Lectura, manipulación y análisis de datos en R

F. Tusell*

Curso 2.004-2.005

Índice

1. Introducción **2**

2. Lectura e importación de datos **2**
2.1. Introducción directa 2
2.2. Lectura directa en formatos propios de R 2
2.3. Importación de formatos propios de otros programas 3
2.4. Conexión a gestores de base de datos 3

3. Datos fechados **4**
3.1. Datos simplemente fechados 4
3.2. Series temporales regulares (equiespaciadas) 4
3.3. Series temporales irregularmente espaciadas 4

4. Comentarios al Ejemplo 1. **5**

5. Comentarios al Ejemplo 2. **13**

6. Información adicional **17**

A. Listado del Ejemplo 1 **19**

B. Listado del Ejemplo 2 **21**

C. Datos en `tomates.csv` (primeras líneas) **23**

D. Datos en `naosoi.dat` (primeras líneas) **23**

*Actualización del día 7 de marzo de 2005. La última versión de este documento, posiblemente más reciente, puede obtenerse de `http://www.et.bs.ehu.es/~etptupaf`

1. Introducción

Con frecuencia, la manipulación de los datos es igual o más laboriosa que el análisis estadístico subsiguiente. Un uso adecuado de las facilidades de importación, transformación y presentación de datos que ofrece R puede redundar en drásticas reducciones de trabajo, además de favorecer la legibilidad y presentación. El resultado son salidas autoexplicativas en que los errores son de comisión menos probable y de detección más sencilla.

Los Anexos A y B hacen uso de los datos en los Anexos C y D para ilustrar el uso de algunas instrucciones. Las Secciones siguientes hasta la Sección 4 introducen algunos conceptos. Las Secciones 4 y 5 comentan los ejemplos en los Anexos.

2. Lectura e importación de datos

Lo que sigue es sólo un resumen: para una discusión más detallada véase [13].

2.1. Introducción directa

Si los datos son pocos, podemos teclearlos directamente en un vector o *dataframe*. Por ejemplo, así:

```
a <- c(1,2,3,4)
b <- matrix(a,2,2,byrow=FALSE)
```

La primera instrucción asigna directamente un vector; la segunda, emplea el contenido del mismo para llenar una matriz 2×2 por columnas.

En ocasiones, tanto para matrices como para *dataframes*, será de utilidad la función `edit`. Una sentencia como

```
d <- edit(b)
```

abre una rudimentaria hoja de cálculo en la que podemos modificar o añadir a la matriz (o *dataframe*) `b` los elementos que deseemos, para luego guardar el resultado en `d` (`b` y `d` pueden coincidir, en cuyo caso el contenido previo de `b` resulta machacado).

Hay que destacar que `b` ha de preexistir. Sus dimensiones, no obstante, pueden modificarse por la función `edit` "al vuelo", añadiendo filas o columnas.

2.2. Lectura directa en formatos propios de R

En ocasiones, los datos puede proceder de un programa que proporciona los datos de forma directamente procesable por R, o en un formato que R puede reconocer. Lo primero no es habitual, siendo el programa econométrico Gretl[1] una notable excepción: produce objetos con sufijo `.R` que pueden directamente leerse en R mediante una instrucción como:

```
obj <- source("objeto.R")
```

[1] Más información acerca de este programa en `http://gretl.sourceforge.net/`.

También S-PLUS puede producir objetos del mismo tipo: desde S-PLUS, la función `dput` producirá un objeto que puede ser recuperado desde R con la función `dget`: pero ello no funcionará en todos los casos ni con todas las versiones de ambos programas, por lo que se impone la prueba[2].

2.3. Importación de formatos propios de otros programas

R no es particularmente bueno importando datos generados por otros programas. No obstante, el paquete recomendado `foreign` lee ficheros en formatos propios de, entre otros, SPSS, SAS, Minitab, Stata, Epi y S. Hay que destacar que los formatos pueden presentar ligeras variaciones de unas versiones a otras (típicamente, las versiones posteriores leen ficheros producidos mediante las precedentes, pero no al revés). Podemos así encontrar que `foreign` nos permite leer, por ejemplo, "worksheets" de Minitab de una versión determinada y las anteriores, pero no las posteriores.

Se impone por tanto también la prueba.

Adicionalmente al paquete citado,

1. Las funciones en el paquete `e1071` permite leer datos en formato OCTAVE (un *look-alike* del más conocido MATLAB).

2. La función `read.xls` en el paquete `gdata` permite leer hojas de cálculo creadas por Excel en *data frames* [3].

2.4. Conexión a gestores de base de datos

Los gestores de base de datos son programas que permiten crear y administrar ficheros accediendo a ellos de forma cómoda, ordinariamente a traves de variantes dialectales de SQL. Si tenemos una tabla de nombre `CENSO` conteniendo las variables `NOMBRE`, `APELLIDOS` y `EDAD` (entre otras), podríamos recuperar todos los casos de edad superior a 45 años así:

```
SELECT NOMBRE,APELLIDOS FROM CENSO WHERE EDAD > 45 ;
```

Sobre SQL puede consultarse [9]; sobre PostgreSQL, [10].

Hay varios paquetes que permiten conectar R a gestores de bases de datos, en la misma o en otra máquina diferente: son `RPgSQL` (para PostgreSQL), `ROracle` (para Oracle), `RMySQL` (para MySQL) y `RODBC` (para cualquier origen de datos ODBC). Una descripción de algunos de estos paquetes puede encontrarse en [11].

Con varios de los paquetes señalados las bases de datos a las que se conecta R pueden estar en la misma u otra máquina: en este último caso, debemos tener algún modo de conectar (normalmente, sobre protocolos TCP/IP).

Con el paquete `RODBC` el origen de datos puede ser una cualquiera de muchas cosas. En particular, puede ser un fichero local de Microsoft Access, lo que proporciona un modo fácil de importar Access a R. Supongamos que tenemos

[2] La pareja de funciones `dump` y `source` ofrece una alternativa que puede funcionar cuando `dput` y `dget` no lo hacen.

[3] Hay que notar que para ellos se necesita Perl instalado en el sistema, lo que es completamente "standard" en Unix/Linux, pero no en Windows. Adicionalmente, hojas en formato `.xls` creadas por otros programas como Open Office, pueden no ser leidas. Una vez más, se impone la prueba.

una base de datos llamada `Gastos` y en ella una tabla de nombre `Empleados`, que suponemos ubicada en la raíz del disco `C:` de una máquina Windows. Para importar su contenido a una *dataframe* de igual nombre, bastaría teclear:

```
library(RODBC)
canal1    <- odbcConnectAccess("C:\\Gastos")
Empleados <- sqlQuery(canal1,"select * from Empleados")
```

Nótese que como lenguaje de interrogación se emplea SQL. La facilidad para poblar una *dataframe* realizando una consulta a una base de datos externa, permite tratar ficheros muy grandes de los que sólo se importan las observaciones/variables que interesan. SQL es además una herramienta excelente para seleccionar casos que verifiquen condiciones complejas de expresar en R.

3. Datos fechados

3.1. Datos simplemente fechados

R posee clases específicas de objetos para manejar fechas, y lo hace con gran flexibilidad. Podemos definir una variable como de clase `POSIXt` o `POSIXct` (dos *standares* para la representación de fechas) mediante la instrucción `as.Date` (véase el uso de la misma en el Anexo A). Las fechas pueden ser días (ejemplo: 20-04-2003) o días más tiempos (ejemplo: 20-04-2003 10:56:03) con una resolución de segundos, e incluir una zona horaria.

Con objetos de estas clases se pueden hacer algunos cálculos como la diferencia entre dos fechas consecutivas, etc. Véase una descripción en [12].

Además de estas clases de objetos para representar fechas, que son las propias y nativas de R, hay otros paquetes (`chron` y `dates`) que ofrecen facilidades análogas.

Podemos guardar datos en *dataframes* una de cuyas columnas contiene fechas de esta clase. Tenemos así datos fechados, pero *no* de serie temporal.

3.2. Series temporales regulares (equiespaciadas)

Es frecuentemente el caso con series mensuales. R puede construir objetos de tipo `ts` como ilustra el ejemplo en la Sección 5. Para dichos objetos hay métodos específicos.

3.3. Series temporales irregularmente espaciadas

En multitud de campos, notablemente en Economía y en Finanzas, las series temporales no son equiespaciadas: hay días laborables y festivos, que introducirían valores `NA` en una serie equiespaciada. A veces, simplemente, las observaciones se presentan en momentos cualesquiera —como las transacciones en un mercado—.

Hay varias posibilidades para el manejo de este tipo de datos. Diferentes paquetes definen diferentes tipos de series irregulares. Así, `tseries` (orientado a Finanzas) define series de la clase `irts`. Los paquetes `fBasics` y `fSeries` —también orientados a Finanzas— definen objetos de clase `timeDate` y `timeSeries`. El paquete `its` define objetos de clase `its`. El paquete `zoo`, por su parte, define aún otra clase de objetos para representar objetos indexados.

4. Comentarios al Ejemplo 1.

Comentamos a continuación algunas de las órdenes que aparecen en el ejemplo extenso del Apéndice A. Los números de línea hacen referencia a dicho Apéndice, en que las instrucciones comentadas pueden verse en contexto.

Los datos corresponden a tomates recogidos de cada una de veinte plantas, algunas de las cuales habían sido sometidas a un tratamiento (despunte, dejando una sóla flor de cada grupo) y otra no. El objetivo era examinar si el dejar menos flores por planta favorecía la producción de tomates de mayor tamaño y peso.

En la línea 1 se ha empleado un `read.table` para leer los datos en el fichero `tomates.csv` (las primeras líneas aparecen en el Apéndice C. La opción `header=T` especifica que la primera línea de dicho fichero proporciona los nombres de las variables.

```
> tomates <- read.table(file = "tomates.csv", header = TRUE)
```

Podemos examinar si la lectura ha producido los resultados apetecidos, listando una pocas líneas de la *dataframe* resultante y viendo su estructura:

```
> tomates[1:3, ]
```

```
  Planta      Fecha Peso
1     11 2003-07-07   90
2      1 2003-07-08  236
3      4 2003-07-09  133
```

```
> str(tomates)
```

```
'data.frame':        291 obs. of  3 variables:
 $ Planta: int  11 1 4 10 8 8 6 11 9 5 ...
 $ Fecha : Factor w/ 26 levels "2003-07-07","20..",..: 1 2 3 4 4 4 5 5 6 7 ...
 $ Peso  : int  90 236 133 183 105 52 91 197 176 27 ...
```

Observamos así que las fechas han sido tomadas como literales generando una v ariable cualitativa con 26 niveles (un *factor* en la terminología de R). Para hacer que R las considere como fechas a todos los efectos, podemos hacer:

```
> tomates[, "Fecha"] <- as.Date(as.character(tomates[, "Fecha"]),
+     format = "%Y-%m-%d")
```

tras de lo cual, las líneas 6 y 7 muestran los cambios experimentados por la *dataframe*:

```
> tomates[1:3, ]
```

```
  Planta      Fecha Peso
1     11 2003-07-07   90
2      1 2003-07-08  236
3      4 2003-07-09  133
```

```
> str(tomates)
```

```
'data.frame':        291 obs. of  3 variables:
 $ Planta: int  11 1 4 10 8 8 6 11 9 5 ...
 $ Fecha :Class 'Date'  num [1:291] 12240 12241 12242 12244 12244 ...
 $ Peso  : int  90 236 133 183 105 52 91 197 176 27 ...
```

La línea 12,

```
> attach(tomates)
```

permite referirnos en lo sucesivo a las clumnas de la *dataframe* `tomates` como si fueran variables en nuestro espacio de trabajo. Las líneas que siguen realizan Las líneas que siguen permiten aprender algo sobre los datos:

```
> summary(tomates)
```

```
     Planta          Fecha                Peso
 Min.   : 1.00   Min.   :2003-07-07   Min.   : 21.00
 1st Qu.: 6.00   1st Qu.:2003-08-01   1st Qu.: 55.00
 Median :10.00   Median :2003-08-16   Median : 73.00
 Mean   :10.62   Mean   :2003-08-18   Mean   : 79.55
 3rd Qu.:16.00   3rd Qu.:2003-08-26   3rd Qu.: 97.00
 Max.   :20.00   Max.   :2003-10-05   Max.   :236.00
```

```
> length(unique(Planta))
```

```
[1] 20
```

```
> length(unique(Fecha))
```

```
[1] 26
```

```
> xtabs(Peso ~ Planta)
```

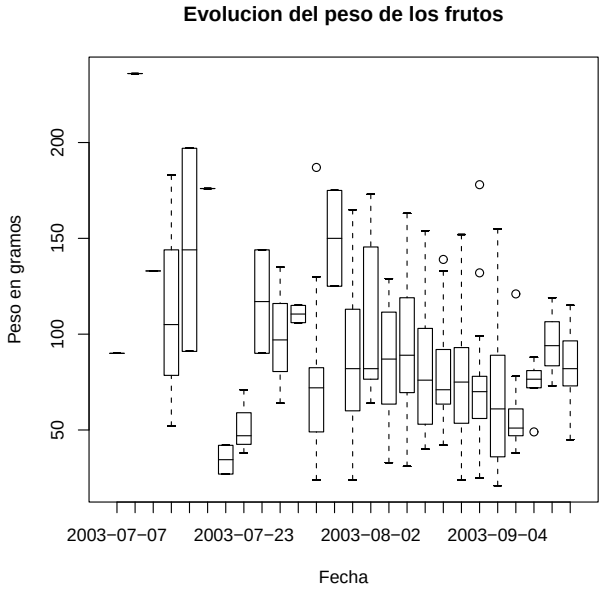
```
Planta
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
1582  835 1166  658  650  907 1139 1468 1627 1699 1364  344  252 1049  899 1159
  17   18   19   20
1616 1883 1337 1515
```

```
> xtabs(Peso ~ Fecha)
```

```
Fecha
2003-07-07 2003-07-08 2003-07-09 2003-07-11 2003-07-12 2003-07-17 2003-07-18
        90        236        133        340        288        176         69
2003-07-23 2003-07-24 2003-07-26 2003-07-27 2003-07-28 2003-07-30 2003-07-31
       156        234        296        221       2053        300       1895
2003-08-02 2003-08-09 2003-08-11 2003-08-14 2003-08-16 2003-08-22 2003-08-26
       763       1270       1533       1826       1207       2926       2413
2003-09-04 2003-09-12 2003-09-17 2003-10-02 2003-10-05
      2547        540        443        286        908
```

La instrucción siguiente genera un boxplot.

```
> boxplot(Peso ~ as.Date(Fecha), main = "Evolucion del peso de los frutos",
+     xlab = "Fecha", ylab = "Peso en gramos")
```



Podemos totalizar el peso total recolectado en cada fecha mediante:

```
> Grs <- aggregate(Peso, by = list(Fecha), sum)
> dimnames(Grs)[[2]] <- c("Fecha", "Total Gramos")
> print(Grs[1:5, ])
```

```
       Fecha Total Gramos
1 2003-07-07           90
2 2003-07-08          236
3 2003-07-09          133
4 2003-07-11          340
5 2003-07-12          288
```

Obsérvese que `by=` ha de ser una lista (aunque tenga un único componente). Análogamente podemos obtener el peso medio por fruto para cada fecha (línea 37):

```
> GrsMedio <- aggregate(Peso, by = list(Fecha), mean)
> dimnames(GrsMedio)[[2]] <- c("Fecha", "Media Gramos")
> print(GrsMedio[1:5, ])
```

```
       Fecha Media Gramos
1 2003-07-07      90.0000
2 2003-07-08     236.0000
3 2003-07-09     133.0000
4 2003-07-11     113.3333
5 2003-07-12     144.0000
```

Algunas de las veinte plantas recibieron un tratamiento:

```
> Tratadas <- c(1, 2, 5, 6, 9, 10, 13, 14, 17, 18)
> Controles <- (1:20)[-Tratadas]
> Status <- ifelse(Planta %in% Tratadas, "Tratada", "Control")
> Status <- as.factor(Status)
```

Podemos comparar los pesos de los frutos en el grupo de plantas tratadas y no tratadas mediante un contraste formal,

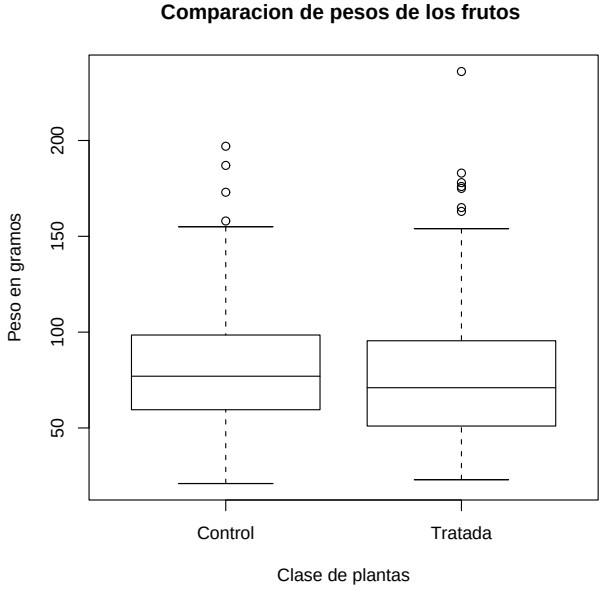
```
> t.test(Peso[Status == "Tratada"], Peso[Status == "Control"])

        Welch Two Sample t-test

data:  Peso[Status == "Tratada"] and Peso[Status == "Control"]
t = -1.0335, df = 288.61, p-value = 0.3022
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -12.432113   3.871429
sample estimates:
mean of x mean of y
 77.56410  81.84444
```

o haciendo un gráfico de cajas:

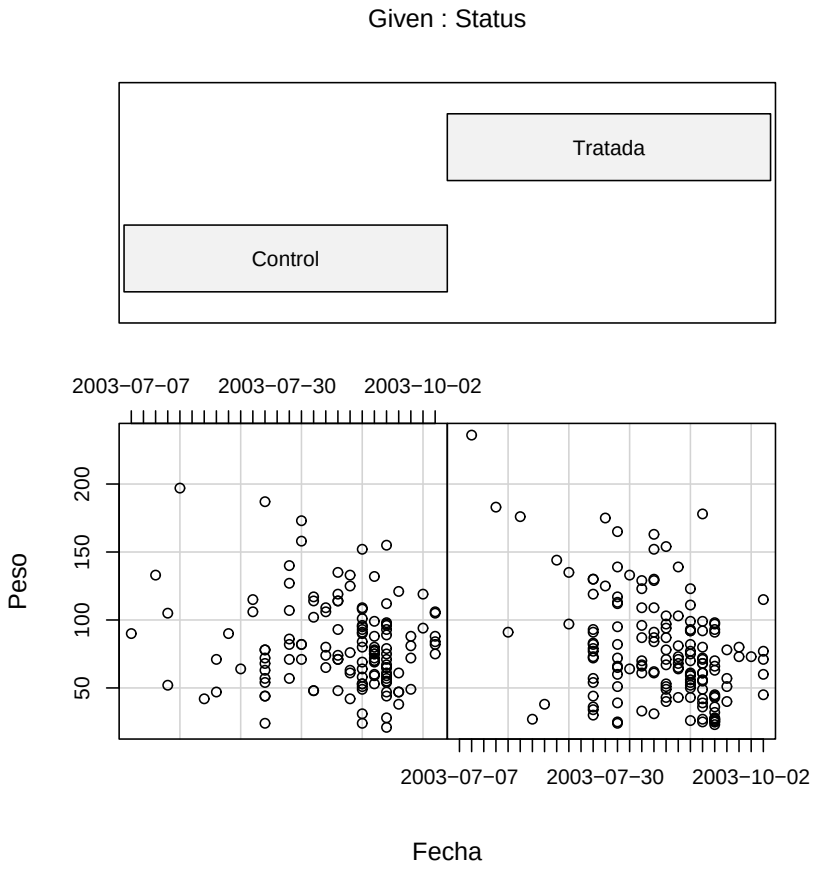
```
> boxplot(Peso ~ Status, main = "Comparacion de pesos de los frutos ",
+     xlab = "Clase de plantas", ylab = "Peso en gramos")
```



No parece haber diferencia en los pesos entre plantas tratadas y no tratadas.

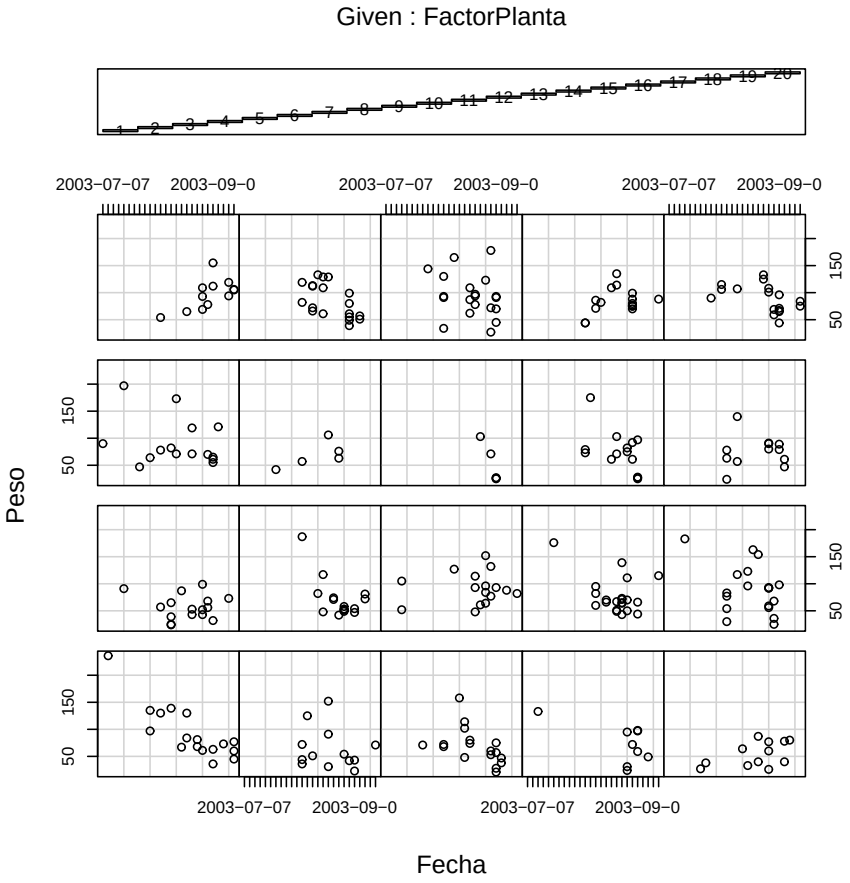
Alternativamente, los gráficos condicionados permiten ver como varía una magnitud (o varias, o la relación entre varias) en función de los valores que toma una o dos variables condicionantes:

```
> FechaFactor <- as.factor(as.Date(Fecha))
> coplot(Peso ~ FechaFactor | Status, xlab = "Fecha")
```



Podríamos tener gráficamente la misma información desglosada por planta:

```
> FactorPlanta <- as.factor(Planta)
> coplot(Peso ~ FechaFactor | FactorPlanta, xlab = "Fecha")
```

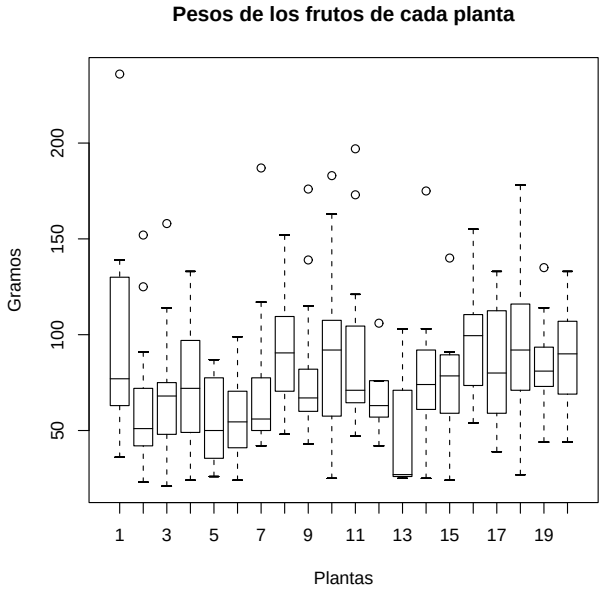


El gráfico anterior proporciona la evolución de los pesos en el tiempo para cada una de las plantas.

Un gráfico en que la variable en abscisas es un factor se transforma automáticamente en un boxplot. Si quisiéramos ver la distribución de pesos de los frutos de cada planta, podríamos escribir:

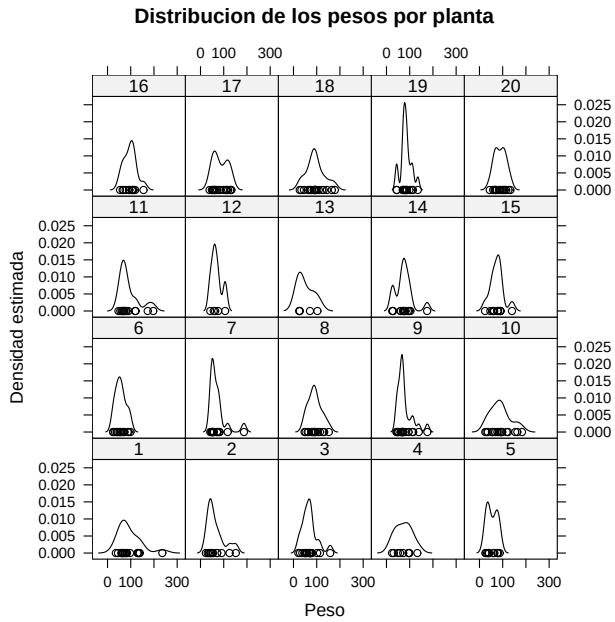
```
> plot(FactorPlanta, Peso, main = "Pesos de los frutos de cada planta",
+     ylab = "Gramos", xlab = "Plantas")
```

para obtener:

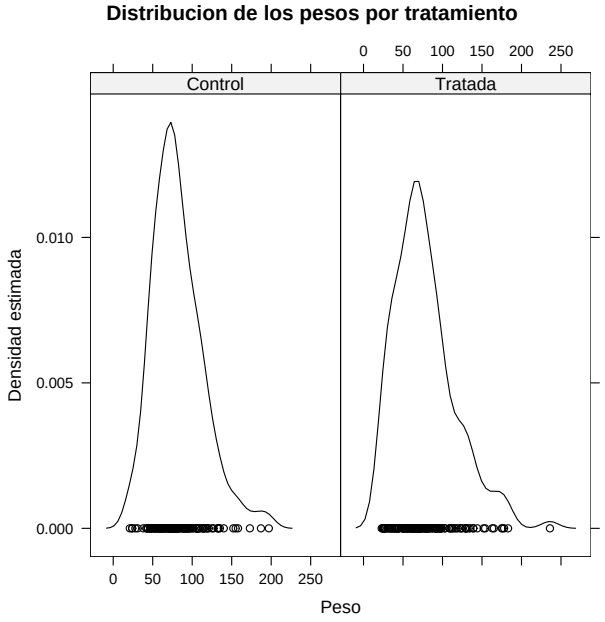


El paquete `lattice` ofrece otras posibilidades de representación gráfica:

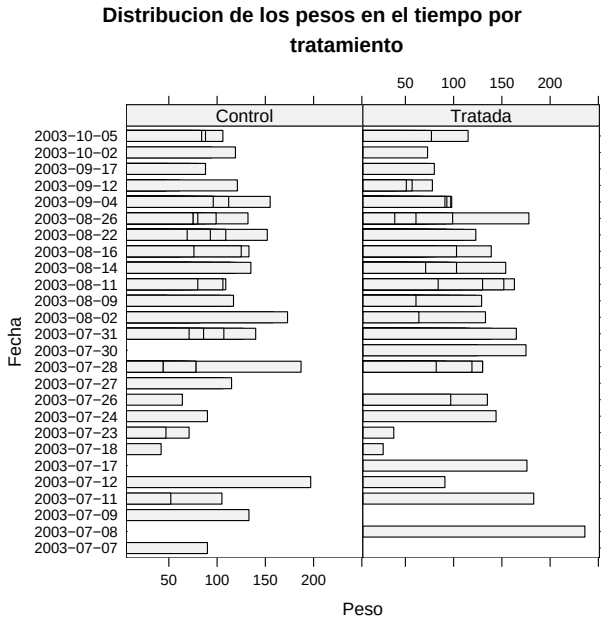
```
> library(lattice)
> print(densityplot(~Peso | FactorPlanta, layout = c(5, 4), xlab = "Peso",
+     ylab = "Densidad estimada", main = "Distribucion de los pesos por planta"))
```




```
> print(densityplot(~Peso | Status, layout = c(2, 1), xlab = "Peso",
+     ylab = "Densidad estimada", main = "Distribucion de los pesos por tratamiento"))
```




```
> print(barchart(as.character(as.Date(Fecha)) ~ Peso | Status,
+     layout = c(2, 1), xlab = "Peso", ylab = "Fecha", main = "Distribucion de los pesos e
```



Obsérvese que hemos de recurrir a `as.character(as.Date(Fecha))` para que las fechas aparezcan con su formato.

5. Comentarios al Ejemplo 2.

Las instrucciones,

```
> nao <- read.table(file = "naosoi.dat")
> nao[1:3, ]
```

```
         NAO SOI
Nov1899   NA  NA
Dec1899   NA  NA
Jan1900 1.38  -1
```

```
> attach(nao)
> par(mfrow = c(2, 1))
```

leen los datos, muestran las tres primeras observaciones y especifican el modo en que queremos los gráficos a continuación. La serie `NAO` (North Atlantic Oscillation) recoge la diferencia de presión barométrica entre Islandia y lasn Azores. `SOI` (Southern Oscillation Index) es un índice análogo que se supone explicativo del fenómeno recurrente frente a las costas peruanas conocido como *El Niño* (un calentamiento superficial del agua del mar con alteraciones climáticas concomitantes)

El bucle que sigue reconstruye la *dataframe* `nao` de modo que cada una de sus columnas sea un objeto de clase `ts`.

```
> cols <- names(nao)
> lcols <- length(nao)
> for (i in 1:lcols) {
+     nao[[i]] <- temp <- ts(nao[[i]], start = c(1900, 1), frequency = 12)
+     temp <- na.omit(temp)
+     temp2 <- (attributes(temp)$tsp)[1]
+     anyo <- floor(temp2)
+     mes <- round(1 + 12 * (temp2%%1))
+     ts.plot(window(temp, start = c(anyo, mes)), gpars = list(ylab = cols[i],
+         xlab = "Año", main = paste(cols[i], ": datos brutos",
+             sep = "")))
+ }
```

Diseccionemos dicho bucle. La primera línea,

```
> nao[[i]] <- temp <- ts(nao[[i]], start = c(1900, 1), frequency = 12)
```

se limita a redefinir las columnas de la *dataframe* `nao` como series temporales. La estructura del objeto resultante es muy simple, y puede ponerse en evidencia tecleando

```
> temp <- na.omit(temp)
> temp2 <- (attributes(temp)$tsp)[1]
```

Vemos que en `$tsp` se almacenan las abscisas temporales en forma de años fraccionarios. Hemos tomado cada una de las columnas de `nao`, las hemos guardado temporalmente en `temp` y hemos prescindido de sus observaciones faltantes. Como estas observaciones faltantes pueden estar al principio, se impone recomputar el año y mes de inicio de los datos no faltantes, Es lo que hacen las órdenes:

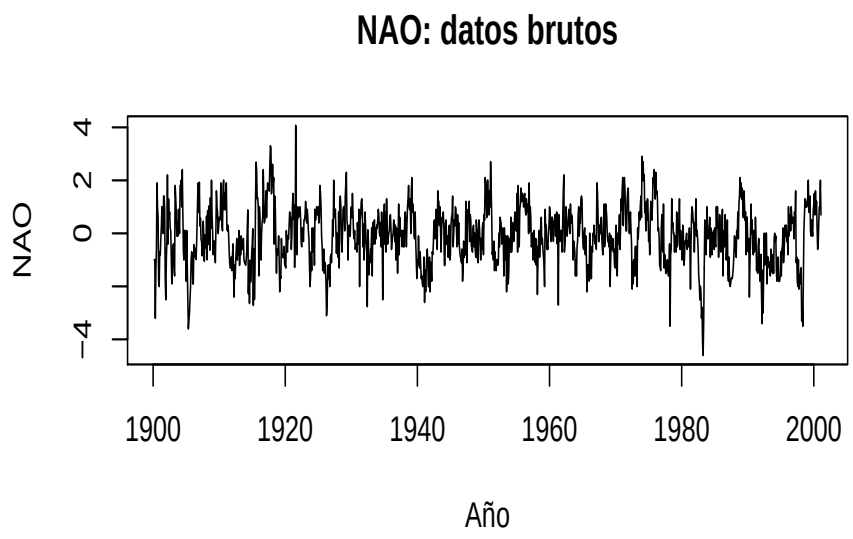
```
> anyo <- floor(temp2)
> mes <- round(1 + 12 * (temp2%%1))
```

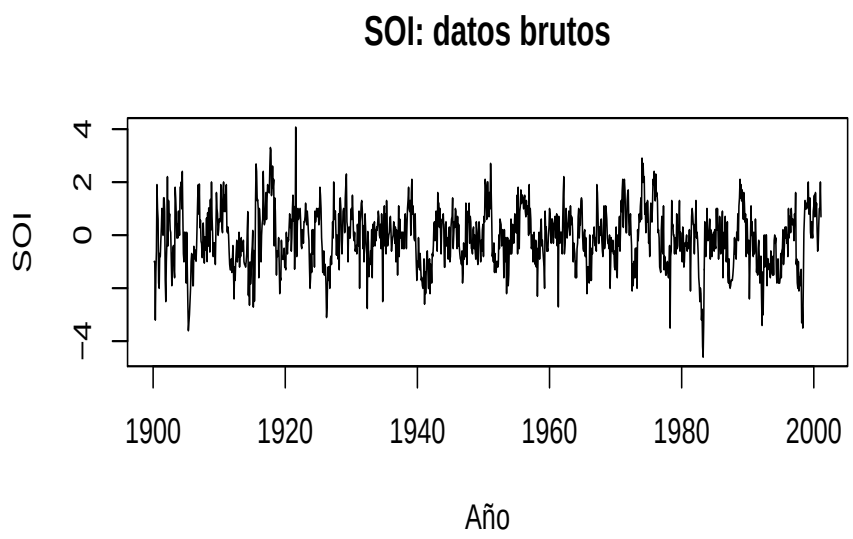
que de los datos presentes en `temp2` computan el año y mes de inicio.

La instrucción `ts.plot` al final genera el gráfico a continuación. Aunque sólo hay dos columnas en la *dataframe* y podría perfectamente haberse escrito dos veces la misma orden, el modo de hacerlo mediante un bucle se generaliza mejor al caso en que tenemos multitud de series temporales como columnas de una misma *dataframe* y queremos obtener gráficos para todas.

Obsérvese que se hace uso del hecho de que una *dataframe* es en realidad una lista, cuyas columnas pueden referenciarse de diversos modos. Por ejemplo, podemos acceder a la segunda colimna de `nao` de una cualquiera de las siguientes formas:

```
> nao$SOI
> nao[, "SOI"]
> nao[[2]]
> nao[, 2]
```

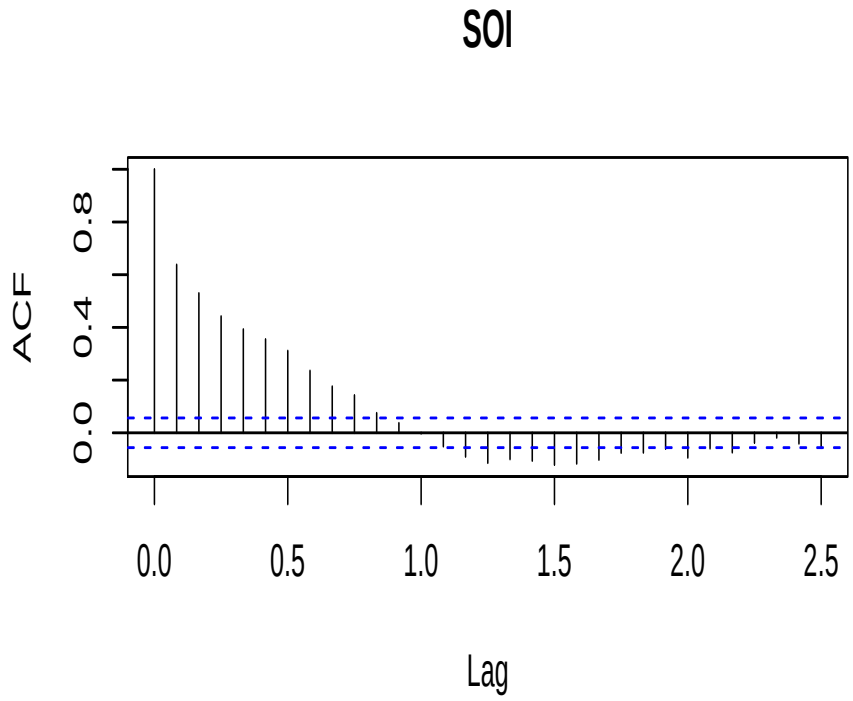


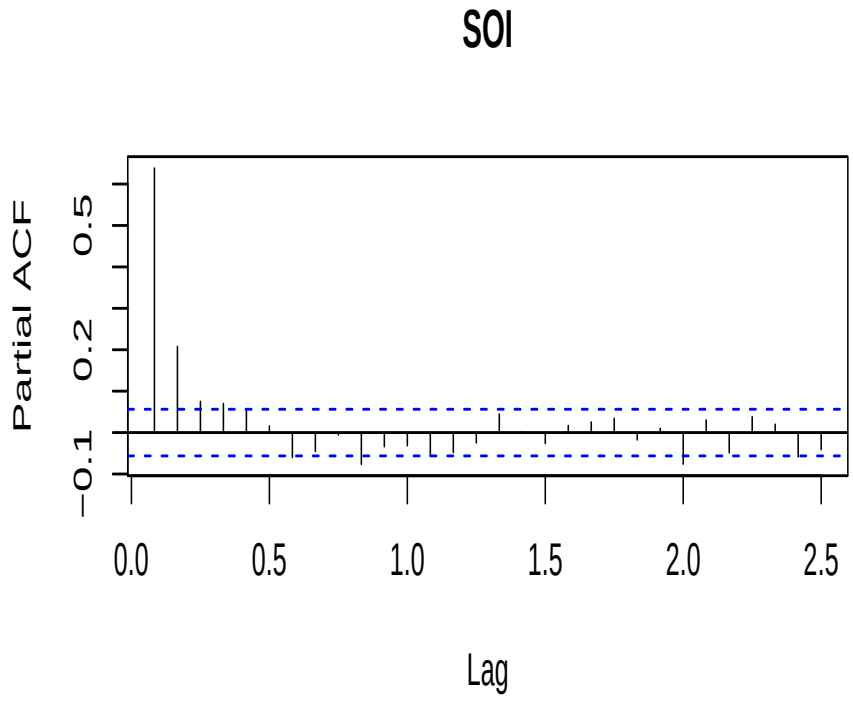


Podemos emplear la misma técnica para obtener las funciones ACF y PACF de cada una de las series mediante:

```
> for (i in 1:lcols) {
+     acf(nao[[i]], na.action = na.omit, main = cols[i])
+     pacf(nao[[i]], na.action = na.omit, main = cols[i])
+ }
```

(a continuación se representan sólo las de `nao$SOI`).


SOI
ACF
0.0
0.4
0.8
0.0
0.5
1.0
1.5
2.0
2.5
Lag


SOI
Partial ACF
−0.1
0.2
0.5
0.0
0.5
1.0
1.5
2.0
2.5
Lag

6. Información adicional

Hay ahora una relativa abundancia de libros y fuentes de información acerca de R. Continúa estando entre los mejores el libro [14]. Dispone de complementos *on line* que lo mantienen actualizado: [15]. Libros introductorios son [6], [1], [17] (que se refieren a S-Plus, pero son bastante aplicables a R), y [6].

Específicamente sobre importación y exportación de datos se dispone de [13].

Sobre R para regresión y modelos lineales en general se dispone de [8] y [7]. Aunque escrito para S-Plus, es de utilidad también [2].

La referencia original describiendo S (lenguaje del cuál R es una implementación) es [3], aún utilizable pero muy desfasado. Más moderno es [5]

Referencia mucho más avanzadas que convendrán a programadores son [4] y [16].

Referencias

[1] A.Krause and M.Olson. *The Basics of* S *and* S-Plus. Springer Verlag, 1997. Signatura: 519.682 KRA.

[2] Douglas M. Bates and Jose C. Pinheiro. *Mixed-effects models in S and S-PLUS*. Springer-Verlag, 2000. Signatura: 519.233.4.

[3] R.A. Becker, J.M. Chambers, and A.R. Wilks. *The New S Language. A Programming Environment for Data Analysis and Graphics*. Wadsworth & Brooks/Cole, Pacific Grove, California, 1988.

[4] J.M. Chambers. *Programming with Data*. Mathsoft, 1998.

[5] J.M. Chambers and T.J. Hastie. *Statistical Models in S*. Wadsworth & Brooks/Cole, Pacific Grove, Ca., 1992.

[6] P. Dalgaard. *Introductory statistics with R*. Statistics and Computing. Springer-Verlag, 2002. Signatura: 519.682 DAL.

[7] J.J. Faraway. *Linear Models with R*. Chapman & Hall/CRC, 2004. Signatura: 519.233 FAR.

[8] J. Fox. *An R and S-Plus companion to applied regression*. Sage Pub., 2002.

[9] J. Melton and A. Simon. *Understanding the new SQL: a complete guide*. Morgan Kaufmann, San Francisco, Ca., 1993.

[10] Bruce Momjian. *PostgreSQL : introduction and concepts*. Addison-Wesley, 2001. Signatura: 681.3.069 MOM.

[11] Brian D. Ripley. Using databases with R. *R News*, 1(1):18–20, January 2001.

[12] Brian D. Ripley and Kurt Hornik. Date-time classes. *R News*, 1(2):8–11, June 2001.

[13] R Development Core Team. *R Data Import/Export*. CRAN. Disponible en `http://cran.at.r-project.org/manuals.html`, visitada el 10-Sep-2003.

[14] W.N. Venables and B.D. Ripley. *Modern Applied Statistics with* S-Plus. Springer-Verlag, New York, third edition, 1999.

[15] W.N. Venables and B.D. Ripley. 'R' complements to Modern Applied Statistics with S-Plus. En `http://www.stats.ox.ac.uk/pub/MASS3`, 1999.

[16] W.N. Venables and B.D. Ripley. *S Programming*. Springer-Verlag, 2000.

[17] M.D. Ugarte y A.Fdez. Militino. *Estadística Aplicada con S-Plus*. Universidad Pública de Navarra, 2001.

A. Listado del Ejemplo 1

```
tomates <- read.table(file="tomates.csv",header=TRUE)
tomates[1:3,]
str(tomates)
tomates[,"Fecha"] <- as.Date(as.character(tomates[,"Fecha"]),
                             format="%d-%m-%Y")
tomates[1:3,]
str(tomates)
#
# Hagamos ahora un breve análisis descriptivo de los
# datos en la data frame tomates
#
attach(tomates)
summary(tomates)                    # Media, desv. típica, etc.
length(unique(Planta))              # ¿Cuántas plantas hay?
length(unique(Fecha))               # ¿Cuántas fechas de recogida?
xtabs(Peso ~ Planta)                # ¿Cuánto ha dado cada planta?
xtabs(Peso ~ Fecha)                 # ¿Cuánto se ha recogido por fecha?

boxplot(Peso ~ as.Date(Fecha),      # Evolución de los pesos el el tiempo.
        main="Evolucion del peso de los frutos",
        xlab="Fecha",ylab="Peso en gramos")

                                    # Necesitaremos ahora etiquetas para
                                    # TODAS las fechas en la muestra (no
                                    # una para cada fecha distinta).


Grs <- aggregate(Peso,              # Peso total recolectado en cada
                 by=list(as.Date(Fecha)),
                      sum)
                                    # fecha. El argumento "by="ha de
                                    # ser una lista.

dimnames(Grs)[[2]] <- c("Fecha","Total Gramos")
print(Grs[1:5,])

GrsMedio <- aggregate(Peso,
                 by=list(Fecha),    # Peso medio por fruto en cada
                                    # fecha. El argumento "by="ha de
                 mean)              # ser una lista.

dimnames(GrsMedio)[[2]] <- c("Fecha","Media Gramos")
print(GrsMedio[1:5,])
                                    # Especificamos ahora cuáles de las
                                    # plantas recibieron un determinado
                                    # tratamiento.

Tratadas  <- c(1,2,5,6,9,10,13,14,17,18)
Controles <- (1:20)[-Tratadas]

                                    # "Status"es un factor dando el
                                    # status de cada observación: "Tratada"
```

```
                                        # si corresponde a una planta tratada,
                                        # control si corresponde a un control.

Status <- ifelse(Planta %in% Tratadas,"Tratada","Control")
Status <- as.factor(Status)

t.test(Peso[Status=="Tratada"],Peso[Status=="Control"])

boxplot(Peso ~ Status,
        main="Comparacion de pesos de los frutos ",
        xlab="Clase de plantas",ylab="Peso en gramos")


FechaFactor <- as.factor(as.Date(Fecha))
coplot(Peso ~ FechaFactor | Status,
          xlab="Fecha")
                                        # Evolución del peso con el
                                        # paso del tiempo para los dos grupos.

FactorPlanta <- as.factor(Planta)     # Creamos un factor para obtener
                                        # desgloses por planta.

coplot(Peso ~ FechaFactor | FactorPlanta,
       xlab="Fecha")
                                        # Evolución del peso de los frutos
                                        # con el paso del tiempo para cada
                                        # planta.


plot(FactorPlanta,Peso,main="Pesos de los frutos de cada planta",
     ylab="Gramos",
     xlab="Plantas")                    # Un plot cuyo eje X es un factor se
                                        # transforma automáticamente en en
                                        # boxplot. Este permite examinar la
                                        # distribución de los pesos de los
                                        # frutos de cada planta.

library(lattice)                        # Ofrece gráficos alternativos.

densityplot(~ Peso | FactorPlanta,
            layout=c(5,4),xlab="Peso",ylab="Densidad estimada",
            main="Distribucion de los pesos por planta")

densityplot(~ Peso | Status ,
            layout=c(2,1),xlab="Peso",ylab="Densidad estimada",
            main="Distribucion de los pesos por tratamiento")

barchart(as.character(as.Date(Fecha)) ~ Peso | Status,
          layout=c(2,1),xlab="Peso",ylab="Fecha",
            main="Distribucion de los pesos en el tiempo por tratamiento")

# q()
```

B. Listado del Ejemplo 2

```
nao <- read.table(file="naosoi.dat")
nao[1:3,]

attach(nao)
#
# Redefinición como series temporales, y representación gráfica
#
par(mfrow=c(2,1))
cols  <- names(nao)
lcols <- length(nao)
 for (i in 1:lcols) {
   nao[[i]] <- temp <- ts(nao[[i]],start=c(1900,1),frequency=12)
   temp <- na.omit(temp)
   temp2 <- (attributes(temp)$tsp)[1]
   anyo <- floor(temp2) ;  mes <- round(1+12*(temp2 %% 1))
   ts.plot(window(temp,start=c(anyo,mes)),
           gpars=list(ylab=cols[i],xlab="Año",
             main=paste(cols[i],": datos brutos",sep="")))
 }
#
# Veamos la forma de sus funciones ACF y PACF
#
 for (i in 1:lcols) {
   acf(nao[[i]],na.action=na.omit,main=cols[i])
   pacf(nao[[i]],na.action=na.omit,main=cols[i])
 }
par(mfrow=c(1,1))
ccf(nao$NAO,nao$SOI,na.action=na.omit,main="CCF de NAO y SOI")

#
# Claramente, necesitaremos alguna diferenciación estacional y/o
# parámetro autoregresivo estacional. En primer lugar, ajustaremos
# un modelo "largo", para darnos una idea de que fracción del pasado
# es relevante en la modelización. Emplearemos modelos AR de orden
# creciente dibujando la gráfica del AIC para cada serie.
#
par(mfrow=c(2,1))
for (i in 1:length(cols)) {
  fit.ar <- ar(nao[[i]],method="ols",order.max=96,na.action=na.omit)
#
# Veamos ahora la pinta que tienen los valores AIC
#
  plot(fit.ar$aic,type="l",xlab="Orden p",ylab="Valor AIC",
       main=paste("Criterio AIC para modelos AR(p) \n Serie: ",
       cols[i],sep=""))
}
#
# SOI parece tener una "memoria"bastante más larga...
# Podríamos a la luz de la ACF y PACF de NAO sospechar algunos
# modelos ARIMA buenos candidatos a ajustar bien. O podemos
# recurrir a una solución de fuerza bruta que consiste en ajustar
# todos los modelos a priori razonables y seleccionar uno o varios
```

```
# de entre ellos con ayuda del criterio AIC.
#
# Nótese el uso de try() para encapsular errores que podrían abortar
# de otro modo una iteración muy costosa.
#
  i <- 0
  NAO.modelos <- vector("list", 450)
  for (sar in 0:4) {
    for (sma in 0:4) {
      for (d in 0:1) {
        for (ar in 0:4) {
          for (ma in 0:4) {
            if((ar+ma)==0) next ;
            if((sar+sma+d)==0) next ;
            i <- i + 1
            NAO.modelos[[i]] <- fit <- try(arima0(NAO,order=c(ar,0,ma),
                                seasonal=list(order=c(sar,d,sma))))
            if (class(fit)=="try-error") next ;
          }
        }
      }
    }
  }
save.image()
q()
```

C. Datos en `tomates.csv` (primeras líneas)

```
"Planta" "Fecha" "Peso"
11 2003-07-07  90
 1 2003-07-08 236
 4 2003-07-09 133
10 2003-07-11 183
 8 2003-07-11 105
 8 2003-07-11  52
 6 2003-07-12  91
20 2003-10-05  75
19 2003-10-05  88
20 2003-10-05  84
 1 2003-10-05  77
```

D. Datos en `naosoi.dat` (primeras líneas)

```
"NAO" "SOI"
"Jan1900" 1.38 -1
"Feb1900" -2.36 -1
"Mar1900" -3.12 -3.2
"Apr1900" 2.61 -1.4
"May1900" -0.57 -0.6

.....   datos omitidos  .....

"Jun2000" 0.9 -0.6
"Jul2000" -2.99 -0.4
"Aug2000" 0.78 0.4
"Sep2000" -1.09 1
"Oct2000" 2.26 1
"Nov2000" -0.24 2
"Dec2000" -1.41 0.7
```