Una introducción a R & RStudio

Daniel Barráez, Alejandro Labarca

Mayo, 2018

Contenido

1. Unidad I: El entorno de R & RStudio

- Tema 1: Fundamentos básicos
  - Entorno, instalación y comandos básicos
  - Manipulaciones simples: números y vectores.
  - Objetos, sus modos y atributos.
  - Arreglos y matrices.
  - Listas y data frames.
  - Estructuras básicas de programación: Bucles, condicionantes y funciones.
  - Packages destacados: base, ggplot2, astsa, R Markdown, dplr, Shiny.
  - Importación y exportación de datos en distintos formatos.

Contenido

2. Unidad II: Análisis estadísticos en R

- Tema2: Estadística descriptiva y elaboración básica de gráficos.
- Tema3: Pruebas de hipótesis. Listas cuáles.
- Tema4: Modelo lineal uni-variado y multivariado. Estimación y diagnóstico.
- Tema5: Series de tiempo. La librería astsa. Modelos ARMA, ARIMA, SARIMA. Estimación y diagnóstico.

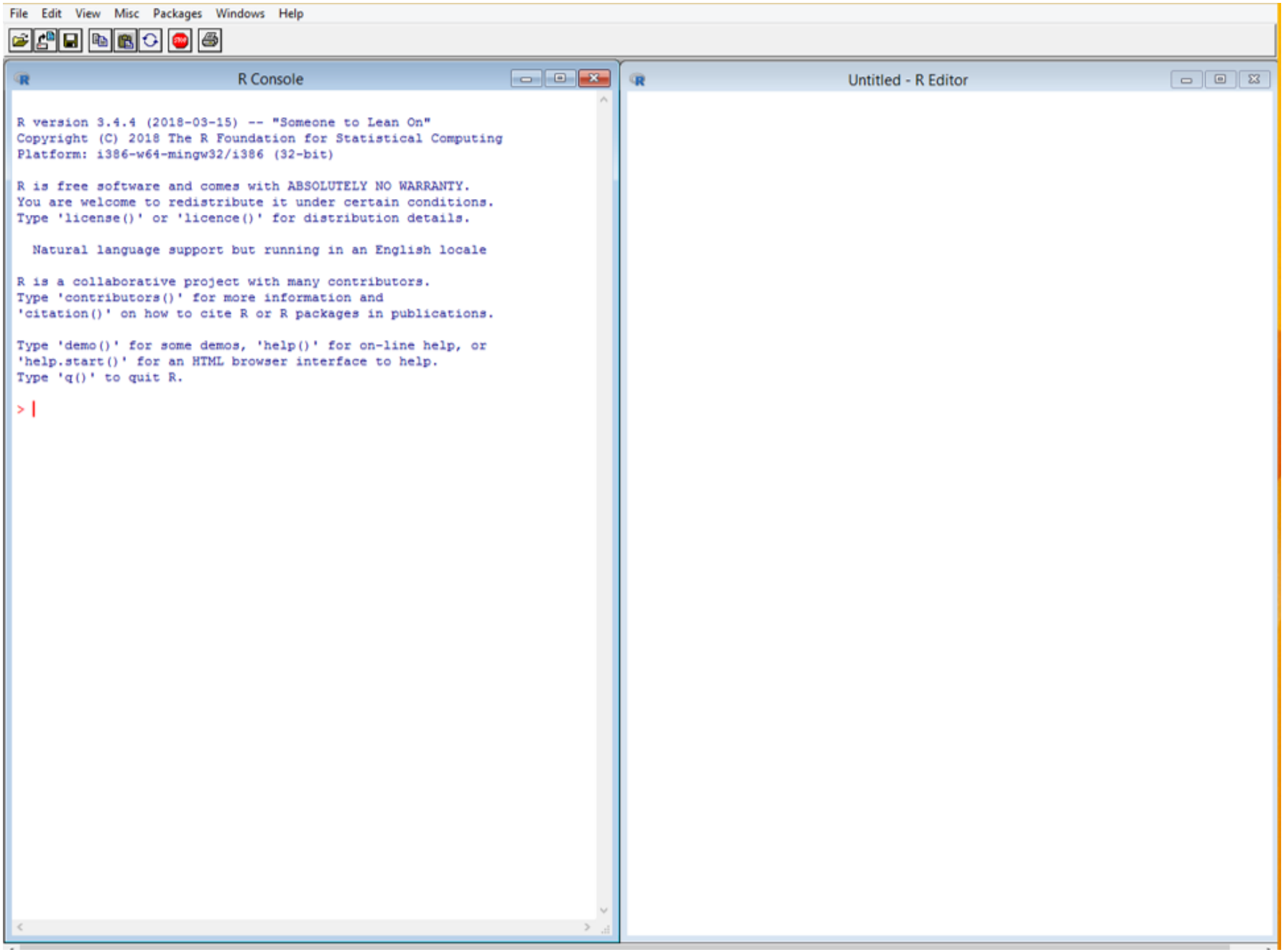
Historia

- Ross Isaka: "... descubrí un maravilloso libro de Hal Abelson y Gerald Sussman llamado THE STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS..."
- Ross Isaka y el lenguaje S de Rick Becker y John Chambers: similitudes y diferencias entre S y Scheme.
- Ross Isaka y Robert Gentleman: Universidad de Auckland. Interés en la informática estadística.
- En agosto de 1993 fué el primer anuncio en el laboratorio de enseñanza. En junio de 1995 lanzan el código fuente R como "software libre"

El entorno de R

R es un entorno de software libre para computación y gráficos estadísticos. Entre otras cosas, posee:

1. Una efectiva manipulación y facilidad de almacenamiento de los datos.
2. Operadores para el cálculo en arreglos, en particuar matrices.
3. Una amplia, coherente, colección integrada de herramientas para el análisis de los datos.
4. Facilidades gráficas para el análisis y presentación de los datos.
5. Un buen desarrollador, simple y efectivo lenguaje de programación (llamado 'S') que incluye condicionales, bucles, funciones recursivas definidas por el usuario y entradas y salidas con gran facilidad. (en efecto, la mayoría de las funciones suministradas por el sistema esta escrito en lenguaje S).


File Edit View Misc Packages Windows Help
R Console
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Untitled - R Editor

Ranking peers

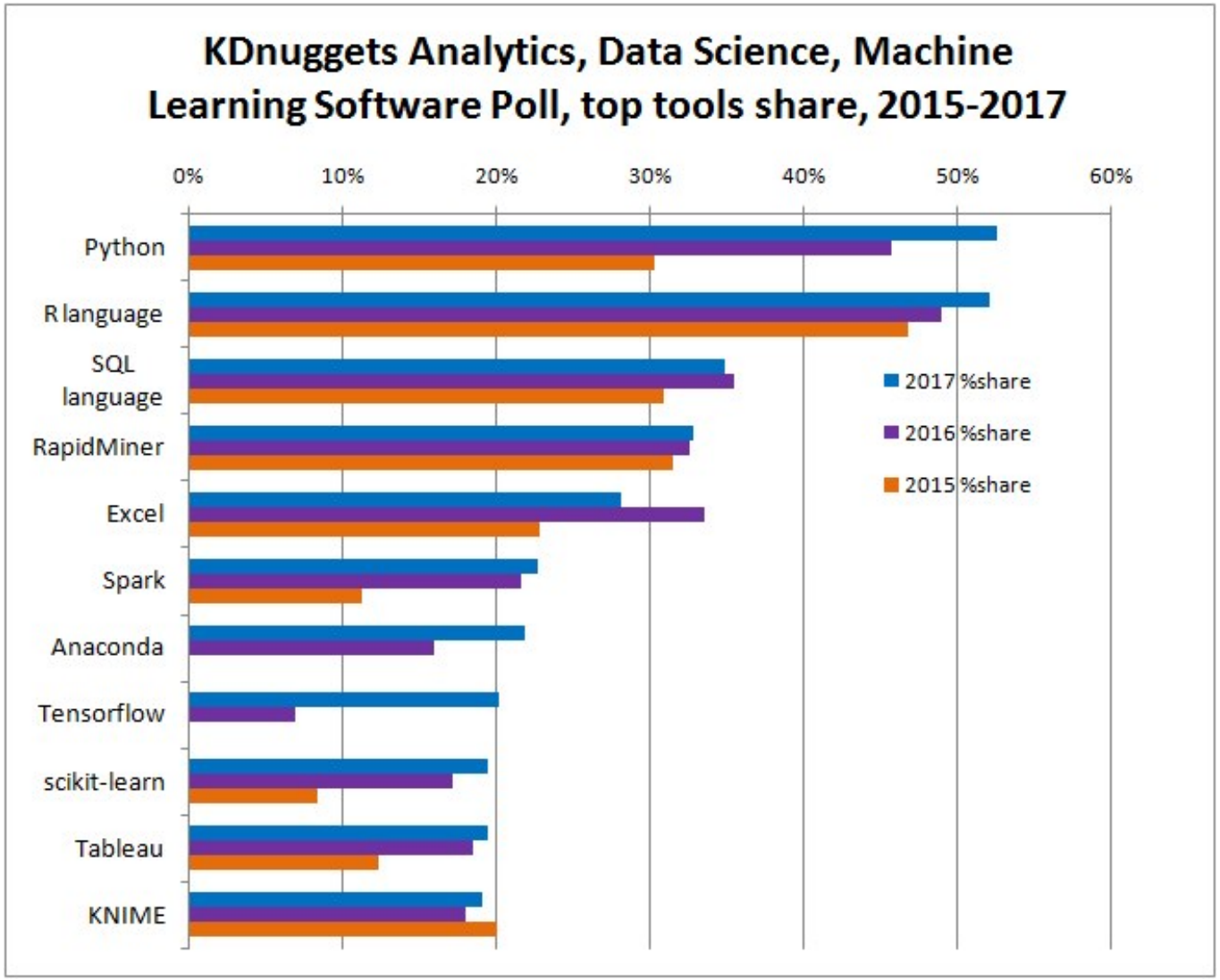


Figure 2:

La distribución R posee una versión HTML (https://stat.ethz.ch/R-manual/) actualizada donde se disponen los manuales de referencia:

1.- An Introduction to R.

2.- R Data Import/Export.

3.- R Installation and Administration.

4.- Writing R Extensions.

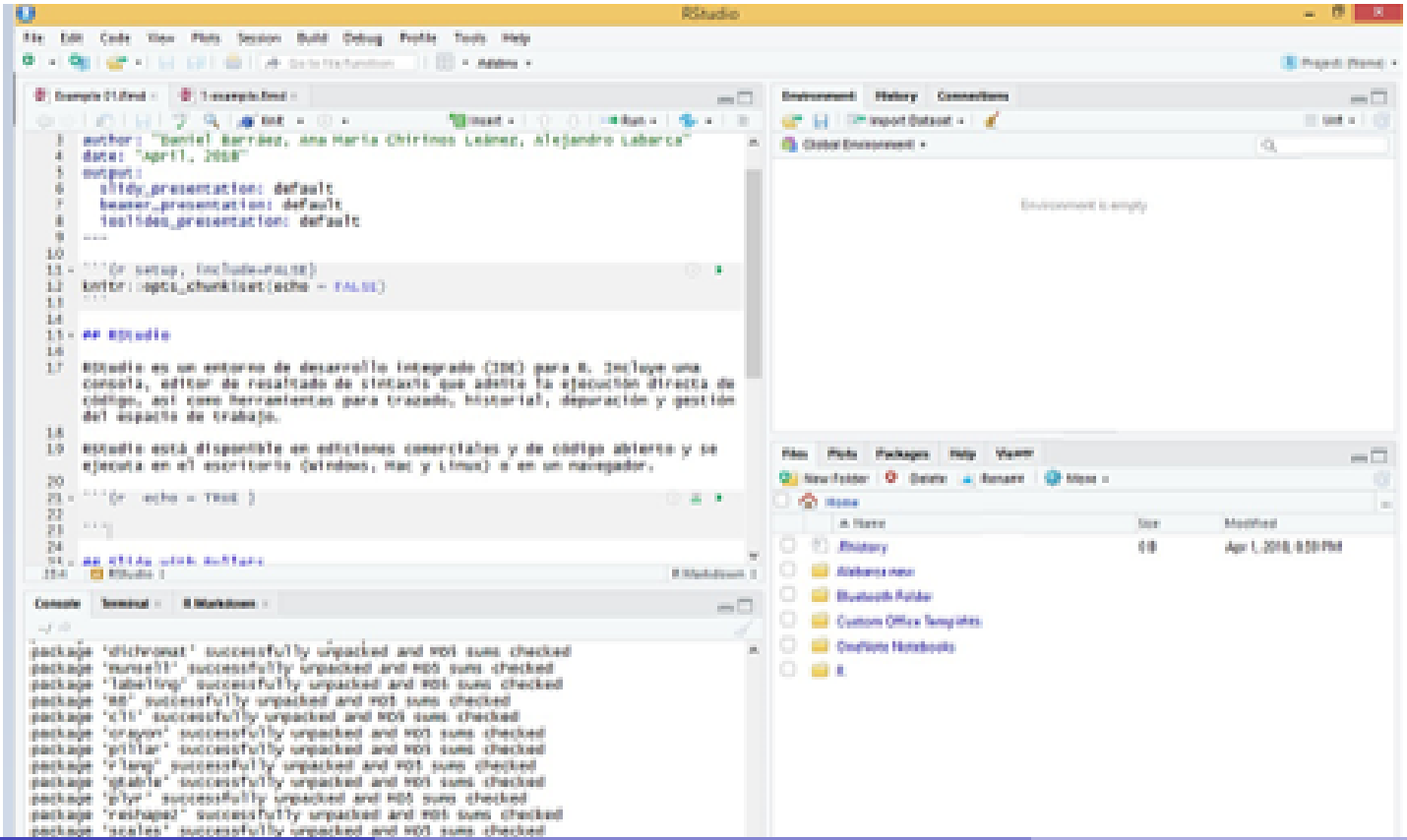
5.- R Internals.

6.- The R Reference Index.

Foros, blogs, grupos, seminarios: #rstats hashtag, R-Ladie, Rweekly, R-bloggers, DataCarpentry y Software Carpentry, Stack Overflow, R Conferences, Github, The R Consortium.

RStudio

RStudio es un entorno de desarrollo integrado (IDE) para R. Incluye una consola, editor de resaltado de sintaxis que admite la ejecución directa de código, así como herramientas para trazado, historial, depuración y gestión del espacio de trabajo.



Consideraciones previas

Para instalar los programas, es necesario comenzar con R y luego RStudio. Debemos descargar las últimas versiones en archivos ejecutables que proporcionan las páginas oficiales, a través de los siguientes enlaces: https://www.r-project.org/ ; https://www.rstudio.com/

Luego de la instalación, es necesario comprender algunos requerimientos básicos para comenzar a programar:

1. En la consola de R el símbolo '>', indica que R está listo para recibir un comando.
2. R es un lenguaje Orientado a Objetos, siendo la sintáxis muy simple e intuitiva. Ejemplo (regresión lineal): lm(y~x).
3. El nombre de un objeto debe comenzar con una letra (A-Z and a-z) y puede incluir letras,dígitos (0-9), y puntos (.). R discrimina entre letras mayúsculas y minúsculas para el nombre de un objeto, de tal manera que x y X se refiere a objetos diferentes (inclusive bajo Windows).

4. El usuario puede modificar o manipular estos objetos con operadores (aritméticos, lógicos, y comparativos) y funciones (que a su vez son objetos)
5. Las funciones disponibles estan guardadas en una librería localizada en el directorio R HOME/library (R HOME es el directorio donde R esta instalado).
6. El paquete (conjunto de funciones) denominado base constituye el nucleo de R y contiene las funciones básicas del lenguaje para leer y manipular datos, algunas funciones gráficas y algunas funciones estadísticas (regresión lineal y análisis de varianza)
7. Para que una función sea ejecutada en R debe estar siempre acompañada de paréntesis, inclusive en el caso que no haya nada dentro de los mismos (por ej., ls()).

Comandos básicos

Creación, listado y remoción de objetos en memoria

1. El comando más simple es escribir el nombre de un objeto para visualizar su contenido. Por ejemplo, si un objeto n contiene el valor 10:

```
n<-10
n
```

```
## [1] 10
```

El dígito 1 indica que la visualizacion del objeto comienza con el primer elemento de n. Este comando constituye un uso implícito de la función print, y el ejemplo anterior es similar a print(n).

2. Un objeto puede ser creado con el operador "asignar" el cual se denota como una flecha con el signo menos y el símbolo "$>$" o "$<$" dependiendo de la dirección en que asigna el objeto:

```
n<-15
n
```

```
## [1] 15
```

```
5->n
n
```

```
## [1] 5
```

```
x<-1
X<-10
x
```

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

Si el objeto ya existe, su valor anterior es borrado después de la asignación (la modificación afecta solo objetos en memoria, no a los datos en el disco). El valor asignado de esta manera puede ser el resultado de una operación y/o de una función:

```
n<- 10 + 2
n
```

```
## [1] 12
```

```
n<-3 + rnorm(1)
n
```

```
## [1] 2.63686
```

La función rnorm(1) genera un dato aleatorio cuya distribución de probabilidad es una normal con media 0 y varianza 1.

3. La función ls lista los objetos en memoria: sólo arroja los nombres de éstos.

```
name<-"Carmen"; n1<-10; n2<-100; m<-0.5
ls()
```

```
## [1] "m"     "n"     "n1"    "n2"    "name" "x"     "X"
```

4. Note el uso del punto y coma para separar comandos diferentes en la misma línea. Si se quiere listar solo aquellos objetos que contengan un caracter en particular, se puede usar la opción pattern (que se puede abreviar como pat):

```
ls(pat="m")
```

```
## [1] "m"     "name"
```

5. La función ls.str() muestra algunos detalles de los objetos en memoria:

```
ls.str()
```

```
## m :  num 0.5
## n :  num 2.64
## n1 :  num 10
## n2 :  num 100
## name :  chr "Carmen"
## x :  num 1
## X :  num 10
```

La opción pattern se puede usar de la misma manera con ls.str(). Otra opción útil en esta función es max.level la cual especifica el nivel de detalle para la visualización de objetos compuestos. Por defecto, ls.str() muestra todos los detalles de los objetos en memoria, incluyendo las columnas de los marcos de datos ("data frames"), matrices y listas.

```r
M<-data.frame(n1,n2,m)
ls.str(pat="M")
```

```
## M : 'data.frame':    1 obs. of  3 variables:
##  $ n1: num 10
##  $ n2: num 100
##  $ m : num 0.5
```

6. Para borrar objetos en memoria, utilizamos la función rm(): rm(x) elimina el objeto x, rm(x,y) elimina ambos objetos x y y, y rm(list=ls()) elimina todos los objetos en memoria; las mismas opciones mencionadas para la función ls() se pueden usar para borrar selectivamente algunos objetos: rm(list=ls(pat="^m"))

La ayuda en línea

Proporciona información muy útil de cómo utilizar las funciones. La ayuda se encuentra disponible directamente para una función dada. Por ejemplo: ?lm o help(lm) mostrará dentro de R, ayuda para la función lm() (modelo lineal)

Manipulaciones simples: Números y vectores

Vectores. Asignación

R tiene cinco clases básicas o "atomic" para objetos:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

El tipo de objeto más simple en R es un vector. En realidad, hay una sola regla sobre vectores en R: Un vector solo puede contener objetos de la misma clase. La excepción a la regla previa la contienen las listas, que veremos un poco más adelante. Una lista se representa como un vector, pero puede contener objetos de diferentes clases. De hecho, esa es la razón por la que los usamos!

La función c() se puede usar para crear vectores de objetos mediante la concatenación de elementos.

```
x <- c(0.5, 0.6)        ## numérico
x <- c(TRUE, FALSE)     ## lógico
x <- c(T, F)            ## lógico
x <- c("a", "b", "c")   ## caracter
x <- c(1+0i, 2+4i)      ##complejo
x <- 4:9               ## entero (secuencia de números)
```

Con la siguiente asignación, crea un vector con 11 elementos consistentes en dos copias de x con un cero entre ambas.

```
y <- c(x, 0, x)
```

El comando as.factor() es útil para "etiquetar" los elementos de un objeto. Un comando que permite visualizar los elementos de un objeto, excluyendo las repeticiones, es a través del comando unique()

Si queremos extraer elementos de un vector utilizamos [ ]:

```
y[1:3]
```

```
## [1] 4 5 6
```

Generación de sucesiones

Si queremos que la secuencia de números se realice cada 2 pasos, utilizamos el comando seq():

```
seq(from=1,to=15,by=2)
```

```
## [1]  1  3  5  7  9 11 13 15
```

Para mostrar vectores cuyas entradas son iguales, se puede utilizar el comando rep(). Es útil, para reducir tiempo de programación.

```
rep(x=5,times=15)
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

Mezcla de objetos

Cuando diferentes objetos se mezclan en un vector, la coerción ocurre de modo que cada elemento en el vector es de la misma clase:

```
c(1.7, "a")    ## caracter
```

```
## [1] "1.7" "a"
```

```
c(TRUE, 2)     ## numérico
```

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

```r
c("a", TRUE)  ## caracter
```

```
## [1] "a"    "TRUE"
```

Los objetos pueden ser forzados explícitamente de una clase a otra usando las funciones as. *, Si están disponibles.

```r
x <- 0:6
class(x)
```

```
## [1] "integer"
```

```r
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5 6
```

```r
as.logical(x)
```

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

```r
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

Aritmética vectorial

- Los vectores pueden usarse en expresiones aritméticas, en cuyo caso las operaciones se realizan elemento a elemento.
- Los operadores aritméticos elementales son los habituales +, -, *, / y ^ para elevar a una potencia. Además están disponibles las funciones log, exp, sin, cos, tan, sqrt, bien conocidas.

Objetos: modos y atributos

En general, los objetos R pueden tener "attributes", que son como metadatos para el objeto. Estos metadatos pueden ser muy útiles ya que ayudan a describir el objeto. Por ejemplo, los nombres de una columna en un marco de datos ayudan a decirnos qué datos están contenidos en cada una de las columnas. Algunos ejemplos de atributos de objetos R son:

- names, dimnames
- dimensions (por ejemplo, matrices, arreglos)
- class (por ejemplo, entero, numérico)
- length
- Otros atributos / metadatos definidos por el usuario

Se puede acceder a los atributos de un objeto (si corresponde) utilizando la función attributes(). No todos los objetos R contienen atributos, en cuyo caso la función attributes() devuelve NULL.

Ejercicios

1. Construir un vector de clase caracter compuesto por los siguientes elementos : 4,5,6,7,. . . ,90,92,94,96,98,100,1,4,6,“a”,“ab”,“abc”. Contar el número de elementos menores a 13. Sugerencia: Utilizar el operador lógico $<$ ó $>$ y la función sum()
2. Considerar el vector de tipo numérico compuesto por los siguientes elementos: 34,56,55,87,NA,4,77,NA,21,NA,21. (i) Contar el número de elementos iguales a NA y eliminarlos. (ii) Determinar cuántos niveles “etiquetas” tiene el vector. Sugerencia: Utilizar la función is.na() en (i).

Matrices y arreglos

Matrices

Las matrices son vectores que posee la dimensión de un vector de tamaño 2 como atributo, el cual especifica el número de filas y columnas.

```
m <- matrix(nrow = 2, ncol = 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```
m <- matrix(1:6,nrow=2,ncol=3)
m
```

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

Pueden ser creadas a través de funciones alternativas:

- Vectores, añadiendo la dimensión atributo.

```
m <- 1:10
m
```

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

```
dim(m)<-c(2,5)
m
```

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

- Enlace de columnas y filas: funciones cbind() y rbind()

```
x <- 1:3
y <- 10:12
cbind(x, y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```
rbind(x, y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

Arreglos

Un arreglo es una colección de datos indexada por varios índices. Las matrices, son un caso particular de esta colección de datos.En R, hay dos maneras de crear arreglos:

- Definiendo un vector y luego estableciendo las dimensiones a través del comando dim(),

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

```
## , , 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
```

- Utilizando el comando array(),

```
array(z,dim = c(3,4,2))
```

```
## , , 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
```

Ejercicios

3. Crear un vector x,y,z de tipo entero de tres elementos cada uno. (i) Combinar los tres vectores en una matriz 3x3, que llamaremos A, donde cada columna es un vector. (ii) Cambiar el nombre de las columnas por a,b,c. (iii) Visualizar la segunda fila y el elemento A_{3X2}. Sugerencia: En (ii) utilizar el comando dimnames()[[2]].
4. Crear un vector con 12 enteros. Convertir el vector en una matriz 4x3, denominada B usando el comando matrix(). (i) Cambiar el nombre de las columnas por x,y,z y las filas por a,b,c,d. (ii) Realizar la transpuestas de B y calcular la multiplicación de matrices t(B)xB. (iii) Extraer los elementos distintos de cero. Sugerencia: utilizar el comando dimnames() en (i), el operador aritmético %*% en (ii), el operador lógico != en (iii).

Listas y data frames

En R, una lista es un objeto cuyos elementos son una colección ordenada de objetos, conocidos como componentes. No es necesario que los componentes sean del mismo modo.El siguiente es un ejemplo de una lista:

```
Lst<-list(nombre="Pedro", esposa="María",no.hijos=3)
Lst
```

```
## $nombre
## [1] "Pedro"
##
## $esposa
## [1] "María"
##
## $no.hijos
## [1] 3
```

- Los componentes siempre están numerados y pueden ser referidos por dicho número. En este ejemplo, Lst es el nombre de una lista con cuatro componentes, cada uno de los cuales puede ser referido, respectivamente, por Lst[[1]], Lst[[2]], Lst[[3]] y Lst[[4]]. Como, además, Lst[[4]] es un vector, Lst[[4]][1] refiere su primer elemento. La función length() aplicada a una lista devuelve el número de componentes (del primer nivel) de la lista.
- La función length() aplicada a una lista devuelve el número de componentes (del primer nivel) de la lista.
- Los componentes de una lista pueden tener nombre, en cuyo caso pueden ser referidos también por dicho nombre, mediante una expresión de la forma nombre de lista$nombre de componente. También es posible utilizar los nombres de los componentes entre dobles corchetes [[]]

- Es muy importante distinguir claramente entre Lst[[1]] y Lst[1]. '[[]]' es el operador utilizado para seleccionar un sólo elemento, mientras que '[]' es un operador general de indexado. Esto es, Lst[[1]] es el primer objeto de la lista Lst, y si es una lista con nombres, el nombre no está incluido. Por su parte, Lst[1], es una sublista de la lista Lst consistente en la primera componente. Si la lista tiene nombre, éste se transfiere a la sublista.

Hoja de datos (Data frames)

La hoja de datos se usan para almacenar datos tabulares en R. Son un tipo importante de objeto en R y se usan en una variedad de aplicaciones de modelado estadístico. El paquete dplyr de Hadley Wickham tiene un conjunto optimizado de funciones diseñadas para funcionar eficientemente con hojas de datos, y las funciones de trazado ggplot2 funcionan mejor con datos almacenados en hojas de datos.

- Se representan como un tipo especial de lista donde cada elemento de la lista debe tener la misma longitud. Cada elemento de la lista se puede considerar como una columna y la longitud de cada elemento de la lista es el número de filas.
- A diferencia de las matrices, las hojas de datos pueden almacenar diferentes clases de objetos en cada columna, a diferencia de las matrices que deben tener cada elemento de la misma clase (por ejemplo, todos los enteros o todos los numéricos).
- Puede construir una hoja de datos utilizando la función data.frame():

```
cont <- data.frame(A=cars$speed, B=cars$dist)
```

Puede forzar que las componentes una lista cumplan las restricciones para ser una hoja de datos, mediante la función as.data.frame(). La manera más sencilla de construir una hoja de datos es utilizar la función read.table() o read.csv, para leerla desde un archivo del sistema operativo. Esta forma se tratará más adelante.

Bucles, condicionantes y funciones

Una ventaja de R comparado con otros programas estadísticos con "menus y botones" es la posibilidad de programar de una manera muy sencilla una serie de análisis que se puedan ejecutar de manera sucesiva.

Supongamos que tenemos un vector x, y para cada elemento de x con valor igual a b, queremos asignar el valor 0 a otra variable y, o sino asignarle 1. En R:

```
y <- numeric(length(x))

for (i in 1:length(x)){

if (x[i] == b) { y[i] <- 0}

else{y[i] <- 1}

}
```

Otra posibilidad es ejecutar una instruccion siempre y cuando se cumpla una cierta condición:

while (myfun $>$ minimum) $\{ \ldots \}$

Sin embargo, este tipo de bucles y estructuras se pueden evitar gracias a una característica clave en R: vectorización. La vectorizacion hace los bucles y condicionantes implícitos en las expresiones. En el ejemplo previo basta con ejecutar:

```
y[x == b] <- 0
y[x != b] <- 1
```

Tambien existen varias funciones del tipo "apply" que evitan el uso de bucles. apply() actua sobre las filas o columnas de una matriz, y su sintaxis es apply(X, MARGIN, FUN,...), donde X es una matriz, MARGIN indica si se van a usar las filas (1), las columnas (2), o ambas (c(1, 2)), FUN es una funcion a ser aplicada, y ... son posibles argumentos opcionales de FUN.

```r
x <- rnorm(10, -5, 0.1)
y <- rnorm(10, 5, 2)
# las columnas de X mantienen los nombres "x" y "y"
X <- cbind(x, y)
apply(X, 2, mean)
```

```
##          x          y
## -4.909422  4.801128
```

```r
apply(X, 2, sd)
```

```
##          x          y
## 0.1325613 2.0976230
```

Otras funciones que actúan de manera similar: lapply(), sapply(), mapply()

Para definir una función debe realizar una asignación de la forma

`NombreDeFuncion <-function(arg 1, arg 2, ...) {exp.}`

donde exp., es una expresión de R que utiliza los argumentos, arg i, para calcular un valor que es devuelto por la función. Consideremos una función que calcule el estadístico t de Student para dos muestras:

```
DosMuestras <- function(y1, y2) {
n1 <- length(y1); n2 <- length(y2)
yb1 <- mean(y1); yb2 <- mean(y2)
s1 <- var(y1); s2 <- var(y2)
s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
tst <- (yb1 - yb2)/sqrt(s2*(1/n1 + 1/n2))
gl<-(((s1/n1)+(s2/n2))^2)/((((s1/n1)^2)+((s2/n2)^2))/(n2-1))
pvalor=dt(tst,df=gl)
M=matrix(nrow=1,ncol=3,data = c(tst,gl,pvalor))
dimnames(M) = list(c("test-t"),c("t-statistic","gl","p-value"))
return(M)
}
```

Una vez definida esta función, puede utilizarse para realizar un contraste de t de Student para dos muestras, del siguiente modo:

```
tStudent <- DosMuestras(sleep[sleep$group==1,]$extra,
sleep[sleep$group==2,]$extra)
tStudent
```

```
##         t-statistic       gl    p-value
## test-t    -1.76451 17.77647 0.08644974
```

Importación y exportación de datos

R brinda una amplia variedad de comandos para trabajar con información que se encuentra en distintos tipos de formatos, permitiendo un alto grado de flexibilidad al momento de realizar análisis específicos con la herramienta. Con el siguiente link descargamos la data VE_INE.csv: https://github.com/alabarca/Curso-R/upload/master

1. .txt/.csv:

   - package base: Si utilizamos la instalación recomendada en R, este paquete ya viene incluido. Comandos:

```
read.table(file = "VE_INE.csv",sep=",",header=TRUE)#importar archivo

write.table(VEN_INE,file="VE_INE2.csv",sep=",",dec=".")
#exportar archivo
```

- package readr: es una buena idea especificar los tipos de columnas explícitamente. Esto descarta cualquier posible error de adivinación por parte de read.table.:

```
read_csv(file = "VE_INE.csv",col_types="ccn",nmax=10)
```

2. .xlsx:

- package readxl: es un paquete diseñado para hacer una sola tarea, importar hojas de Excel a R. Esto hace que sea un paquete ligero y eficiente, a cambio de no contar con funciones avanzadas.

Es compatible con hojas de cálculo de Excel 97-03, con extensión .xls, y con hojas de cálculo de las versiones más recientes de Excel, con extensión .xlsx. Si una celda de una pestaña contiene una fórmula, se importa es el resultado de esa fórmula.

Antes de empezar, necesitamos conocer el contenido de nuestra hoja de cálculo. Podemos usar la función excel_sheets() para conocer qué pestañas contiene nuestra hoja de cálculo sin salir de R

```
excel_sheets("VE_INE.xlsx")
```

```
## [1] "VE_INE"
```

Usaremos la función read_excel() indicando la ruta del documento que queremos importar a nuestro espacio de trabajo de R, sin parámetros adicionales.Asignaremos el resultado de esta función al objeto excel_INE

```
#excel_INE <- read_excel("VE_INE.xlsx",n_max=100)
```

Es posible explorar el objeto a través de los comandos str(), summary(), head(), tail().