# Gráficos con R

Francisco Rodríguez

4 de noviembre de 2018

# Índice general

# 1 Aspectos generales.

R cuenta con un potente conjunto de herramientas y estructuras para la creación de gráficos. Estos gráficos de R se pueden mostrar tanto en modo interactivo como en modo no interactivo, aunque el primero es el más versátil y el que normalmente se va a utilizar, de tal manera que el modo interactivo es el que se activa cuando se comienza R o Rstudio. A pesar de que cuando se inicia Rstudio, por defecto se activa el dispositivo correspondiente para mostrar gráficos, conviene que el lector sepa que este modo se activa mediante la orden X11() , aunque igualmente es válido windows() cuando se está trabajando con Microsoft Windows. Otros dispositivos gráficos también se pueden abrir con una función que depende del tipo de archivo con el que se quiera trabajar: postscript(), pdf(), png(), etc Otros dispositivos gráficos también se pueden abrir con una función que depende del tipo de archivo con el que se quiera trabajar: postscript(), pdf(), png(), etc[1].

Las órdenes para dibujar gráficos en R se pueden dividir en tres grandes grupos:

- **Nivel alto**. En este nivel se crean gráficos, etiquetas, títulos, etc.
- **Nivel Bajo**. Se utilizan para añadir cuestiones complementarias a los gráficos existentes, como pueden ser puntos adicionales, líneas, etc
- **Interactivas**. Sirven para poder interactuar con el gráfico.

En los siguientes apartados se va a exponer más detalles cada uno de los apartados mostrados anteriormente, aunque desde el principio hay que dejar bien claro que todos estos componentes vienen ya con la posibilidad de ser utilizados en el momento de instalar R. Ahora bien, existe en R una librería muy potente en la generación de gráficos que es ampliamente utilizada y que se denomina ggplot2 que se estudiará y expondrá en capítulos posteriores al presente.



## 1.1. Funciones gráficas de alto nivel.

Estas funciones están diseñadas para generar gráficos completos, en base a los datos que se le faciliten en la función empleada. Con estas funciones se generan gráficos con todos sus elementos, con los ejes, títulos, datos etc. y se dibujan en el dispositivo activo, borrando si fuera necesario (y si en los parámetros de la función no se indica lo contrario), el contenido más antiguo. A continuación vamos a ver funciones de este tipo.

[1] Para ver la lista de dispositivos gráficos no hay más que ejecutar en R ?device. Además, la función dev.list() muestra una lista con los dispositivos abiertos y además los números que figuran en esa lista, identifican al dispositivo correspondiente, con el que después puede ser activado, por ejemplo dev.set(3) activa el dispositivo 3. La función dev.off(3) cerrará ese dispositivo.

### 1.1.1. Función plot.

Esta es una de las funciones más utilizadas dentro de R ya que es sumamente versátil, y además el tipo de gráfico que genera se adapta a la clase a la que pertenece el primer argumento (y en ocasiones el segundo). Algunos ejemplos son los siguientes:

- plot(x,y) , si x e y son dos vectores entonces se dibuja un diagrama de dispersión. Si en lugar de pasar dos argumentos, se pasa unos solo en modo de lista con dos elementos o una matriz de dos columnas, el resultado sería el mismo.
- plot(x) , si x es un vector entonces se representan los datos de tal manera que sobre el eje de las x se representa el índice de los mismos. Si x es una serie temporal, se representa como tal serie y por último, si x es un número complejo, se genera un gráfico de la parte real sobre la imaginaria.
- plot(f) ó plot(f,x), con f un factor y x un vector numérico, con la primera forma se representa se genera un diagrama de barras de los niveles de f, y con la segunda un diagrama de cajas de y, para cada nivel de f.
- plot(y~expr), con y un vector de datos numéricos y expr una lista de nombres de símbolos separados por «+» como por ejemplo a+b+c. En este caso se genera un gráfico sobe cada par de cada elemento de «expr» y cada valor de y.

Vamos a trabajar para mostrar los ejemplos con el famoso dataset de R denominado «iris». Veamos algunos datos del mismo.

```
head(iris,3)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

Veamos cómo dibujar un simple scaterplot o diagrama de dispersión ( ver figura 1.1 ).

```
plot(iris$Sepal.Length,iris$Sepal.Width)
```

Se puede dar estilo a la salida de la siguiente manera ( ver figura 1.2).

```
plot(iris$Sepal.Length,iris$Sepal.Width,pch=19,col="gray",cex=2,xlab="Sepal Length",ylab="Sepal Width")
```

Se pueden añadir puntos y líneas ( ver figura 1.3):

```
plot(iris$Sepal.Length,iris$Sepal.Width,pch=19,col="gray",cex=2,
        xlab="Length",ylab="Width",ylim=c(0,5),xlim=c(0,8))
#Se añaden puntos
points(iris$Petal.Length,iris$Petal.Width,pch=19,col="red",cex=2)
# Se añaden lineass
lines(0:5,0:5,col="blue",lwd=2,lty=2)
```

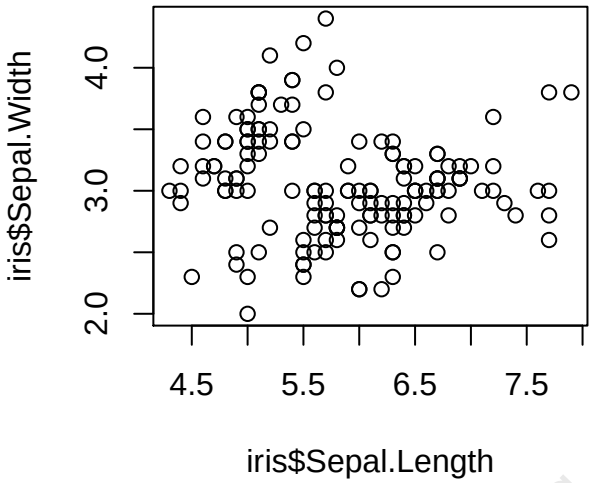


Figura 1.1: Ejemplo de diagrama de dispersión

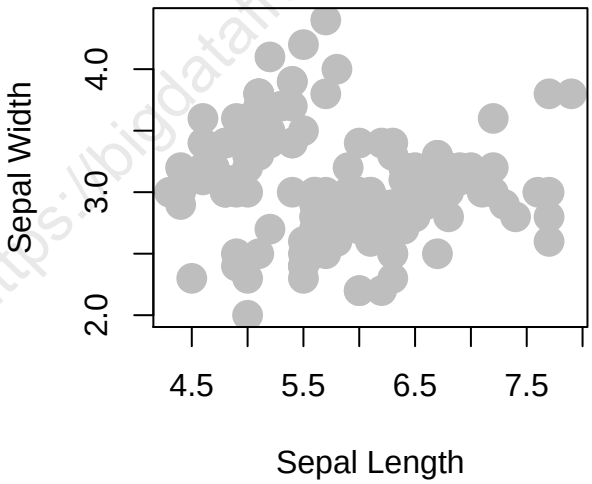


Figura 1.2: Dando cierto estilo

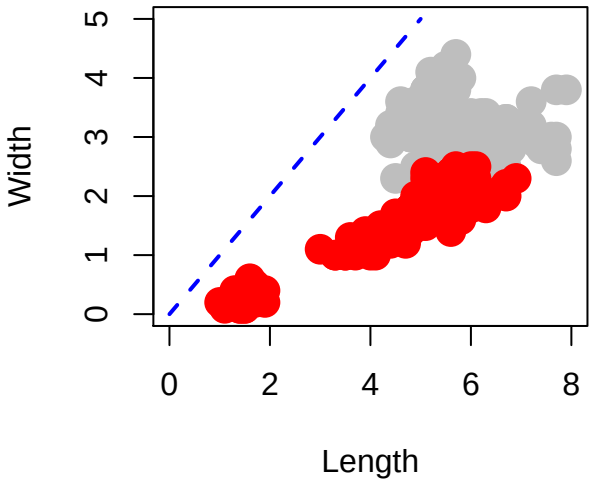


Figura 1.3: Añadiendo puntos y líneas

### 1.1.2. Funciones para datos multivariantes.

En el supuesto de que estemos trabajando con más de dos variables, es decir en los casos más normales, se puede hacer de forma y en un sólo gráfico, una representación de todo el conjunto de datos con el que se trabaje, agrupando de dos en dos las variables. Esto se consigue con la función

- pairs(X)

Siendo X una matriz o dataframe numérico. En este gráfico se genera un diagrama de dispersión por cada pareja de variables numéricas presente en X , en la figura 1.4 podemos ver un ejemplo de este tipo de gráficos.

```
pairs(iris[,1:4])
```

Otra función muy interesante para representar datos, sobre todo cuando se trabaja con tres o más variables, es la función coplot(), que presenta el siguiente formato.

- coplot(a ~ b | c)

Siendo a y b vectores numéricos y b otro vector numérico o un factor. Con esa función se genera un diagrama de dispersión de a sobre b para cada valor de c. En el caso de cuatro variables, se pueden generar diagramas de dispersión para cada cruce de dos variables condicionantes, el formato sería el siguiente:

- coplot(a ~ b | c+d)

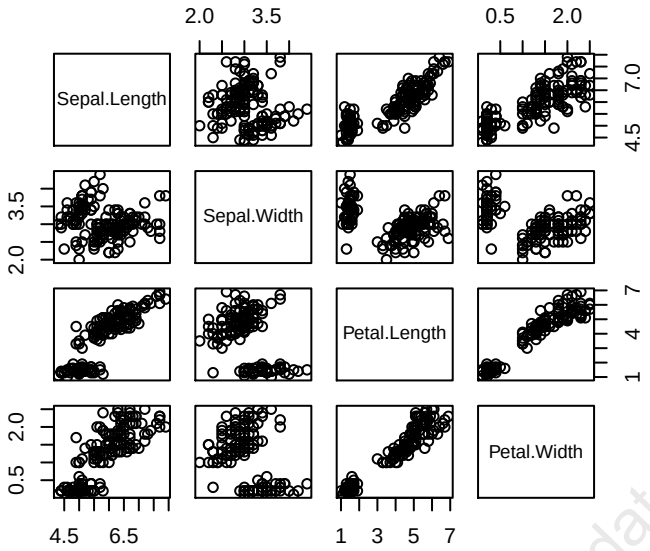


Figura 1.4: Representación por pares de variables

### 1.1.3. Otras funciones gráficas de alto nivel

Otras funciones gráficas de alto nivel pueden ser las siguientes:

- qqnorm(x); qqline(x); qqplot(x,y), estos gráficos son útiles para comparación de distribuciones. Con el primero se representan los cuartiles de la variable x sobre los esperados, en el supuesto que se de normalidad. La segunda función representa una recta que pasa por los cuartiles de la distribución de los datos. La tercera función representa los cuartiles de x sobre los de y para comparar ambas distribuciones.

- hist(x) , Genera un histograma con los datos de la variable numérica x. El número de clases, se puede elegir con el argumento «nclass=n». También se pueden elegir los puntos de corte con el argumento «breaks». Si se indica el argumento «probability=TRUE», se obtendrán frecuencias relativas, en lugar de absolutas (valor por defecto).

- dotchart(x,...), con esta función, en el eje y están las etiquetas de los datos y en el eje x los valores de la variable, un ejemplo de este tipo de gráficos se puede ver en la figura 1.5

- image(x,y,z,...); contour(x,y,z,...); persp(x,y,z,...); Todas estas funciones sirven para hacer gráficos tridimensionales, con image se representa una retícula de rectángulos con colores diferentes, dependiendo del valor de la variable z. Contour sirve para

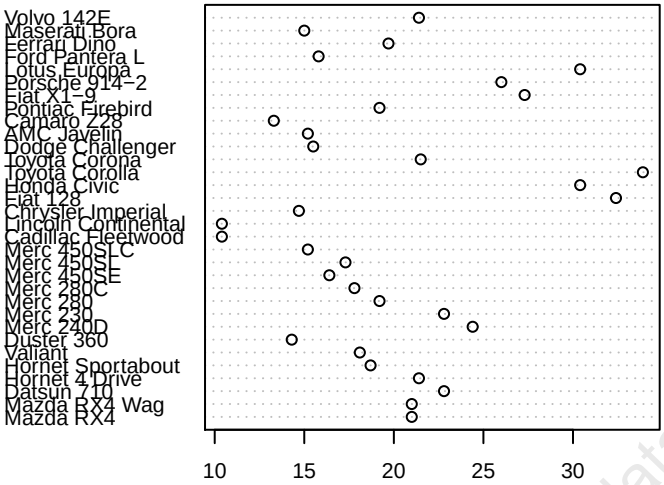


Figura 1.5: Ejemplo de Función dotchart()

representar curvas de nivel de la variable z, y persp para representar una superficie tridimensional (ver figura **??**).

- piechart(x,...); Genera un gráfico de tipo pie, coloquialmente como de tipo pastel, es decir con forma circular y ángulo de los sectores proporcional a la cantidad de la magnitud que representa.
- boxplot(x,...); Gráfico de tipo boxplot, para la variable que figura como parámetro.
- interaction.plot(f1, f2, y); Este tipo de gráficos es una representación visual de la interacción entre dos efectos o factores (f1 y f2) respecto de la variable independiente y. Por defecto lo que hace es poner en la gráfica el promedio de y con respecto a los valores de f1 ( colocados en eje 1) y de f2. El valor del promedio se puede cambiar, utilizando para ello otra función que se declara con el parámetro «fun=».
- marplot(x,y); es un gráfico bivariado de la primera columna de x vs. la primera columna de y, la segunda columna de x vs. la segunda columna de y, etc.
- symbols(x,y,...); Dibuja en las coordenadas dadas por x e y símbolos, tales como círculos, cuadrados, estrellas, etc..

```
dotchart(mtcars$mpg,labels = row.names(mtcars),cex=0.7)
```

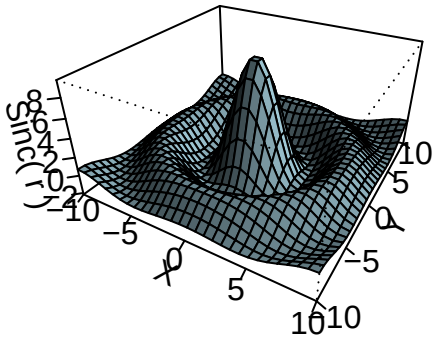


Figura 1.6: Figura tridimensional con r

```
x <- seq(-10, 10, length= 30)
y <- x
f <- function(x, y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z <- outer(x, y, f)
z[is.na(z)] <- 1
op <- par(bg = "white")
#persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue",
      ltheta = 120, shade = 0.75, ticktype = "detailed",
      xlab = "X", ylab = "Y", zlab = "Sinc( r )"
) -> res
```

```
round(res, 3)

##       [,1]   [,2]   [,3]   [,4]
## [1,] 0.087 -0.025  0.043 -0.043
## [2,] 0.050  0.043 -0.075  0.075
## [3,] 0.000  0.074  0.042 -0.042
## [4,] 0.000 -0.273 -2.890  3.890
```

### 1.1.4. Principales argumentos comunes de las funciones de alto nivel.

En todos los gráficos de alto nivel, existen una serie de argumentos que son comunes a todas ellas y que permiten modificar el comportamiento predeterminado del gráfico. A continuación veremos los más usuales:

- add=TRUE; Con esta opción y solo para algunas funciones, lo que se hace es mantener al gráfico anterior y sobre él añadir el que se genere en ese momento.
- axes=FALSE; Con esto se indique se suprima la generación de los ejes. Este parámetro es útil si se quieren crear ejes personalizados mediante el uso de la función «axis» ( ver este enlace o también ver estos ejemplos).
- log="x", log="y", log="xy"; para indicar gráficas en escala logarítmica.
- type= ; controla el tipo de gráfico que se generará:
  - type="p" para dibujar puntos individuales (es el valor predeterminado)
  - type="l" para dibujar líneas
  - type="b" para dibujar puntos y líneas que los unen
  - type="h" para dibujar líneas verticales desde cada punto al eje de las X
  - type="s" ó type="S" para dibujar un gráfico de escalera. Con la primera forma la escalera comienza hacia la derecha, y con la segunda hacia arriba.
  - type="n" en este caso no se dibuja el gráfico, y si no se indica lo contrario sí salen los ejes. Suele utilizarse este método si se quiere construir el gráfico con instrucciones de bajo nivel
- xlab, ylab ; se utiliza para definir las etiquetas que figurarán en los ejes x e y respectivamente.
- main=string; para definir los títulos de los gráficos.
- sub=string, Para definir el subtítulo del gráfico

## 1.2. Funciones gráficas de nivel bajo.

Estas funciones permiten afinar las representaciones gráficas que se hacen en R, permitiendo de esta manera mejorar la presentación de los gráficos. Veamos a continuación algunas funciones de bajo nivel muy utilizadas en las representaciones gráficas de R.

- points(x,y); lines(x,y); permiten insertar puntos o líneas al gráfico actual.
- test(x,y, etiquetas...); Para añadir texto al gráfico en las coordenadas definidas por (x,y). Con carácter general el valor de etiquetas[i] se coloca en el lugar indicado por el par (x[i],y[i]).
- mtext(...); Sirve para escribir texto en uno de los cuatro márgenes del gráfico.
- abline(a,b); abline(h=y); abline(v=x); abline(lm.objeto). Dibuja líneas en el gráfico activo, con la primera forma dibuja una línea de pendiente b y ordenada en el origen a. Con la segunda forma dibuja una línea horizontal que pasa por el punto «y». Con la tercera forma se dibuja una línea vertical que pasa por el punto x ; y con la última forma se dibuja la recta que genera un modelo de regresión lineal.

- polygon(x,y,...); para añadir un polígono al gráfico actual, con vértices en los puntos definidos por los vectores numéricos x e y
- legend( ); para dibujar la leyenda del gráfico. Para entender bien esta función visitar este enlace ( en inglés). Para ver otro tutorial en castellano click en este enlace (pag. 17)
- title(main, sub); Para añadir título o subtítulo en un gráfico.
- axis( ...); Una función muy útil para configurar a medida los ejes. Lo ejes se identifican del 1 al 4, siendo 1 el de la parte inferior y el resto en el sentido de las agujas del reloj.
- segments(x0, y0, x1, y1); dibuja una línea desde el punto (x0,y0) al punto x1,y1).
- arrows(...); permite dibujar flechas en el gráfico.
- locator(...); devuelve las coordenadas (x,y) después que el usuario ha hecho click n veces sobre el gráfico.

### 1.2.1. Añadiendo expresiones matemáticas en los gráficos de R.

Mediante la función expresión es posible añadir fórmulas matemáticas dentro de los gráficos generados por R. Este función expression, se puede insertar en cualquiera de las funciones: text, mtext, axis o title. Un ejemplo puede ser el mostrado en la figura 1.7

```
par(mar = c(4, 4, 2, 0.1))
plot(rnorm(100), rnorm(100),
  xlab = expression(hat(mu)[0]), ylab = expression(alpha^beta),
  main = expression(paste("Plot of ", alpha^beta, " versus ", hat(mu)[0])))
```

Para incluir una variable es una expresión se pueden utilizar las funciones substitute y as.expression, su uso se muestra en las figura 1.8

```
par(mar = c(4, 4, 2, 0.1))
plot(rnorm(100), rnorm(100),
  xlab = expression(hat(mu)[0]), ylab = expression(alpha^beta),
  main = expression(paste("Plot of ", alpha^beta, " versus ", hat(mu)[0])))
```

Por último para ver ayuda de esta características, se pueden utilizar los siguientes comandos:

- help(plotmath)
- example(plotmath)
- demo(plotmath)

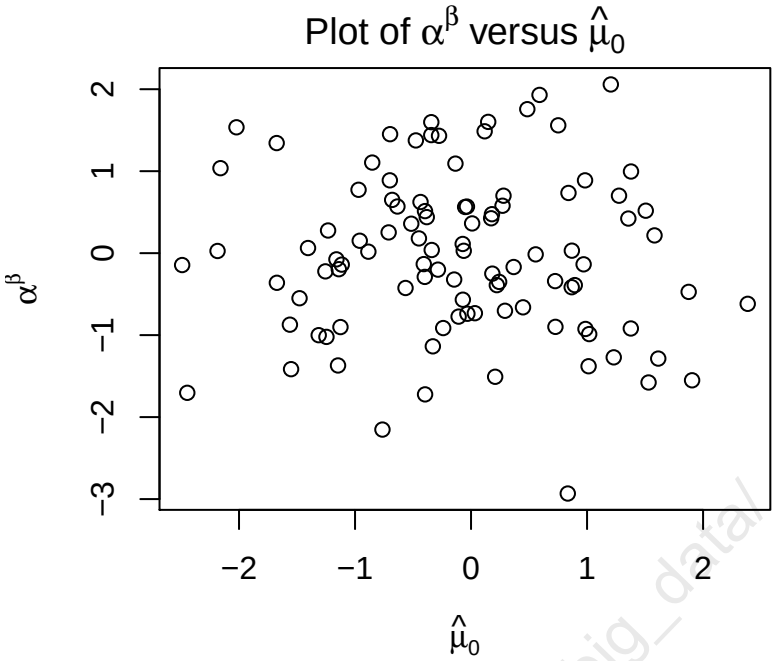


Figura 1.7: Agregando fórmulas matemáticas

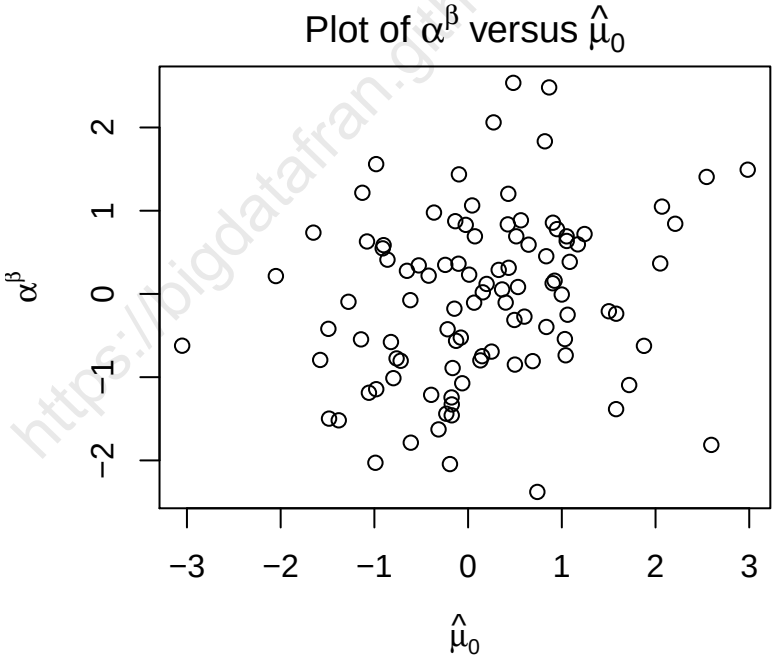


Figura 1.8: Utilizando variables en la fórmula

## 1.3. Funciones gráficas interactivas.

En ocasiones, cuando creamos un gráfico con R, resulta muy interesante poder interactuar con el gráfico. Para conseguir este objetivo, R nos facilita una serie de funciones que nos facilitan esta labor.

- locator(n, type); Con esta función el usuario puede seleccionar posiciones del gráfico, bien hasta n ( el valor predeterminado es n=512) o bien hasta que se haga click con el botón secundario del ratón.
- identify(x,y,etiquetas); sirve para poder identificar cualquiera de los puntos definidos por x e y . Al pulsar el botón secundario del ratón , se devuelven los índices de los puntos seleccionados.

## 1.4. Parámetros gráficos.

Todos los gráficos de R admiten una serie de parámetros que permiten optimizar las presentaciones de los mismos. Los valores de estos parámetros, se pueden modificar de forma temporal ( cuando se les añade en la generación del gráfico ) o de forma permanente, mientras se siga con la sesión ya comenzada.

### 1.4.1. Cambios permanentes.

La función par() es muy útil en R, ya que con ella se pueden acceder a la lista de parámetros gráficos del dispositivo gráfico con el que se trabaja, y además permite modificarlos de forma permanente, mientras dure la sesión. Los parámetros gráficos que se pueden modificar los puedes encontrar este enlace.

Si se quieren ver en la interface de R todos los parámetros gráficos a los que se tiene acceso, no hay más que escribir la instrucción «par()» ( es decir la función par sin argumentos ).

Los valores de los parámetros gráficos se pueden almacenar en variables, de tal manera que si para un determinado gráfico o conjunto de gráficos, se necesitan ciertos valores, y luego se quiere volver sobre los valores anteriores, esto se puede hacer siguiendo el esquema que a continuación se muestra:

- par.anterior<- par(col=2, lty=1)

...ejecución de órdenes para dibujar gráficos ...

- par(par.anterior) (con esta instrucción se volvería a la situación anterior)

Cada vez que se cambia el entorno gráfico con el que se está trabajando, se vuelven a los valores gráficos por defecto. Por lo tanto si ejecutamos la instrucción «dev.off()», entonces se volverían a los parámetros gráficos que R tiene por defecto. Otra forma de conseguirlo, es como se muestra en el ejemplo siguiente:

```
p<-par() #Guardo todos los valores por defecto
par(mfrow=c(2,2))
print(par("mfrow"))
```

```
## [1] 2 2

par(p) #Vuelvo a los valores iniciales

## Warning in par(p): el parámetro del gráfico "cin" no puede ser especificado
## Warning in par(p): el parámetro del gráfico "cra" no puede ser especificado
## Warning in par(p): el parámetro del gráfico "csi" no puede ser especificado
## Warning in par(p): el parámetro del gráfico "cxy" no puede ser especificado
## Warning in par(p): el parámetro del gráfico "din" no puede ser especificado
## Warning in par(p): el parámetro del gráfico "page" no puede ser especificado

print(par("mfrow"))

## [1] 1 1
```

Además de todo lo anterior el paquete settings contiene la función «reset_par» que permite pasar a los valores iniciales de los parámetros gráficos, de un forma rápida y sencilla.

### 1.4.2. Cambio temporales de los parámetros gráficos.

Estos cambios temporales se consiguen utilizando dichos parámetros gráficos, como parámetros de las funciones que permiten hacer las representaciones gráficas. A continuación se muestran los parámetros más usuales.

- **adj**. Se utiliza para la justificación del texto, 0 indica justificado izquierda, 0.5 justificado al centro y 1 justificado a la derecha.
- **bg**. Se utiliza para definir el color del fondo. Hay 657 colores disponibles y la lista se puede ver con la instrucción «colors()», o mejor en este enlace.
- **bty**. Controla el tipo de caja que se dibuja alrededor del gráfico.
- **cex**. Para indicar el tamaño de texto, respecto de un valor que es el usado por defecto. También existe cex.main, cex.axis, cex.lab y cex.sub
- **col**. Para controlar el color de los símbolos. Como en el caso anterior, también existe col.main, col.axis, col.lab y col.sub
- **font**. Toma como valor un entero que indica el estilo del texto ( 1: normal, 2: cursiva, 3: negrilla, 4: egrilla y cursiva ). También existe font.main, font.axis, font.lab y font.sub
- **las**. Un entero que controla la orientación del texto en los ejes (0:paralelo a los ejes, 1: horizontal, 2: perpendicular a los ejes 3:vertical)[2].
- **lty**. Un entero que controla el tipo de líneas. 1: sólida, 2: quebrada, 3:punteada, 4:punto-línea, 5: línea larga-corta, 6: dos líneas cortas. Existe también la opción

[2]Existe la posibilidad de dar un ángulo distinto de 90 grados a la disposición del texto, esto se hace con la función «axis()»

de especificar otro tipo de línea, mediante una secuencia de hasta 8 números que especifican de forma alternada la longitud en pieles de los elementos dibujados..

- **lwd**. Un número que controla la anchura de las líneas dibujadas.
- **mar**. Sirve para controlar los márgenes entre los ejes y el borde de la gráfica. El formato es c(inferior, izquierda, superior, derecho)
- **mfcol y mfrow**. Para dividir la zona gráfica en una matriz c(n1,n2) con n1 filas y n2 columnas.
- **mfg=c(a,b,c,d)**; con a,b,c,d números enteros. Define la posición de la figura actual dentro de la matriz de figuras múltiples. Los dos primeros valores indican la fila y la columna de la figura actual, mientras que los dos últimos son el número de filas y columnas de la matriz de figuras múltiples. Incluso los dos último valores pueden ser distintos de los verdaderos valores, con el fin de obtener figuras de tamaños distintos en la misma página.
- **fig=c(4,9,1,4)/10**; Sirve para colocar la figura en un determinado lugar de la página. Los valores dados son las posiciones de los bordes izquierdo, derecho, inferior y superior respectivamente, medidos como la proporción de página desde la esquina inferior izquierda.
- **pch**. Para controlar el tipo de símbolos. Puede ser un entero entre 1 y 25 o bien un carácter encerrado entre comillas. ( ver figura 1.9 )
- **ps**. Un entero que controla el tamaño ( en puntos) de textos y símbolos.
- **tck**. Un valor que especifica la longitud de los marcadores de ejes como una fracción de la altura o anchura máxima del gráfico, y por lo tanto si tck=1 se dibuja una rejilla.
- **xaxt**. Si se indica xaxt="n" el eje x se dibuja pero no se muestra. Esto se suele utilizar para definir una disposición particular del eje x con axis(side=1, ...)
- **yaxt**. Si se indica yaxt="n" el eje y se dibuja pero no se muestra. Esto se suele utilizar para definir una disposición particular del eje y con axis(side=2, ...)



```r
# Script para dibujar todos los símbolos con pch
generateRPointShapes<-function(){
  oldPar<-par()
  par(font=2, mar=c(0.5,0,0,0))
  y=rev(c(rep(1,6),rep(2,5), rep(3,5), rep(4,5), rep(5,5)))
  x=c(rep(1:5,5),6)
  plot(x, y, pch = 0:25, cex=1.5, ylim=c(1,5.5), xlim=c(1,6.5),
       axes=FALSE, xlab="", ylab="", bg="blue")
  text(x, y, labels=0:25, pos=3)
  par(mar=oldPar$mar,font=oldPar$font )
}
generateRPointShapes()
```

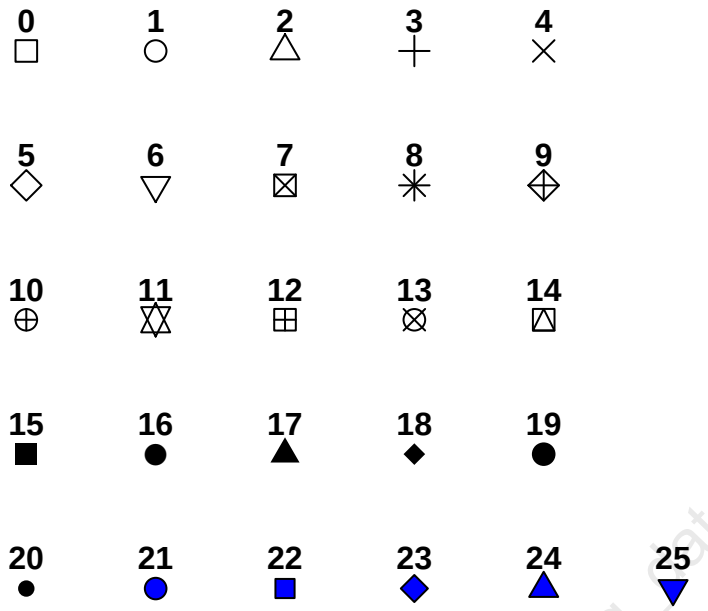


Figura 1.9: Símbolos con pch

Veamos a continuación algunos ejemplos sobre el uso de las funciones de bajo nivel. Primero hacemos una representación simple en la figura 1.10.

```
x<-1:10
y<-x*x*x
plot(x,y,type='b', main = 'Gráfica de la función',
     xlab="Eje x",ylab = "Eje y")
mtext("Eje derecho",side = 4)
```

En la figura 1.10 podemos observar que con mtext() hemos puesto un texto en el eje derecho del gráficos, con main se ha puesto un título y con xlab e ylab un título a cada eje. Imaginemos que queremos rotar las etiquetas de los ejes. Para ello existe un parámetro gráfico que se denomina las que dependiendo del valor que tome coloca la etiqueta de la siguiente forma:

- las=0; Siempre paralelos en la dirección del eje (valor por defecto)
- las=1; siempre horizontal
- las=2; siempre perpendicular a los ejes
- las=3; siempre vertical.

Ahora bien, ¿ cómo se puede conseguir que el ángulo sea diferente a los 90 grados y además se pueda actuar de forma individual en cada eje?. La respuesta no es tan sencilla

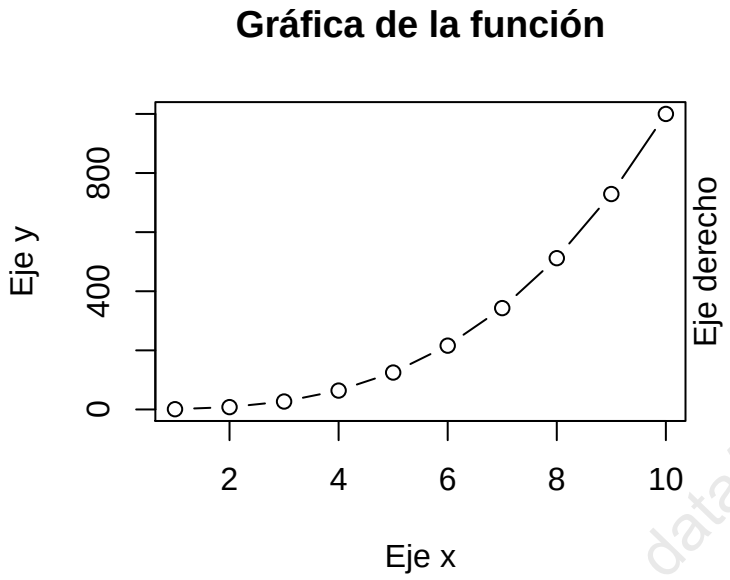


Figura 1.10: Funciones gráficas de bajo nivel



como utilizar un determinado parámetro. A continuación se va a explicar cómo conseguir esto.

Vamos a rotar primero las etiquetas del eje de las x, para ello lo que hacemos es en la función plot añadimos el parámetro «xaxt='n'» para dibujar el eje x sin más. A continuación añadimos la función axis con el siguiente contenido:

```
axis(side = 1,at=seq(1,10,by=1), labels = FALSE)
```

Con lo anterior lo que indicamos es que en el eje de las x se añadan las 10 marcas que definen los puntos de referencia del eje. Ahora, para poner las etiquetas a ese eje utilizaremos la función text, de la siguiente manera:

```
text(seq(1,10,by=1),par("usr")[3]-0.2,labels = c(1:10),srt=-45,pos = 1,
xpd=TRUE)
```

Veamos el significado del script anterior. El primer parámetro, indica .as coordenadas x donde se escribirá. El segundo parámetro está basado en el parámetro gráfico «usr» que hace referencia a un vector de la forma c(x1, x2, y1, y2) que contiene los extremos de la región donde se coloca el gráfico. En este caso se cogerá la coordenada tercera, que indica la zona de abajo del gráfico. Con el parámetro «labels» indicamos las etiquetas que queremos colocar. Con el parámetro «srt=-45» **indicamos el giro** que queremos dar a las etiquetas. Con «pos=1» se indica que la etiqueta se coloque en la zona de abajo. Y por último, con «xpd=TRUE», se indica que se puede dibujar fuera de la zona del gráfico ( si no ponemos esta condición, en el gráfico no se verían estas etiquetas). Una vez hechos estos cambios, el resultado se muestra en la figura 1.11

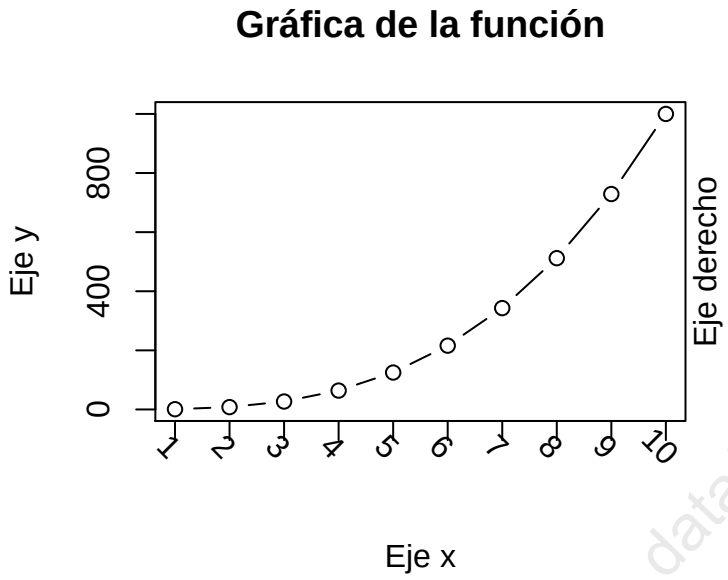


Figura 1.11: Figura con etiquetas eje x rotadas 45 grados

```
x<-1:10
y<-x*x*x
plot(x,y,type='b', main = 'Gráfica de la función',
     xlab="Eje x",ylab = "Eje y", xaxt='n')
mtext("Eje derecho",side = 4)
axis(side = 1,at=seq(1,10,by=1), labels = FALSE)
text(seq(1,10,by=1),par("usr")[3]-0.2,labels = c(1:10),srt=-45,pos = 1,xpd=TRUE)
```

Ahora también modificamos la estética del eje y para poner unas etiquetas diferentes a las mostradas por defecto, el resultado se muestra en la figura 1.12.

```
x<-1:10
y<-x*x*x
plot(x,y,type='b', main = 'Gráfica de la función',
     xlab="Eje x",ylab = "Eje y", xaxt='n',yaxt='n',ylim = c(1,1010))
mtext("Eje derecho",side = 4)
axis(side = 1,at=seq(1,10,by=1), labels = FALSE)
text(seq(1,10,by=1),par("usr")[3]-0.2,labels = c(1:10),srt=-45,pos = 1,xpd=TRUE)
axis(side=2,at=seq(1,1010,by =500 ),labels = FALSE)
text(y=seq(1,1010,by=500),par("usr")[1],labels = c("uno","quinientos","mil"),
     srt=45,pos=2,xpd=TRUE)
```

## 1.5. Gráficos dinámicos

Para crear gráficos dinámicos, existe la posibilidad de utilizar el paquete xgobi que utiliza las posibilidades que nos ofrece la posibilidad desde R de ejecutar las posibilidades

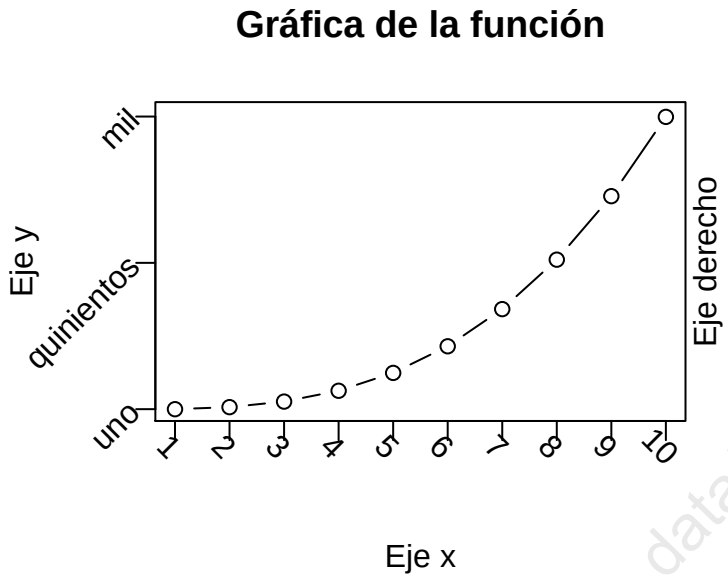


Figura 1.12: Figura dando un ángulo a las etiquetas de los dos ejes

que nos ofrece el sistema XGobi. Otra posibilidad que existe para visualizar de forma dinámica datos en tres dimensiones es el paquete «scatterplot3d» . Igualmente se pueden encontrar otros paquetes con esta finalidad:

- **dygraphs**. Es un paquete de R que se basa en el paquete de javascript muy útil para la representación de series temporales.
- **plotly**. Es otro paquete libre muy útil para la realización de gráficos interactivos y estéticos.
- **animation**. Este paquete tiene muchas clases muy útliles con animaciones prefijadas. También tiene la clase «saveGif», que mediante una serie de imágenes gif las va mostrando una tras otra para conseguir una animación, para conseguir eso utiliza ImageMagick or GraphicsMagick.



## 1.6. Ejemplos de gráficos.

Como complemento de todo lo dicho hasta ahora, a continuación se van a mostrar algunos códigos que nos permiten obtener gráficos con el paquete básico de R. Para construir estos ejemplos se utilizarán dataset's de R contenidos en algunos paquetes que se deberán cargar cuando sea necesario.

### 1.6.1. Variables cualitativas.

Comenzamos por un plot de variables cualitativas, el resultado se muestra en la figura 1.13

```
if(!require("kernlab")) install.packages("kernlab"); require("kernlab")

## Loading required package: kernlab

#Tomamos el data set spam. Para ver contenido ejecutar ?spam como comando R
#Son 4601 e-mails clasificados como spam o noSpam
#Cargamos datos en memoria
data(spam)
#Seperamos variables
#Primero las variables independientes
spam.X <- spam[,1:(ncol(spam)-1)]
#Ahora la variable dpendiente
spam.Y <- spam[,ncol(spam)]
#Obtenemos una distribución de valores absolutos
abs.freq.spam.Y <- table(spam.Y)

# Primero indicamos que las gráficas van en una matriz gráfica 1x2
par(mfrow=c(1,2))

# Creamos un diagrama de barras y un pichart con las frecuencias anteriores
# Primero diagrama de barras
barplot(abs.freq.spam.Y,col="deepskyblue4",main="Barplot de spam",
        ylab="Frecuencias absolutas",xlab="Valores Spam")
#Ahora un pirchart
pie(abs.freq.spam.Y,col=c("deepskyblue4","firebrick4"),
        main="Piechart de spam",radius=1)
```

```
#Borro los objetos de memoria
#rm(spam.X,spam.Y,spam)
```

En la figura 1.14 se crea un plot (que es un un barplot)con los datos de otro dataset del paquete «nutshell»

```
if(!require("nutshell")) install.packages("nutshell"); require("nutshell")

## Loading required package: nutshell
## Loading required package: nutshell.bbdb
## Loading required package: nutshell.audioscrobbler

#contiene datos de nacimientos durant 2006
#Tomamos el data set births2006.smpl. Para ver contenido ejecutar ?births2006.smpl como comando R
#Cargamos datos en memoria
data(births2006.smpl)
#Obtenemos las frecuencias absolutas

# Barplot para las variables SEX y DMEDUC

# Dividimos gráfico en 1x1

par(mfrow=c(1,1))

plot(births2006.smpl$DMEDUC,births2006.smpl$SEX,
        xlab="DMEDUC",ylab="SEX",col=c("deepskyblue4","firebrick4"))
```

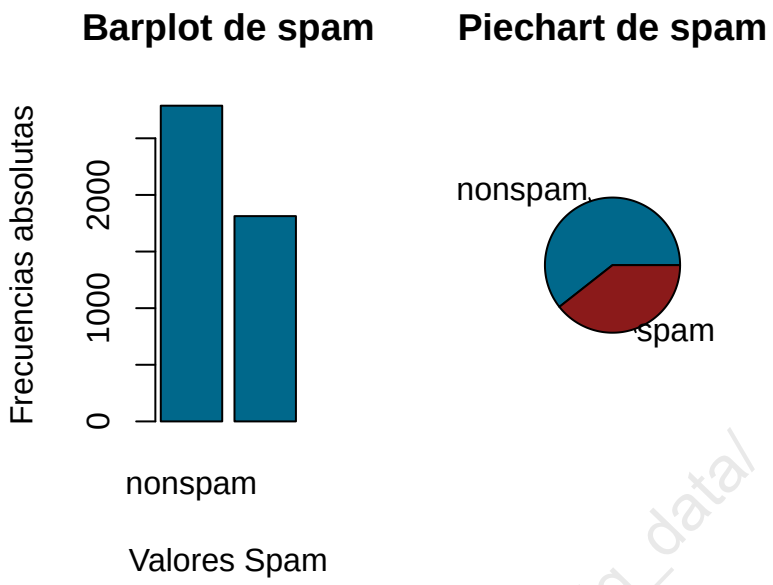


Figura 1.13: Diagrama de barras y Piechart

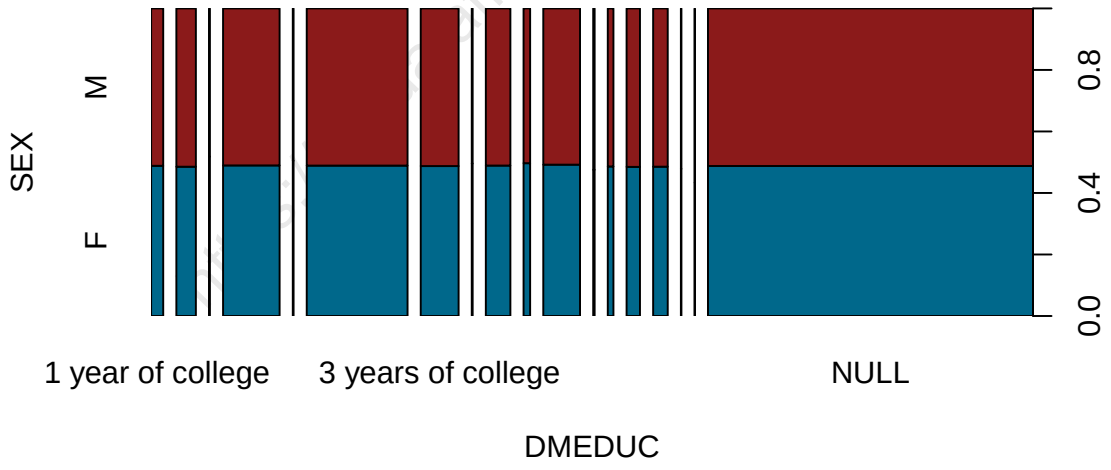


Figura 1.14: Plot de births2006.smpl

Observar en este caso que al representar dos variables de tipo factor, lo que se representa realmente es un diagrama de barras, todas con la misma altura total, pero en cada categoría de la variable y se representa el porcentaje que hay de varones frente a mujeres.

Otros gráficos de tipo barplot muy utilizados son los denominados **Stacked Bar Plot** , que son gráficos de barras pero con datos agrupados, en la figura 1.15 se muestra un ejemplo de los dos tipos de gráficos que se pueden dar de este tipo.

```
#Primero cruzo los datos
tabla<-table(births2006.smpl$SEX,births2006.smpl$DMEDUC)
par(mfrow=c(2,1))
barplot(tabla,col=c("deepskyblue4","firebrick4"),legend = rownames(tabla),
       args.legend = list(x=4,y=200000), main="Dato apilados " )
barplot(tabla,col=c("deepskyblue4","firebrick4"),legend = rownames(tabla), beside=TRUE,
        args.legend = list(x=7,y=80000), main="Datos agrupados")
```

### 1.6.2. Variables cuantitativas.

En esta sección vamos a mostrar algunos ejemplos interesantes que pueden ser utilizados con variables de tipo cuantitativo. En primer lugar vamos a ver un gráfico que nos permite dar una idea aproximada de la distribución de los datos. Este gráfico es el denominado **Boxplot** o en castellano «diagrama de caja» y su interpretación es la siguiente: la parte baja de la caja coincide con el primer quartil, la parte alta con el cuartil 3, la raya del medio de la caja es la mediana y «los bigotes» que se extienden por arriba y por abajo se extienden hasta 1,5 veces el rango intercuartílico (Q3-Q1) . A continuación se muestra un ejemplo para ver cómo calcularlo, para un nuevo dataset, en este caso el denominado NCI60 que viene dentro de la librería «ISLR». Este dataset se utiliza para identificar casos de cáncer con la la ayuda de 6830 variables. La representación de los datos para diferentes variables, lo vemos en la figura 1.16.

```
if(!require("ISLR")) install.packages("ISLR"); require("ISLR")

## Loading required package: ISLR

data(NCI60)
# Boxplot de las primera 10 variables de NCI60 data set
# Definimos el tipo de salida de datos
par(mfrow=c(2,5))
#Dibujamos Boxplot para 10 primeras variables con ayuda de función sapply
sapply(seq(1,10),function(j)boxplot(NCI60$data[,j],main=paste("Var.",colnames(NCI60$data)[j]),
xlab="",col="deepskyblue4"))
```

```
##       [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## stats Numeric,5 Numeric,5 Numeric,5 Numeric,5 Numeric,5 Numeric,5
## n     64        64        64        64        64        64
## conf  Numeric,2 Numeric,2 Numeric,2 Numeric,2 Numeric,2 Numeric,2
## out   Numeric,0 Numeric,3 Numeric,4 Numeric,0 Numeric,3 Numeric,3
## group Numeric,0 Numeric,3 Numeric,4 Numeric,0 Numeric,3 Numeric,3
## names ""        ""        ""        ""        ""        ""
##       [,7]      [,8]      [,9]      [,10]
## stats Numeric,5 Numeric,5 Numeric,5 Numeric,5
## n     64        64        64        64
```

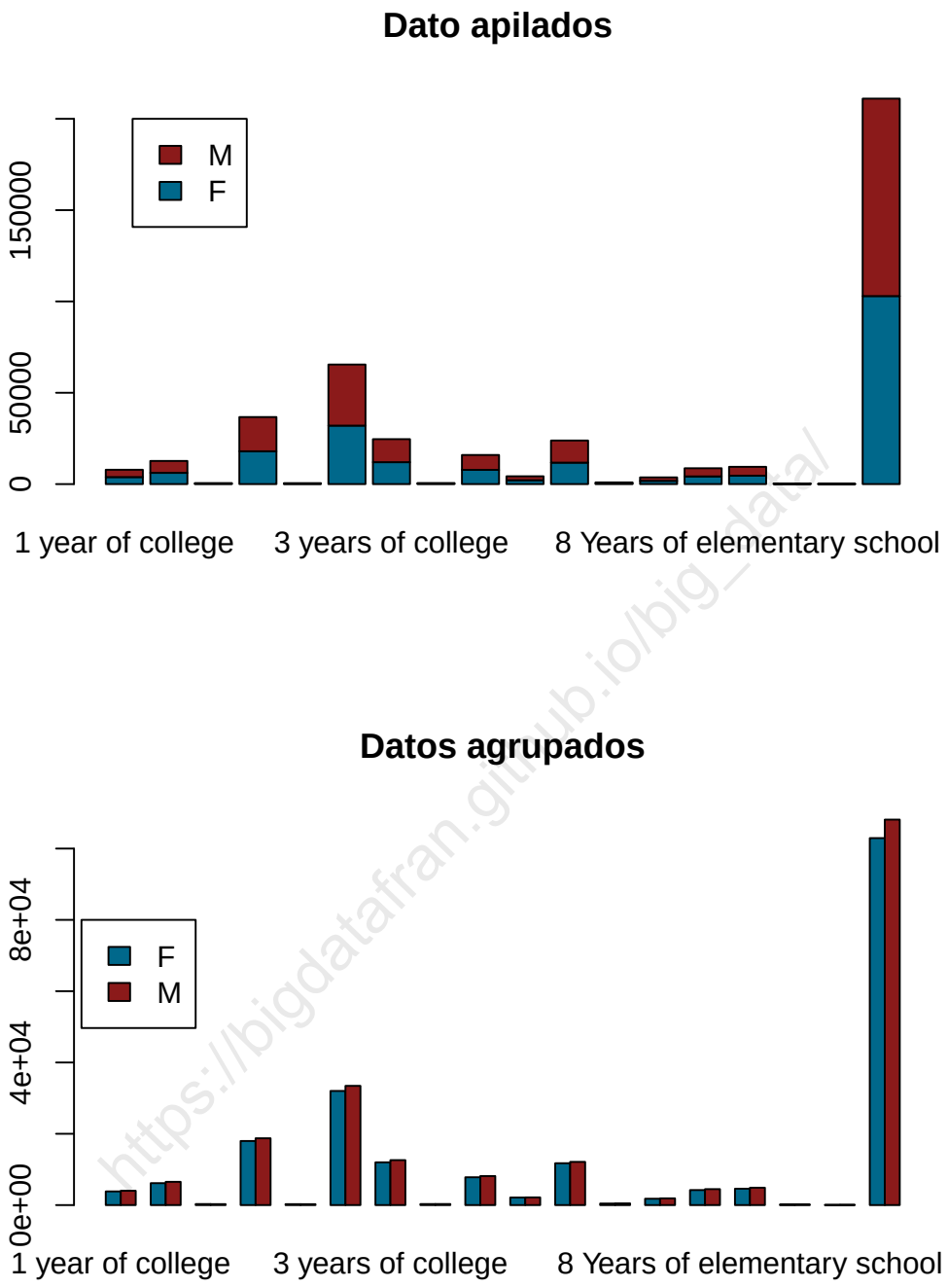




Figura 1.15: Stacked Bar Plot

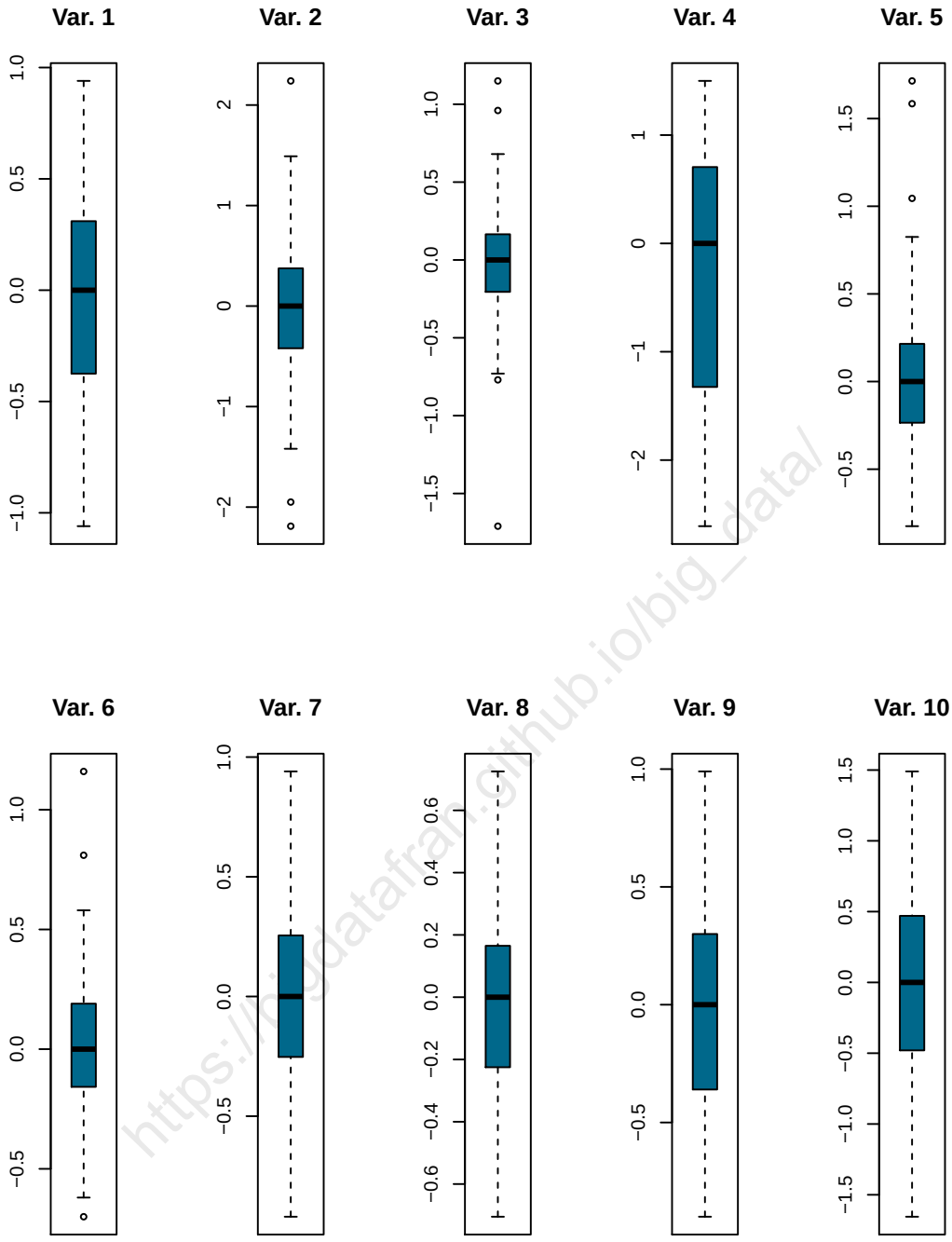




Figura 1.16: Diagrámas de caja, Boxplot

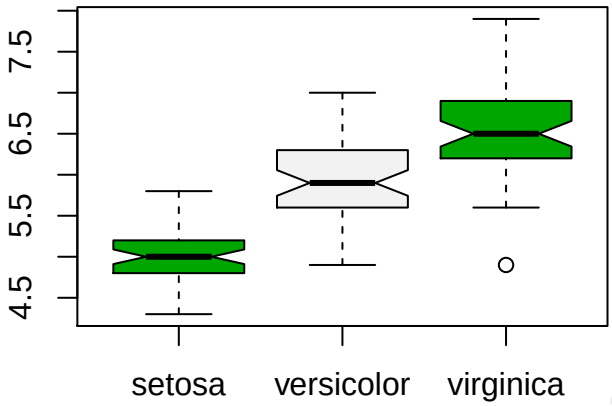


Figura 1.17: Boxplot de tipo violín

```
## conf  Numeric,2 Numeric,2 Numeric,2 Numeric,2
## out   Numeric,0 Numeric,0 Numeric,0 Numeric,0
## group Numeric,0 Numeric,0 Numeric,0 Numeric,0
## names ""        ""        ""        ""
```

Un parámetro muy interesante de la función boxplot es «notch». Con este parámetro con valor TRUE, se obtienen diagramas de caja que tienen una muesca o bisel doble alrededor de la mediana. La anchura de esa muesca viene a ser un análogo de los intervalos de confianza que conocemos, pero aplicado a la mediana en lugar de la media.

El parámetro notch controla la aparición de ese bisel doble o muesca en el diagrama de cajas. Incluir este parámetro nos puede ayudar a ver si las medianas son iguales o no. Al tratarse de muestras grandes y, si además son aproximadamente normales, la media y la mediana son muy similares. Y esos biseles permiten, de forma muy sencilla, evaluar gráficamente si existen diferencias significativas entre las medianas (y por consiguiente, en algunos casos, entre las medias). La regla práctica que se aplica es que si los biseles correspondientes a dos niveles no se solapan entre sí (en vertical), entonces podemos sospechar que existen diferencias significativas entre las medianas correspondientes a esos dos niveles. Si se quiere profundizar más en el uso y significado del parámetro «notch» se recomienda consultar la ayuda de la función boxplot de R. Un ejemplo utilizando este parámetros con el dataset iris lo podemos ver en la figura 1.17

```
boxplot(iris$Sepal.Length ~ iris$Species,col=terrain.colors(2),notch=TRUE)
```

Como puede verse, en este ejemplo tenemos razones para sospechar que existen diferencias significativas, dos a dos, en todos los casos.

Otro gráfico muy utilizado dentro del estudio analítico de los datos es el histograma, ya que su formato, nos puede dar una idea del tipo de distribución que hay detrás de los datos, y eso nos puede ayudar en el modelado de ls mismos. A continuación en la figura 1.18 vamos a representar los histogramas de las 10 primeras variables del conjunto de datos NCI60.

```
par(mfrow=c(2,5))
#plot de los 10 histogramas
sapply(seq(1,10),function(j)hist(NCI60$data[,j],main=paste("var.",colnames(NCI60$data)[j]),
        xlab="",freq=FALSE,col="deepskyblue4"))
```

```
##          [,1]               [,2]               [,3]
## breaks   Numeric,6          Numeric,11         Numeric,8
## counts   Integer,5          Integer,10         Integer,7
## density  Numeric,5          Numeric,10         Numeric,7
## mids     Numeric,5          Numeric,10         Numeric,7
## xname    "NCI60$data[, j]" "NCI60$data[, j]" "NCI60$data[, j]"
## equidist TRUE               TRUE               TRUE
##          [,4]               [,5]               [,6]
## breaks   Numeric,10         Numeric,7          Numeric,11
## counts   Integer,9          Integer,6          Integer,10
## density  Numeric,9          Numeric,6          Numeric,10
## mids     Numeric,9          Numeric,6          Numeric,10
## xname    "NCI60$data[, j]" "NCI60$data[, j]" "NCI60$data[, j]"
## equidist TRUE               TRUE               TRUE
##          [,7]               [,8]               [,9]
## breaks   Numeric,11         Numeric,9          Numeric,11
## counts   Integer,10         Integer,8          Integer,10
## density  Numeric,10         Numeric,8          Numeric,10
## mids     Numeric,10         Numeric,8          Numeric,10
## xname    "NCI60$data[, j]" "NCI60$data[, j]" "NCI60$data[, j]"
## equidist TRUE               TRUE               TRUE
##          [,10]
## breaks   Numeric,8
## counts   Integer,7
## density  Numeric,7
## mids     Numeric,7
## xname    "NCI60$data[, j]"
## equidist TRUE
```

En ocasiones, puede ocurrir que los datos de una variable, se concentren en un intervalo de datos, aunque el rango de valores pueda ser muchos mayor. Esta situación aparece en la distribución de la variable «capitalAve» del conjunto de datos spam.X. En la figura 1.19 se puede ver claramente la concentración de datos entorno a los primeros valores que toma la variable.

```
par(mfrow=c(1,1))
 hist(spam.X$capitalAve,main="Histogram of capitalAve",xlab="",
        col="deepskyblue4",freq=FALSE)
```

En estos casos lo que se suele hacer, sobe todo para hacer más legibles dichos datos, es una especie de estiramiento de la distribución hacia la derecha, y ese estiramiento se

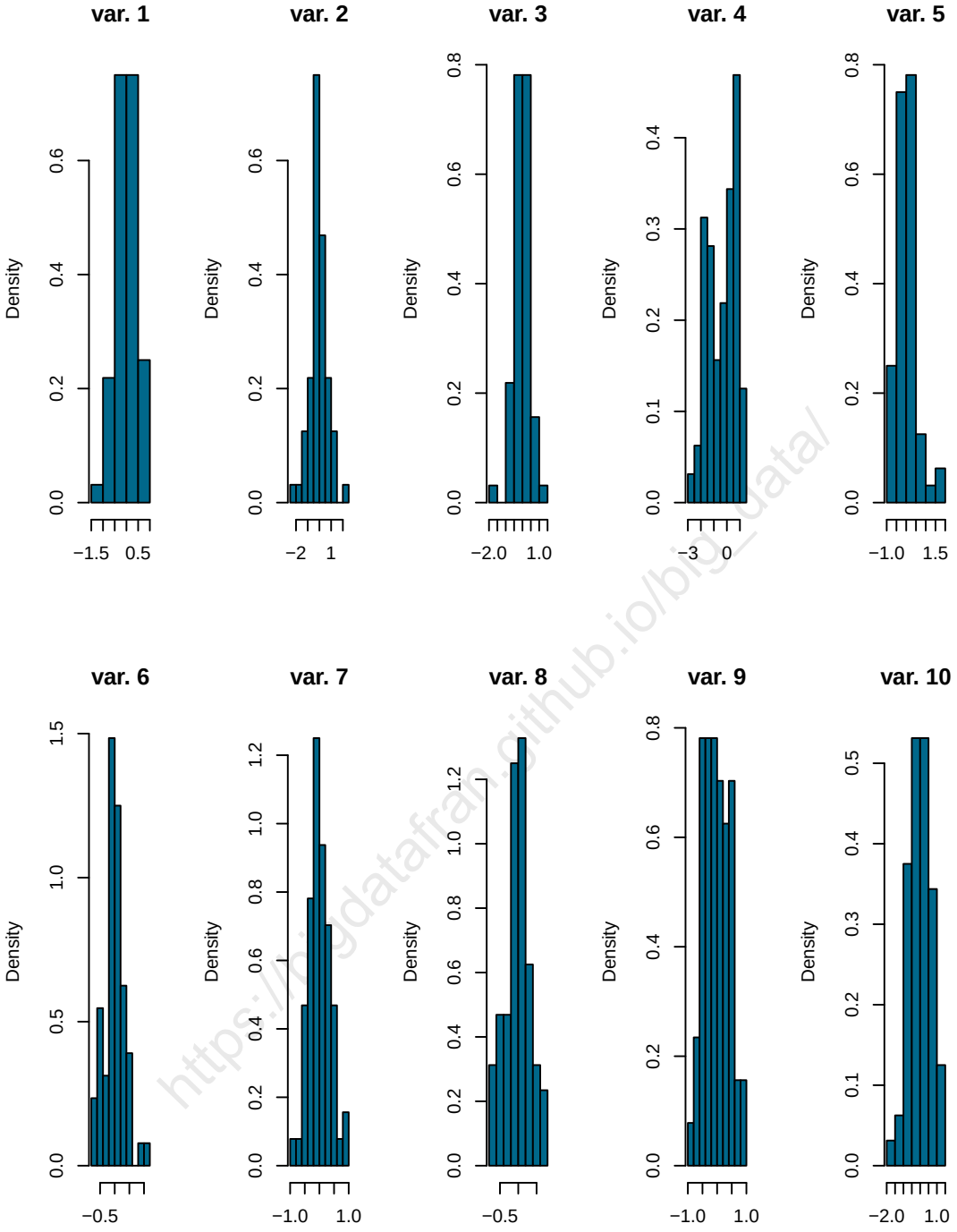




Figura 1.18: Histograma de 10 variables

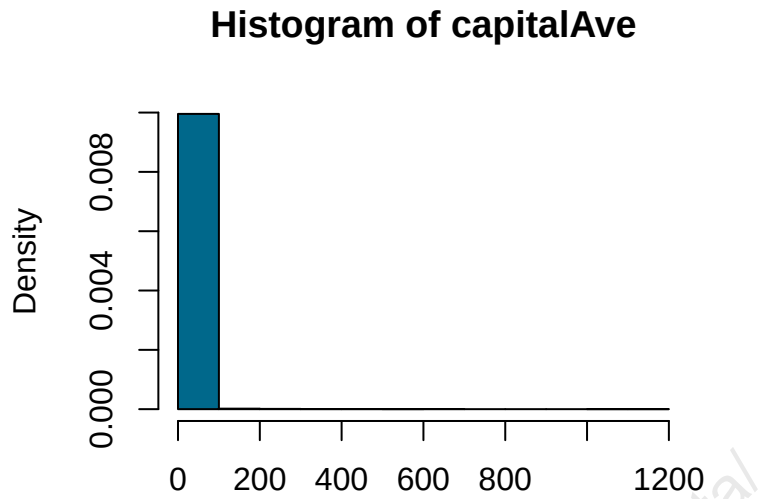


Figura 1.19: Concentración de datos

consigue haciendo una transformación de la variable, en este caso tomando logaritmos, de esa manera el resultado que se consigue se muestra en la figura 1.20 .

```
 hist(log(spam.X$capitalAve),main="Boxplot of logarithm of capitalAve",xlab=""
,col="deepskyblue4",freq=FALSE)
```

Podemos comprobar que los datos están más desplazados hacia la derecha y además el histograma es mucho más legible que el inicial. Otras transformaciones muy usuales son el cálculo del valor inverso, la raiz cuadrada, o elevarlo al cuadrado ( ver enlace en página 4 ) .

En el gráfico 1.21 vemos un gráfico muy interesante como es la superposición de dos funciones de distribución, cada una correspondiente a una variable aleatoria. Observar la construcción que se hace, y en especial, el hecho de que incialmente se utiliza la opción «plot=FALSE» con la finalidad de no dibujar inicialmente el gráfico, sino obtener solo los valores, para luego poder superponer los dos gráficos.

```
hist.spam <- hist(log(spam.X$capitalAve[spam.Y=="spam"]),plot=FALSE)
hist.nonspam <- hist(log(spam.X$capitalAve[spam.Y=="nonspam"]),plot=FALSE)

# Dibujamos los dos histogramas conjuntamente

plot(hist.spam,col=rgb(0,0,1,1/4),xlim=c(0,max(log(spam.X$capitalAve)))
,ylim=c(0,1),freq=FALSE,xlab="",main="Histogram of capitalAve in terms of spam")
plot(hist.nonspam,col=rgb(1,0,0,1/4),xlim=c(0,max(log(spam.X$capitalAve))),
freq=FALSE,add=T)
```

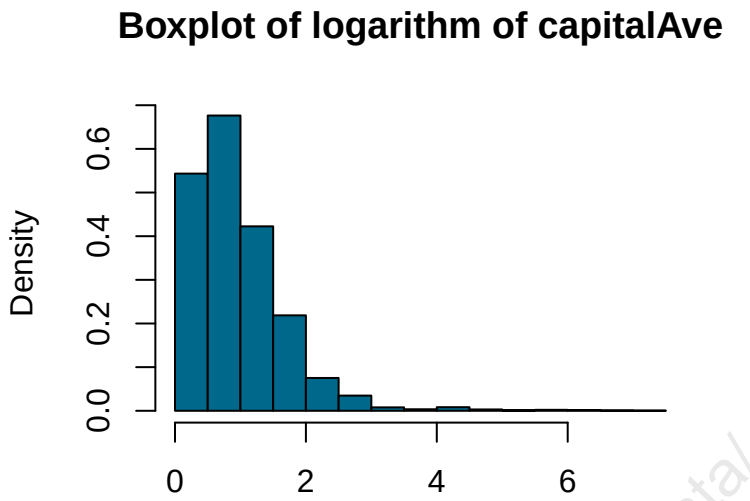


Figura 1.20: Transformación logarítmica

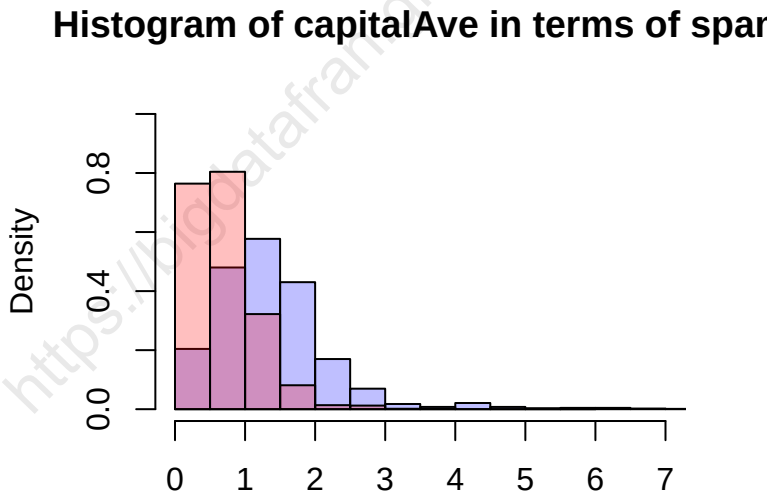


Figura 1.21: Superposición de dos distribuciones

Se puede estudiar y dibujar la estimación de las funciones de densidad, mediante la función «density()», en la figura 1.22 vemos cómo poder hacerlo, pues se representan las 10 estimaciones de la función de densidad de las 10 primeras variables del dataset NCI60.

```
par(mfrow=c(2,5))

# Plot de estimación  10 densidades Gaussian kernel

sapply(seq(1,10),function(j)plot(density(NCI60$data[,j],kernel="gaussian")
,main=paste("var.",colnames(NCI60$data)[j]),xlab="",col="deepskyblue4",lwd=2))
```

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

En el código que sigue, y que da como resultado la figura 1.23 se muestra una función que sirve para comparar dos distribuciones, y nos muestra las funciones de densidad estimada de esas dos distribuciones. Con esta función perfectamente se podría contrastar la hipótesis si dos variables siguen aproximadamente la misma distribución. No obstante con el test de Kolmogorov-Smirnov, y en concreto con la función de R, «ks.test(x, y, ...,)» ( Ver este enlace ), se podría realizar este contraste de forma numérica.

```
#Definimos inicialmente la función
densidades <- function(x,y,titulo) {
  #X e y son vectores con la misma dimensión sobre los que se calcula la densidad
  #titulo es el titulo del gráfico
  d1 <- density(x,kernel="gaussian")
  min.x <- min(d1$x)
  max.x <- max(d1$x)
  min.xy <- min(d1$y)
  max.xy <- max(d1$y)
```

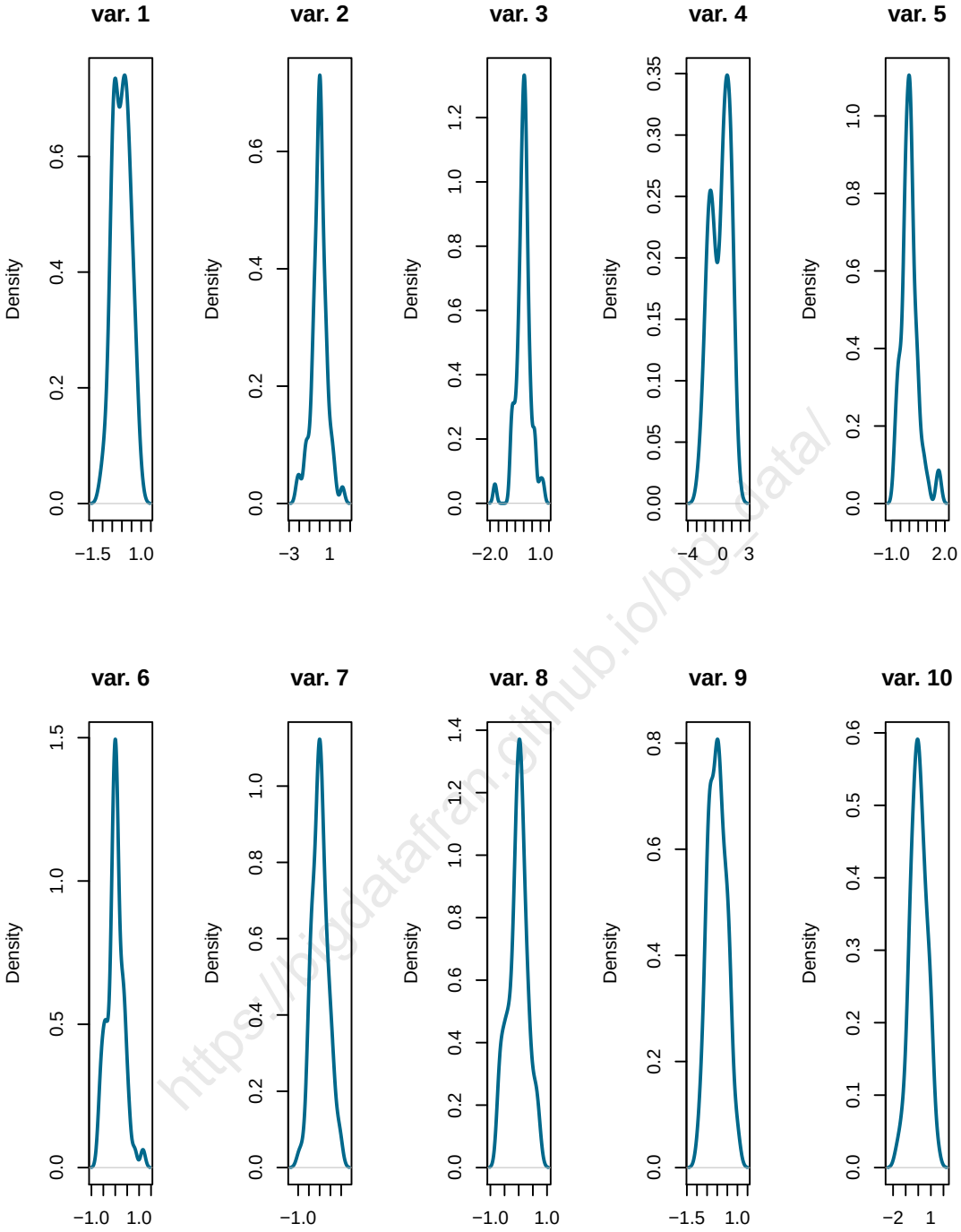




Figura 1.22: Representación estimación densidad

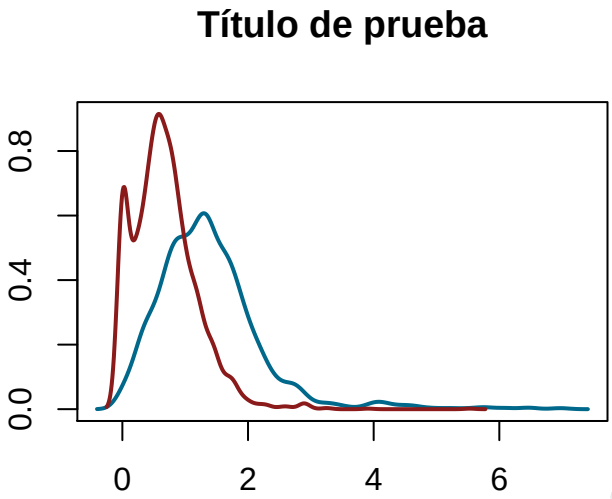


Figura 1.23: Comparación dos densidades

```
  d2 <- density(y,kernel="gaussian")
  min.yx <- min(d2$x)
  max.yx <- max(d2$x)
  min.y <- min(d2$y)
  max.y <- max(d2$y)

  min1.x <- min(c(min.x,min.yx))
  max1.x <- max(c(max.x,max.yx))
  min1.y <- min(c(min.xy,min.y))
  max1.y <- max(c(max.xy,max.y))

  plot(c(min1.x,max1.x),c(min1.y,max1.y),xlab="",ylab="",main=titulo,type="n")
  lines(d1$x,d1$y,col="deepskyblue4",lwd=2)
  lines(d2$x,d2$y,col="firebrick4",lwd=2)

}

#ahora llamamos a la función
densidades(log(spam.X$capitalAve[spam.Y=="spam"]),
        log(spam.X$capitalAve[spam.Y=="nonspam"]),"Título de prueba")
```

### 1.6.3. Presentación de datos en tres dimensiones.

Hasta ahora, todos los gráficos que se han presentado son en tres dimensiones. En muchas ocasiones nos interesa presentar la información en tres dimensiones, para ello existe la librería «scatterplot3d», que a continuación vamos a cargar y utilizar. En el

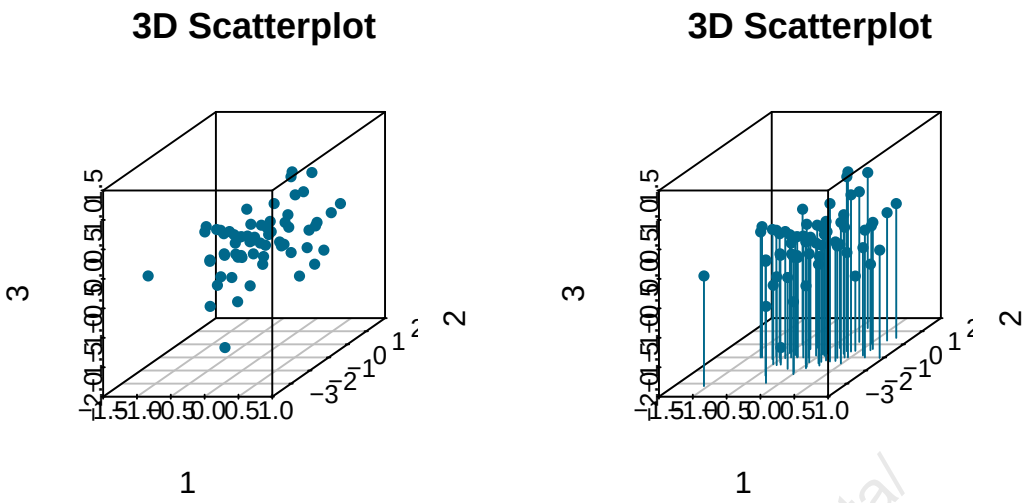


Figura 1.24: Representación en tres dimensiones

gráfico 1.24 podemos ver los resultados.

```
if(!require("scatterplot3d")) install.packages("scatterplot3d"); require("scatterplot3d")

## Loading required package: scatterplot3d

par(mfrow=c(1,2))
scatterplot3d(NCI60$data[,1:3],pch=20,color="deepskyblue4",main="3D Scatterplot")
scatterplot3d(NCI60$data[,1:3],pch=20,color="deepskyblue4",main="3D Scatterplot",
        type="h")
```

La librería «rgl» también nos permite hacer gráficos tridimensionales y además como valor añadido a la anterior, estos gráficos se pueden girar online y de forma dinámica, con lo que facilita enormemente la posibilidad de ver los datos desde diversas perspectivas. En el ejemplo que sigue, se muestra el código de ejemplo, pero no su salida, pues el resultado obtenido se podrá ver desde la pantalla del ordenador donde se ejecuta, pero no en este documento.

```
if(!require("rgl")) install.packages("rgl"); require("rgl")

## Loading required package: rgl

# Apertura de una ventana en tres dimesiones.
open3d()

plot3d(NCI60$data[,1:3],size=5,col="deepskyblue4")
```

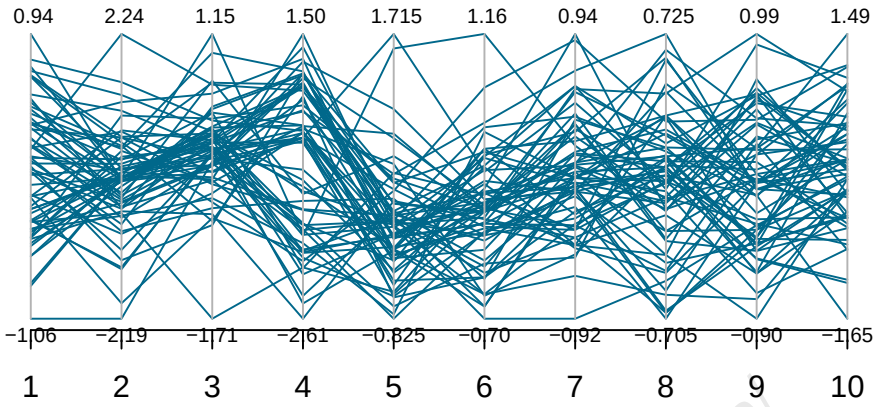


Figura 1.25: Parallel coordinates plot

## 1.7. Parallel coordinates plots.

Es un gráfico muy ilustrativo sobre posibles dependencias y tendencias de las variables, se debe cargar previamente la librería «MASS», y el resultado que se obtiene se puede ver en la figura 1.25,

```
if(!require("MASS")) install.packages("MASS"); require("MASS")

## Loading required package: MASS

#Sacamos el gráfico para 10 variables
parcoord(NCI60$data[,1:10],col="deepskyblue4",var.label=TRUE)
```

### 1.7.1. Mosaic plot.

Un mosaic plot ( diagrama de mosaico) es un método gráfico para visualizar datos de dos o más variables cualitativas. Este tipo de gráficos está dispuesto en celdas de tal manera que el área de cada celda es proporcional a la frecuencia correspondiente que hay en la casilla de la tabla de contingencia. En la figura 1.26 se ve un ejemplo de ese tipo de gráfico.

```
#Primero cruzo los datos
tabla<-table(births2006.smpl$SEX,births2006.smpl$DMETH_REC)
print(tabla)
```

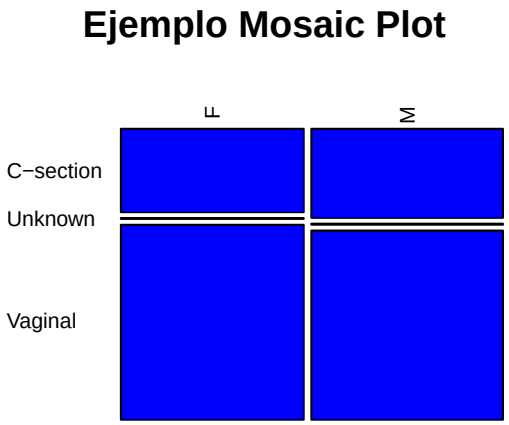


Figura 1.26: Ejemplo de mosaic plot

```
##
##     C-section Unknown Vaginal
##   F     62327     693  145633
##   M     69741     824  148105

if(!require("graphics")) install.packages("graphics"); require("graphics")
mosaicplot(tabla,main="Ejemplo Mosaic Plot",las=2,col='blue')
```

### 1.7.2. plot de correlaciones.

Como ya sabemos, la correlación es un valor estadístico que varía entre -1 y 1 y es una medida de relación lineal entre las dos variables. La función «cor( )» de R es la que se encarga de calcular este valor. Pero en ocasiones y soobre todo cuando estamos trabajando con muchas variables, es más estético y expresivo expresar estas correlación mediante un gráfico. Esto es lo que se consigue con el paquete «corrplot». En el gráfico 1.27 se muestra un ejemplo de la salida que se tiene utilizando el dataset iris.

```
if(!require("corrplot")) install.packages("corrplot"); require("corrplot")

## Loading required package: corrplot
## corrplot 0.84 loaded

#Calculamos las correlaciones entre las variables
corr<-cor(iris[,1:4])
#Imprimimos la matriz de correlaciones
print(corr)
```

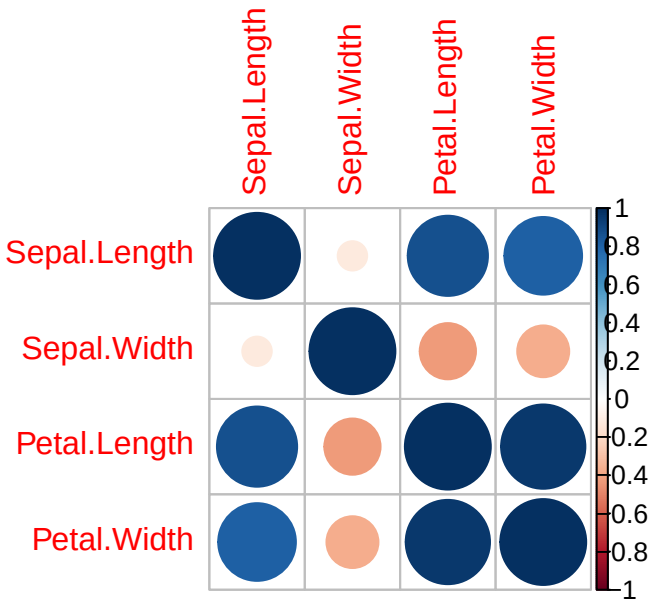


Figura 1.27: Plot de correlaciones

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000

#Sacamos el gráfico de correlaciones
corrplot(corr, method="circle")
```

### 1.7.3. chord diagram

Un chord diagram es un gráfico que nos permite ver las interrelaciones que hay entre dos variables categóricas con las datos que ofrecen la tabla de contingencia correspondiente. En la figura 1.28 se puede ver un ejemplo de esta figura.

```
if(!require("circlize")) install.packages("circlize"); require("circlize")

## Loading required package: circlize
## ========================================
## circlize version 0.4.4
## CRAN page: https://cran.r-project.org/package=circlize
## Github page: https://github.com/jokergoo/circlize
## Documentation: http://jokergoo.github.io/circlize_book/book/
##
## If you use it in published research, please cite:
## Gu, Z. circlize implements and enhances circular visualization
```

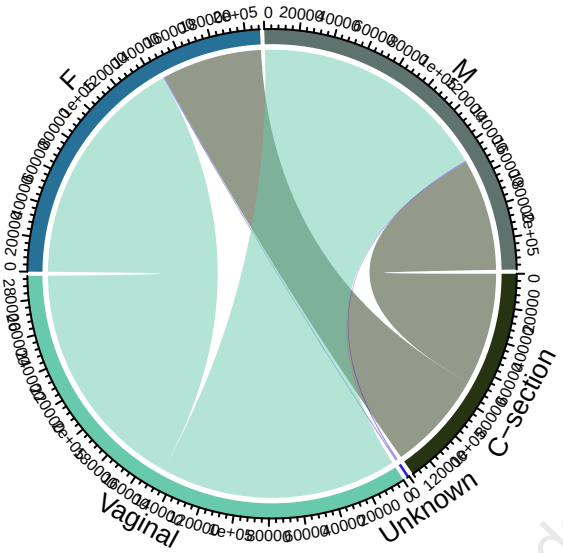


Figura 1.28: Construcción de un chord diagram

```
##  in R. Bioinformatics 2014.
## ======================================

tabla<-table(births2006.smpl$DMETH_REC,births2006.smpl$SEX)
b<-as.data.frame.matrix(tabla)
print(b)




chordDiagram(as.matrix(b))
```

```
#Borro los objetos de memoria
#rm(spam.X,spam.Y,spam,)
```

# Bibliografía

[https://www.r-graph-gallery.com/heatmap/]

[https://plot.ly/r/]

[https://rstudio.github.io/dygraphs/]

# Índice de figuras

# Índice alfabético