AIS: Capacitación de R

Oliab Herrera Coria

2023-02-09

Contents

Información del curso **5**

Temario 5

1 Introducción a R. **7**

1.1 Instalación de R y R Studio. 7

1.2 Entorno de trabajo de RStudio. 9

1.3 Configuración del espacio de trabajo 14

1.4 Opción 2 : Proyecto de R. 14

1.5 Instalación de paquetes. 15

1.6 Ayuda en R. 17

2 Manipulación y visualización de datos **19**

2.1 Comandos básicos de R. 19

2.2 Manipulación y limpieza de datos. 32

Información del curso

- Este es un curso de Introducción a R.
- Al final serán capaces de utilizar R para cargar datos, arreglarlos, hacer gráficos y tablas, e informes en Rmarkdown.
- Intentaremos que el curso sea fundamentalmente práctico.
- En lugar de presentar todos los pormenores de R de manera lineal, se irán presentando distintos aspectos de R conforme se vayan necesitando; es decir, no vamos a presentar R como un lenguaje de programación sino como una herramienta para hacer análisis estadísticos.

E- n la carpeta del curso están todos los materiales: tutoriales, algunos datos, etc….

Ligas

Notas: https://fastidious-brioche-36f6dd.netlify.app/ Correo: oliabherrera@gmail.com

Temario

1. **Introducción a R**

- Instalación de R y R Studio.
- Entorno de trabajo de RStudio.
- Instalación de paquetes.
- Ayuda en R.

2. **Manipulación y visualización de datos**

- Comandos básicos de R.

- Manipulación y limpieza de datos.
- Visualización de datos.
- Temas selectos de programación en R.

3. **Reportes POS**

- Descarga de datos.
- Scripts principales.
- Proyecciones.

Software

- https://www.r-project.org
- https://www.rstudio.com

Chapter 1

Introducción a R.

El objetivo de este tutorial es familiarizarnos con el entorno de trabajo que proporciona R y RStudio. Al finalizar este tutorial también deberemos ser capaces de instalar y cargar los paquetes que vayamos a necesitar para realizar nuestros análisis de datos.

1.1 Instalación de R y R Studio.

Para instalar R vamos a la página web de R project: http://www.r-project.org

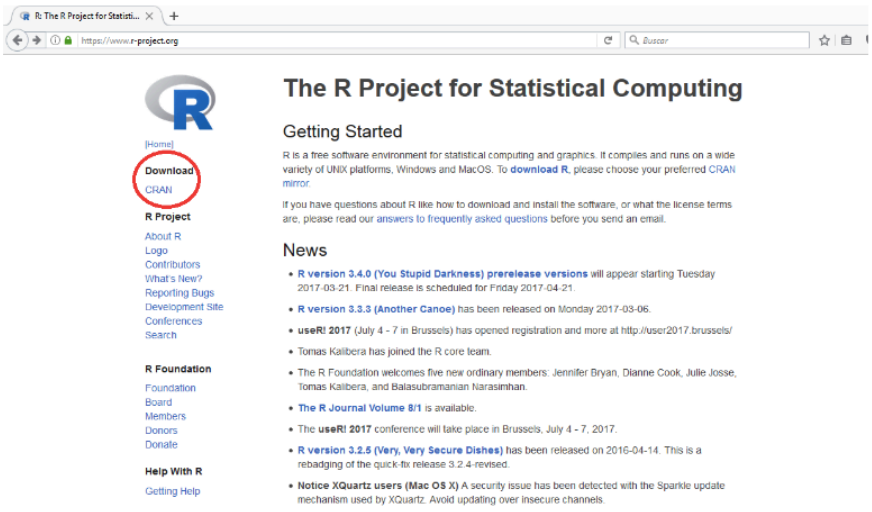


Figure 1.1: Figura 1

Para descargar la aplicación hacemos clic en Cran y pinchamos sobre el enlace del "espejo" más próximo a nuestra ubicación, México. Seleccionemos la URL de, por ejemplo (https://cran.itam.mx/).

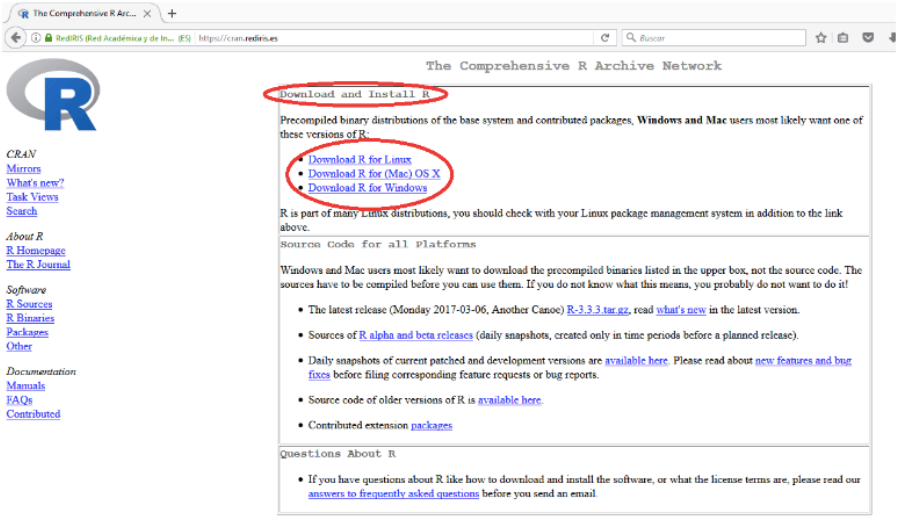
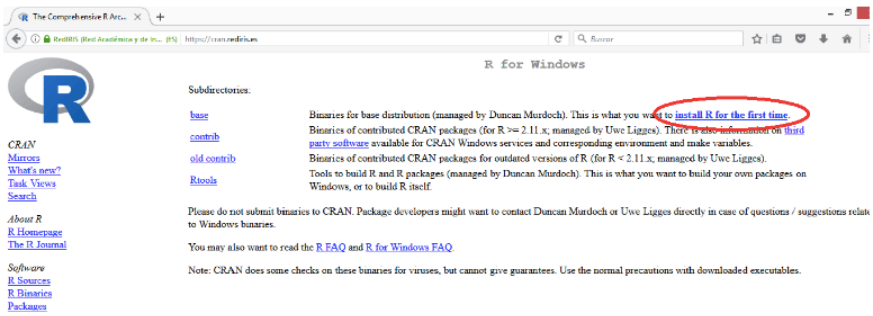


Figure 1.2: Figura 2

Ahora, en función del tu sistema operativo, seleccionar la correspondiente opción.



Instalar R en Windows.

Al hacer clic sobre Download R for Windows iremos a la página que se reproduce más abajo. Hacer clic sobre *install R for the first time.*

En la siguiente ventana, hacer clic sobre *Download R 3.3.3 for Windows* y guardar el archivo de instalación.

Ejecutar el archivo descargado para proceder a la instalación de R.

1.1.1 Instalar R en Mac.

Al hacer clic sobre Download R for (Mac) OS X iremos a la página que se reproduce más abajo. Hacer clic sobre install R for the first time.

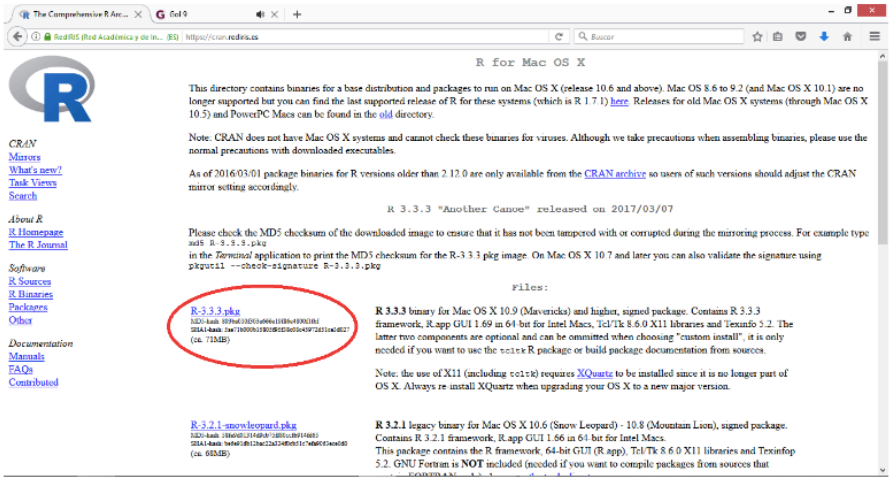


Figure 1.3: Figura 4

Hacer clic sobre R-3.3.3.pkg y guardar el archivo de instalación. Ejecutar el archivo descargado para proceder a la instalación de R.

1.1.2 Instalar RStudio

Descargamos la aplicación desde la página web de RStudio aquí según nuestra plataforma de trabajo:

- RStudio 1.0.136 - Windows Vista/7/8/10.
- RStudio 1.0.136 - Mac OS X 10.6+ (64-bit).

Una vez guardado el archivo, lo ejecutamos para instalar RStudio. Sigue las instrucciones de instalación.

1.2 Entorno de trabajo de RStudio.

En general trabajamos con la interfaz de RStudio antes que con la de R porque la primera es “más amigable”.

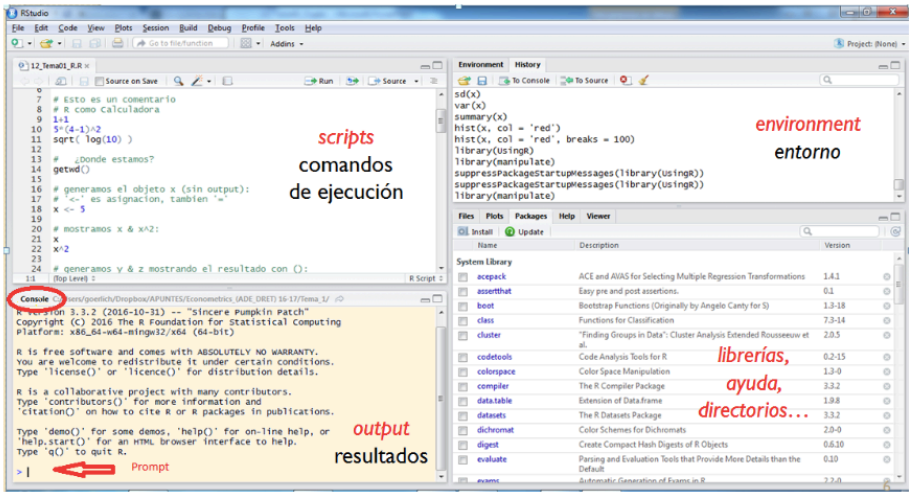


Figure 1.4: Figura 5

Si abrimos RStudio vamos a ver algo parecido a lo que se muestra en la siguiente imagen:

Una vez estamos en RStudio, podemos escribir y ejecutar las órdenes de varias formas:

- directamente en la consola.
- a través de un script (.R).
- con ficheros Rmarkdown (.Rmd).

Como podemos ver, RStudio está (normalmente) dividido en 4 paneles.

1.2.1 Consola

Por defecto, la consola se encuentra en el panel inferior-izquierdo. ¿Vemos la pestaña que pone Console? Inmediatamente debajo aparece un texto informativo y, finalmente, el símbolo ">". Aquí es donde R espera que le demos instrucciones. Para ejecutarlas y obtener el resultado pulsamos enter.

Vamos a hacer este ejemplo:

```
2+2
```

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

```
5*(3-1)^2
```

```
## [1] 20
```

```
sqrt(4)
```

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

1.2.2 Scripts

Trabajar en la consola es muy limitado ya que las instrucciones se han de introducir una a una. Lo habitual es trabajar con scripts o ficheros de instrucciones. Estos ficheros tienen extensión **.R**.

Se puede crear una script con cualquier editor de texto (uno de los más populares es Tinn-R), pero nosotros lo haremos desde RStudio. Para ello, seleccionamos la siguiente ruta de menús: **File > New File > R script**

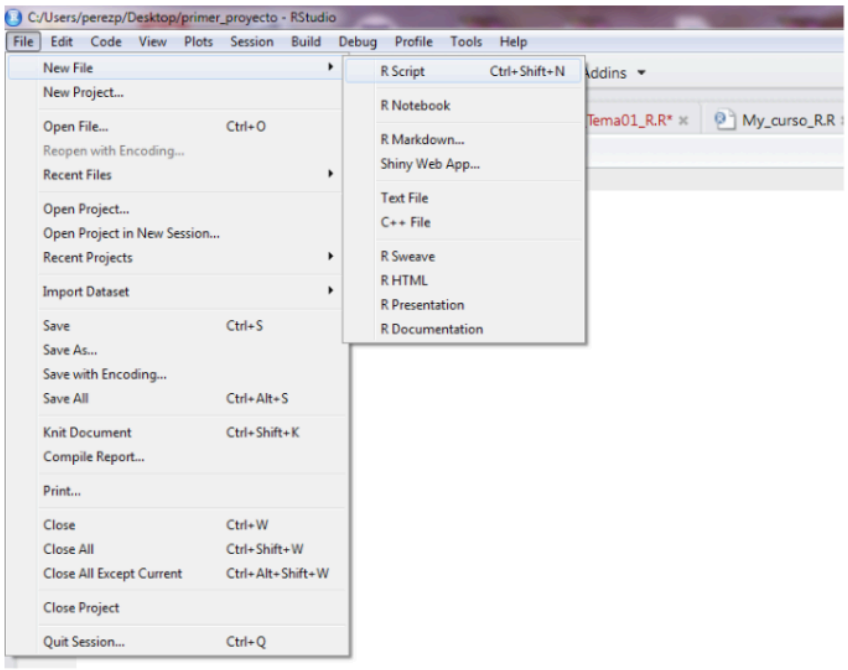


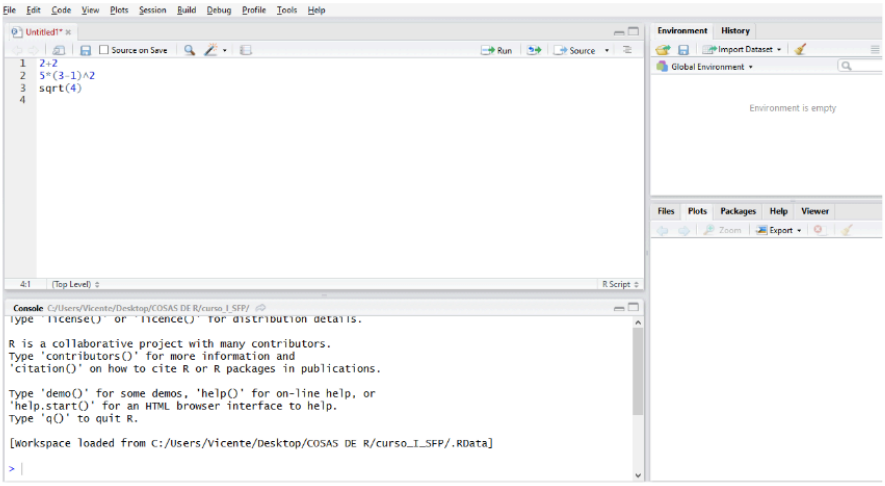
Figure 1.5: Figura 5

El panel del script se sitúa en la parte superior-izquierda de RStudio. Ahora podemos escribir las instrucciones línea por línea. Las instrucciones las podemos

ejecutar una a una o las podemos seleccionar y ejecutar en bloque. Para ejecutar las instrucciones tenemos varias alternativas:

- Hacemos clic en el botón: **Run** (botón situado en la parte derecha de las opciones del panel de script)
- Pulsamos Ctrl+r
- Ejecutamos el código desde las opciones del menú Code. Sinceramente, esto nunca lo hemos utilizado. ¡Cuestión de comodidad!

Como se muestra en la imagen más abajo, vamos a escribir nuestro primer script.



Para guardar el script:

- File > Save as.. y seleccionar la ruta donde se quiere guardar el fichero.
- Hacer clic en el botón Guardar que se encuentra en la parte izquierda de la cinta de opciones del script.

1.2.3 Entorno

El panel, llamémoslo, de entorno esta compuesto de dos pestañas: Environment y History.

- En el Environment se irán registrando los objetos que vayamos creando en la sesión de trabajo. También tenemos la opción de cargar y guardar una sesión de trabajo, importar datos y limpiar los objetos de la sesión. Estas opciones están accesibles a través de la cinta de opciones de la pestaña.

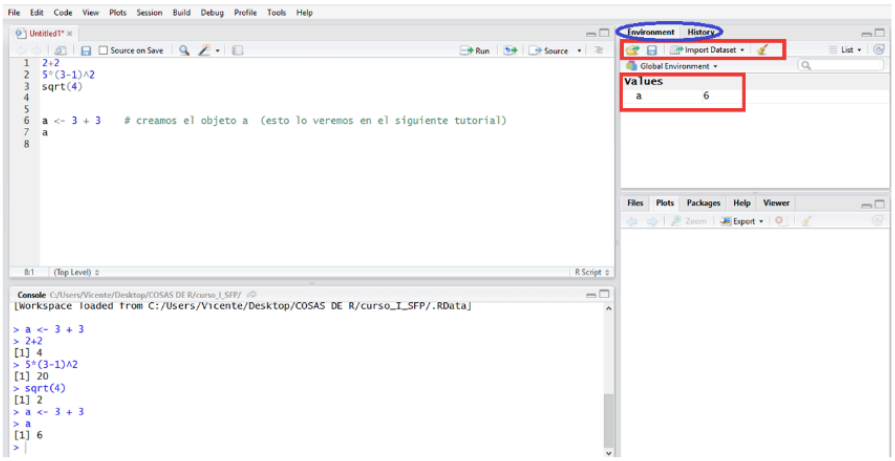


Figure 1.6: Figura 7

- En la pestaña History se registran las instrucciones ejecutadas. Como opciones, podemos cargar y guardar el historial de la sesión, seleccionar una o más instrucciones y enviarlas bien a la consola bien al script, y limpiar el historial.

1.2.4 Otros recursos

Con el nombre de **Otros recursos** nos referimos al panel que se encuentra en la parte inferior-derecha del escritorio de RStudio. ¡No sabía cómo llamarlo!

En este panel cabe destacar las siguientes pestañas, cada una con diferentes opciones:

- Files: es una especie de explotador de ficheros.
- Plots: donde se visualizan los gráficos que creamos. Entre las opciones disponibles se encuentran:
  - Zoom: para agrandar el gráfico y verlo en otra ventana.
  - Export: para exportar/guardar el gráfico. Se puede guardar el gráfico como imagen, pdf o copiarlo al portapapeles.
- Packages: proporciona un listado de los paquetes instalados en R y los que han sido cargado en la sesión. A través de las opciones de esta pestaña podemos instalar nuevos paquetes o actualizar los existentes.
- Help: Para obtener ayuda sobre una determinada función.

1.3 Configuración del espacio de trabajo

Antes de comenzar a trabajar debemos fijar el directorio donde queremos guardar nuestros ficheros. Básicamente, dos alternativas.

1.3.1 Opción 1: Fijar directorio

Opción 1. Indicamos a R la ruta donde queremos trabajar y la fijamos con la función setwd().

```
#setwd("C:/ruta del directorio de trabajo")
```

Para comprobar el directorio de trabajo utilizamos la función getwd():

```
getwd()
```

```
## [1] "/Users/user/Documents/proyectos/capacitacion/capacitacion_R"
```

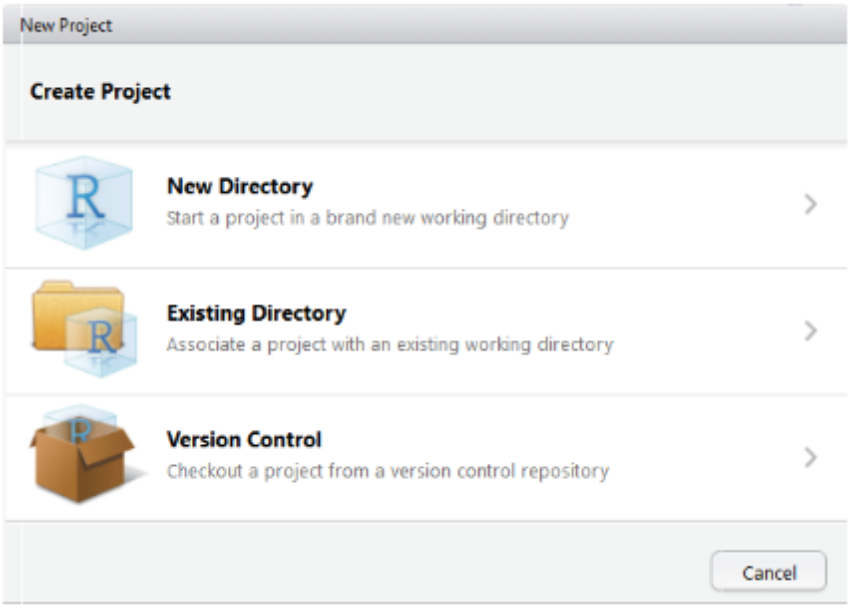
Para obtener un listado de los ficheros que contiene la ruta establecida se usa la función dir().

```
dir()
```

```
##  [1] "_book"              "_bookdown_files"    "_bookdown.yml"
##  [4] "_main_files"        "_main.Rmd"          "_output.yml"
##  [7] "book.bib"           "capacitacion.Rproj" "css"
## [10] "data"               "imagenes"           "index.Rmd"
## [13] "packages.bib"       "preamble.tex"       "README.md"
## [16] "style.css"
```

1.4 Opción 2 : Proyecto de R.

Al crear un proyecto todos los ficheros quedan vinculados directamente al proyecto. Para crear un proyecto selección **File > New project...** Se abrirá la siguiente ventana:



Para crear un proyecto en un nuevo directorio, hacemos clic en el botón New Directory. Seguidamente, seleccionamos el tipo de proyecto, en nuestro caso Empty Project. Ahora, asignamos un nombre al directorio (carpeta) que se va a crear y que al mismo tiempo será el nombre del proyecto de R. Para terminar, hacemos clic en el botón Create Project. Al seguir este proceso se habrá creado una carpeta en Documentos y un fichero nombre_carpeta.Rproj.

Para crear un proyecto en una carpeta que ya existe, hacemos clic en el botón Existing Directory y después seleccionamos la carpeta ayudándonos del Browse.. si fuera necesario. Una vez elegida la carpeta, clicamos en Create Project.

Para abrir un proyecto hacemos doble clic sobre el archivo con extensión .Rproj o lo abrimos desde el menú de RStudio: File > Open Project…

Ventaja de los proyectos: cualquier fichero que creemos (script de R, documento de Rmarkdown, etc.) y guardemos se guardará en la carpeta del proyecto.

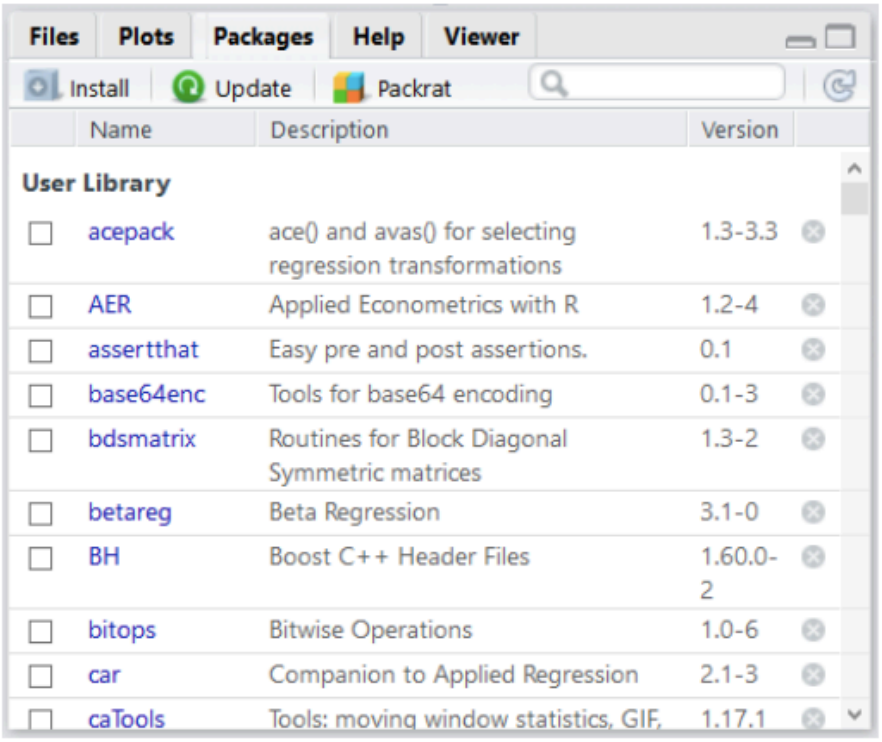
1.5 Instalación de paquetes.

R está compuesto por un sistema base, pero para extender su funcionalidad es necesario instalar paquetes adicionales.

Podemos instalar paquetes de varias formas:

- A través del menú: Tools > Install packages…

- En el escritorio de RStudio: Packages/Install. Vemos los paquetes que tenemos actualmente instalados y aquellos que se encuentran cargados.



- Utilizando la función install.packages(). El nombre del paquete que queremos instalar debe ir entre comillas.

```
#install.packages("dplyr") # dplyr es un paquete que se utiliza para manipular/gestion
```

Una vez instalado el paquete, hay que cargarlo para poderlo utilizar. Esto se hace con la función library().

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

1.6 Ayuda en R.

En muchas ocasiones necesitamos ayuda sobre cómo funciona una determinada función, cuáles son sus argumentos, etc. Hay varias formas de pedir la ayuda de R. Vamos a pedir la ayuda de la función mean().

```
help(mean)
?mean
```

Si ejecutamos directamente la función library() se abrirá una ventana listando los paquetes que tenemos instalados en R. En el escritorio de RStudio, en la pestaña Packages también tenemos en listado de paquetes instalados (organizados en dos bloques: User Library y System Library)

```
library()
```

Para obtener ayuda sobre un determinado paquete...

```
library(help="foreign")
```

Pero sin duda, una de las mejores fuentes de ayuda en R nos la proporciona internet. Bien haciendo directamente en google la búsqueda sobre el tema que estamos interesados, bien acudiendo a algunas de las muchas webs que ofrecen ayuda. Algunas de las más populares y recomendables webs son:

- R-Bloggers
- Stack Overflow

Chapter 2

Manipulación y visualización de datos

2.1 Comandos básicos de R.

2.1.1 Introducción

El objetivo de este tutorial es familiarizarnos con los conceptos básicos de R. ¿Qué es un objeto en R? ¿Con qué clases/tipos de objetos se trabaja en R? Aprenderemos a definir vectores y operar con ellos; a crear matrices, listas y data frames; a seleccionar elementos, añadir filas y columnas, etc. Como lo que se pretende es que se entienda la filosofía y la práctica del trabajo con R, todos los conceptos que se introducen se ilustran con ejemplos muy sencillos. No obstante, la selección de funciones que se realiza en este tutorial tienen una aplicación directa en el tratamiento real de datos.

Vamos a realizar paso a paso este sencillo ejercicio para introducir algunos conceptos importantes.

```
3+4
```

```
## [1] 7
```

```
## [1] 7
log(10)
```

```
## [1] 2.302585
```

```
## [1] 2.302585
x <- 3+4
x  # x es un vector cuya primera componente es 7. Enseguida vamos con los vectores!
```

```
## [1] 7
```

```
## [1] 7
y = 2+6
y
```

```
## [1] 8
```

```
## [1] 8
z <- c(x,y)
z
```

```
## [1] 7 8
```

```
## [1] 7 8
mean(z)
```

```
## [1] 7.5
```

```
## [1] 7.5
w <- mean(z)
w
```

```
## [1] 7.5
```

```
## [1] 7.5
round(w, digits=0)
```

```
## [1] 8
```

```
## [1] 8
```

R utiliza funciones para realizar operaciones. Una función es, por ejemplo, mean(). Para utilizar una función deben especificarse unos argumentos, que es lo que escribimos dentro de los paréntesis. En el caso de la función round() hemos especificado dos argumentos: el vector que queremos redondear (w) y el número de decimales del redondeo (digits).

El símbolo <- es el operador para asignar. También se puede utilizar = (o menos frecuente ->), aunque es preferible utilizar el <-.

El símbolo # se utiliza para introducir un comentario. **Todo lo que quede a la derecha de # no se ejecutará.**

Cuando se realiza una asignación se obtiene un objeto. Podemos ver el resultado o contenido de un objeto de varias formas. Por ejemplo, para ver qué es el objeto x podemos escribir en la consola:

- x
- print(x)
- (x <- 3+4)

2.1.2 Vectores

Básicamente R trabaja con los siguientes tipos de objetos:

- VECTORES
- MATRICES y ARRAYS (variables indexadas)
- LISTAS
- FACTORES
- DATA FRAMES
- FUNCIONES

Empezaremos viendo los objetos más sencillos, los vectores. Poco a poco iremos viendo el resto de objetos.

La mayoría de las operaciones (+, -, *, /) y funciones en R están definidas con carácter vectorial. ¿Qué significa esto? Que R opera componente a componente.

Antes de entender el concepto "caracter vectorial", vamos a ver cómo se define/crea un vector.

Para crear un vector se utiliza la función c() (c de concatenate). Por ejemplo:

```
x <- c(1,2,3,4)
x                  # x es un vector que tiene cuatro componentes
```

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

```
y <- c(5,6,7,8)
y
```

```
## [1] 5 6 7 8
```

Volvemos sobre el tema del carácter vectorial, es decir, se opera componente a componente. Pensemos, si

```
z <- x + y
```

¿Qué resultado espero obtener para z?

Exacto!!! Como la operación se realiza vectorialmente (componente a componente, muy importante!) el resultado es:

```
z
```

```
## [1]  6  8 10 12
```

Vamos a ver si lo entendemos de verdad. Supongamos que x e y son los siguientes vectores:

```
x <- c(1,2,3,4)
y <- c(1,2,3)
```

¿Qué longitud tienen los vectores x e y? Aquí la respuesta está clara, pero en aplicaciones reales utilizaríamos la función length().

```
length(x)                # esta función es muy útil, conviene recordarla.
```

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

```
## [1] 4
length(y)
```

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

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

Los vectores no tienen la misma longitud, entonces.. ¿Cuál será el resultado de z <- x + y?

```
z <- x+y
```

```
## Warning in x + y: longer object length is not a multiple of shorter object
## length
```

```
z
```

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

R nos da un mensaje de aviso (warning), no es lo mismo que un error. Nos avisa que hay algo que no cuadra pero…realiza la operación que nosotros queremos.

Una cuestión muy importante que siempre tenemos que tener en cuenta cuando trabajamos con vectores es que en un vector sólo podemos concatenar elementos del mismo tipo. ¿Qué tipos/clases de elementos (o datos) tenemos en R?

- Carácter
- Numéricos
- Enteros
- Complejos
- Lógicos

Veamos algunos ejemplos…

```
x <- c(1,2,3,4)     # creamos el vector x
class(x)            # devuelve el tipo de objeto
```

```
## [1] "numeric"
```

```
y <- c("a","b")
class(y)
```

```
## [1] "character"
```

```
z <- c(1L,2L,3L)    # escribimos L detrás del número para obligar a que sea entero
class(z)
```

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

```
w <- c(TRUE, F)    # en general, puede escribirse TRUE/FALSE o T/F
class(w)
```

```
## [1] "logical"
```

```
t <- c(1+2i, 1+3i)
class(t)
```

```
## [1] "complex"
```

En los ejemplos anteriores hemos definido un vector en el que todos sus elementos eran del mismo tipo. Pero....¿qué pasa si tenemos los siguientes vectores?

```
x <- c(1,2,"a")
y <- c(FALSE, 1)
z <- c("a",T)
```

¿De qué tipo son ahora los vectores x, y, z?

```
class(x)
```

```
## [1] "character"
```

```
## [1] "character"
class(y)
```

```
## [1] "numeric"
```

```
## [1] "numeric"
class(z)
```

```
## [1] "character"
```

```
## [1] "character"
```

R ha forzado a que todos los elementos del vector sean del mismo tipo. A esto se le llama implicit coercion. Fijémonos cúal es el resultado de los vectores que hemos definido antes.

```
x
```

```
## [1] "1" "2" "a"
```

```
y
```

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

```
z
```

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

En ocasiones somos nosotros los que estamos interesados en forzar que todos los elementos del vector sean del mismo tipo (esto es la explicit coercion). Para ello utilizamos las funciones as.numeric() , as.character(), as.logical() … Si el resultado no tiene sentido R producirá un mensaje de error o warning. Un ejemplo:

```
x <- c(1,2,"a")
x
```

```
## [1] "1" "2" "a"
```

```
## [1] "1" "2" "a"
as.numeric(x)
```

```
## Warning: NAs introduced by coercion
```

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

```
## [1]  1  2 NA
as.character(x)
```

```
## [1] "1" "2" "a"
```

```
## [1] "1" "2" "a"
```

Por último, podemos evaluar el tipo/clase de objeto con las funciones is.numeric(), is.character(), etc.

2.1.2.1 Acceder a un elemento de un objeto

Para seleccionar/acceder a un elemento de un objeto se suelen emplear: **[]**, **$**, **[[]]**.

Vamos a crear el objeto x que será un vector de cuatro componentes formado por los cuatro primeros números pares. Así:

```
x <- c(2,4,6,8)
```

Si queremos acceder/seleccionar/extraer al/el segundo componente de x

```
x[2]
```

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

2.1.2.2 Enlistar y borrar objetos

Las funciones ls() y objects() hacen lo mismo: listan los objetos que hemos definido en la sesión.

```
ls()
```

```
## [1] "t" "w" "x" "y" "z"
```

```
objects()
```

```
## [1] "t" "w" "x" "y" "z"
```

Si queremos borrar objetos utilizamos la función rm()

```
rm(z)  # borramos el objeto z
ls()
```

```
## [1] "t" "w" "x" "y"
```

y si estamos interesados en borrar todos los objetos que hemos definido:

```
rm(list=ls()) # o también rm(list=objects())
ls()
```

```
## character(0)
```

2.1.3 Tipos de objetos

2.1.3.1 Matrices

La función matrix() permite organizar los datos en una matriz con tantas filas y columnas como se indiquen.

Vamos a pedir ayuda a R sobre la función matrix()

```
?matrix # una forma de obtener ayuda en R es escribiendo ? delante de la función de la que solici
```

2.1.3.1.1 Cómo crear y trabajar con matrices Una vez hemos leído en la ayuda cómo usar la función matrix, vamos a practicarlo con el siguiente ejemplo. Pero antes de ejecutarlo, pensemos en qué es lo que queremos hacer y, sobre todo, pensemos en cuál es el resultado que esperamos obtener.

```
x <- matrix (data= c(1,2,3,4), nrow=2, ncol=2) # o x <- matrix (c(1,2,3,4), nrow=2, ncol=2)
x
```

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

Observemos que se ha creado una matrix de 2x2 (2 filas y 2 columnas) y, además, muy importante, ¿cómo se ha completado la matriz? ¡Exacto! La matriz se ha rellenado por columnas. Si queremos que se rellene por filas hay que incluir el argumento byrow en los argumentos de la función.

```
y <- matrix (c(1,2,3,4), nrow=2, ncol=2, byrow=T)
y
```

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

Fíjense en la diferencia entre los objetos x e y.

Una forma más simple para definir una matrix es:

```
y <- matrix (c(1,2,3,4), 2, 2, byrow=T)
y
```

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

aunque también podíamos haber omitido el argumento relativo al número de filas o de columnas, porque conocida una dimensión R completaría la matrix dados los datos con los que se trabaja.

```
y <- matrix (c(1,2,3,4), 2, byrow=T)  # no especificamos nrow porque por defecto es el
y
```

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

```
x <- matrix(c(1,2,3,4,5,6), ncol=3)
x
```

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

2.1.3.2 Seleccionando elementos de una matriz

Lo primero, vamos a crear el objeto A que será una matriz.

```
A <- matrix(1:16,4,4)
A
```

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

Para seleccionar elementos de una matriz utilizamos el símbolo de los corchetes: [].

Pensemos un momento en el posible resultado de estos ejemplos antes de efectuarlos.

```
A[2,3]
```

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

```
A[c(1,2),c(2,4)]
```

```
##      [,1] [,2]
## [1,]    5   13
## [2,]    6   14
```

```
A[1:3,2:4]
```

```
##      [,1] [,2] [,3]
## [1,]    5    9   13
## [2,]    6   10   14
## [3,]    7   11   15
```

```
A[1,]
```

```
## [1]  1  5  9 13
```

```
A[1:2,]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
```

```
A[,2:3]
```

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

2.1.4 Listas

A diferencia de los vectores o matrices, las listas pueden contener elementos/componentes de distinto tipo. Observemos esta lista que tiene 5 componentes (pueden ser matrices, vectores, dataframes,..).

```
x <- list(c(1,2,3,4), "Curso", F, 1+2i, 3L,A)
x
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] "Curso"
##
## [[3]]
## [1] FALSE
##
## [[4]]
## [1] 1+2i
##
## [[5]]
## [1] 3
##
## [[6]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

Utilizamos el doble corchete [[]] para acceder al contenido concreto de una lista.

```
x[[3]]  # accedemos al tercer componente de la lista
```

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

```
x[[1]][2] # accedemos al segundo elemento del primer componente de la lista
```

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

```
x[[6]][1,2]
```

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

Vamos a crear otra lista para practicar.

```
y <- list( Titulacion = c("Economía", "Sociología", "Derecho"), Edad =c(25,26,27))

y
```

```
## $Titulacion
## [1] "Economía"   "Sociología" "Derecho"
##
## $Edad
## [1] 25 26 27
```

Fijémonos en la diferencia de presentación de las listas x e y. Como en la lista y hemos nombrado los componentes, estos aparecen al ejecutar el objeto precedidos del símbolo $. Ahora también podemos acceder a un componente de la lista por su nombre.

```
y$Titulacion
```

```
## [1] "Economía"   "Sociología" "Derecho"
```

```
y[[1]]
```

```
## [1] "Economía"   "Sociología" "Derecho"
```

Evidentemente, también podemos realizar operaciones con listas.

```
y[[2]]*3
```

```
## [1] 75 78 81
```

2.1.5 Data Frame

Los data frame se usan para almacenar datos en forma de tablas (filas / columnas), como estamos habituados en Excel, Spss, etc.

Los data frame pueden almacenar objetos/datos de distinto tipo: numéricos, carácter, … En las matrices todos los elementos tenían que ser enteros o numéricos.

Los data frame pueden entenderse como un tipo especial de lista donde cada elemento de la lista tiene que tener la misma longitud. Cada elemento de la lista sería una columna y la longitud de cada elemento de la lista serían las filas.

Aunque normalmente los data frame los creamos al cargar/leer una base de datos (ver el tutorial), vamos crear una data frame para ver su estructura.

```
x <- data.frame(Titulacion = c("Economía", "ADE", "Sociología", "Magisterio"), Edad = c
x
```

```
##   Titulacion Edad
## 1   Economía   25
## 2        ADE   27
## 3 Sociología   25
## 4 Magisterio   24
```

2.1.6 Importar y exportar datos

2.2 Manipulación y limpieza de datos.