

Ramírez Ramírez, L.L.

Notas sobre R

Enero 2015

Contenido

Contenido	ii
Sobre R	1
Variables y objetos en R	2
Escalaes	2
Tipos	2
Algunas operaciones y funciones entre escalaes	3
Vectores	5
La función “c”	5
La función “:”	5
La función seq	6
La función rep	6
Simulación de variables aleatorias	6
Algunas operaciones y funciones especiales entre vectores y escalaes	7
Factores	10
Matrices	11
Arreglos	17
Bases de datos (Data Frames)	18
Listas	24
Otros	25
Funciones	26
Ayuda	29
Organización	32
Paquetes	33
Manejo de bases de datos	34

Sobre R

R es un lenguaje de programación abierto (open source) que incorpora una interfase grafica (GUI) orientado a realizar análisis estadístico y gráficos. El lenguaje es ampliamente utilizado entre estadísticos para realizar programas estadísticos y análisis de datos.

R toma mucho del lenguaje S, desarrollado por Rick Becker, John Chambers y colegas en Bell Labs en los años setentas y ochentas. Los creadores describen a S como un lenguaje y un ambiente de programación interactiva para el análisis de datos y graficación. Aún más, decían “S le alienta a calcular, mirar los datos y programar de manera interactiva, con una respuesta rápida que le permite entender y aprender”. Esta forma de interactuar con los datos, a diferencia de hacerlo por lotes, revolucionó la manera de hacer análisis estadístico.

R está desarrollado por el Equipo de Desarrollo Central (Development Core Team) en el que Chambers es un miembro. Este programa toma su nombre considerando los nombres de los dos primeros autores de R (Robert Gentleman and Ross Ihaka), y por otra parte como contraparte del nombre de S.

R está principalmente escrito en C, Fortran y R. Debido a que forma parte del proyecto GNU, se hace disponible a cualquier persona bajo la licencia Publica General GNU y puede proveer de versiones precompiladas del programa para diversos sistemas operativos.

La página oficial es: <http://cran.r-project.org/>

Al instalar R se instala una interface gráfica, pero existen otros proyectos (Tinn-R, RStudio, por ejemplo).

R es un lenguaje que se denomina interpretado porque no genera un “ejecutable” sino que un programa intérprete, almacenado en el sistema operativo del disco, o incluido de manera permanente dentro de la máquina, convierte cada proposición del programa fuente (de alto nivel) en lenguaje de máquina conforme vaya siendo necesario durante el proceso de los datos

```
2+3

## [1] 5

x<- 2+3
x

## [1] 5

exp(-4*4/2)/sqrt(2*pi)

## [1] 0.0001338302
```

Variables y objetos en R

Escalares

Las variables son el equivalente de elementos en tu calculadora, pero en R se tiene la posibilidad de utilizar casi un número ilimitado de ellas y pueden tener el nombre que nosotros queramos. Los nombres pueden obtener letras, números, punto, pero algunas restricciones son las palabras reservadas y símbolos especiales como , %.

Tipos

Entero

```
x<-3
x
## [1] 3
```

Doble

```
pi
## [1] 3.141593
runif(1)
## [1] 0.6585816
```

Caracter

```
Nombre<-"Laura"
Nombre
## [1] "Laura"
```

Lógica

```
w<-x==2
w
## [1] FALSE
```

Catórica o “Factor”

```
F.1<-factor("Masculino")
F.1

## [1] Masculino
## Levels: Masculino
```

Otros tipos

Valor faltante (NA) Pueden surgir cuando se lee una base de datos con valores no registrados.

```
var (8)

## [1] NA
```

Infinito (Inf)

```
1/0

## [1] Inf
```

Invalido (NaN)

```
log(-1)

## Warning in log(-1): Se han producido NaNs

## [1] NaN
```

Algunas operaciones y funciones entre escalares

```
#Suma:
x+pi

## [1] 6.141593

#Producto
x*pi

## [1] 9.424778

#División:
x/pi

## [1] 0.9549297

#Exponenciación:
x^2

## [1] 9

x^(1/3)
```

```
## [1] 1.44225

#Funciones matemáticas
round(pi,2)

## [1] 3.14

ceiling(pi)

## [1] 4

floor(pi)

## [1] 3

exp(x)

## [1] 20.08554

log(x)  #Ojo, este es el logaritmo natural

## [1] 1.098612

log(x,base=2)  #Logaritmo base 2

## [1] 1.584963

#Funciones lógicas
x>3

## [1] FALSE

x<3

## [1] FALSE

x>=3

## [1] TRUE

x<=3

## [1] TRUE

x==3

## [1] TRUE

Nombre=="Laura"

## [1] TRUE
```

Vectores

Los vectores, al igual que los escalares pueden ser de diversos tipos:

```
xv<-c(1,2,3,4)
yv<-c(pi,2*pi, 3*pi)
Nombrev<-c("Jose", "Laura", "Alfonso")
wv<-(xv>=2)
F.lv<-factor(Nombrev)
```

Otra forma de construir vectores es usando funciones especiales como:

La función “c”

Esta función recibe su nombre de la acción de concatenar. En los ejemplos anteriores concatenábamos escalares, pero también se pueden concatenar vectores

```
c(1,2,3,4)

## [1] 1 2 3 4

c(xv,yv)

## [1] 1.000000 2.000000 3.000000 4.000000 3.141593 6.283185 9.424778
```

La función “:”

Crea una serie donde el valor inicial esta a la izquierda y el final a la derecha. Los incrementos son siempre de una unidad.

```
1:5

## [1] 1 2 3 4 5

0:5

## [1] 0 1 2 3 4 5

-10:5

## [1] -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5

2.5:10.5

## [1] 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5 10.5

2.5:10

## [1] 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5

5:1

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

La función seq

Es una generalización a “:” ya que genera series cuyo salto esta especificado o cuyo numero de elementos es fijo

```
seq(from=0,to=5,by=1)           #equivalente a seq(0,5,1)

## [1] 0 1 2 3 4 5

seq(from=0,to=5,length=10)

## [1] 0.0000000 0.5555556 1.1111111 1.6666667 2.2222222 2.7777778 3.3333333
## [8] 3.8888889 4.4444444 5.0000000

seq(3,1,-0.5)

## [1] 3.0 2.5 2.0 1.5 1.0

seq(1.1,3.2,0.1)

## [1] 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7
## [18] 2.8 2.9 3.0 3.1 3.2

v<-1; w<-3
seq(v,w,length=10)

## [1] 1.000000 1.222222 1.444444 1.666667 1.888889 2.111111 2.333333
## [8] 2.555556 2.777778 3.000000
```

La función rep

Esta función replica un valor o vector el numero de veces señalado

```
rep(0,3)

## [1] 0 0 0

rep(xv,2)

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

rep(xv,each=2)

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

Simulación de variables aleatorias

Por ejemplo, la función “runif” obtiene simulaciones de una variable aleatoria con distribución uniforme. Así como existe runif para la distribución uniforme, también existen rnorm, rexp, rpois, rbinom, etc. para las distribuciones estadísticas más comunes. Estas se presentan más adelante.


```
a<-runif(10) #obtiene 10 simulaciones de una U(0,1)
runif(10,2,4) #obtiene 10 simulaciones de una U(2,4)

## [1] 2.957492 3.274963 2.652255 3.678730 3.254729 2.996208 3.251123
## [8] 2.625958 3.459860 2.596820
```

Algunas operaciones y funciones especiales entre vectores y escalares

```
#Suma:
xv+x          #vector y escalar

## [1] 4 5 6 7

xv+(0:3)      #vector y vector

## [1] 1 3 5 7

#Producto:
xv*2

## [1] 2 4 6 8

xv*(0:3)

## [1] 0 2 6 12

#División:
xv/2

## [1] 0.5 1.0 1.5 2.0

xv/(1:4)

## [1] 1 1 1 1

#Exponenciación:
xv^2

## [1] 1 4 9 16

2^{1:3}

## [1] 2 4 8

#Funciones matemáticas:
exp(xv)

## [1] 2.718282 7.389056 20.085537 54.598150

log(xv) #Ojo, este es el logaritmo natural
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944

#Funciones lógicas:
xv>3; xv<3;      xv>=3; xv<=3;xv==3

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

#Funciones para vectores:
#Longitud del vector
length(xv)

## [1] 4

length(x) #un escalar es en particular un vector de longitud 1

## [1] 1

#Valores max y min
max(xv); min(xv)

## [1] 4
## [1] 1

#Ordenar
sort(a) #ordena en forma ascendente

## [1] 0.1162483 0.3598760 0.3730265 0.4640631 0.5071465 0.5514037 0.5760065
## [8] 0.6574654 0.6733821 0.8436838

sort(a,decreasing=TRUE)#ordena en forma descendente

## [1] 0.8436838 0.6733821 0.6574654 0.5760065 0.5514037 0.5071465 0.4640631
## [8] 0.3730265 0.3598760 0.1162483

#suma y suma acumulada
sum(xv); cumsum(xv)

## [1] 10
## [1] 1 3 6 10

#producto y producto acumulado
prod(xv); cumprod(xv)

## [1] 24
## [1] 1 2 6 24
```

```
#los elementos únicos y duplicados
unique(xv); duplicated(c(xv,xv))

## [1] 1 2 3 4
## [1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE

#Funciones estadísticas
#Media
mean(a)

## [1] 0.5122302

#Varianza
var(a)

## [1] 0.04047375

#Desviacion Estándar
sd(a)

## [1] 0.2011809

#Percentiles
median(a); quantile(a,prob=.5)

## [1] 0.5292751
##      50%
## 0.5292751

#Resumen
summary(a)

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.1162  0.3958   0.5293   0.5122  0.6371   0.8437

#Muestras
sample(1:100,3); sample(1:3,10,replace=TRUE)

## [1] 88 25 37
## [1] 2 3 2 1 3 1 2 3 2 3

#Seleccionando elementos se realiza con el uso de paréntesis cuadrados
xv[1]; xv[3:4]; a[3:4];xv[c(3,1,2,4)]

## [1] 1
## [1] 3 4
## [1] 0.5071465 0.6574654
## [1] 3 1 2 4

#Substituyendo el valor de elementos
xv[1]<-0
xv[1:3]<-(-3):(-1)
```

Factores

Los factores son variables que solo pueden tomar un numero finito de valores discretos (numérico o carácter). Los factores normalmente ocurren en vectores y la diferencia con los vectores es que a pesar de poder tener valores numéricos éstos no son tratados jamás como vectores.

Los factores fueron creados para definir a variables como categóricas y no cometer errores como el intentar aplicar funciones numéricas a éstas. ¿Qué sentido tiene la media de la religión en México?

Los factores se pueden establecer de vectores de carácter o numéricos aplicándoles la función “factor”

```
xvf<-factor(xv)
F.1v<-factor(Nombrev)
#Algunas operaciones y funciones para factores
#Tabla de contingencia:
table(xvf)

## xvf
## -3 -2 -1  4
##  1  1  1  1
```

Aunque las tablas de contingencia también pueden obtenerse para valores numéricos (enteros o dobles) tiene mayor sentido hacerlo para un vector de factores. ¿Por qué?

```
length(table(xv))

## [1] 4

length(table(runif(1000)))

## [1] 1000
```

Crear un factor de un vector numérico puede ser útil cuando lo que se quiere estudiar son las categorías existentes. Para hacer esto considere la función “cut”

```
cut(a,c(0,.20,.40,.60,.70,1)) #crea un factor

## [1] (0.4,0.6] (0.7,1] (0.4,0.6] (0.6,0.7] (0.2,0.4] (0,0.2] (0.2,0.4]
## [8] (0.4,0.6] (0.4,0.6] (0.6,0.7]
## Levels: (0,0.2] (0.2,0.4] (0.4,0.6] (0.6,0.7] (0.7,1]

table(cut(a,c(0,.20,.40,.60,.70,1)))

##
## (0,0.2] (0.2,0.4] (0.4,0.6] (0.6,0.7] (0.7,1]
##      1      2      4      2      1

#Contrastar los resultados siguientes
cut(a,c(0,.20,.40,.60,.70,1))

## [1] (0.4,0.6] (0.7,1] (0.4,0.6] (0.6,0.7] (0.2,0.4] (0,0.2] (0.2,0.4]
## [8] (0.4,0.6] (0.4,0.6] (0.6,0.7]
## Levels: (0,0.2] (0.2,0.4] (0.4,0.6] (0.6,0.7] (0.7,1]
```

```
cut(a,c(0,.20,.40,.60,.70,1),label=FALSE)

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

¿Es un factor?

```
is.factor(x)

## [1] FALSE

is.factor(Nombre)

## [1] FALSE

is.factor(xvf)

## [1] TRUE
```

Matrices

Al igual que los vectores, las matrices pueden ser de diversos tipos. Las siguientes son formas de crear matrices matrix

```
matrix(0,3,3)

##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0

matrix(1:9,3,3)

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

matrix(1:9,3,3,byrow=TRUE)

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

matrix(1:10,3,3) #extiende una advertencia

## Warning in matrix(1:10, 3, 3): la longitud de los datos [10] no es un submúltiplo o múltiplo
del número de filas [3] en la matriz
```

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

matrix(1:4,3,3) #extiende advertencia y recila el vector

## Warning in matrix(1:4, 3, 3): la longitud de los datos [4] no es un submúltiplo o múltiplo
del número de filas [3] en la matriz

##      [,1] [,2] [,3]
## [1,]    1    4    3
## [2,]    2    1    4
## [3,]    3    2    1

matrix(c("hola", "adios"),3,3)

## Warning in matrix(c("hola", "adios"), 3, 3): la longitud de los datos [2] no es un submúltiplo
o múltiplo del número de filas [3] en la matriz

##      [,1] [,2] [,3]
## [1,] "hola" "adios" "hola"
## [2,] "adios" "hola" "adios"
## [3,] "hola" "adios" "hola"
```

cbind y rbind

```
cbind(xv,2*xv,3*xv)

##      xv
## [1,] -3 -6 -9
## [2,] -2 -4 -6
## [3,] -1 -2 -3
## [4,]  4  8 12

cbind(xv,1:3)

## Warning in cbind(xv, 1:3): number of rows of result is not a multiple of vector length (arg
2)

##      xv
## [1,] -3  1
## [2,] -2  2
## [3,] -1  3
## [4,]  4  1

rbind(xv,2*xv,3*xv)

##      [,1] [,2] [,3] [,4]
## xv    -3  -2  -1    4
```

```
##      -6  -4  -2   8
##      -9  -6  -3  12

b<-rbind(xv,c("hola", "adios"))
```

¿Es caracter?

```
is.character(b)

## [1] TRUE
```

Algunas operaciones y funciones especiales entre vectores y escalares

```
mat<-matrix(round(runif(9),2),3,3)

#Suma:
mat+1          #matriz y escalar

##      [,1] [,2] [,3]
## [1,] 1.64 1.39 1.85
## [2,] 1.27 1.86 1.40
## [3,] 1.45 1.41 1.79

mat+(1:3)      #matriz y vector

##      [,1] [,2] [,3]
## [1,] 1.64 1.39 1.85
## [2,] 2.27 2.86 2.40
## [3,] 3.45 3.41 3.79

mat+mat        #matriz y matriz

##      [,1] [,2] [,3]
## [1,] 1.28 0.78 1.70
## [2,] 0.54 1.72 0.80
## [3,] 0.90 0.82 1.58

#Producto
mat*2          #compara con mat+mat

##      [,1] [,2] [,3]
## [1,] 1.28 0.78 1.70
## [2,] 0.54 1.72 0.80
## [3,] 0.90 0.82 1.58

mat*(0:2)

##      [,1] [,2] [,3]
## [1,] 0.00 0.00 0.00
## [2,] 0.27 0.86 0.40
## [3,] 0.90 0.82 1.58
```

```

mat*mat

##          [,1]  [,2]  [,3]
## [1,] 0.4096 0.1521 0.7225
## [2,] 0.0729 0.7396 0.1600
## [3,] 0.2025 0.1681 0.6241

#División:
mat/2

##          [,1]  [,2]  [,3]
## [1,] 0.320 0.195 0.425
## [2,] 0.135 0.430 0.200
## [3,] 0.225 0.205 0.395

mat/(1:3)

##          [,1]      [,2]      [,3]
## [1,] 0.640 0.3900000 0.8500000
## [2,] 0.135 0.4300000 0.2000000
## [3,] 0.150 0.1366667 0.2633333

mat/mat

##          [,1] [,2] [,3]
## [1,]      1      1      1
## [2,]      1      1      1
## [3,]      1      1      1

#Exponenciación:
mat^2                                #compara con mat*mat

##          [,1]  [,2]  [,3]
## [1,] 0.4096 0.1521 0.7225
## [2,] 0.0729 0.7396 0.1600
## [3,] 0.2025 0.1681 0.6241

#Funciones matemáticas
exp(mat);      log(mat)

##          [,1]      [,2]      [,3]
## [1,] 1.896481 1.476981 2.339647
## [2,] 1.309964 2.363161 1.491825
## [3,] 1.568312 1.506818 2.203396
##          [,1]      [,2]      [,3]
## [1,] -0.4462871 -0.9416085 -0.1625189
## [2,] -1.3093333 -0.1508229 -0.9162907
## [3,] -0.7985077 -0.8915981 -0.2357223

```



```
#transpuesta:
```

```
t(mat)

##      [,1] [,2] [,3]
## [1,] 0.64 0.27 0.45
## [2,] 0.39 0.86 0.41
## [3,] 0.85 0.40 0.79
```

```
#producto matricial (y con vector) %*%
```

```
mat%*%mat #comparar con mat*mat
```

```
##      [,1] [,2] [,3]
## [1,] 0.8974 0.9335 1.3715
## [2,] 0.5850 1.0089 0.8895
## [3,] 0.7542 0.8520 1.1706
```

```
mat%*%(1:3) #el vector se toma como matriz columna
```

```
##      [,1]
## [1,] 3.97
## [2,] 3.19
## [3,] 3.64
```

```
mat%*%t(1:3) #el prod tiene que ser de elementos concordantes por lo que se tiene error aca
```

```
## Error in mat %*% t(1:3): argumentos no compatibles
```

```
#Funciones lógicas
```

```
mat>0.5;          mat<=0.5; mat==0.5
```

```
##      [,1] [,2] [,3]
## [1,] TRUE FALSE TRUE
## [2,] FALSE TRUE FALSE
## [3,] FALSE FALSE TRUE
##      [,1] [,2] [,3]
## [1,] FALSE TRUE FALSE
## [2,] TRUE FALSE TRUE
## [3,] TRUE TRUE FALSE
##      [,1] [,2] [,3]
## [1,] FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE
## [3,] FALSE FALSE FALSE
```

```
#Funciones para matrices
```

```
#dimensiones
```

```
dim(mat); dim(x)[1]; dim(x)[2] #Los ultimos dos reportan el No de renglones y de columnas, respectiv
```

```
## [1] 3 3
## NULL
## NULL
```

```

#Valores max y min
max(mat); min(mat)

## [1] 0.86
## [1] 0.27

#Funciones estadísticas
#Media, varianza (insesgada) y Desv Estándar
mean(mat); var(mat); sd(mat)

## [1] 0.5622222
##           [,1]      [,2]      [,3]
## [1,] 0.03423333 -0.04311667 0.04135
## [2,] -0.04311667 0.07063333 -0.06470
## [3,] 0.04135000 -0.06470000 0.05970
## [1] 0.2254317

#Percentiles
median(mat); quantile(mat,prob=.5)

## [1] 0.45
## 50%
## 0.45

#Seleccionando elementos. Se realiza en forma analoga con el caso de vectores
mat[1,1];      mat[1,]; mat[,3]; mat[1:2,c(3,1,2)]

## [1] 0.64
## [1] 0.64 0.39 0.85
## [1] 0.85 0.40 0.79
##           [,1] [,2] [,3]
## [1,] 0.85 0.64 0.39
## [2,] 0.40 0.27 0.86

#Ordenar. Se puede hace con la función order
mat; mat[order(mat[,1]),]; mat[order(1:3),] #que pasa?

##           [,1] [,2] [,3]
## [1,] 0.64 0.39 0.85
## [2,] 0.27 0.86 0.40
## [3,] 0.45 0.41 0.79
##           [,1] [,2] [,3]
## [1,] 0.27 0.86 0.40
## [2,] 0.45 0.41 0.79
## [3,] 0.64 0.39 0.85
##           [,1] [,2] [,3]
## [1,] 0.64 0.39 0.85
## [2,] 0.27 0.86 0.40
## [3,] 0.45 0.41 0.79

```

```
mat[order(3:1),]

##      [,1] [,2] [,3]
## [1,] 0.45 0.41 0.79
## [2,] 0.27 0.86 0.40
## [3,] 0.64 0.39 0.85

mat[,order(mat[2,])] #que hace?

##      [,1] [,2] [,3]
## [1,] 0.64 0.85 0.39
## [2,] 0.27 0.40 0.86
## [3,] 0.45 0.79 0.41
```

Funciones aplicadas a columnas o renglones de la matriz
 Esto puede realizarse con funciones especificas o con apply

```
s<- matrix(1:9,3,3)
colSums(s);          rowSums(s)

## [1]  6 15 24
## [1] 12 15 18

colMeans(s);         rowMeans(s)

## [1]  2 5 8
## [1]  4 5 6

#No existe colVars ni colSds !!!!!!!!!
apply(s,1,var)

## [1]  9 9 9

apply(mat,1,var)      #y compara con var(mat[1,])

## [1] 0.05303333 0.09610000 0.04360000

apply(mat,2,sd)

## [1] 0.1850225 0.2657693 0.2443358

apply(s,2,sum)        #y compara con colSums(s)

## [1]  6 15 24
```

Arregos

Un arreglo de dos dimensiones es una matriz!!

```
#En lugar de usar: arr<-array(0,2,3,4), se usa:
arr<-array(0,c(2,3,4))

dim(arr)

## [1] 2 3 4
```

Bases de datos (Data Frames)

Los “Data Frame” al igual que matrices y arreglos son justamente un arreglos de datos, pero la diferencia que guardan con éstas es que los elementos no tienen que ser todos del mismo tipo.

Estos objetos se utilizan principalmente para almacenar lo que se conocen como base de datos. Arreglos que casi generalmente consideran por renglones a cada “individuo” y en cada columna alberga el valor de alguna variable registrada para cada uno (“estatura”, “sexo”, “estado civil”, etc.)

Estas bases de información normalmente se crean en R a partir de leer una base que puede estar en formato txt, Excel, spss, etc.

```
is.data.frame(mat)

## [1] FALSE

mat[1,1]<-“hola”
is.numeric(mat[,2])

## [1] FALSE

matf<-as.data.frame(mat) #transformamos como data frame
matf[1,1]<-“hola”
matf

##      V1    V2    V3
## 1 hola 0.39 0.85
## 2 0.27 0.86  0.4
## 3 0.45 0.41 0.79

is.numeric(matf[,2])

## [1] FALSE

is.data.frame(matf)

## [1] TRUE

#str describe el objeto que es y en el caso de de data frames, elista la naturaleza
#de sus columnas
str(matf)

## 'data.frame': 3 obs. of  3 variables:
##  $ V1: Factor w/ 3 levels "0.27","0.45",...: 3 1 2
##  $ V2: Factor w/ 3 levels "0.39","0.41",...: 1 3 2
##  $ V3: Factor w/ 3 levels "0.4","0.79","0.85": 3 1 2
```

Algunas operaciones y funciones especiales para bases de datos
Nombres de vectores y renglones

```
names(matf)

## [1] "V1" "V2" "V3"

names(matf)<-c("uno","dos","tres")
matf

##      uno  dos tres
## 1 hola 0.39 0.85
## 2 0.27 0.86  0.4
## 3 0.45 0.41 0.79

#y las llamadas a sus columnas pueden hacerse a través de sus nombres
#Esto es muy conveniente para leer en forma más natural el código
matf$dos

## [1] 0.39 0.86 0.41
## Levels: 0.39 0.41 0.86

matf$dos+matf$tres

## Warning in Ops.factor(matf$dos, matf$tres): '+' not meaningful for factors
## [1] NA NA NA

rownames(matf)

## [1] "1" "2" "3"

rownames(matf)<-c("primero","segundo","tercero")
matf

##           uno  dos tres
## primero hola 0.39 0.85
## segundo 0.27 0.86  0.4
## tercero 0.45 0.41 0.79
```

mtcars es una base de datos (data frame) incluido en R. Es un arreglo de 32 registros con 11 variables

```
names(mtcars)

## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"

rownames(mtcars)

## [1] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710"
## [4] "Hornet 4 Drive" "Hornet Sportabout" "Valiant"
```

```
## [7] "Duster 360"      "Merc 240D"      "Merc 230"
## [10] "Merc 280"        "Merc 280C"      "Merc 450SE"
## [13] "Merc 450SL"      "Merc 450SLC"    "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial" "Fiat 128"
## [19] "Honda Civic"      "Toyota Corolla"  "Toyota Corona"
## [22] "Dodge Challenger" "AMC Javelin"     "Camaro Z28"
## [25] "Pontiac Firebird" "Fiat X1-9"       "Porsche 914-2"
## [28] "Lotus Europa"     "Ford Pantera L"  "Ferrari Dino"
## [31] "Maserati Bora"    "Volvo 142E"
```

Cuando la base de datos tienen asignado nombres, es más fácil hacer referencia a las columnas por nombre y no por posición.

```
mtcars[,1]      #compara con mtcars$mpg

## [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

table

```
table(mtcars$hp)

##
##  52  62  65  66  91  93  95  97 105 109 110 113 123 150 175 180 205 215
##   1   1   1   2   1   1   1   1   1   1   3   1   2   2   3   3   1   1
## 230 245 264 335
##   1   2   1   1

table(mtcars$gear)

##
##  3  4  5
## 15 12  5

table(mtcars$gear, mtcars$hp)

##
##      52 62 65 66 91 93 95 97 105 109 110 113 123 150 175 180 205 215 230
##  3  0  0  0  0  0  0  0  1  1  0  1  0  0  2  2  3  1  1  1
##  4  1  1  1  2  0  1  1  0  0  1  2  0  2  0  0  0  0  0  0
##  5  0  0  0  0  1  0  0  0  0  0  0  1  0  0  1  0  0  0  0
##
##      245 264 335
##  3  2  0  0
##  4  0  0  0
##  5  0  1  1

table(mtcars$gear, mtcars$hp, mtcars$cyl)
```

```
## , , = 4
##
##
##      52 62 65 66 91 93 95 97 105 109 110 113 123 150 175 180 205 215 230
## 3  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0
## 4  1  1  1  2  0  1  1  0  0  1  0  0  0  0  0  0  0  0
## 5  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0  0
##
##      245 264 335
## 3  0  0  0
## 4  0  0  0
## 5  0  0  0
##
## , , = 6
##
##
##      52 62 65 66 91 93 95 97 105 109 110 113 123 150 175 180 205 215 230
## 3  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0  0
## 4  0  0  0  0  0  0  0  0  0  0  2  0  2  0  0  0  0  0
## 5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
##
##      245 264 335
## 3  0  0  0
## 4  0  0  0
## 5  0  0  0
##
## , , = 8
##
##
##      52 62 65 66 91 93 95 97 105 109 110 113 123 150 175 180 205 215 230
## 3  0  0  0  0  0  0  0  0  0  0  0  0  2  2  3  1  1  1
## 4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##
##      245 264 335
## 3  2  0  0
## 4  0  0  0
## 5  0  1  1
```

ftable

```
ftable(mtcars$gear, mtcars$hp,mtcars$cyl)

##      4 6 8
##
## 3 52  0 0 0
## 62  0 0 0
## 65  0 0 0
```

```
## 66 0 0 0
## 91 0 0 0
## 93 0 0 0
## 95 0 0 0
## 97 1 0 0
## 105 0 1 0
## 109 0 0 0
## 110 0 1 0
## 113 0 0 0
## 123 0 0 0
## 150 0 0 2
## 175 0 0 2
## 180 0 0 3
## 205 0 0 1
## 215 0 0 1
## 230 0 0 1
## 245 0 0 2
## 264 0 0 0
## 335 0 0 0
## 4 52 1 0 0
## 62 1 0 0
## 65 1 0 0
## 66 2 0 0
## 91 0 0 0
## 93 1 0 0
## 95 1 0 0
## 97 0 0 0
## 105 0 0 0
## 109 1 0 0
## 110 0 2 0
## 113 0 0 0
## 123 0 2 0
## 150 0 0 0
## 175 0 0 0
## 180 0 0 0
## 205 0 0 0
## 215 0 0 0
## 230 0 0 0
## 245 0 0 0
## 264 0 0 0
## 335 0 0 0
## 5 52 0 0 0
## 62 0 0 0
## 65 0 0 0
## 66 0 0 0
## 91 1 0 0
## 93 0 0 0
```



```
##    95    0 0 0
##    97    0 0 0
##   105    0 0 0
##   109    0 0 0
##   110    0 0 0
##   113    1 0 0
##   123    0 0 0
##   150    0 0 0
##   175    0 1 0
##   180    0 0 0
##   205    0 0 0
##   215    0 0 0
##   230    0 0 0
##   245    0 0 0
##   264    0 0 1
##   335    0 0 1

tt<- table(mtcars$gear, mtcars$hp)
is.table(tt)

## [1] TRUE
```

Convertir a vector o matriz lo que hay en tabla

```
dim(tt)

## [1]  3 22

ttn<-matrix(as.numeric(tt),3,22)
is.table(ttn)

## [1] FALSE

is.numeric(ttn)

## [1] TRUE
```

grep y subset

```
a <- data.frame( x = c('red','blue1','blue2', 'red2','redblue5', "yellow"))
grep("blue", a$x) #regresa los lugares donde la cadena "blue" se encuentra

## [1] 2 3 5

#Si se quiere encontrar los subconjuntos de a, se puede usar:
result <- a[ grep("blue", a$x) , ]
#o usando la function grepl(), que regresa un vector logico:
subset(a, grepl("blue", a$x))
```

```
##          x
## 2      blue1
## 3      blue2
## 5 redblue5

a[grepl("^[ry]", a$x),] #Despliega los renglones que en la variable a$x contenga una r o y

## [1] red      red2      redblue5 yellow
## Levels: blue1 blue2 red red2 redblue5 yellow
```

Listas

Dentro de los objetos de R las listas son las más generales. Una lista se puede pensar como un vector donde cada elemento es un objeto y estos pueden ser de tipos totalmente diferente. Al igual que los vectores las listas tienen una longitud, pero en contraste con los vectores, cada elemento no es llamado con `[.]` sino con `[[.]]`.

```
li<-as.list(rep(0,5))
length(li)

## [1] 5

li[[1]]

## [1] 0

li[[1]]<-"hola"
li[[2]]<-mat
li[[3]]<-matf
li[[4]]<-FALSE
li[[5]]<-pi

dim(li[[2]])

## [1] 3 3
```

Formas de crear listas

Una es como la anterior. Otras son:

```
Empl1<-list( "Ana", "Fred", 3, c(4,7,9))
Empl2 <- list(empleada = "Ana", pareja = "Fred", hijos = 3, edadesHijos = c(4,7,9))
#Contrasta Empl1 con Empl2
#Otra forma es concatenando listas
r<-c(list(rep(0,3)),list("hola"))
r

## [[1]]
## [1] 0 0 0
##
```

```
## [[2]]
## [1] "hola"

r<-c(una=list(rep(0,3)),dos=list("hola"))
r

## $una
## [1] 0 0 0
##
## $dos
## [1] "hola"
```

Cuando usamos la segunda opción, entonces además de poder hacer referencia al primer objeto como `r[[1]]`, también se puede llamar por nombre: `r$una`. Esto es particularmente importante cuando se conocen los nombres de los elementos pero se puede desconocer su posición dentro del ordenamiento de la lista.

Otros

Entre otros: Fechas, Resultados de funciones particulares, como el ajustes de modelo lineal. Las fechas pueden tratarse como elementos carácter, pero cuando así están almacenadas, no son fáciles de manipular. En cambio cuando se especifican como con formato “fecha” se pueden realizar fácilmente algunas operaciones, como la más común que es la diferencia de fechas.

```
#La fecha en el sistema de la maquina
Sys.Date()

## [1] "2020-03-05"

#Asignar formato de fecha usando 'as.Date'
miFecha<-as.Date("10-01-1993",format="%d-%m-%Y")
as.numeric(miFecha) #que corresponde al número de días desde el 1ro de Enero de 1970.

## [1] 8410

#Cambio del formato de fecha:
format(Sys.Date(), "%a %b %d") #otros formato por ejemplo "%Y %m %d"

## [1] "jue. mar. 05"

#Diferencia entre dos fechas
Sys.Date()-miFecha

## Time difference of 9916 days
```

Funciones

El formato de las funciones es el siguiente

```
myfunction<-function(parametro1,parametro2,...){  
  #comentarios  
  operaciones]  
  list(resultado1,resultado2)  
}
```

Los valores que puede recibir una función puede ser cualquier objeto R. Incluso, una función puede no requerir un valor para ejecutarse.

Si la función regresa una variable, se puede omitir el uso de “list” pero si se requiere que la función regrese más de un objeto, el uso de list es obligatorio.

A diferencia de otros softwares, R no requiere que se declaren las variables antes de utilizarlas.

Ejemplos:

```
saludo<-function(){  
  cat("Buen dia \n")  
}  
  
saludo()  
  
## Buen dia  
  
holamundo<-function(saludo,n){  
  #saludo es lógico  
  #n es entero  
  if(saludo){  
    cat(rep(" *Hola Mundo* ",n), "\n")  
  }else{  
    cat("Hola")  
  }  
}  
  
holamundo(TRUE,3)  
  
##  *Hola Mundo*   *Hola Mundo*   *Hola Mundo*  
  
holamundo(FALSE,3)
```

```
## Hola

holamundo(TRUE)

## *Hola Mundo*

holamundo(FALSE)

## Hola
```

Se pueden definir valores default en una función, así como lo hace R con funciones como `dnorm`. Ejemplo:

```
hipotenusa<-function(a=1,b=1){
  #a y b son las longitudes de los catetos
  sqrt(a^2+b^2)
}
hipotenusa()

## [1] 1.414214

hipotenusa(a=3,b=5)

## [1] 5.830952

hipotenusa(3,5)

## [1] 5.830952
```

Encontrando los errores en el código (debugging)

`browser()` hace una pausa dentro de la función que permite evaluar los valores que esta utilizando.

`traceback()` Identifica que función está generando errores o advertencias.

Ejemplo:

```
mmde<-function(vector,limite){      #funcion que obtiene estadísticas del vector, eliminando los valores
medias<-medianas<-des<-rep(0,length(limite))
for(i in 1:length(limite)){
  nuevo.vec<- vector[vector<=limite[i]]
  medias[i]<-mean(nuevo.vec)
  browser() # cuando se ejecute la función, pausará aca y todas las
            # variables internas estarán activas, con lo que se puede
            # examinar sus valores
  medianas[i]<-median(nuevo.vec)
  des[i]<-sd(nuevo.vec)
}
list(medias=medias,medianas=medianas,des=des)
}
```

```
res<- mmde(1:10,5:10)  
res
```

Ayuda

Lee los manuales en el sitio de R project.

```
#?help  
#help.search("variance")  
#help(nombre del comando) #si sabes exactamente el nombre del comando  
#_&&' #para operadores o palabras como if y for  
#help.start() for R online documentation
```

Sitios en internet, como:

- RSeek <http://www.rseek.org/>
- Revolutions <http://blog.revolutionanalytics.com/>
- R-bloggers <http://www.r-bloggers.com/>
- R Graph Gallery <http://addictedtor.free.fr/graphiques/>

Más sobre operaciones

```
#Operaciones Aritméticas  
#suma: + resta: -, división: /, producto: *, exponenciación: ^  
#modulo: %%  
#división entera: %/%  
#Operaciones Lógicas  
#Igual: == , No igual: != , Mayor, menor, mayor igual, menor igual: >, <, >=, <=  
#No: ! , ó: |, || y: &, &&  
3!=2  
  
## [1] TRUE  
  
3<2|4>3  
  
## [1] TRUE  
  
z1<-c(TRUE,FALSE,TRUE); z2<-c(FALSE,TRUE,TRUE)  
z1|z2  
  
## [1] TRUE TRUE TRUE  
  
z1||z2 #solo considera el primer elemento
```

```
## [1] TRUE

z1&z2

## [1] FALSE FALSE TRUE

z1&& z2

## [1] FALSE

z1[1]<-TRUE
z1|z2

## [1] TRUE TRUE TRUE

z1||z2 #solo considera el primer elemento

## [1] TRUE

z1&z2

## [1] FALSE FALSE TRUE

z1&& z2

## [1] FALSE
```

Funciones para manejar caracteres

```
#paste: combina elementos que son caracteres
paste("hola", "adios")

## [1] "hola adios"

paste("hola", "adios", sep="")

## [1] "holaadios"

paste("hola", "adios", sep="-")

## [1] "hola-adios"

paste("hola", 1:3)

## [1] "hola 1" "hola 2" "hola 3"

paste("hola", 1:3, collapse="")

## [1] "hola 1hola 2hola 3"

#substr: separa los caracteres
tt<-c( "21 22 23 24")
substr(tt, 1, 2)
```



```
## [1] "21"

strsplit(tt,split=" ")

## [[1]]
## [1] "21" "22" "23" "24"

unlist(strsplit(tt,split=" "))

## [1] "21" "22" "23" "24"

#cat: Despliega en pantalla caracteres u objetos
i<-4
cat("i= ",i,"\n")

## i= 4
```

Organización

Como organizar mis proyectos?

Directorio de trabajo

Uno, establece el directorio de trabajo para tu proyecto. ¿Cómo verlo? `getwd()`

¿Cómo cambiarlo? `setwd()`

Images

Las imagenes guardan todas las variables que se especifiquen para poder ser utilizadas nuevamente en otra sesion

```
save(x, mat, file = "xmat.RData")      #se especifican que variables guardar
load('xmat.RData')                    #para cargar la imagen al iniciar nuevamente la sesión o después
                                      #de haber borrado o sobrescrito una variable
save.image()                          #guarda el espacio de trabajo
                                      #(todas las variables en uso
                                      #durante esta sesión y R escribe el archivo .RData)
#alternativamente: File/Save Workspace. Cuando se inicia R, esta imagen se \carga" automáticamente.
```

Administración de objetos

Una sencilla forma de ver cuales objetos están en uso es usando `ls()`

Para ver cuales son los principales atributos de un objeto: `str(nombreObjeto)`

Paquetes

Cuando iniciamos R se cargan diversos paquetes que contienen las funciones más comunes para que el usuario pueda utilizarlas al momento.

Pero para optimizar en los recursos R no carga todas las funciones disponibles inmediatamente, sino que conjuntos de funciones se llaman cuando se quiere usar una función miembro.

El conjunto de funciones se llama genéricamente “Paquetes” y la forma de actualizarlas y llamarlas es por nombre.

Los paquetes agregan conjuntos de funciones que se orientan a hacer una tarea específica (graficar, calcular modelos lineales generalizados, hacer ajuste de rectas usando polinomios, etc). Y estos se tienen que instalar desde el sitio en R usando el menú de la interfase o `install.packages()`.

Una vez instalados se llaman con el comando `library(nombrePaquete)`

Para actualizar las versiones de los paquetes se utiliza `update.packages()`

Los manuales básicos de cada paquete se encuentran también en el sitio de R dentro de la página de cada uno.

`help(barchart)` `library(lattice)` `help(barchart)`—

Manejo de bases de datos

Archivos txt

Leer En un editor de texto capture la siguiente información: Juan 8 Ernesto 9 Marlene 8 Hector 7 Graciela 8 y guarde en el archivo ejercicio.txt `datos<-read.table("ejercicio.txt")`

Nombre Calif Juan 8 Ernesto 9 Marlene 8 Hector 7 Graciela 8 y guarde en el archivo ejercicio.txt