar con él. En las sesiones de prácticas describiremos casos prácticos motivando e ilustrando los gráficos descriptivos y exploratorios más habituales.

Según el tipo de datos y gráfico la siguiente tabla muestra las funciones básicas correspondientes de R.

`Función`	Descripción
	Datos cualitativos
`barplot`	Diagrama de barras
`pie`	Diagrama de sectores
	Una variable cuantitativa
`hist`	Histograma
`boxplot`	Diagrama de cajas
`qqnorm`, `qqplot`	Gráficos normalidad y cuantil-cuantil
`density`	Densidad suavizada (junto con `plot`)
	Varias variables cuantitativas
`plot`, `pairs`	Diagramas de dispersión
`sunflowerplot`	Medidas repetidas, datos agrupados
`persp`	Gráficos tridimensionales
`contour`	Gráficos de contornos
`image`	Gráficos de calor (*heat map*)

Tabla 4.4: Funciones para la creación de gráficos estadísticos elementales

4.1.8. Paquetes específicos con funciones gráficas

R ofrece en la actualidad varios paquetes para la creación de gráficos avanzados. Entre ellos destacamos *ggplot2* y *lattice*. El paquete *ggplot2*, creado por Hadley Wickham, implementa un sistema coherente para describir y construir gráficos, denominado gramática de gráficos (aprendiendo un único sistema se puede aplicar a diferentes ámbitos). Utilizando la función `ggplot()` se pueden realizar gráficos estadísticos habituales de gran calidad y de una manera sencilla. Deepayan Sarkar es el creador del paquete *lattice* que tiene como finalidad ofrecer mejoras sobre las funciones gráficas en el sistema base de R, así como capacidades adicionales para la visualización de relaciones multivariantes y gráficos condicionales (*trellis graphs*).