Introducción a R

Sesión 3
Estructuras de Datos y Gráficos

Joaquín Ortega Sánchez

Centro de Investigación en Matemáticas, CIMAT
Guanajuato, Gto., Mexico

Verano de Probabilidad y Estadística
Junio-Julio 2008

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

Arreglos

Un arreglo (`array`) de datos es un objeto que puede ser concebido como una matriz multidimensional (hasta 8 dimensiones). Una ventaja de este tipo de objeto es que sigue las reglas que hemos descrito para las matrices. La sintaxis para definir un arreglo es

```
array(data, dim).
```

Las componentes `data` y `dim` deben presentarse como una sola expresión, por ejemplo `c(2,4,6,8,10)` o `c(2,3,2)`:

```
> x <- array (1:24, c(3,4,2))
```

produce un arreglo tridimensional: la primera dimensión tiene tres niveles, la segunda tiene cuatro y la tercera tiene dos. Al imprimir el arreglo `R` comienza con la dimensión mayor y va bajando hacia la dimensión menor, imprimiendo matrices bidimensionales en cada etapa.

Arreglos

```
> x
, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
, , 2
     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

Arreglos

Otra manera de crear este arreglo es crear primero un vector con las componentes y luego asignarle las dimensiones adecuadas:

```
> x <- 1:24
> dim(x) <- c(3,4,2)
> x
```

Arreglos

Si asignamos el valor 2 al segundo índice obtenemos la matriz 3 x 2

```
> x[ ,2, ]
     [,1] [,2]
[1,]    4   16
[2,]    5   17
[3,]    6   18

> x[,2,1]
[1] 4 5 6

> x[1,,]
     [,1] [,2]
[1,]    1   13
[2,]    4   16
[3,]    7   19
[4,]   10   22
```

La Función `aperm`

La función `aperm` es una extensión de la trasposición de matrices que permuta las dimensiones de un arreglo. La sintaxis es `aperm(arreglo, perm, resize=TRUE)` donde `arreglo` es el arreglo a permutar, `perm` es el vector de la permutación y `resize` es un indicador sobre si el vector de datos debe ser redimensionado, en caso de ser necesario. Como ejemplo, el arreglo `iris3` que tiene dimensiones $50 \times 4 \times 3$ puede ser redimensionado a un arreglo de dimensiones $4 \times 3 \times 50$ con la instrucción

```
> aperm(iris3, c(2,3,1))
```

La Función `apply`

Esta función aplica de manera sucesiva una función a cada fila (primera dimensión), columna (segunda dimensión) o cada nivel de una dimensión superior. La sintaxis es

```
> apply(datos, dim, función, ...)
```

`datos` es el nombre de la matriz o arreglo y función es el nombre de cualquier función de `R`. Para una matriz bidimensional , la opción `dim` puede tomar el valor `1` ó `2` para indicar las filas o las columnas, respectivamente. Los puntos suspensivos indican opciones que pueden ser necesarias para la función en cuestión.

La Función `apply`

Como ejemplo, podemos usar esta función para determinar el máximo de las variables de nuestro conjunto de datos petroleros.

```
> apply( petro.datos, 2, max)
Prod. Reservas Refin. Ventas
 4.3      21500   6.6      8.9
```

La función max calcula el máximo para cada columna del conjunto de valores.

La Función `apply`

Para ver otro ejemplo del uso de esta función vamos a trabajar con el archivo `iris3` que tiene la información del conjunto de datos `iris` que hemos usado anteriormente pero en el formato de un arreglo de dimensiones $50 \times 4 \times 3$. La tercera dimensión corresponde a las especies, y para cada una de las tres especies hay una matriz de dimensión 50×4 con los valores de las cuatro variables que ya conocemos para las 50 plantas de la especie correspondiente.
Para obtener el valor medio de cada variable para todo el conjunto de datos escribimos

```
> apply(iris3, 2, mean)
Sepal L. Sepal W. Petal L. Petal W.
5.843333 3.057333 3.758000 1.199333
```

La Función `apply`

Si queremos las medias para cada especie entonces en el lugar de la dimensión a la cual vamos a aplicar la función hay que poner un vector `c(2,3)` de la siguiente manera:

```
> apply(iris3, c(2,3), mean)
            Setosa  Versicolor  Virginica
 Sepal L.    5.006       5.936      6.588
 Sepal W.    3.428       2.770      2.974
 Petal L.    1.462       4.260      5.552
 Petal W.    0.246       1.326      2.026
```

La Función `sweep`

La función `sweep` se usa con frecuencia junto con la función `apply`. Por ejemplo, supongamos que queremos restar la media que calculamos en el ejemplo anterior a todas las componentes del arreglo `iris3`, es decir, en cada caso queremos restar la media que corresponda a la variable y a la especie de cada dato. La forma de hacer esto es la siguiente:

La Función `sweep`

```
> iris.medias <- apply(iris3, c(2,3), mean)
> sweep(iris3, c(2,3), iris.medias,'-')
, , Setosa
```

	Sepal L.	Sepal W.	Petal L.	Petal W.
[1,]	0.094	0.072	-0.062	-0.046
[2,]	-0.106	-0.428	-0.062	-0.046
[3,]	-0.306	-0.228	-0.162	-0.046
⋮	⋮	⋮	⋮	⋮

La Función `outer`

Otra función importante para matrices, cuadros de datos y arreglos en general es la función `outer` o producto exterior. Si `A` y `B` son dos arreglos, su producto exterior es un nuevo arreglo cuya dimension es la concatenación de los vectores de dimensión de los arreglos A y B, en el orden del producto, y cuyas entradas se obtienen formando todos los posibles productos de los elementos de A con los elementos de B. Por ejemplo, el producto exterior de un vector de longitud 2 con otro de longitud 3 produce una matriz de dimensiones 2×3; el producto de una matriz de dimensiones $m \times n$ por otra matriz de dimensiones $p \times q$ produce un arreglo de dimensión $m \times n \times p \times q$. Veamos ejemplos.

La Función `outer`

```
> (aa <- 1:4)
[1] 1 2 3 4
> (bb <- c(2,4,6))
[1] 2 4 6
> (ab <- aa %o% bb)
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    4    8   12
[3,]    6   12   18
[4,]    8   16   24
```

En términos de álgebra lineal, el producto exterior `%o%` de los vectores x y y es el producto xy^t, mientras que el producto de matrices `%*%` representa $x^t y$, y requiere que los vectores tengan igual longitud.

La Función `outer`

```
> (A <- matrix(1:12, ncol=3))
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> (B <- 9:10)
[1] 9 10
```

La Función `outer`

```
> (AB <- A %o% B)
, , 1
     [,1] [,2] [,3]
[1,]    9   45   81
[2,]   18   54   90
[3,]   27   63   99
[4,]   36   72  108
, , 2
     [,1] [,2] [,3]
[1,]   10   50   90
[2,]   20   60  100
[3,]   30   70  110
[4,]   40   80  120
```

La Función `outer`

Otra sintaxis posible para la función `outer` que es mas flexible es

```
> outer(arreglo1, arreglo2, FUN='*')
```

donde `arreglo1, arreglo2` son los arreglos que se van a multiplicar y `FUN` es la operación que se va a usar para obtener el nuevo arreglo, que por defecto es la multiplicación, pero que puede ser cualquier otra operación o cualquier función de dos variables. Veamos ejemplos

La Función `outer`

```
> (AB <- outer(A,B))
, , 1
     [,1] [,2] [,3]
[1,]    9   45   81
[2,]   18   54   90
[3,]   27   63   99
[4,]   36   72  108
, , 2
     [,1] [,2] [,3]
[1,]   10   50   90
[2,]   20   60  100
[3,]   30   70  110
[4,]   40   80  120
```

La Función `outer`

Supongamos que queremos evaluar los valores de la función

$$\frac{1}{2\pi}\exp\{-\frac{x^2+y^2}{2}\}$$

que es la densidad Gaussiana bivariada estándar, en el cuadrado de lado $[-3,3]$. Tomamos un reticulado de 100 puntos entre -3 y 3 en cada lado:

```
> x <- y <- seq(-3,3,length=100)
> f <- function(x,y)
> {
> exp(-(x^2+y^2)/2)/(2*pi)
> }
> z <- outer(x,y,f)
 num [1:100, 1:100] 1.96e-05 2.35e-05 2.80e-05
...
```

La Función `outer`

Veamos como otro ejemplo la creación de una tabla de multiplicar:

```
> outer(1:9,1:9)
     [,1] [,2] [,3] [,4] [,5] [6] [,7] [,8] [,9]
[1,]    1    2    3    4    5    6    7    8    9
[2,]    2    4    6    8   10   12   14   16   18
[3,]    3    6    9   12   15   18   21   24   27
[4,]    4    8   12   16   20   24   28   32   36
[5,]    5   10   15   20   25   30   35   40   45
[6,]    6   12   18   24   30   36   42   48   54
[7,]    7   14   21   28   35   42   49   56   63
[8,]    8   16   24   32   40   48   56   64   72
[9,]    9   18   27   36   45   54   63   72   81
```

Ejercicio

Ejercicio 1

1. A partir del archivo `iris3` cree un arreglo con la misma información pero que tenga dimensiones $3 \times 50 \times 4$.
2. Usando las funciones `apply` y `sweep` calcule la desviación típica de las variables numéricas del arreglo `iris3` separadas por especie y luego obtenga un arreglo con cada dato dividido por el valor de la desviación típica correspondiente.
3. Calcule los valores de la siguiente función para un reticulado de valores de las variables x y de -1 a 1 en pasos de 0.1:

$$f(x, y) = \frac{\cos(y)}{1 + x^2}$$

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

Cuadros de Datos

Los cuadros de datos permiten mezclar distintos tipos de datos en una misma estructura, de modo que se puede tener acceso a los datos como en una matriz. La sintaxis es

```
> data.frame (datos1, datos2, ...)
```

donde los puntos suspensivos indican que pueden especificarse tantos conjuntos de datos como sea necesario.

Cuadros de Datos

Veamos un ejemplo:

```
> prueba.df <- data.frame(7:10,
    c('a','b','c','d'))
> prueba.df
> prueba.df <- data.frame(prueba.df,
    20+(1:4)/4)
> prueba.df
```

Para asignar nombres a las dimensiones usamos las funciones `colnames` y `rownames`:

```
> colnames(prueba.df) <- c('V1', 'V2', 'V3')
> row.names(prueba.df) <- c('S1','S2',
    'S3','S4')
> prueba.df
```

Cuadros de Datos

Para ver un ejemplo de la utilidad de los cuadros de datos volvamos al ejemplo de las compañías petroleras. Observamos que tres de ellas, Exxon Mobil, BP Amoco y Total ELF, son producto de fusiones entre compañías que antes eran independientes. Queremos incluir esta información en nuestra estructura de datos añadiendo un vector con una nueva variable de caracteres. Pondremos `F` si la compañía es producto de una fusión y `NF` si no lo es. Inicialmente trataremos de incluir esta información en la matriz, tal como lo hicimos anteriormente.

Cuadros de Datos

```
> Fusion <- c("F", "NF", "F", "F", "NF")
> petro.datos1 <- cbind (petro.datos, Fusion)
> petro.datos1
             Prod. Reservas Refin. Ventas Fusion
Exxon Mobil   "4.3"   "21500" "6.6"    "8.9"  "F"
Shell         "3.7"   "20500" "3.4"    "5.7" "NF"
BP Amoco      "4.1"   "19400" "3.3"    "5.4"  "F"
Total ELF     "2.1"    "9600" "2.4"    "3.2"  "F"
Chevron       "1.5"    "6200" "1.6"    "2.0" "NF"
```

Cuadros de Datos

El problema con este procedimiento es que, debido a la estructura de manejo de distintos modos de datos, los datos han sido transformados a modo 'carácter' y no podemos hacer operaciones aritméticas con ellos. Por ejemplo, si tratamos ahora de buscar el máximo

```
> apply(petro.datos1, 2, max)
Error in max(..., na.rm = na.rm) : invalid
    'mode' of argument
```

R produce un mensaje de error porque los datos no son numéricos. Este problema se resuelve usando un cuadro de datos como estructura.

Cuadros de Datos

```
> petro.frame <- data.frame (petro.datos, Fusion)
> petro.frame
             Prod. Reservas Refin. Ventas Fusion
Exxon Mobil   4.3    21500    6.6    8.9      F
Shell         3.7    20500    3.4    5.7     NF
BP Amoco      4.1    19400    3.3    5.4      F
Total ELF     2.1     9600    2.4    3.2      F
Chevron       1.5     6200    1.6    2.0     NF
```

Cuadros de Datos

Podemos tener acceso a las variables individuales usando dos notaciones distintas,
`petro.frame$Fusion` o `petro.frame["Fusion"]`.

```
>petro.frame$Fusion
[1] F  NF F  F  NF
Levels: F NF
petro.frame["Fusion"]
             Fusion
Exxon Mobil       F
Shell            NF
BP Amoco          F
Total ELF         F
Chevron          NF
```

Cuadros de Datos

Podemos tener acceso a las variables individuales usando dos notaciones distintas,
`prueba.df$V1` o `prueba.df["V1"]`.

```
> prueba.df$V1
[1]  7  8  9 10
> prueba.df['V1']
    V1
S1   7
S2   8
S3   9
S4  10
```

Cuadros de Datos

Ambas son largas y pueden resultar tediosas si hay que usarlas con frecuencia. Sin embargo, hay una alternativa que resulta útil cuando vamos a usar el mismo conjunto de datos repetidas veces: usar el comando `attach` ('pegar'). Si se usa este comando con un cuadro de datos, es posible trabajar usando sólo los nombres de las variables, sin hacer referencia al cuadro de datos en uso, hasta que se separe usando el comando `detach`. Al usar el comando attach hay que tener en cuenta que si existen variables que tengan el mismo nombre que las variables del cuadro de datos, puede haber conflictos.

Cuadros de Datos

Veamos un ejemplo,

```
> attach(prueba.df)
> V1
[1] 7 8 9 10
> S2
Error: object "S2"not found
```

Cuadros de Datos

Veamos un ejemplo

```
> attach (petro.frame)
    The following object(s) are masked _by_
    .GlobalEnv :
    Fusion
> Ventas
Exxon Mobil Shell BP Amoco Total ELF Chevron
        8.9   5.7      5.4       3.2       2
> Fusion
[1] "F" "NF" "F" "F" "NF"
```

Cuadros de Datos

Observe que al dar la instrucción `attach`, `R` emite un alerta indicando que el objeto `Fusion` esta 'cubierto' por otro en el ambiente de trabajo, y al pedir `Fusion` el programa no nos da como respuesta los valores de esta variable en el cuadro de datos, ya que los valores aparecen entre comillas y no aparecen abajo los niveles, como anteriormente. La respuesta del programa es la variable local '`Fusion`', que construimos previamente a su inclusión en el cuadro de datos. Para ver la variable del cuadro de datos debemos separar (`detach`) el cuadro, eliminar (`rm`) la variable `Fusion` y volver a pegar (`attach`) el cuadro de datos.

Cuadros de Datos

```
> detach()
> rm(Fusion)
> attach (petro.frame)
> Fusion
[1] F NF F F NF
Levels: F NF
> detach()
> Fusion
Error: Object "Fusion" not found
```

Cuadros de Datos

La versión local de una variable precede a la versión que viene a través de la función `attach`.
Para ver el efecto que tiene la instrucción `attach`, podemos ver la trayectoria de búsqueda usando el comando `search()`:

```
> attach (prueba.df)
> search()
[1] ".GlobalEnv" "prueba.df"
"package:methods"
[4] "package:stats" "package:graphics"
[6] "package:grDevices" "package:utils"
[8] "package:datasets" "Autoloads"
[10] "package:base"
```

Observamos que el cuadro de datos `prueba.df` esta ubicado en el segundo lugar de la trayectoria de búsqueda.

Cuadros de Datos

Para eliminarlo de la trayectoria de búsqueda usamos la instrucción `detach`

```
> detach()
> search()
[1] ".GlobalEnv" "package:methods"
"package:stats"
[4] "package:graphics" "package:grDevices"
[6] "package:utils" "package:datasets"
[8] "Autoloads" "package:base"
```

Cuadros de Datos

La función `attributes` permite listar las características de cualquier objeto en R.

```
> attributes(prueba.df)
$names
[1] "V1" "V2" "V3"
$row.names
[1] "S1" "S2" "S3" "S4"
$class
[1] "data.frame"
```

Cuadros de Datos

La función `str` muestra la estructura del conjunto de datos.

```
> str(prueba.df)
'data.frame': 4 obs. of 3 variables:
$ V1: int 7 8 9 10
$ V2: Factor w/ 4 levels "a","b","c","d": 1 2
    3 4
$ V3: num 20.3 20.5 20.8 21.0
```

Ejercicio

Ejercicio 2

1. Construya un cuadro de datos llamado `notas` con la siguiente información

	Examen1	Examen2	Tareas
Antonio	7	9	8
Berenice	6	6	7
Carlos	8	8	9

2. Use la función `apply` para obtener el promedio de todas las filas. Agregue una nueva columna de nombre `Def.` que tenga los valores promedios.
3. Agregue una nueva columna que indique si el estudiante aprobó (`Ap.`) o reprobó (`Rep.`).

La Función `reshape`

La función `reshape` permite cambiar el formato de un cuadro de datos de ancho a largo y viceversa. Cuando tenemos medida repetidas para un mismo sujeto, en el formato ancho estas distintas mediciones se encuentran en columnas distintas que corresponden a las distintas mediciones. En el formato largo, las mediciones están en un misma columna y en la siguiente columna hay una variable o un factor que identifica de cuál medición se trata.

La Función `reshape`

Como ejemplo de un cuadro de datos largo tenemos `Indometh`, un archivo que contiene resultados sobre la farmacokinesis del compuestoo indometicina. El archivo tiene tres columnas: `Subject, time, conc`. Para cada sujeto se midió la concentración en 10 intervalos de tiempo.
En el archivo los seis sujetos se listan en la primera columna, cada uno repetido 10 veces. La segunda columna indica el tiempo de la medición y la tercera la concentración resultado de la medición. Usando la función `reshape` vamos a cambiar el formato del cuadro de datos para tener una sola fila por sujeto y una columna para cada tiempo.

La Función `reshape`

La sintaxis de la instrucción `reshape` es

```
reshape(data, varying=NULL, v.names=NULL,
timevar='time', idvar='id', direction, ...)
```

donde

- `data` es el cuadro de datos, `varying` son los nombres de conjuntos de variables en formato ancho que corresponden a una sóla variable en el formato largo,
- `v.names` son los nombres de variables en el formato largo que corresponden a multiples variables en el formato ancho,
- `timevar` es la variable en el formato largo que diferencia registros múltiples para el mismo individuo o grupo,
- `idvar` son los nombres de una o más variables en el formato largo que identifican los registros múltiples para el mismo individuo o grupo y
- `direction` es una palabra que identifica el tipo de formato que se quiere, `wide` o `long`.

La Función `reshape`

Veamos como ejemplo el uso de esta función para cambiar el formato del cuadro de datos `Indometh` de `long` a `wide`.

```
> str(Indometh)
> summary(Indometh)
> Indometh
> Indowide <- reshape(Indometh,
v.names='conc',
     idvar='Subject', timevar='time',
     direction='wide')
> Indowide
> reshape(Indowide, direction="long")
> reshape(Indowide, idvar="Subject", varying=
     list(names(wide)[2:12]), v.names=çonc",
     direction="long")
```

La Función `merge`

La función `merge` permite unir dos cuadros de datos en base a los valores comunes de ciertas columnas o filas. Para ver un ejemplo vamos a usar los conjuntos de datos `Animals` y `mammals` del paquete `MASS`. El primero tiene el peso promedio del cerebro y del cuerpo para 28 especies de animales terrestres mientras que el segundo tiene la misma información para 62 especies de mamíferos terrestres. Vamos a usar la instrucción uniendo las entradas con nombres de fila iguales.

```
> library(MASS)
> Animals
> mammals
> merge(Animals, mammals, by='row.names')
```

La Función `aggregate`

La función `aggregate` divide los datos en subconjuntos, calcula alguna función estadística para cada uno de ellos y devuelve el resultado en un formato apropiado. La sintaxis es `agregate(x,by,FUN,...)` donde `x` es el cuadro de datos, `by` es una lista (en formato `list` que veremos a continuación) de los elementos que se usarán para crear los grupos y `FUN` es la funcón que se quiere calcular. Veamos un ejemplo con el conjuntos de datos `crabs`.

```
> str(crabs)
> aggregate(crabs[, 4:8], list(sp=crabs$sp,
    sex=crabs$sex), median)
  sp sex    FL    RW    CL    CW    BD
1  B   F 13.15 12.20 27.90 32.35 11.60
2  O   F 18.00 14.65 34.70 39.55 15.65
3  B   M 15.10 11.70 32.45 37.10 13.60
4  O   M 16.70 12.10 33.35 36.30 15.00
```

La Función `by`

La función `by` tiene un comportamiento similar. El cuadro de datos se divide por filas en cuadros de datos de acuerdo al valor de uno o varios factores, y se usa una funcón sobre cada uno de estos cuadros de datos resultantes. La sintaxis es `by(x,INDICES,FUN,...)`. Veamos un ejemplo usando el mismo conjunto de datos `crabs`.

La Función by

```
> by(crabs[, 4:8], list(sp=crabs$sp,
    sex=crabs$sex), mean)
sp: B
sex: F
FL RW CL CW BD
13.270 12.138 28.102 32.624 11.816
------------------------------------------------------------
sp: O
sex: F
FL RW CL CW BD
17.594 14.836 34.618 39.036 15.632
------------------------------------------------------------
sp: B
sex: M
FL RW CL CW BD
14.842 11.718 32.014 36.810 13.350
------------------------------------------------------------
sp: O
sex: M
FL RW CL CW BD
16.626 12.262 33.688 37.188 15.324
```

La Función `order`

La función `sort` permite ordenar las componentes de un vector. Para ordenar estructuras más complejas como una matriz o cuadro de datos existe la función `order`. El resultado de usar esta función sobre un vector es una permutación que reacomoda el vector en orden ascendente o descendente, según la opción `decreasing` que por defecto es cierta. Veamos un ejemplo

```
> (x <- c(4:6,2,1,3,7))
[1] 4 5 6 2 1 3 7
> order(x)
[1] 5 4 6 1 2 3 7
```

El resultado quiere decir que para ordenar el vector `x` hay que colocar en el primer lugar la componente 5, en segundo lugar la componente 4, en tercer lugar la componente 6, y así sucesivamente.

La Función `order`

Si queremos ordenar el vector `x` ejecutamos la instrucción

```
> x[order(x)]
[1] 1 2 3 4 5 6 7
```

Lo interesante es que esta función permite utilizar los índices obtenidos para ordenar varios vectores usando el criterio correspondiente al vector `x`.

```
> > y <- 7:1
> z <- 1:7
> rbind(x,y,z)[,order(x)]
  [,1] [,2] [,3] [,4] [,5] [,6] [,7]
x    1    2    3    4    5    6    7
y    3    4    2    7    6    5    1
z    5    4    6    1    2    3    7
```

La Función `order`

Veamos un ejemplo del uso de esta función para ordenar un conjunto de datos. Vamos a tomar las 10 primeras filas de las variables numéricas del conjunto iris y las vamos a ordenar según la primera columna.

```
> (iris10 <- iris[1:10,1:4])
> ordiris <- order(iris10[,1])
> iris10[ordiris,]
```

Cuadros de Datos

El uso de operaciones aritméticas o matriciales con cuadros de datos grandes puede resultar mucho más lenta que la operación equivalente con matrices, incluso para operaciones sencillas. Por esta razón es conveniente usar la función `as.matrix` que convierte temporalmente el cuadro de datos en matriz, para efectuar las operaciones necesarias.
Por ejemplo, en lugar de multiplicar los cuadros de datos `A1` y `B2` resulta más eficiente escribir

```
> as.matrix(A1) * as.matrix(B2)
```

Ejercicio

Ejercicio 3

1. El conjunto de datos `immer` del paquete `MASS` tiene un formato 'ancho'. Usando la función `reshape` cambie el formato a 'largo' usando como variables a convertir en factor discriminador `Y1` y `Y2`.
2. Usando la función `by` calcule la media para las variables numéricas del conjunto `iris` separadas por especie.
3. Ordene las variables numéricas del conjunto `iris` según los valores de `Sepal.Length`.

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

Listas

Las listas son objetos que permiten unir información que no tiene la misma estructura. Recordemos que en los cuadros de datos, los datos podían ser de distintos modos pero debían tener la misma estructura.

Regresemos al ejemplo `prueba.df` y supongamos que queremos incluir dos números de teléfono. Esta información no tiene la misma estructura que la anterior, pero usando una lista podemos almacenarlas conjuntamente.

```
> Telefonos <- c('(473) 7329900',
    '(477) 7726543')
> prueba.list <- list( prueba.df, Telefonos)
```

Listas

```
> prueba.list
[[1]]:
   V1  V2     V3
S1  7   a  20.25
S2  8   b  20.50
S3  9   c  20.75
S4 10   d  21.00

[[2]]
[1] '(473) 7329900' '(473) 7726543'
```

Listas

Buscamos ahora algunos valores en la lista:

```
> prueba.list$Telefonos
NULL
> prueba.list[[1]]['S1', 'V3']
[1] 20.25
> prueba.list[[1]][, 'V1']
[1] 7 8 9 10
> prueba.list[[2]]
[1] '(473) 7429900' '(477) 7726543'
```

Listas

El comando `list` simplemente 'pega' las distintas piezas incluidas en el comando. Crea un item para cada pieza, que se listan secuencialmente y pueden ser recuperados usando corchetes dobles.
Observe que la sintaxis usada con los cuadros de datos no funciona con listas y el programa devuelve el valor `NULL`. Para tener acceso a datos individuales tenga en cuenta que los corchetes dobles deben ser tratados como parte del nombre de la estructura de interés. El comando `prueba.list[[2]]` produce el segundo elemento de la lista. De allí en adelante las referencias son como si se tratara de una matriz:

```
> prueba.list[[1]][2,3]
[1] 20.5
```

Listas

Una lista es un tipo de estructura que tiene gran flexibilidad para guardar datos. Su característica principal es que puede almacenar cualquier tipo de objeto como una de sus variables. Los objetos en una lista son completamente independientes y pueden ser de tipos y tamaños distintos. Para construir una lista usamos la función `list` y enumeramos los objetos que deseamos incluir. Veamos un ejemplo:

```
> L <- list (matrix(1:6,ncol=2), c(T,F),
    c("Buenos", "Dias"))
```

Esta lista tiene tres elementos: Una matriz numérica, un vector con los valores lógicos `TRUE` y `FALSE` y dos palabras.

Listas

Para ver el contenido de la lista escribimos su nombre, al igual que hacemos con otras estructuras:

```
> L
[[1]]
, , 1

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

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

Listas

```
[[2]]
[1] TRUE FALSE

[[3]]
[1] 'Buenos' 'Dias'
```

Listas

La presentación del contenido de L nos indica como podemos tener acceso a los elementos de la lista:

```
> L[[2]]
[1] T F
> L[[3]][1]
[1] 'Buenos'
```

Los elementos de un lista pueden ser asignados y modificados tal como se hace con otras estructuras.

Listas

Para añadir un elemento nuevo a una lista basta colocarlo en alguna posición que no esté en uso. La lista L tiene tres elementos (esto puede determinarse como de costumbre usando la función `length()`). Podemos añadir un nuevo elemento asignándolo a un índice libre.

```
> L[[4]] <- c('El', 'nuevo', 'elemento')
```

Se puede usar cualquier índice, pero es mejor usar el próximo que se encuentre libre, ya que a los elementos intermedios se les asigna el valor NULL. Si añadimos un elemento con índice 10 a una lista de 3 elementos, estamos creando automáticamente los elementos del lugar 4 al 9.

Listas

Se puede usar el siguiente comando para añadir el nuevo elemento en el primer lugar libre:

```
> L[[length(L)+1]] <- nuevo elemento
```

donde `nuevo elemento` puede ser cualquier tipo de objeto de datos.
Si asignamos un índice existente, el elemento es reemplazado.
Para eliminar un elemento existente hay que asignarle el valor NULL. La instrucción

```
> L[[2]] <- NULL
```

elimina el segundo elemento de la lista, de modo que no vuelve a aparecer.

Listas

```
> L[[2]] <- NULL
> L
[[1]]
, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2
     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
[[2]]:
[1] 'Buenos' 'Dias'
[[3]]:
[1] 'El' 'nuevo' 'elemento'
```

Listas

Como observamos, al eliminar un elementos los demás se mueven hacia adelante y sus índices se reducen en una unidad. Es imposible eliminar varios elementos de una lista simultáneamente. Tampoco se puede usar índices negativos para suprimir elementos.

Ejercicio

Ejercicio 4

1. Definimos la siguiente lista:

```
> LL <- list(1:10, 11:22, 44:24,
    'string', T)
```

¿Cuál es el efecto de la siguiente instrucción?

```
> for (i in 1:4) { LL[[2]] <- NULL }
```

2. Las siguientes instruciones tienen como objetivo borrar el segundo, tercer y quinto elementos de la lista L ¿Qué hace realmente el programa?

```
> T <- list(1:11,'a','b','c','d','e')
> T[[2]] <- NULL
> T[[3]] <- NULL
> T[[5]] <- NULL
```

Listas

Al igual que ocurre con matrices y vectores, podemos asignarle nombres a los elementos de una lista. La función correspondiente es `names`. Luego podemos usar los nombres de los elementos para tener acceso a ellos. Veamos un ejemplo:

```
> L <- list( numero= 1:10, boole=c(T,F),
    mensaje = c('Buenos', 'Dias'))
> L
$numero:
[1] 1 2 3 4 5 6 7 8 9 10
$boole:
[1] T F
$mensaje:
[1] 'Buenos' 'Dias'
```

Listas

Se obtiene el mismo resultado si asignamos los nombres después de haber creado la lista:

```
> L <- list( 1:10, c(T,F), c('Buenos',
    'Dias'))
> names(L) <- c('numero', 'boole', 'mensaje')
```

Ahora podemos usar los nombres para tener acceso a los elementos sin necesidad de recordar los índices. Por ejemplo,

```
> L$boole
[1] T F
```

Si asignamos un valor a un elemento cuyo nombre no existe, `R` crea un nuevo elemento con ese nombre y le asigna el valor:

```
> L$nuevo <- 'elemento nuevo'
```

Listas

Se pueden modificar los nombres existentes usando la función `names`.

```
> names(L)
[1] 'numero' 'boole' 'mensaje' 'nuevo'
> names(L)[2] <- 'valores logicos'
> names(L)
[1] 'numero' 'valores logicos' 'mensaje'
    'nuevo'
```

La función `lapply`

La función `lapply` permite usar la misma función sobre todos los elementos de una lista. Por ejemplo, podemos querer calcular la media de los elementos de la lista. Veamos un ejemplo.

```
> L <- list (vec = 1:10, mat=matrix (99:88,3,4))
> L
$vec:
[1] 1 2 3 4 5 6 7 8 9 10
$mat:
     [,1] [,2] [,3] [,4]
[1,]   99   96   93   90
[2,]   98   95   92   89
[3,]   97   94   91   88
> lapply (L, mean)
$vec:
[1] 5.5
$mat:
[1] 93.5
```

Listas

También podemos, por ejemplo, restar 1 a cada elemento de la lista:

```
> lapply (L, "-", 1)
$vec:
[1] 0 1 2 3 4 5 6 7 8 9

$mat:
     [,1] [,2] [,3] [,4]
[1,]   98   95   92   89
[2,]   97   94   91   88
[3,]   96   93   90   87
```

Listas

Las funciones `apply` y `sapply` son similares.

```
> sapply(mtcars, range)
      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
[1,] 10.4   4  71.1  52 2.76 1.513  14.5  0  0    3    1
[2,] 33.9   8 472.0 335 4.93 5.424  22.9  1  1    5    8
```

```
> apply(mtcars,2, range)
      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
[1,] 10.4   4  71.1  52 2.76 1.513  14.5  0  0    3    1
[2,] 33.9   8 472.0 335 4.93 5.424  22.9  1  1    5    8
```

Listas

La sintaxis de estas funciones es

```
apply(<arreglo>, <dimensión>, <función>)
lapply(<lista>, <función>)
sapply(<lista>, <función>)
```

El resultado de `lapply` es una lista. El de `sapply` se simplifica a un vector o una matriz, si es posible.

Listas

En ciertas ocasiones es necesario separar los elementos de una lista para recomponerlos de otra manera o en una estructura diferente. La función `unlist` toma todos los elementos de la lista y los combina en la estructura de datos más global. Por ejemplo, una lista con vectores y matrices numéricas se transforma en un vector que contiene todos los números. Una lista con valores numéricos y de caracteres se transforma en una variable de modo carácter. En el ejemplo anterior, al aplicar la función media a los elementos de lista, nos puede interesar tener un vector de medias en lugar de una lista. Podemos usar la función `unlist` para esto:

```
> unlist (lapply (L, mean))
vec mat
5.5 93.5
```

Conversión de Objetos

Como hemos visto, las diferencias entre algunos tipos de objetos son pequeñas. Es posible convertir objetos de un tipo a otro cambiando algunos de sus atributos. Las instrucciones para este tipo de cambio tienen un formato común del tipo `as.`*algo*. `R` tiene más de 90 funciones de este tipo. El resultado de una conversión depende de los atributos del objeto. Algunos de los casos se resumen en la siguiente tabla.

Conversión de Objetos

Conversión a	Función	Reglas
numérico	`as.numeric`	`FALSE` → 0 `TRUE` → 1 `'1','2',...` → 1,2,... `'A'` → `NA`

lógico	`as.logical`	0 → `FALSE`
		otros números → `TRUE`
		`'FALSE', 'F'` → `FALSE`
		`'TRUE', 'T'` → `TRUE`
		otros caracteres → `NA`

carácter	`as.character`	1,2,... → `'1', '2', ...`
		`FALSE` → `'FALSE'`
		`TRUE` → `'TRUE'`

Tabla 1.9

Listas

R incluye funciones para calcular una variedad de estadísticas de una muestra. Para datos numéricos tenemos, entre otras, las siguientes

Función	Operación
`mean()`	Promedio
`median()`	Mediana
`fivenum()`	Resumen de 5 números
`summary()`	Resumen numérico
`min(), max()`	Menor y mayor valor en la muestra
`quantile()`	Cuantiles muestrales
`var(), sd()`	Varianza y desviación típica
`cov(), cor()`	Covarianza y correlación

Tabla 1.10.

Listas

Algunas de estas funciones dan el resultado esperado cuando se aplican sobre matrices o cuadros de datos:

```
> round(mean(mtcars),1)
 mpg  cyl  disp   hp drat  wt qsec  vs  am gear carb
20.1  6.2 230.7 146.7  3.6 3.2 17.8 0.4 0.4  3.7  2.8
```

Problema

Ejercicio 5

1. Construya una lista que contenga el vector `equipaje`, el vector `np`, la matriz `AA`, y el cuadro de datos `paises`, conservando el mismo nombre para cada componente. Ejecute las siguientes acciones sobre la lista:
2. Elimine los tres últimos elementos de la componente `equipajes`
3. Obtenga los elementos $(2, 3)$ y $(4, 2)$ de la matriz `AA`
4. Obtenga los datos para `Alemania` y la `Inflacion` para todos los paises.
5. Usando la función `apply` o alguna de sus variantes, calcule la mediana de las columnas de `AA`.

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

Introducción

Una de las mayores ventajas de `R` es su capacidad gráfica. Para ver una demostración usamos los siguientes comandos:

```
> demo(graphics)
> demo(persp)
> demo(image)
```

Las ventanas gráficas se abren automáticamente al ejecutar una función gráfica durante la sesión, y cualquier nueva gráfica se presenta en la misma pantalla y sustituye a la anterior, a menos que se de una instrucción para superponerla o se abra una nueva ventana gráfica. Para hacer esto en Windows usamos la instrucción `windows()`.

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

plot

Posiblemente es la función gráfica más usada en `R`. Su efecto depende de la sintaxis y el argumento que usemos. Para ver algunos ejemplos trabajaremos con el cuadro de datos `iris` que ya usamos anteriormente.

```
> attach(iris)
> plot(Sepal.Length, Sepal.Width)
```

Con esta instrución obtenemos una gráfica de `Sepal.Length` (eje *x*) contra `Sepal.Width` (eje *y*). El mismo resultado se obtiene con las siguientes instrucciones.

```
> plot(Sepal.Width ~ Sepal.Length)
> plot( Sepal.Length + Sepal.Width)
```

Es importante observar que en la primera de estas dos instrucciones el orden de las variables se invierte: se indica primero la variable que aparece en el eje *y* y luego la del eje *x*.

plot

En estas tres instrucciones usamos vectores como argumento de la función `plot`. Si, en cambio, usamos un factor como argumento, como por ejemplo

```
> plot(Species)
```

obtenemos un diagrama de barras o `barplot` del factor `Species`. En este caso, como hay el mismo número de individuos de cada especie (50) el gráfico no es muy interesante.

La instrucción

```
> plot(Species, Sepal.Length)
```

produce un diagrama de caja (`boxplot`) de la variable `Sepal.Length` para cada especie de iris. En este caso el primer argumento es un factor mientras que el segundo es un vector numérico.

plot

La instrucción

```
> plot(iris)
```

produce una matriz de gráficos donde aparecen todas las combinaciones posibles de las 5 variables Por otro lado, si escribimos

```
> plot(Petal.Length ~ Sepal.Width + Sepal.Length)
```

`R` produce dos gráficos, uno de `Petal.Length` contra `Sepal.Width` y otro de `Petal.Length` contra `Sepal.Length` (en ambos casos `Petal.Length` aparece en el eje *y*). Sin embargo, no los produce de inmediato. Para que los gráficos aparezcan hay que apretar la tecla 'enter', pues ambos aparecen en la misma ventana. En cambio,

```
> plot(~ Sepal.Width + Sepal.Length + Petal.Width)
```

produce una matriz 3×3 de gráficos donde aparecen las combinaciones posibles de las tres variables presentes en la instrucción.

plot

Recordemos que el archivo `cars` es una matriz de datos con dos columnas, la primera corresponde a la velocidad y la segunda a la distancia necesaria para frenar el vehículo. Si escribimos

```
> plot(cars)
```

obtenemos un gráfico de la primera columna (velocidad) en el eje *x* y la segunda columna (distancia) en el eje *y*.
También es posible usar la instrucción `plot` con un objeto de clase `ts`, es decir, una serie de tiempo. Para mostrar un ejemplo usamos el archivo `nhtemp`, que tiene la temperatura media anual en grados Fahrenheit en New Haven, Connecticut, de 1912 a 1971.

```
> plot(nhtemp)
```

Como hemos visto, la función `plot` puede usarse con una gran variedad de argumentos y su efecto depende del argumento.

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

Argumentos

Veamos ahora algunos de los argumentos que pueden incluirse en la función `plot`. El argumento `type` controla el tipo de gráfico de acuerdo a las siguientes posibilidades

`type = 'p'`	Dibuja puntos. Es la opción por defecto.
`type = 'l'`	Dibuja curvas.
`type = 'b'`	Dibuja puntos unidos por curvas.
`type = 'o'`	Dibuja puntos con curvas superpuestas.
`type = 'h'`	Dibuja rectas verticales hasta el eje x.
`type = 's'`	Dibuja funciones escalera. El punto corresponde al extremo superior del segmento vertical.
`type = 'S'`	Dibuja funciones escalera. El punto corresponde al extremo inferior del segmento vertical.
`type = 'n'`	No dibuja la gráfica pero presenta los ejes cuyas coordenadas se determinan de acuerdo a los datos.

Argumentos

Veamos ejemplos de cada uno de estos casos, usando `cars`.

```
> plot(cars, type='p')
> plot(cars, type='l')
> plot(cars, type='b')
> plot(cars, type='o')
> plot(cars, type='h')
> plot(cars, type='n')
```

Para ver la diferencia entre las opciones `'s'` y `'S'` vamos a hacer dos gráficos en una página usando el archivo `pressure` y superponiendo los datos.

```
> par(mfrow=c(1,2))
> plot(pressure,type='s')
> points(pressure)
> plot(pressure,type='S')
> points(pressure)
> par(mfrow=c(1,1))
```

Argumentos

Para etiquetar los ejes se usan las instrucciones `xlab =` *'etiqueta'* , `ylab =` *'etiqueta'* :

```
> plot(cars, type='l', xlab='Velocidad',
    ylab='Distancia')
```

Para colocar un título en la gráfica usamos `main =` *'titulo'* mientras que `sub =` *'subtitulo'* crea un subtítulo que se coloca debajo del eje *x* con tamaño de letra más pequeño:

```
> plot(cars, type='l', xlab='Velocidad',
    ylab= 'Distancia', main ='Distancia de.
    Frenado vs Velocidad', sub='Datos tomados
    en 1920')
```

Argumentos

Podemos también controlar el tipo, grosor y color de las lineas con las instrucciones `lty`, `lwd` y `col`.

```
> plot(cars, type='l', xlab='Velocidad',
    ylab= 'Distancia', main = 'Distancia de
    Frenado vs. Velocidad', sub='Datos
    tomados en 1920', lty=2, lwd = 2, col=3)
```

El tipo de línea se puede especificar usando un código numérico o el nombre en inglés del tipo de trazo, colocado entre comillas. De igual manera el color se puede especificar con un múmero o usando el nombre en inglés, entre comillas. Para ver una lista de los colores disponibles se puede usar la instrucción `colors()`. Presentamos a continuación dos tablas que muestran los códigos y nombres en inglés para los tipos de línea y algunos colores básicos.

Argumentos

0	*blank*	(invisible)
1	*solid*	(continua)
2	*dashed*	– – –
3	*dotted*	. . . . .
4	*dotdash*	– · – · –
5	*longdash*	—— ——
6	*twodash*	—— – ——

1	*black*	negro
2	*red*	rojo
3	*green*	verde
4	*blue*	azul
5	*cyan*	cyan
6	*magenta*	magenta
7	*yellow*	amarillo
8	*grey*	gris

Tabla 3.1 Códigos para tipos y colores de línea.

Argumentos

Si se trata de una gráfica de puntos, de manera similar podemos especificar el tipo de símbolo a usar, su tamaño y color. Las opciones `pch` y `cex` permiten seleccionar el símbolo a usar y su tamaño. Veamos algunos ejemplos.

```
> plot(cars, pch=8, cex=1.2, col = 4)
> plot(cars, pch=15, cex=2, col = 5)
> plot(cars, pch=4, cex=0.5, col = 6)
```

También es posible usar cualquier carácter colocándolo entre comillas:

```
> plot(cars, pch='r', cex=1.2, col = 2)
> plot(cars, pch='~', cex=1.5, col = 1)
```

y modificar las escalas de los ejes con las instrucciones `xlim` y `ylim`:

```
> plot(cars,xlim=c(0,30), ylim=c(0,150))
```

Argumentos

Otros argumentos que se pueden usar son los siguientes

`log = 'x'`	
`log = 'y'`	
`log = 'xy'`	Cambia la escala de los ejes seleccionados a logarítmica.
`add = TRUE`	Superimpone el gráfico a la figura que se encuentre en la ventana activa.
`axes = FALSE`	Suprime la generación de los ejes. Es útil para añadir luego ejes particulares con la instrucción `axis()`. Por defecto se tiene `axis = TRUE`.

Ejercicio

Ejercicio

1. Haga una gráfica del conjunto de datos `women` con líneas y puntos superpuestos. Coloque 'Altura' como etiqueta en el eje x y 'Peso' en el y. Como título ponga 'Valores promedio de altura y peso', subtitulo: 'Mujeres de 30 a 39 años'.
2. Repita el gráfico anterior con una línea cortada de color azul y grueso 2.

Outline

Arreglos

Cuadros de Datos

Listas

Gráficos

La función plot

Argumentos

Otras Funciones

Otras Funciones

En `R` hay tres tipos de funciones gráficas:

- Funciones gráficas de alto nivel. Crean una nueva gráfica en la ventana, posiblemente con ejes, titulo, etiquetas, etc. El comando `plot` es un comando de alto nivel.
- Funciones gráficas de bajo nivel. Sirven para añadir información a una gráfica existente, tales como puntos, lineas, etiquetas, etc.
- Funciones gráficas interactivas, que permiten añadir o extraer información interactivamente de una gráfica existente, usando algún mecanismo como el ratón.

Veamos otras funciones de alto nivel que permiten crear otros tipos de gráficos.

curve

Para graficar una función continua sobre un conjunto específico de valores se puede usar la función `curve` cuya sintaxis es `curve(expr, from, to, add=FALSE,...)`.
Los argumentos son

`expr`	Fórmula escrita usando `x` como variable,
`from, to`	Rango de valores sobre el cual se hará la gráfica de la función,
`add`	Valor lógico. Si es cierto (`'TRUE'`), la curva se añade a la gráfica activa.

Veamos un ejemplo

```
> curve(x^3-3*x,-2,2)
> curve(x^2-2, add=TRUE, col='blue')
```

Ejercicio

Ejercicio

1. Haga una gráfica de la función $\cos(3x)$ de 0 a 3, de color azul.
2. Superponga la gráfica de $\sin(2x)$ de color rojo.

hist

La función para crear histogramas en `R` es `hist` y su sintaxis es

```
hist(x, breaks = "Sturges", freq = NULL,
        probability = !freq,...).
```

Los argumentos son
`breaks` que puede ser

- un vector con los extremos de las celdas del histograma,
- un número que indica cuántas celdas debe tener el histograma,
- una serie de caractéres que corresponden al nombre de un algoritmo para calcular el número de celdas,
- Una función para calcular el número de celdas.

En los últimos tres casos el valor se toma como una sugerencia.

hist

`freq` Parámetro lógico. Si es cierto (`'TRUE'`) el histograma representa el número de datos que hay en cada clase. Si es falso (`'FALSE'`) representa frecuencias relativas.
`probability` Es lo contrario de `frequency`.

Hay argumentos adicionales que se pueden ver usando `help(hist)`.

hist

Para obtener el número de celdas hay tres algoritmos: `Sturges`, que se usa por defecto, `Scott` y `FD`, por Friedman-Diaconis. El algoritmo de Sturges usa la siguiente fórmula para calcular el número de intervalos

$$\lceil \log_2(n) + 1 \rceil$$

donde n es el tamaño de la muestra y $\lceil\ \rceil$ es el operador techo, que calcula el menor entero mayor o igual a la cantidad entre los símbolos. La fórmula de Scott está basada en la desviación típica σ: $3.5\sigma n^{-1/3}$, mientras que la de Friedman y Diaconis se basa en el rango inter-cuartil (*ric*): $2 \cdot ric \cdot n^{-1/3}$.

hist

Veamos algunos ejemplos,

```
> hist(Sepal.Width)
> hist(Sepal.Width,breaks='Scott')
> hist(Sepal.Width,breaks='FD')
> hist(Sepal.Width,breaks=18,freq=FALSE)
```

Vemos que los tres primeros gráficos son iguales. En cuanto al último, observamos que en lugar de 18 clases se tomaron 24. Si cambiamos la variable a `Petal.Length` obtenemos

```
> hist(Petal.Length)
> hist(Petal.Length,breaks='Scott')
> hist(Petal.Length,breaks='FD')
> hist(Petal.Length,breaks=18,freq=FALSE)
```

Vemos ahora que las fórmulas de Scott y FD dan un menor número de clases que la de Sturges y, de nuevo, la última gráfica no tiene exactamente 18 clases.

Densidades

Aunque no existe una función residente en `R` para graficar la densidad estimada a partir de un conjunto de datos, es posible combinar las funciones `density` y `plot` para obtener una gráfica de la densidad. Estas gráficas con frecuencia son preferibles a los histogramas, pero al igual que ellos, su forma puede depender bastante de los parámetros seleccionados. La sintaxis de la instrucción es

```
density(x, bw = "nrd0", adjust = 1, kernel =
 c("gaussian", "epanechnikov","rectangular",
    "triangular", "biweight", "cosine",
              "optcosine"), ...).
```

Los argumentos son:

`bw`	El ancho de la ventana (*bandwidth*) para la estimación de la densidad.
`adjust`	Un número que se multiplica por el ancho de la ventana.
`kernel`	Indica el tipo de núcleo de usar para la estimación. Por defecto se toma el núcleo Gaussiano.

Densidades

En general resulta más conveniente modificar el ancho de la ventana de estimación a través del parámetro `adjust`. Tanto el ancho de la ventana como la selección de núcleo pueden alterar la apariencia del gráfico.

```
> plot(density(Sepal.Length))
> plot(density(Sepal.Length,adjust=2))
> plot(density(Sepal.Length,adjust=0.5))

> plot(density(Sepal.Length,kernel='epa'))
> plot(density(Sepal.Length,kernel='rec'))
> plot(density(Sepal.Length,kernel='tri'))
> plot(density(Sepal.Length,kernel='biw'))
> plot(density(Sepal.Length,kernel='cos'))
> plot(density(Sepal.Length,kernel='opt'))
```

Ejercicio

Ejercicio

1. Para el conjunto de datos `trees` haga histogramas para cada una de las variables con 6, 12 y 18 clases.
2. Haga una gráfica de la densidad estimada para las variables del mismo conjunto usando distintos valores de `adjust` (0.5, 0.75, 1).

boxplot

El rango intercuartil se usa para construir el diagrama de caja o "*boxplot*". Este tipo de diagramas tiene varias versiones, pero en general se representa el rango intercuartil por una caja o rectángulo, de modo que los extremos del rectángulo están ubicados en el primer y tercer cuartil, tal como se indica en la figura 1. Dentro del rectángulo se indica por una línea o un punto la ubicación de la mediana. Fuera del rectángulo se dibujan dos segmentos, llamados 'bigotes' que llegan hasta los datos más lejanos que estén a una distancia menor o igual a $1.5 \times (ric)$ del rectángulo, donde *ric* representa el rango intercuartil. Cualquier punto que no esté incluido en este rango se representa individualmente y se considera un punto atípico (outlier).

boxplot

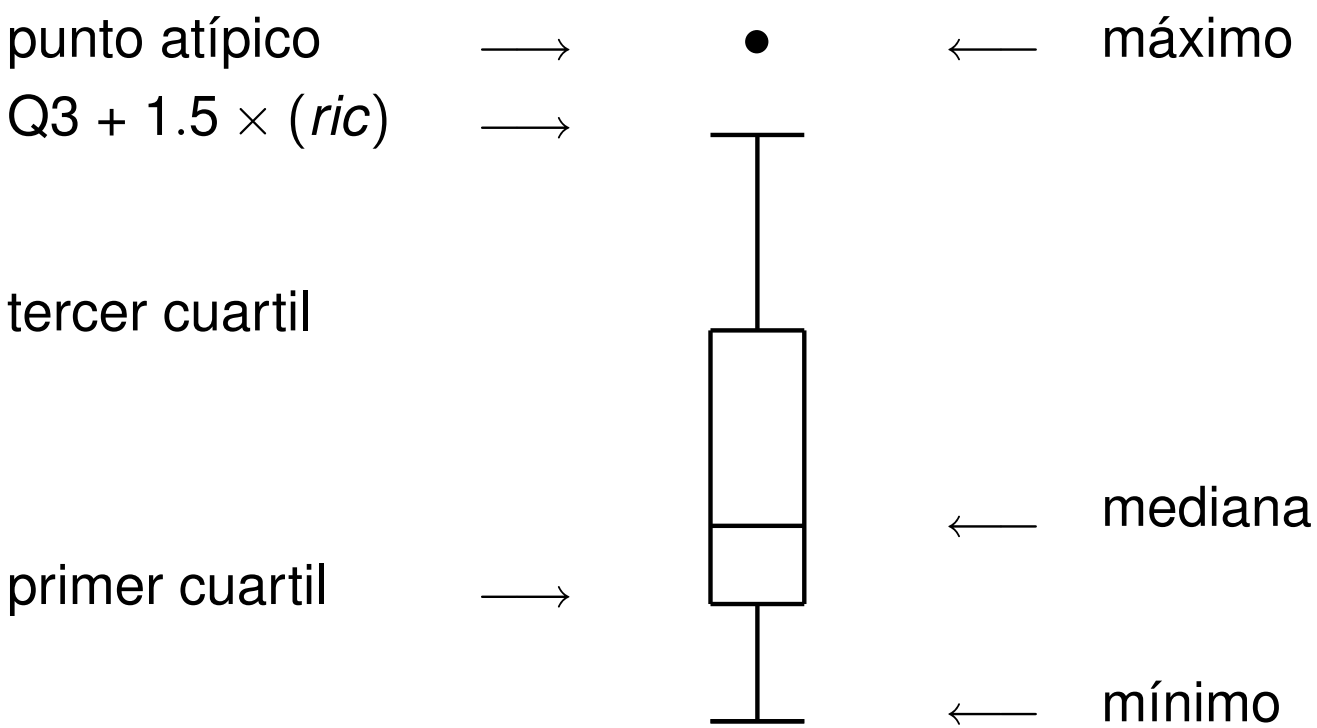

Figura 1
Diagrama de Cajas o 'box plot'.

boxplot

Los diagramas de caja son muy populares porque dan, de manera visual, una gran cantidad de información sobre la distribución de los datos. A continuación generamos una matriz de 10 columnas, en cada una de las cuales hay un vector de tamaño 100 de números aleatorios generados a partir de la distribución Gaussiana y luego hacemos los boxplots correspondientes. Finalmente, hacemos los boxplots para los datos `iris`.

```
> normat <- matrix(rnorm(1000),ncol=10)
> boxplot(data.frame(normat))
> boxplot(iris[,1:4])
```

qqnorm

Un qq-plot normal es un método gráfico para estudiar la bondad de ajuste de un conjunto de datos a la distribución normal. Si $q_i = i/(n+1)$ graficamos los puntos

$$(F_n^{-1}(q_i), \Phi^{-1}(q_i)), \qquad i = 1, \ldots, n.$$

donde F_n es la función de distribución empírica, es decir, hacemos una gráfica de los cuantiles empíricos de los datos vs. los cuantiles correspondientes de la distribución normal. Observe que no es necesario seleccionar los parámetros de ubicación y escala, μ y σ, ya que $F_n^{-1}(q_i) = x_{(i)}$, donde $x_{(i)}$ es el i-ésimo estadístico de orden la muestra, y

$$F_n^{-1}(q_i) \approx \Phi_{\mu,\sigma}^{-1}(q_i) = \mu + \sigma\Phi^{-1}(q_i)$$

implican que la gráfica de

$$(\Phi^{-1}(q_i), x_{(i)})$$

estará cerca de la recta $y = \mu + \sigma x$.

qqnorm

El punto de corte con el eje *y* y la pendiente de la recta nos dan estimaciones de μ y σ.

En R la función `qqnorm(x)` hace un qq-plot normal de los datos `x`. La función `qqline(x)` traza una línea recta que pasa por el primer y tercer cuartil.

```
> qqnorm(normat)
> qqline(normat)
```

qqplot

Esta instrucción es similar a la anterior pero en lugar de comparar los cuantiles de la muestra con los de la distribución normal, permite comparar dos muestras. Es posible combinarla con la función `q`*dist*, que calcula los cuantiles de la distribución *dist*, para comparar los cuantiles de una muestra con los cuantiles de una distribución dada.
Como ejemplo, generamos dos muestra de tamaño 100 de la distribución normal y las comparamos y luego comparamos con los cuantiles de la distribución `t` con dos grados de libertad:

```
> muestra.1 <- rnorm(100)
> muestra.2 <- rnorm(100)
> qqplot(muestra.1,muestra.2)
> cuantiles.t <- qt(1:100/100,df=2)
> qqplot(muestra.1,cuantiles.t)
> qqplot(muestra.2,cuantiles.t)
```

stem

El diagrama de tallo y hojas (*stem and leaf chart*) es una manera ingeniosa de obtener información sobre la distribución de un conjunto de datos, similar a la que se obtiene con un histograma, pero sin perder la información numérica. Para ver como se construye consideramos los siguientes datos, que representan la circunferencia de 31 arboles medida a 4 pies y 6 pulgadas del suelo. Estos datos corresponden a la variable `Girth` del archivo de datos `trees`

8.3	8.6	8.8	10.5	10.7	10.8	11.0	11.0
11.1	11.2	11.3	11.4	11.4	11.7	12.0	12.9
12.9	13.3	13.7	13.8	14.0	14.2	14.5	16.0
16.3	17.3	17.5	17.9	18.0	18.0	20.6	

Para hacer un diagrama de tallo y hoja dividimos los números de la tabla en dos partes, la izquierda es el tallo y la derecha la hoja.

stem

En este caso dividimos los números en el punto decimal, de modo que para el diámetro 11.7 el tallo es 11 y la hoja es 7. En un diagrama de este tipo listamos todos los tallos a la izquierda una sola vez y todas sus hojas van en la misma fila, a la derecha. Para obtener el diagrama correspondiente a los datos anteriores escribimos

```
 > stem(trees$Girth, scale = 2)
The decimal point is at the |
 8 | 368
 9 |
10 | 578
11 | 00123447
12 | 099
13 | 378
14 | 025
15 |
16 | 03
17 | 359
18 | 00
19 |
20 | 6
```

stem

Si repetimos la instrucción sin la opción `scale` obtenemos una versión diferente.

```
> stem(trees$Girth)
The decimal point is at the |
 8 | 368
10 | 57800123447
12 | 099378
14 | 025
16 | 03359
18 | 00
20 | 6
```

barplot

El diagrama de barras (`barplot`) es útil para representar en barras el número de veces que ocurre un evento. Es posible usarla con datos numéricos o con datos categóricos para los cuales hemos usado la función `table`. Para mostrar un primer ejemplo, vamos a generar una muestra de 100 números de la distribución de Poisson con parámetro 5 y luego hacemos el diagrama. La segunda versión del diagrama usa la instrucción `rainbow` para darle color a las barras.

```
> tN <- table(Ni <- rpois(100, lambda=5))
> barplot(tN)
> r <- barplot(tN, col=rainbow(20))
```

barplot

Para el segundo ejemplo usamos el archivo `esoph` que contiene datos sobre un estudio de cancer en el esófago. Las variables son la edad `agegp` (en 6 categorías), consumo de alcohol `alcgp` y de tabaco `tobgp` (en 4 categorías cada uno).

```
> barplot(table(esoph$agegp))
> barplot(table(esoph$agegp,esoph$tobgp))
> barplot(table(esoph$agegp,esoph$alcgp))
> barplot(table(esoph$tobgp,esoph$alcgp),
    beside=TRUE)
```

pairs

Esta función produce una matriz de gráficos. El argumento puede ser una matriz o cuadro de datos o también una fórmula, como por ejemplo `~ x + y + z`. En este caso cada sumando representa una variable a ser usada en la construcción de los pares de gráficos. Por ejemplo,

```
> pairs(iris)
> pairs( ~ Sepal.Length + Petal.Length +
    Petal.Width)
> pairs(iris[1:4], main = "Anderson's Iris
    Data - 3 species", pch = 21, bg =
    c("red", "green3", "blue")
    [unclass(iris$Species)])
> pairs(~ Fertility + Education + Catholic,
    data = swiss, subset = Education < 20,
    main = "Swiss data, Education < 20")
> pairs(USJudgeRatings)
```

coplot

Esta instrucción produce gráficas condicionadas. La sintaxis es `coplot(fórmula, datos, ...)`. La fórmula describe la forma de la gráfica condicionada. La fórmula `y ~ x | a` indica que se harán gráficas de `y` contra `x`, condicionando por el valor de la variable `a`. En cambio, una fórmula del tipo `y ~ x | a * b` indica que las gráficas deben condicionarse en los valores de las dos variables `a` y `b`. Veamos algunos ejemplos tomados de la ayuda de `R`.

```
> coplot(lat ~ long | depth, data = quakes)
> given.depth <- co.intervals(quakes$depth,
    number=4, overlap=.1)
> coplot(lat ~ long | depth, data = quakes,
    given.v=given.depth, rows=1)
```

coplot

```
> ll.dm <- lat  long | depth * mag
> coplot(ll.dm, data = quakes)
> coplot(ll.dm, data = quakes, number=c(4,7),
    show.given=c(TRUE,FALSE))
> coplot(ll.dm, data = quakes, number=c(3,7),
    overlap=c(-.5,.1))
```

Ejercicio

Ejercicio

1. El archivo `mtcars` tiene información sobre 11 variables para 32 modelos de carros. Use la ayuda u otra función para identificar los nombres de las variables.
2. Utilice la función `pairs` para obtener graficas a pares de todas las variables del archivo.
3. Haga ahora gráficas a pares del rendimiento, desplazamiento, potencia y peso.
4. Usando la instrucción `coplot` haga una gráfica de rendimiento contra desplazamiento condicional a los valores del número de cilindros y el tipo de transmisión.
5. Para mejorar la gráfica anterior, use los condicionantes como factores.

persp

Esta función dibuja la perspectiva de superficies en el plano *xy*. La sintaxis es `persp(x = seq(0, 1, len = nrow(z)), y = seq(0, 1, len = ncol(z)), z, ...)`. Los vectores `x, y` tienen la ubicación de las rectas que forman la retícula sobre la cual se miden los valores de `z`. Deben estar en orden ascendente y por defecto se toman valores igualmente espaciados entre 0 y 1. Si `x` es una matriz se toman los valores `x$x` y `x$y` por `x`, `y`, respectivamente. `z` es una matriz que contiene los valores a ser graficados.

```
> 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
> persp(x,y,z, theta=30, phi=30, expand=0.5,
    col='lightblue')
```

contour

Esta función crea una gráfica de contornos o añade contornos a una gráfica. La sintaxis es

```
contour(x = seq(0,1, len = nrow(z)), y =
seq(0,1, len = ncol(z)), z, nlevels = 10,
levels = pretty(zlim, nlevels), labels =
NULL,...).
```

Las variables `x, y, z` tienen las mismas características que para la función `persp`. El argumento `nlevels` indica el número de niveles para el trazado de los contornos, a menos que se indiquen los niveles deseados usando el argumento `levels`. Finalmente `labels` es un vector con las etiquetas correspondientes a las curvas de contorno. Si no se asigna ningún valor a este argumento, se usan los niveles como etiquetas.

contour

```
> x <- y <- seq(-4*pi, 4*pi, len = 27)
> r <- sqrt(outer(x^2, y^2, "+"))
> opar <- par(mfrow=c(1,1),
    mar=c(5.1,4.1,4.1,2.1))
> par(mfrow = c(2, 2), mar = rep(0, 4))
> for(f in pi^(0:3))
> contour(cos(r^2)*exp(-r/f), drawlabels =
    FALSE, axes = FALSE, frame = TRUE)
> par(opar)
```

Funciones Gráficas

`sunflowerplot(x,y)`	Gráfica bivariada. Puntos de coordenadas similares se representan como flores y el número de pétalos equivale al número de puntos
`stripchart(x)`	Dibuja los puntos de `x` en una línea
`matplot(x,y)`	Gráficas bivariadas de las columnas de `x` vs. columnas de `y`
`dotchart(x)`	Gráfica de puntos para un cuadro de datos `x`
`plot.ts(x)`	Gráfica de un objeto de clase `ts` contra tiempo
`filled.contour(x,y,z)`	Gráfica de contornos con áreas en color
`image(x,y,z)`	Similar a la gráfica de contornos con los datos representados por colores
`stars(x)`	Gráfica con forma de una estrella con rayos proporcionales a los valores de las coordenadas de `x`

Ejercicio

Ejercicio

1. Haga una gráfica tipo `sunflowerplot` para las variables `Petal.Length` y `Petal.Width` de `iris`
2. Usando el comando `stripchart` y el conjunto de datos `Orange` haga una gráfica de `circumference` como función de `age`.
3. Haga un diagrama `stars` para las 7 primeras variables del archivo `mtcars`. Use la opción `full=FALSE` para obtener un gráfico más claro.