Tema 01.02 Programando con R

Análisis Estadístico de series económicas

Xavi Barber

Centro de Investigación Operativa
Universidad Miguel Hernández de Elche

2018-02-15

1. Tipos de Datos y su control R
2. Funciones

Tipos de Datos y su control R

Material bibliográfico

Data Input

Vectores

```
a <- c(1, 2, 5.3, 6, -2, 4)  # numeric vector
b <- c("one", "two", "three")  # character vector
c <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE)  #logical vector

a
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
b
```

```
## [1] "one"   "two"   "three"
```

```
c
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE
```

Matrices

```
# generates 5 x 4 numeric matrix
y<-matrix(1:20, nrow=5,ncol=4)
y
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

```
y<-matrix(1:20, nrow=5,ncol=4, byrow = TRUE)
y
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

```r
# another example
cells <- c(1,26,24,68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
mymatrix <- matrix(cells, nrow=2,
                   ncol=2, byrow=TRUE,
                dimnames=list(rnames, cnames))
mymatrix
```

```
##    C1 C2
## R1  1 26
## R2 24 68
```

Arrays

Son un objeto peculiar, pues permite crear matrices de más de dos dimensiones:

```
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),
                dimnames = list(row.names,column.names, matrix.names))
print(result)
```

```
## , , Matrix1
##
##      COL1 COL2 COL3
## ROW1    5   10   13
## ROW2    9   11   14
## ROW3    3   12   15
##
## , , Matrix2
##
##      COL1 COL2 COL3
## ROW1    5   10   13
```

Data Frames

Se trata de uno de los principales "contenedores" de información. Equivale a lo que es una "Hoja" en las Excel(c) o similar.

```
a <- c(1,2,3,4)
b <- c("one","two","three", "four")
d <-  c(1,2,5.3,6)
e <- c("red", "white", "red", NA)
f <- c(TRUE,TRUE,TRUE,FALSE)

mydata <- data.frame(a,b,d,e,f)
names(mydata) <- c("ID","number", "wieght","Color","Passed") # variable names
```

```
##   ID number wieght Color Passed
## 1  1    one    1.0   red   TRUE
## 2  2    two    2.0 white   TRUE
## 3  3  three    5.3   red   TRUE
## 4  4   four    6.0  <NA>  FALSE
```

Seleccionando elementos de un data.frame

```
myframe[,3:5] # columns 3,4,5 of data frame
myframe[,c("ID","Age")] # columns ID and Age from data frame
myframe$X1 # variable x1 in the data frame
sel<-c(1,4,5)
myframe[,sel] # columns 1,4,5 of data frame
myframe[1:3,] # row 1,2,3 of data frame
```

Listas

Son colecciones de objetos ordenadas, es el gran desconocido por el usuario "poco iniciado" en R

```
# example of a list with 4 components -
# a string, a numeric vector, a matrix, and a scaler
w1 <- list(name="Fred", mynumbers=a, mymatrix=b, age=5.3)
w2 <- list(name="Fred", mynumbers=a, mymatrix=b, age=5.3)

# example of a list containing two lists
v <- c(w1,w2)
```

```
## $name
## [1] "Fred"
##
## $mynumbers
## [1] 1 2 3 4
##
## $mymatrix
## [1] "one"   "two"   "three" "four"
##
## $age
## [1] 5.3


## $name
## [1] "Fred"
##
## $mynumbers
## [1] 1 2 3 4
##
## $mymatrix
## [1] "one"   "two"   "three" "four"
```

Para seleccionar elementos de una lista debemos utilizar el doble corchet: [[X]]

```
v[[2]] # 2nd component of the list
```

```
## [1] 1 2 3 4
```

```
w2[["mynumbers"]] # component named mynumbers in list
```

```
## [1] 1 2 3 4
```

Factores

Una de las grandes ventajas que tiene Rs su forma de tratar las variables cualitativas.

```
# variable gender with 20 "male" entries and
# 30 "female" entries
gender <- c(rep("male",20), rep("female", 30))
gender <- factor(gender)
# stores gender as 20 1s and 30 2s and associates
# 1=female, 2=male internally (alphabetically)
# R now treats gender as a nominal variable
summary(gender)
```

```
## female   male
##     30     20
```

```
rating<-c("small associates", "large", "small associates", "medium",
rating<-factor(rating)
table(rating)
```

```
## rating
##              large             medium small associates
##                  3                  1                  2
```

```
rating2 <- ordered(rating)
# recodes rating to 1,2,3 and associates
# 1=large, 2=medium, 3=small internally
# R now treats rating as ordinal
rating2
```

```
## [1] small associates large                small associates medium
## [5] large                 large
## Levels: large < medium < small associates
```

Problemas con el orden alfabético:

```
mons = c("March","April","January","November","January",
"September","October","September","November","August",
"January","November","November","February","May","August",
"July","December","August","August","September","November",
"February","April")
mons = factor(mons)
table(mons)
```

```
## mons
##     April    August  December  February   January      July     March
##         2         4         1         2         3         1         1
##       May  November   October September
##         1         5         1         3
```

Ordenando los niveles:

```
mons = factor(mons, levels = c("January", "February", "March",
    "April", "May", "June", "July", "August", "September", "October",
    "November", "December"), ordered = TRUE)
table(mons)
```

```
## mons
##   January  February     March     April       May      June      July
##         3         2         1         2         1         0         1
##    August September   October  November  December
##         4         3         1         5         1
```

Variable labels

```
library(Hmisc)
label(mydata$Color) <- "Variable label for variable Color"
describe(mydata)
```

```
## mydata
##
##  5  Variables      4  Observations
## ---------------------------------------------------------------------------
## ID
##        n  missing distinct     Info     Mean      Gmd
##        4        0        4        1      2.5    1.667
##
## Value          1    2    3    4
## Frequency      1    1    1    1
## Proportion  0.25 0.25 0.25 0.25
## ---------------------------------------------------------------------------
## number
##        n  missing distinct
##        4        0        4
##
## Value        four   one three   two
## Frequency       1     1     1     1
## Proportion   0.25  0.25  0.25  0.25
## ---------------------------------------------------------------------------
## wieght
##        n  missing distinct     Info     Mean      Gmd
##        4        0        4        1    3.575     3.05
```

Value Labels

```
# variable Color2 is coded 1, 2 or 3
mydata$Color2<-c(1,1,3,2)
# we want to attach value labels 1=red, 2=blue, 3=green

mydata$v1 <- factor(mydata$Color2,
levels = c(1,2,3),
labels = c("red", "blue", "green"))
```

```
# variable Weight2 is coded 1, 3 or 5
mydata$weight2<-c(5,1,3,3)
# we want to attach value labels 1=Low, 3=Medium, 5=High

mydata$v1 <- ordered(mydata$weight2,
                     levels = c(1,3, 5),
                     labels = c("Low", "Medium", "High"))
```

Missing Data

Para saber dónde hay *missings*:

```
is.na(c) # returns TRUE of x is missing
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
y <- c(1,2,3,NA)
is.na(y) # returns a vector (F F F T)
```

```
## [1] FALSE FALSE FALSE  TRUE
```

Si venimos de SPSS suele ser habitual el famoso 999

```
# recode 99 to missing for variable v1
# select rows where v1 is 99 and recode column v1
mydata$v1[mydata$v1==99] <- NA
```

Excluir los valores ausentes del análisis:

```
x <- c(1,2,NA,3)
mean(x) # returns NA
```

```
## [1] NA
```

```
mean(x, na.rm=TRUE) # returns 2
```

```
## [1] 2
```

```
# list rows of data that have missing values
mydata[!complete.cases(mydata),]
```

```
##   ID number wieght This is the label for variable 3 Passed Color2     v1
## 4  4   four      6                              <NA>  FALSE      2 Medium
##   weight2
## 4       3
```

```
# create new dataset without missing data
newdata <- na.omit(mydata)
```

Funciones interesantes

```
length(object) # number of elements or components
str(object)    # structure of an object
class(object)  # class or type of an object
names(object)  # names

c(object,object,...)        # combine objects into a vector
cbind(object, object, ...) # combine objects as columns
rbind(object, object, ...) # combine objects as rows

object      # prints the object

ls()        # list current objects
rm(object) # delete an object

newobject <- edit(object) # edit copy and save as newobject
fix(object)               # edit in place
```

```r
# list levels of factor v1 in mydata
levels(mydata$v1)

# print mydata
mydata


# print first 10 rows of mydata
head(mydata, n=10)

# print last 5 rows of mydata
tail(mydata, n=5)
```

Para practicaR estos conceptos podeis realizar el siguiente curso online gratuito:

Curso

Data Values

Las fechas son un problema en cualquier programa "estadístico".

```
# use as.Date( ) to convert strings to dates
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
# number of days between 6/22/07 and 2/13/04
days <- mydates[1] - mydates[2]
```

`Sys.Date( )` devuelve la fecha de "hoy".

`date()` devuelve la fecha y hora "actual".

Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a	abbreviated weekday	Mon
%A	unabbreviated weekday	Monday
%m	month (00-12)	00-12
%b	abbreviated month	Jan
%B	unabbreviated month	January
%y	2-digit year	07
%Y	4-digit year	2007

```r
# print today's date
today <- Sys.Date()
format(today, format="%B %d %Y")
```

```
## [1] "febrero 15 2018"
```

```r
"June 20 2007"
```

```
## [1] "June 20 2007"
```

Data Conversion

```
# convert date info in format 'mm/dd/yyyy'
strDates <- c("01/05/1965", "08/16/1975")
dates <- as.Date(strDates, "%m/%d/%Y")
```

```
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
```

```
# convert dates to character data
strDates <- as.character(dates)
```

Para practicar

Data Management

Creando nuevas variables

```
# Three examples for doing the same computations

mydata$sum <- mydata$x1 + mydata$x2
mydata$mean <- (mydata$x1 + mydata$x2)/2

attach(mydata)
mydata$sum <- x1 + x2
mydata$mean <- (x1 + x2)/2
detach(mydata)

mydata <- transform( mydata,
sum = x1 + x2,
mean = (x1 + x2)/2
)
```

Recodificando variables

```r
# create 2 age categories
mydata$agecat <- ifelse(mydata$age > 70,
c("older"), c("younger"))

# another example: create 3 age categories
attach(mydata)
mydata$agecat[age > 75] <- "Elder"
mydata$agecat[age > 45 & age <= 75] <- "Middle Aged"
mydata$agecat[age <= 45] <- "Young"
detach(mydata)
```

Renombrando variables

```r
# rename interactively
fix(mydata) # results are saved on close

# rename programmatically
library(reshape)
mydata <- rename(mydata, c(oldname="newname"))

# you can re-enter all the variable names in order
# changing the ones you need to change.the limitation
# is that you need to enter all of them!
names(mydata) <- c("x1","age","y", "ses")
```

Funciones matemáticas y lógicas

Operadores

Operator	Description
$+$	addition
$-$	subtraction
$*$	multiplication
$/$	division
x^y or $2**3$	exponentiation
$x\%\%y$	modulus (3 mod 2) is 1
$5\%/\%2$	integer division $5\%/\%2$ is 2

Operadores lógicos

Operador	Descripción
$<$	less than
$<=$	less than or equal to
$>$	greater than
$>=$	greater than or equal to
$==$	exactly equal to
$!=$	not equal to
$!x$	Not x
$x \mid y$	x OR y
$x\&y$	x AND y
isTRUE(x)	test if X is TRUE

Funciones matemáticas

Función	Descripción
abs(x)	absolute value
sqrt(x)	square root
ceiling(x)	ceiling(3.475)=$\lceil 3.475 \rceil$ is 4
floor(x)	floor(3.475)=$\lfloor 3.475 \rfloor$ is 3
trunc(x)	trunc(5.99) is 5
round(x, digits=n)	round(3.475, digits=2) is 3.48
signif(x, digits=n)	signif(3.475, digits=2) is 3.5
cos(x), sin(x), tan(x)	also acos(x), cosh(x), acosh(x), etc.
log(x)	natural logarithm
log10(x)	common logarithm
exp(x)	e^x

Estructuras de control

if-else

```
if (cond) expr
if (cond) expr1 else expr2
```

Ejemplo:

```
  if (!is.matrix(x)) {
    warning("Esto no es una matriz")
  }else{
    print("Todo Correcto!!!!!")
  }
```

for

Para realizar búcles donde conocemos el principio y final del bucle.

Sintaxis: `for (var in seq) expr`

Ejemplo:

```
for (i in 1:5) print(i)
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```r
# o para cuando queremos poner más cosas dentro
for (i in 1:3 ) {
   texto<-paste("el valor de i es=",i)
    print(texto)
}
```

```
## [1] "el valor de i es= 1"
## [1] "el valor de i es= 2"
## [1] "el valor de i es= 3"
```

while

Para bucles donde se tienen que cumplir una condición para su detención. Sintaxis: `while (cond) expr`

Ejemplo:

```
x <- 1
while(x < 5) {
  x <- x+1;
  print(x);
 }
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```r
x <- 1
while(x < 5) {
    x <- x+1;
    if (x == 3) break;
    print(x);
    }
```

```
## [1] 2
```

switch

Evalua una expresión y elige uno de los argumentos de la lista. Sintaxis: `switch(EXPR, ...)`

Ejemplo:

```
EXPRESION<-2
x <- switch( EXPRESION,
             "first", "second", "third","fourth")
print(x)
```

```
## [1] "second"
```

```
EXPRESION<-4
x <- switch(EXPRESION,
             "first", "second", "third","fourth")
print(x)
```

```
## [1] "fourth"
```

ifelse

Un función para crear nuevas variables, recodificar en un tiempo muy rápido. Es una función computacionalmente eficiente.

Sintaxis: `ifelse(test, yes, no)`

```
x <- c(-2,1,-5,3,3,4,-6)
y <- ifelse(x>0, 1, -1)
print(x)
```

```
## [1] -2  1 -5  3  3  4 -6
```

```
print(y)
```

```
## [1] -1  1 -1  1  1  1 -1
```

Ejemplo completo

```r
# Transponer una matriz creando una función 'pobre' parecita
# a t()

mytrans <- function(x) {
    if (!is.matrix(x)) {
        warning("Esto no es una matriz: devuelve NA")
        return(NA_real_)
    }
    y <- matrix(1, nrow = ncol(x), ncol = nrow(x))
    for (i in 1:nrow(x)) {
        for (j in 1:ncol(x)) {
            y[j, i] <- x[i, j]
        }
    }
    return(y)
}
```

```
# try it
z <- matrix(1:10, nrow=5, ncol=2)
tz <- mytrans(z)
tz
```

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

Funciones

Funciones de usuario

Si existe en Rna forma de ahorrar tiempo y que nuestro código sea reutilizable es gracias a las *funciones de usuario*.

Estas funciones las creamos nosotros mismos y su estructura es:
`nombre.Funcion<-function(parametros) {acciones}`

Ejemplo básico:

```
mifuncion <- function(x) {
    sqrt(x)
}

mifuncion(81)
```

```
## [1] 9
```

Funciones de R

```
apply(X, MARGIN, FUN, ...)
```

```
lapply(X, FUN, ...)
```

```
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

Web de ayuda

Otra web de ejemplos

apply

```
# Genero una matriz aleatoriamente
m <- matrix(data = cbind(rnorm(5, 0), rnorm(5), rnorm(5, 5)),
    nrow = 5, ncol = 3)
m
```

```
##              [,1]       [,2]     [,3]
## [1,] -0.9479654 -0.4280244 4.858259
## [2,]  0.9613545 -1.2512863 3.449658
## [3,]  0.3303340 -2.1581889 6.070787
## [4,] -0.8792417 -0.8330960 5.077991
## [5,] -1.1837979 -0.1522127 2.753201
```

```r
apply(m, 1, mean)  ## la media por filas
```

```
## [1] 1.1607563 1.0532419 1.4143107 1.1218843 0.4723969
```

```r
apply(m, 2, mean)  ## la media por columnas
```

```
## [1] -0.3438633 -0.9645617  4.4419790
```

```r
# create a list with 2 elements
l <- list(a = 1:10, b = 11:20)
l
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```r
# # the mean of the values in each element
sapply(l, mean)
```

```
##    a    b
##  5.5 15.5
```

```r
# create a list with 2 elements
l <- list(a = 1:10, b = 11:20)
l
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```r
# the mean of the values in each element
lapply(l, mean)
```

```
## $a
## [1] 5.5
##
## $b
## [1] 15.5
```

```r
# create a list with 2 elements
l <- list(a = 1:10, b = 11:20)
l
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```r
# the mean of the values in each element
sapply(l, mean, simplify = F)
```

```
## $a
## [1] 5.5
##
## $b
## [1] 15.5
```

```r
# create a list with 2 elements
l <- list(a = 1:10, b = 11:20)
l
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```r
# the mean of the values in each element
sapply(l, mean, simplify = T)
```

```
##    a    b
##  5.5 15.5
```

```r
l <- list(a = 1:10, b = 11:20)
# log2 of each value in the list
rapply(l, log2)
```

```
##       a1       a2       a3       a4       a5       a6       a7       a8
## 0.000000 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355 3.000000
##       a9      a10       b1       b2       b3       b4       b5       b6
## 3.169925 3.321928 3.459432 3.584963 3.700440 3.807355 3.906891 4.000000
##       b7       b8       b9      b10
## 4.087463 4.169925 4.247928 4.321928
```

```r
l <- list(a = 1:10, b = 11:20)
l
# log2 of each value in the list
rapply(l, log2, how = "list")
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
##  [1] 11 12 13 14 15 16 17 18 19 20


## $a
##  [1] 0.000000 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355
##  [8] 3.000000 3.169925 3.321928
##
## $b
##  [1] 3.459432 3.584963 3.700440 3.807355 3.906891 4.000000 4.087463
##  [8] 4.169925 4.247928 4.321928
```

```r
attach(iris)
summary(iris)
# mean petal length by species
tapply(Petal.Length, Species, mean)
detach(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##

##     setosa versicolor  virginica
##      1.462      4.260      5.552
```

```r
l1 <- list(a = c(1:10), b = c(11:20))
l2 <- list(c = c(21:30), d = c(31:40))
# sum the corresponding elements of l1 and l2
mapply(sum, l1$a, l1$b, l2$c, l2$d)
```

```
##  [1]  64  68  72  76  80  84  88  92  96 100
```

```r
l <- list(a = 1:10, b = 11:20)
l
# fivenum of values using vapply
l.fivenum <- vapply(l, fivenum,
                    c(Min.=0, "1st Qu."=0, Median=0,
                      "3rd Qu."=0, Max.=0))
class(l.fivenum)

# let's see it
l.fivenum
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```
## [1] "matrix"
```

```
##                a    b
## Min.         1.0 11.0
## 1st Qu.      3.0 13.0
## Median       5.5 15.5
## 3rd Qu.      8.0 18.0
## Max.        10.0 20.0
```