



# Fundamentos de R



- ☰ I. Introducción y objetivos
- ☰ II. El lenguaje de programación en R
- ☰ III. Entorno de programación RStudio
- ☰ IV. Sintaxis básica de R
- ☰ V. Estructura de datos
- ☰ VI. Herramientas de control de flujo
- ☰ VII. Funciones
- ☰ VIII. Resumen
- ☰ IX. Caso práctico con solución

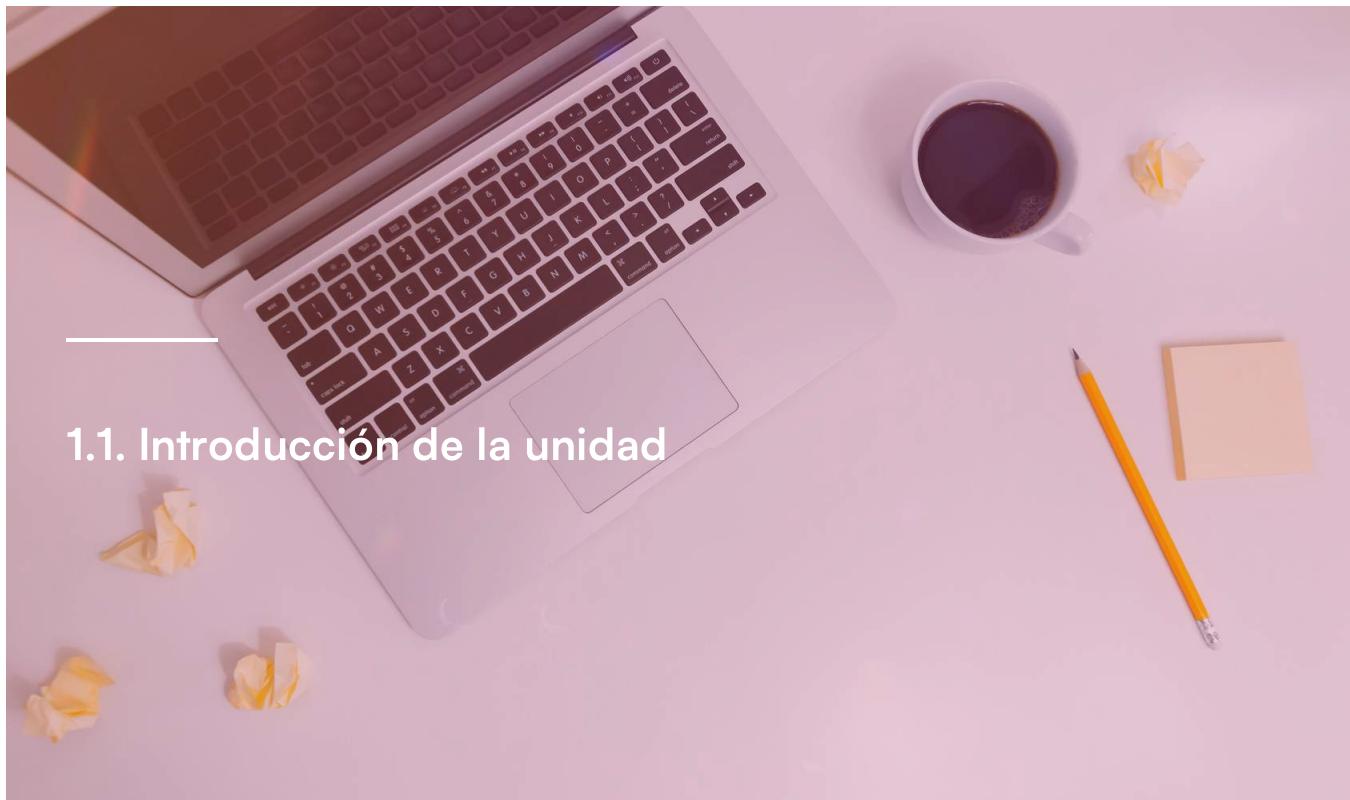
 X. Enlaces de interés

 XI. Glosario

 XII. Bibliografía

# I. Introducción y objetivos

---



## 1.1. Introducción de la unidad

Dentro de las **herramientas que puede encontrar un científico de datos para realizar proyectos o simplemente, investigar sobre toda la información que podemos extraer de un conjunto de datos, tarde o temprano se encontrará con R**.

---

**En la actualidad, además de mantener su rivalidad con Python dentro del mundo de la ciencia de datos, es considerado uno de los mejores lenguajes de programación en lo que a modelado y aprendizaje estadístico se refiere.**

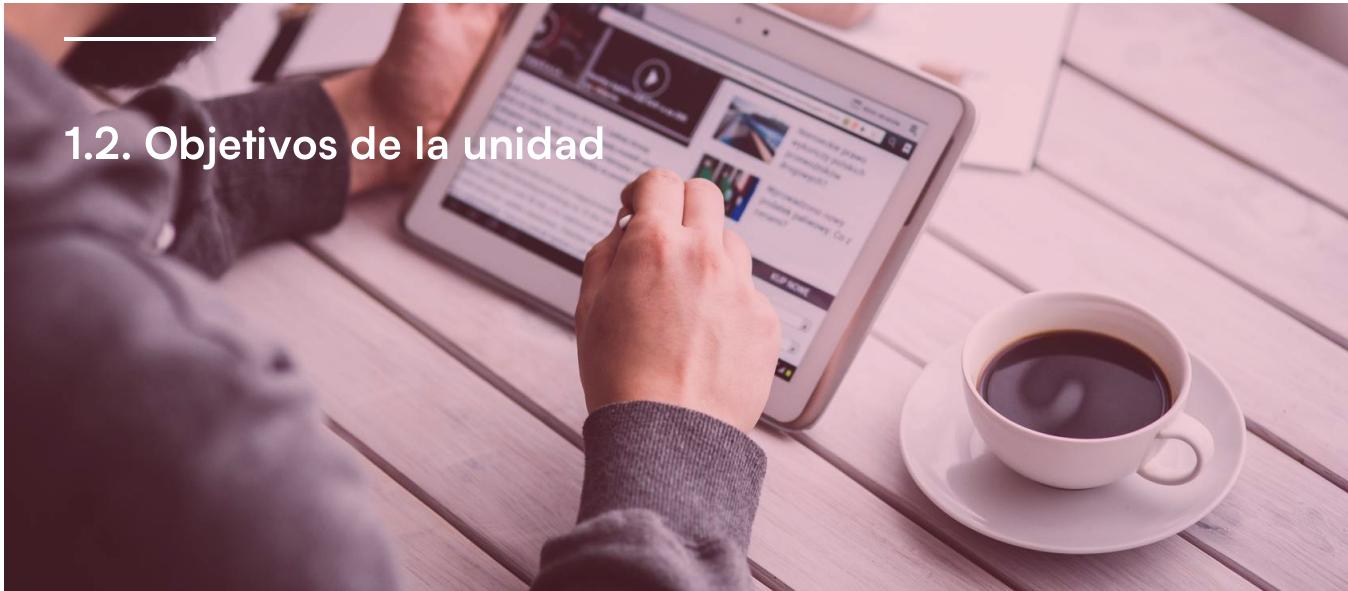
Aprovecharemos la infinidad de **posibilidades que ofrece R**.

Especialmente, podremos usar su enorme cantidad de paquetes para realizar análisis exploratorio sobre *datasets*.

- i** Se introducirá al alumno sobre este lenguaje, primero mediante una breve visión sobre qué es R, instalación de R y RStudio, la creación de *scripts* y *notebooks*, y después el aprendizaje de los fundamentos de R, la utilización e instalación de paquetes para realizar transformación y limpieza de datos y, finalmente, se trabajará con las librerías más populares que existen en R para la visualización de datos, como son los gráficos base, Ggplot2 y Plotly.

**CONTINUAR**





## 1.2. Objetivos de la unidad

Tras finalizar esta unidad, el alumno conocerá los siguientes conceptos y habrá adquirido las competencias mencionadas a continuación:

- 1 Conocer qué es el lenguaje de programación R y habituarse al IDE de programación RStudio.
- 2 Aprender sobre la sintaxis básica de R, desde la creación de *scripts* y *notebooks* en donde el alumno podrá realizar anotaciones en Markdown, realizar comentarios, conocer los principales operadores e implementar variables y conocer los diferentes tipos que existen en R.
- 3 Distinguir y saber utilizar las siguientes estructuras de datos en R: vectores, *arrays*, factores, listas, matrices y *dataframes*.
- 4 Saber controlar el flujo de un programa a través de sentencias condicionales y bucles.
- 5 Desarrollar funciones propias y diferenciar parámetros de entrada y salida, así como utilizar funciones propias que integra R internamente.

## II. El lenguaje de programación en R

---

R<sup>1</sup> es un **lenguaje de programación** interpretado de distribución de código abierto (con licencia GNU). Su **uso principal** está **centrado en la estadística**. Esta es una de las mayores razones por las que en los últimos se ha intensificado su uso en el ámbito de la ciencia de datos.

---

<sup>1</sup> Proyecto R. [En línea] URL disponible en este [enlace](#).



**Figura 1.** Logo de R.

Fuente: [CRAN](#).

---

**Los comienzos de R se remontan a 1992.** Fue creado por Ross Ihaka y Robert Gentleman. En sus inicios, se llamó S. En la actualidad, sus **principales características** son:

- Lenguaje interpretado.

- Operadores para cálculo sobre variables indexadas (*arrays*, *listas*, *matrices*, *vectores*, *dataframes*, etc.).
- Mantenido por la comunidad de R.
- Sintaxis básica.

## Repositorio de paquetes

Uno de sus **puntos fuertes** se centra en el repositorio de paquetes, que nos brinda la posibilidad de **encontrar cualquier función o solución que deseemos desarrollar ya realizada en un paquete**. Tanto los paquetes como manuales, documentación o preguntas frecuentes, noticias y, por supuesto, la descarga de R, se encuentran en el CRAN<sup>2</sup> (*The comprehensive R archive network*). El repositorio de paquetes disponible en CRAN es tan grande que, por ejemplo, si tomamos como fecha abril de 2020, nos encontramos con más de 15 mil paquetes disponibles para su descarga y uso.

2 R CRAN. [En línea] URL disponible en [este enlace](#).

## Contributed Packages

### Available Packages

Currently, the CRAN package repository features 15514 available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

**Figura 2.** Contribución de paquetes en CRAN.

Fuente: [CRAN](#).

## Available CRAN Packages By Date of Publication

Date	Package	Title
2020-04-04	<a href="#">ALA4R</a>	Atlas of Living Australia (ALA) Data and Resources in R
2020-04-04	<a href="#">arules</a>	Mining Association Rules and Frequent Itemsets
2020-04-04	<a href="#">auRoc</a>	Various Methods to Estimate the AUC
2020-04-04	<a href="#">fs</a>	Cross-Platform File System Operations Based on 'libuv'
2020-04-04	<a href="#">mritc</a>	MRI Tissue Classification
2020-04-04	<a href="#">RCT</a>	Assign Treatments, Power Calculations, Balances, Impact Evaluation of Experiments

**Figura 3.** Vista de algunos paquetes en CRAN.

Fuente: [CRAN](#).

### Saber más

Además de la bibliografía de esta unidad, si se desea profundizar en el desarrollo e investigación a través de R o, simplemente, se desea tener un mayor acceso a manuales oficiales de la comunidad de R, dentro del proyecto R existe una sección dedicada a documentación y manuales. [En línea] URL disponible en [este enlace](#).

## III. Entorno de programación RStudio

---

### 3.1. Instalación de R



CONTINUAR

Antes de tener disponible RStudio y trabajar en este IDE, hay que **descargar R base**, que incluye el intérprete de R.

## Paso 1

### The Comprehensive R Archive Network

#### Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

El primer paso es ir a CRAN y, en la pantalla principal, aparecerán links para descargar R tanto para Windows, Mac OS X y Linux. En esta ocasión, la instalación se realizará desde Windows.

**Figura 4.** CRAN. Links para descarga de R.

Fuente: [CRAN](#).

## Paso 2

### R for Windows

Subdirectories:

- base Binaries for base distribution. This is what you want to [install R for the first time](#).
- contrib Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on [third party software](#) available for CRAN Windows services and corresponding environment and make variables.
- old contrib Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
- Rtools Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Pulsamos sobre el link que corresponda a nuestro sistema operativo y aparecerá la siguiente pantalla:

**Figura 5.** CRAN. Selector de versión de R base.

Fuente: [CRAN](#).

### Paso 3

#### R-3.6.3 for Windows (32/64 bit)

[Download R 3.6.3 for Windows](#) (83 megabytes, 32/64 bit)

[Installation and other instructions](#)

[New features in this version](#)

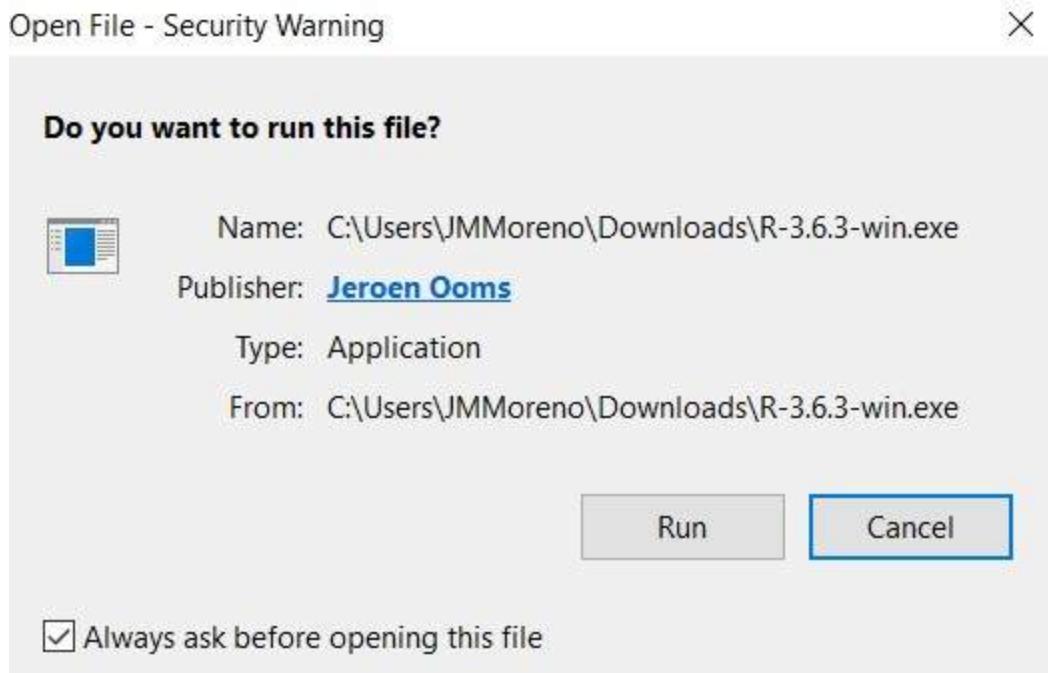
If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [fingerprint](#) on the master server. You will need a version of md5sum for windows: both [graphical](#) and [command line versions](#) are available.

Pulsamos sobre el link “Install R for the first time”. Nos llevará a la siguiente pantalla:

**Figura 6.** CRAN. Descarga de R base.

Fuente: [CRAN](#).

#### Paso 4



Pulsamos sobre el link “Download R X.X.X for S.O”. Aunque en la imagen aparece la versión 3.6.3, el alumno descargará la más reciente. Al pulsar sobre el enlace, comenzará automáticamente la descarga del instalador de R base.

Al lanzar el instalador, aparecerá la siguiente pantalla:

**Figura 7.** Permisos de ejecución R.

*Fuente:* elaboración propia.

## Paso 5

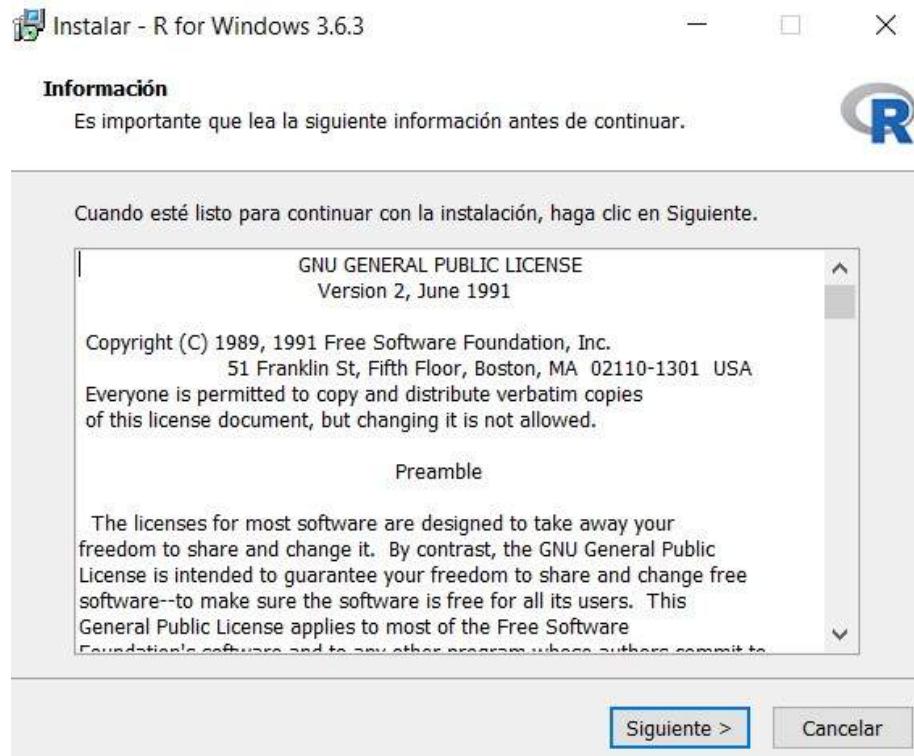


Al asignar permisos de ejecución, se lanzará el asistente de instalación, en primer lugar aparecerá el selector de idioma.

**Figura 8.** Selección de idioma.

*Fuente:* elaboración propia.

## Paso 6

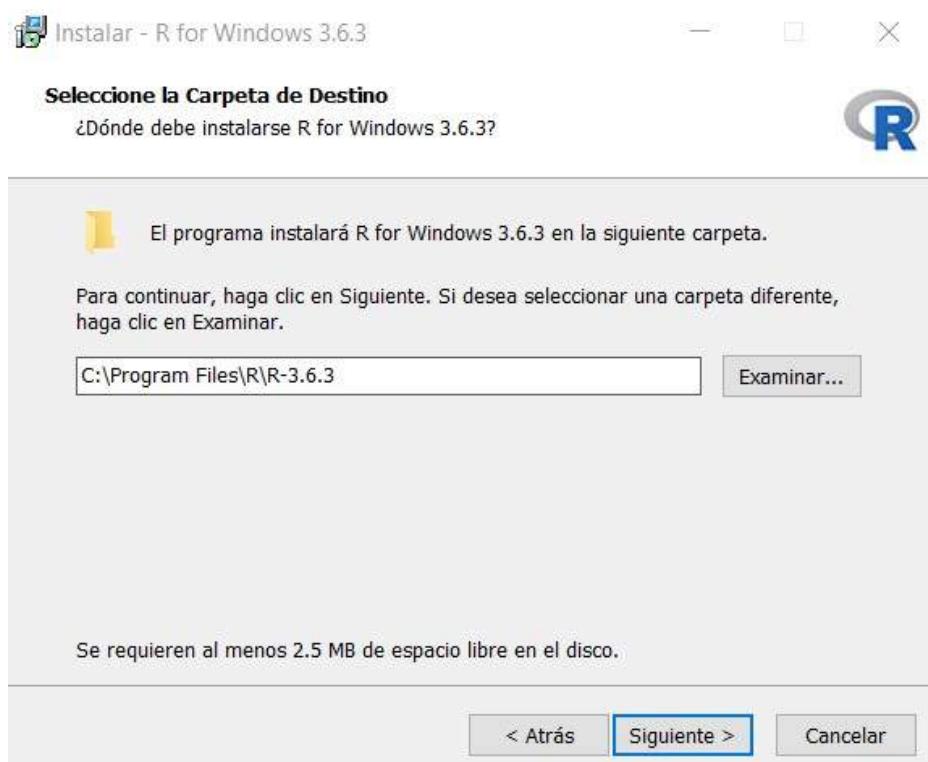


Posteriormente, aparecerá la licencia GNU y pulsamos en “Siguiente”.

**Figura 9.** Licencia GNU.

Fuente: elaboración propia.

## Paso 7

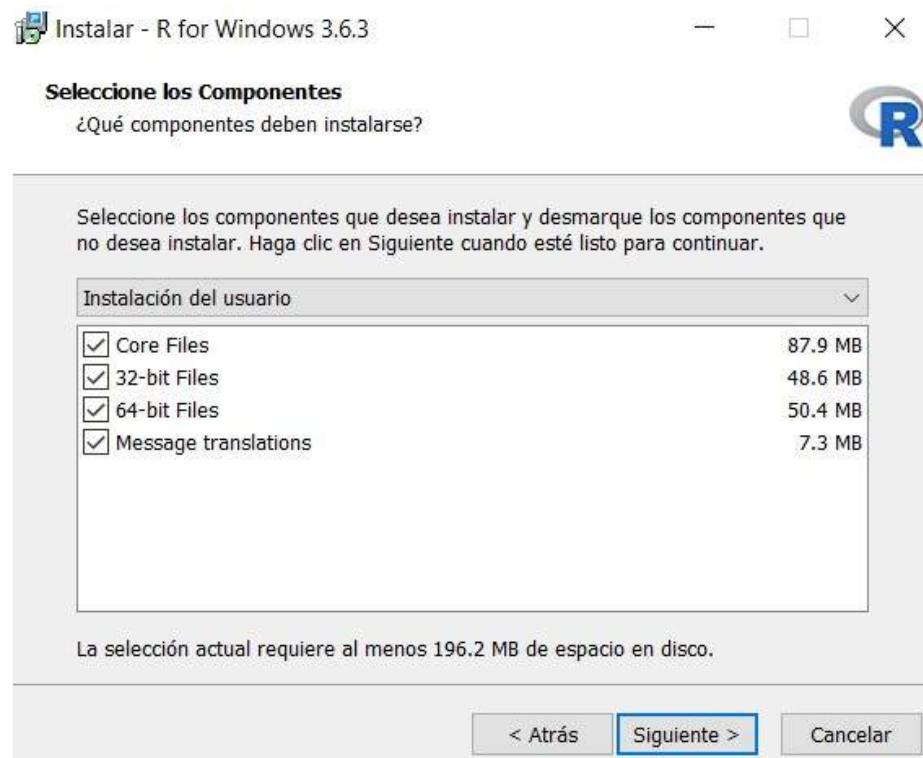


El siguiente paso será seleccionar la ubicación de instalación de R base. Dejamos la que aparezca por defecto.

**Figura 10.** Ruta de instalación.

*Fuente:* elaboración propia.

## Paso 8

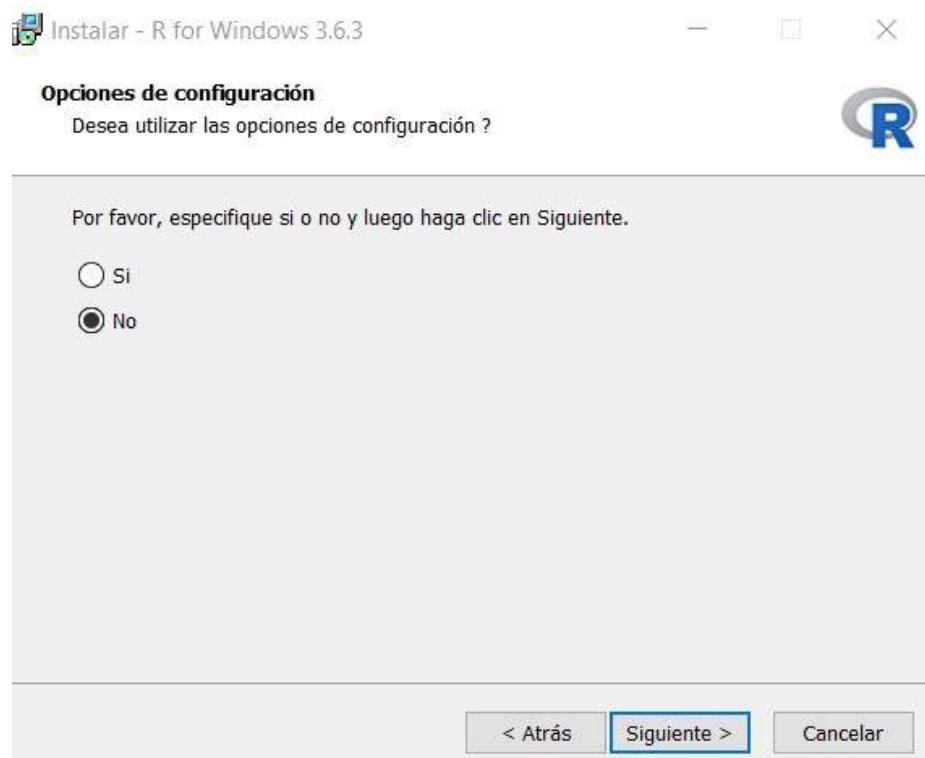


Seguidamente, aparecerá la instalación y selección de componentes. Podemos dejar los que aparecen por defecto y pulsamos "Siguiente".

**Figura 11.** Selección de componentes.

Fuente: elaboración propia.

## Paso 9

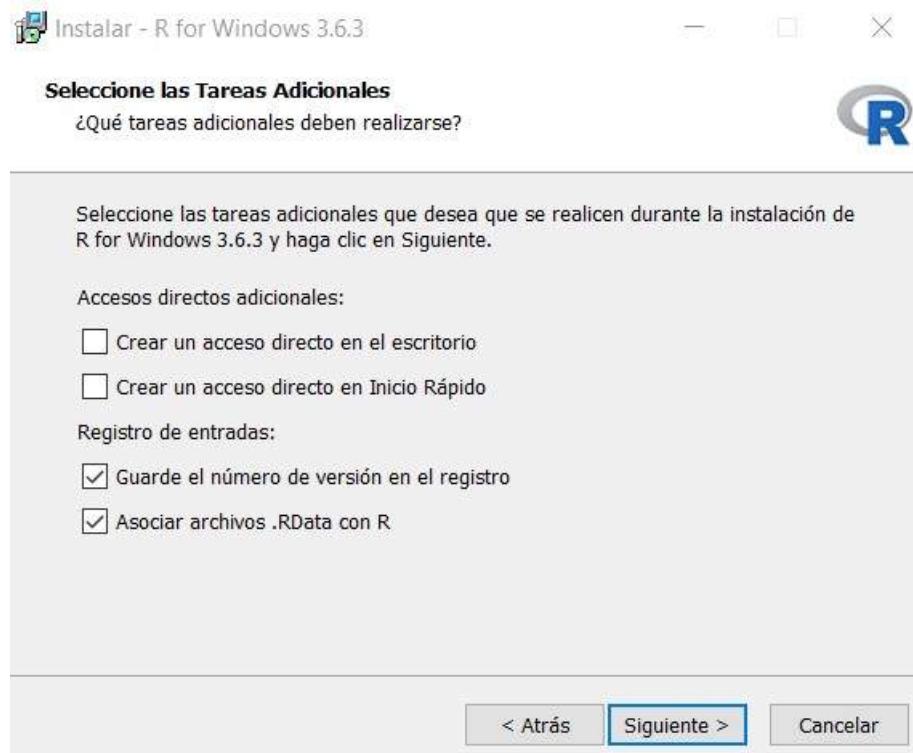


En las opciones de configuración, dejamos la opción que viene por defecto (“No”) y pulsamos “Siguiente”.

**Figura 12.** Opciones de configuración.

*Fuente:* elaboración propia.

## Paso 10

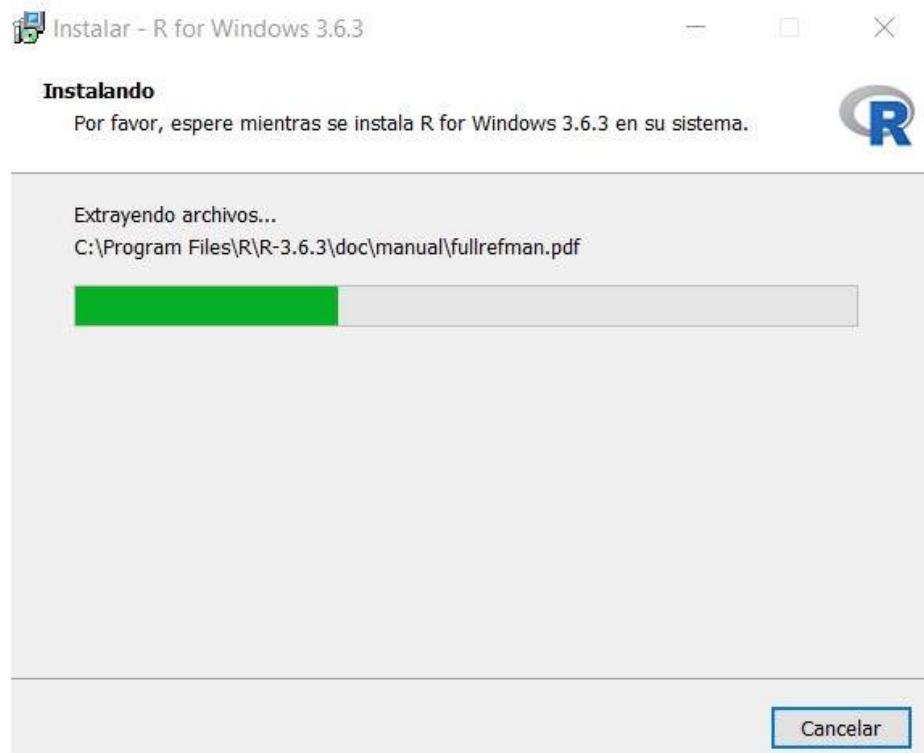


Seleccionamos si queremos crear accesos directos en el escritorio. Es mejor que no marquemos estas opciones, ya que no vamos a utilizar R base, sino RStudio. Simplemente dejamos marcadas las opciones de “Registro de entradas” y pulsamos “Siguiente”.

**Figura 13.** Tareas adicionales.

Fuente: elaboración propia.

## Paso 11



La instalación de R comenzará automáticamente.

**Figura 14.** Proceso de instalación de R base.

*Fuente:* elaboración propia.

## Paso 12



Una vez finalizado el proceso de instalación, pulsamos en “Finalizar”.

**Figura 15.** Instalación de R completa.

*Fuente:* elaboración propia.

Ya tendríamos listo el intérprete de R para Windows.

**CONTINUAR**

## 3.2. Instalación de RStudio

Tras la finalización de R base, instalaremos RStudio<sup>3</sup> para disponer de un entorno de programación, más visual y mucho más fácil de utilizar. Para su instalación, acudiremos a la web de <https://rstudio.com/> y pulsamos en “Download”.

---

<sup>3</sup> Web de RStudio. [En línea] URL disponible en: <https://rstudio.com/>



**Figura 16.** Web de RStudio.

Fuente: <https://rstudio.com/>.

---

**¿Cómo instalar RStudio?**

## Paso 1

RStudio Desktop	RStudio Desktop	RStudio Server	RStudio Server Pro
Open Source License	Commercial License	Open Source License	Commercial License
<b>Free</b>	<b>\$995 /year</b>	<b>Free</b>	<b>\$4,975 /year</b>
			(5 Named Users)
<b>DOWNLOAD</b>	<b>BUY</b>	<b>DOWNLOAD</b>	<b>BUY</b>
<a href="#">Learn more</a>	<a href="#">Learn more</a>	<a href="#">Learn more</a>	<a href="#">Evaluation</a>   <a href="#">Learn more</a>

Escogemos la versión gratuita (*free*) y pulsamos en “Download”.

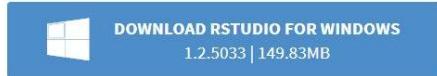
**Figura 17.** Selección de RStudio Desktop Free.

Fuente: <https://rstudio.com/products/rstudio/download/>

## Paso 2

### RStudio Desktop 1.2.5033 - [Release Notes](#)

- 1.** Install R. RStudio requires R 3.0.1+.
- 2.** Download RStudio Desktop. Recommended for your system:



Requires Windows 10/8/7 (64-bit)

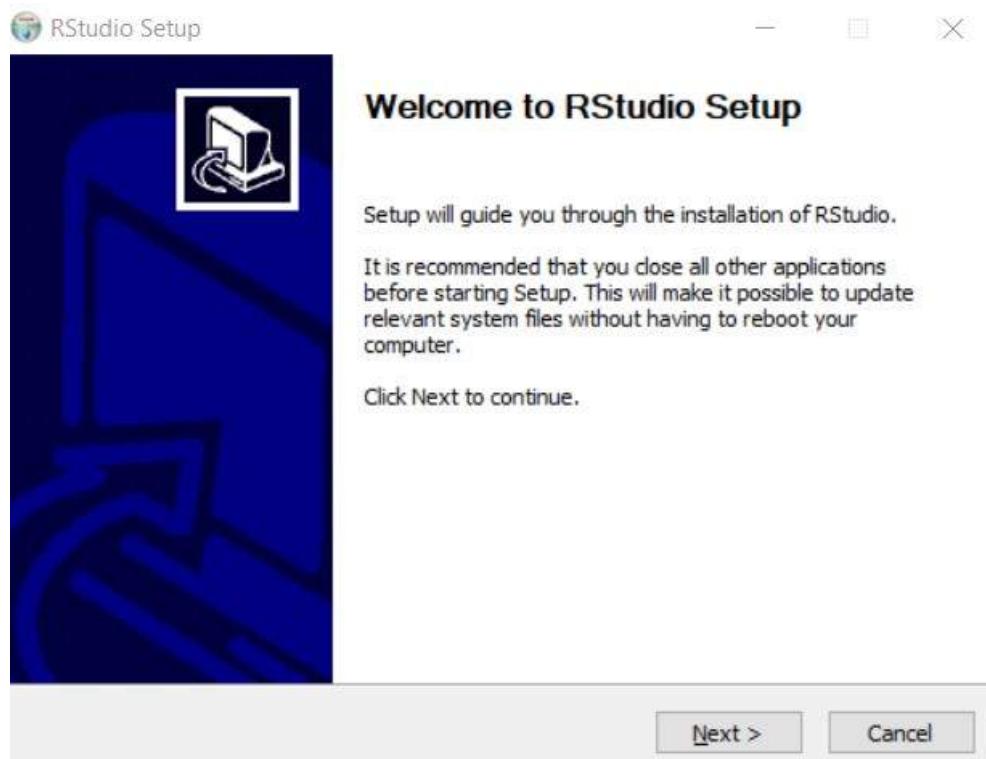


Automáticamente, detectará nuestro sistema operativo, en este caso, Windows. Pulsamos en “Download Rstudio For Windows 1.2.5033”. El alumno descargará la mejor versión. Automáticamente, iniciará la descarga de RStudio.

**Figura 18.** Descarga de RStudio.

Fuente: <https://rstudio.com/products/rstudio/download/#download>

### Paso 3

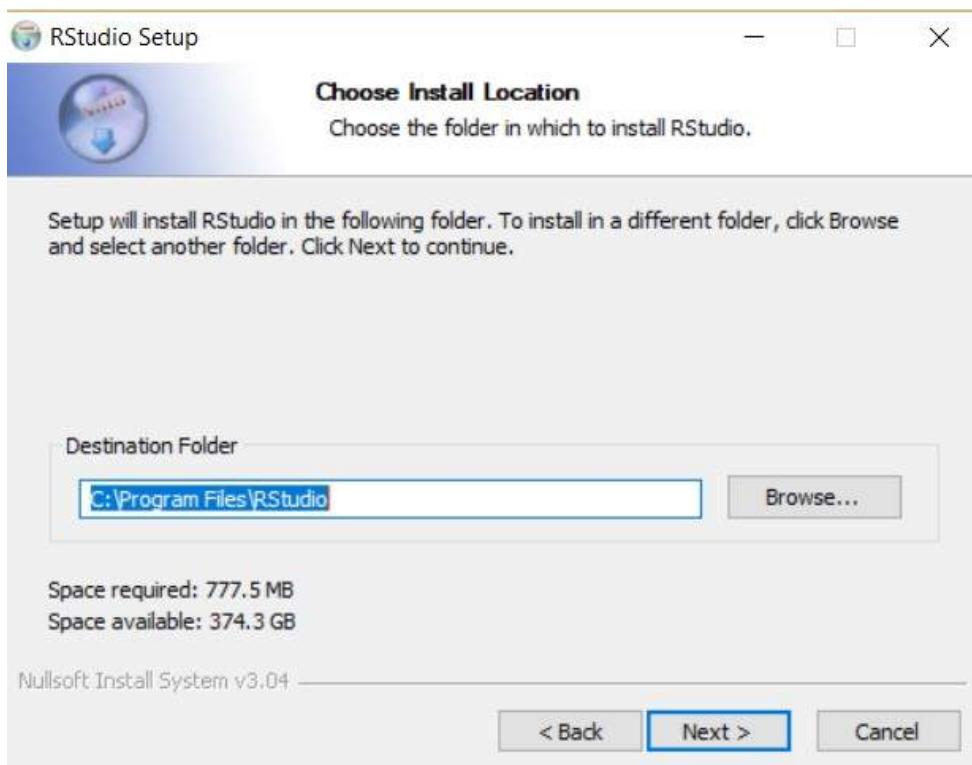


Al iniciar el asistente de instalación, pulsamos en “Next”.

**Figura 19.** Instalación de RStudio (I).

*Fuente:* elaboración propia.

## Paso 4

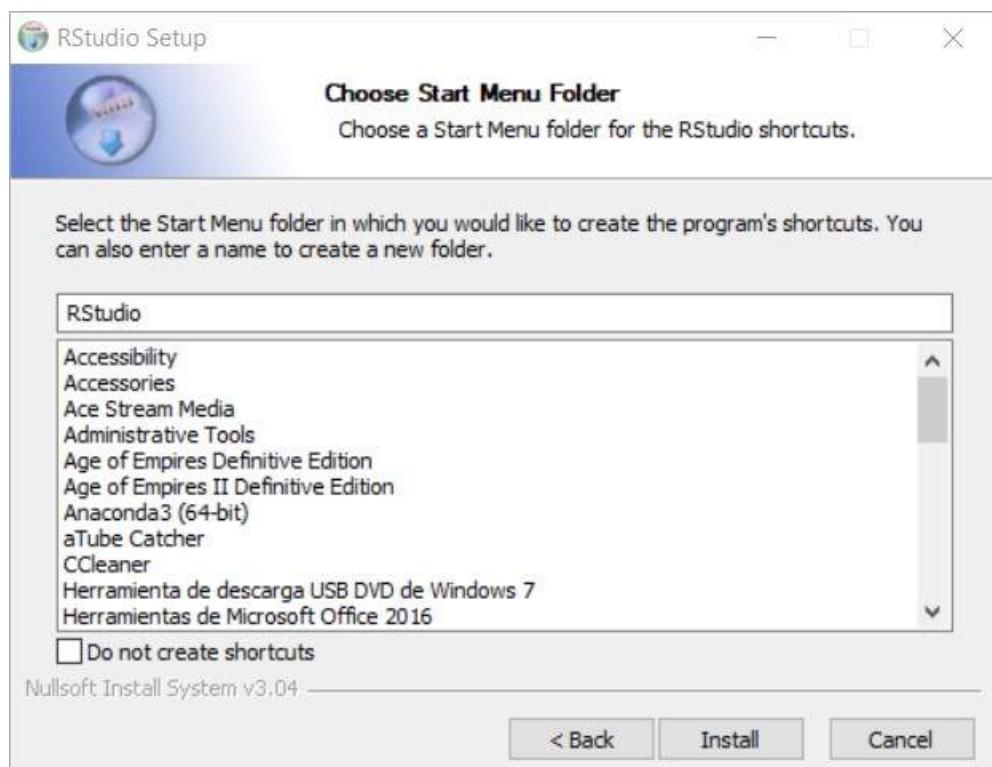


Escogemos la ruta de instalación y pulsamos “Next”.

**Figura 20.** Instalación de RStudio (II).

*Fuente:* elaboración propia.

## Paso 5

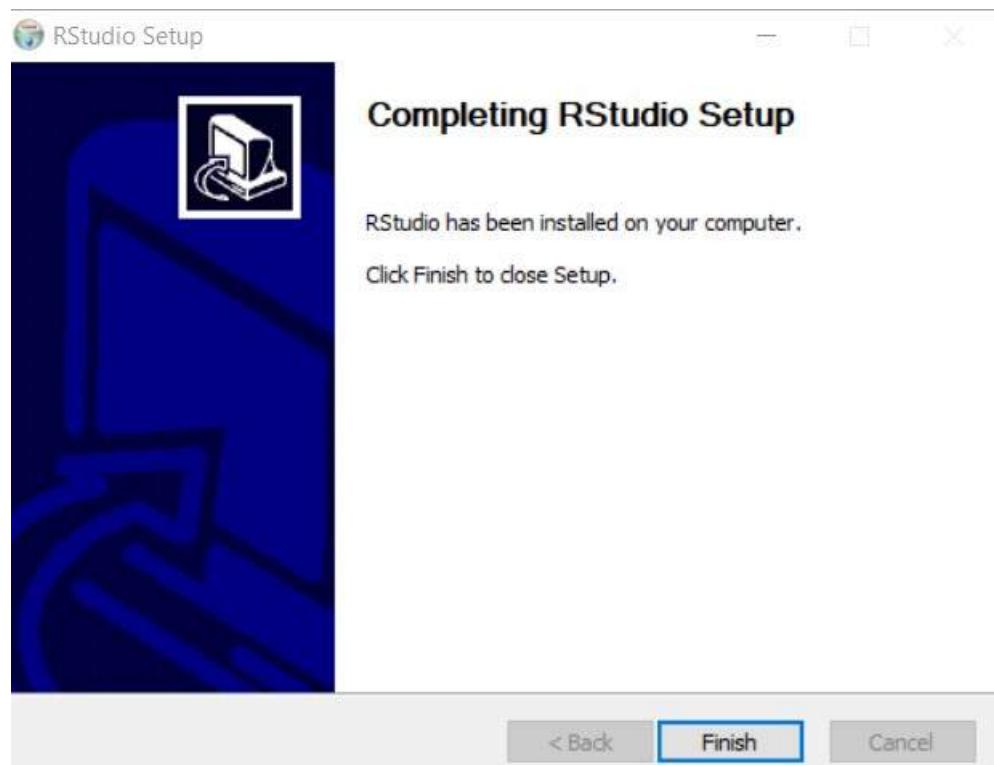


En la ventana de selección del menú de inicio, lo dejamos por defecto y pulsamos “Install”.

**Figura 21.** Instalación de RStudio (III).

*Fuente:* elaboración propia.

## Paso 6



El proceso de instalación comenzará automáticamente. Una vez finalizada la instalación, pulsamos en “Finish”.

**Figura 22.** Instalación de RStudio (IV).

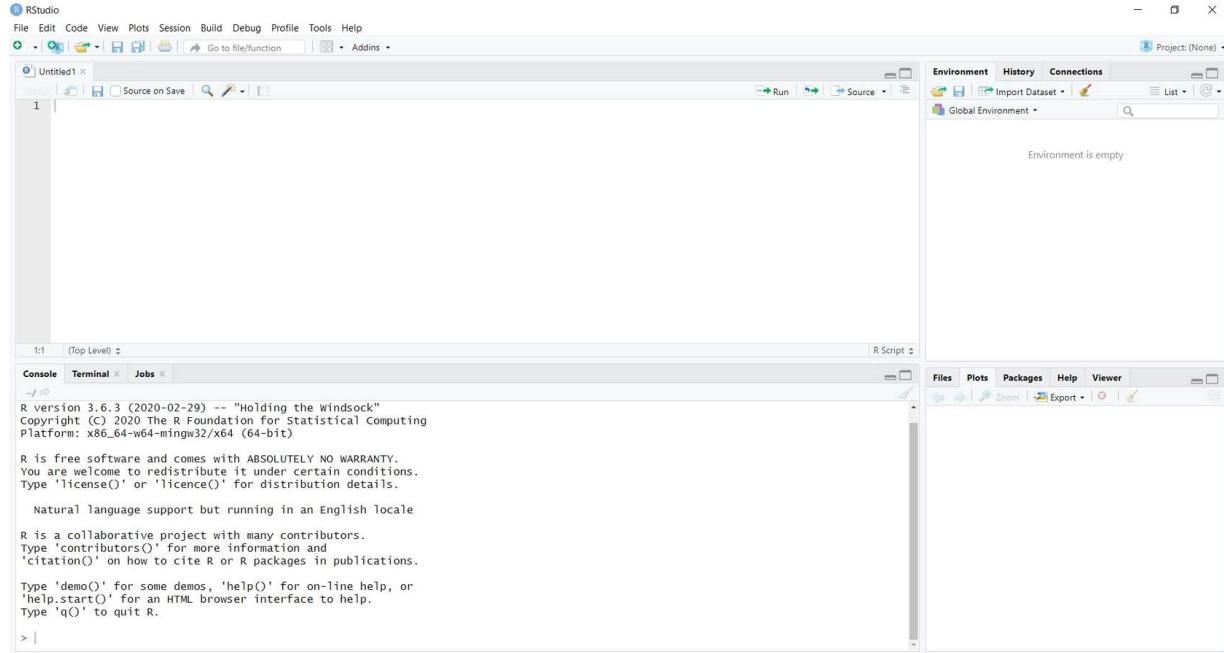
*Fuente:* elaboración propia.

Tras la finalización del proceso, ya podremos **iniciar la aplicación de RStudio**.

**CONTINUAR**

### **3.3. El entorno de programación RStudio**

Una vez instalados R base y RStudio, lanzamos RStudio. Aparecerá la siguiente ventana:



**Figura 23.** Vista de RStudio.

Fuente: elaboración propia.

Dentro del IDE de Rstudio, se distinguen cuatro cuadrantes principales:

### Zona de scripting

Donde se escribe el código fuente. Principalmente, se utiliza en modo de *script* o a través de *notebooks*.

### Zona de entorno e historial de comandos

Cuando se declaren variables o se carguen datos aparecerán en esta sección; también cualquier comando ejecutado se irá registrando en la sesión de R y se podrá acceder a cada comando a través del historial.

### Zona de terminal de consola de comandos

Se puede ejecutar código en este espacio y de cada comando que se ejecute en la zona de *scripting* aparecerá aquí incluido el resultado.

### Zona de archivos, paquetes, ayuda y gráficos

Cuando se acceda a la ayuda de una función, podrá visualizarse en este cuadrante. Se podrán ver los archivos que existan en el directorio a través del cuál se iniciará la sesión y, finalmente, cuando se visualice una gráfica, aparecerá en esta sección.

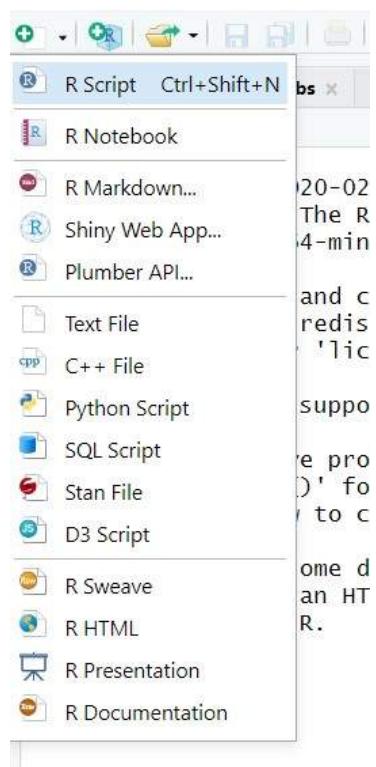
**CONTINUAR**

## 3.4. Primeros pasos a la hora de realizar *scripts* y *notebooks*

Para comenzar a trabajar con RStudio, lo primero será realizar pequeño *script* antes de comenzar a utilizar *notebooks*.

**¿Cómo crear un script?**

## Paso 1



Para crear un *script*, hay que pulsar en el icono de nuevo archivo y, posteriormente, sobre “R Script”.

**Figura 24.** Creación de un nuevo script.

Fuente: elaboración propia.

## Paso 2

Automáticamente, aparecerá la zona de scripting. Aunque todavía no se haya introducido la sintaxis básica de R, pegaremos las siguientes variables en el script.

```
a <- 5
```

```
b <- 2.0
```

```
c <- '5'
```

### Paso 3

También agregaremos algunos comentarios:

#### # Comentario de una línea

"

#### Comentario de varias líneas

de

código"

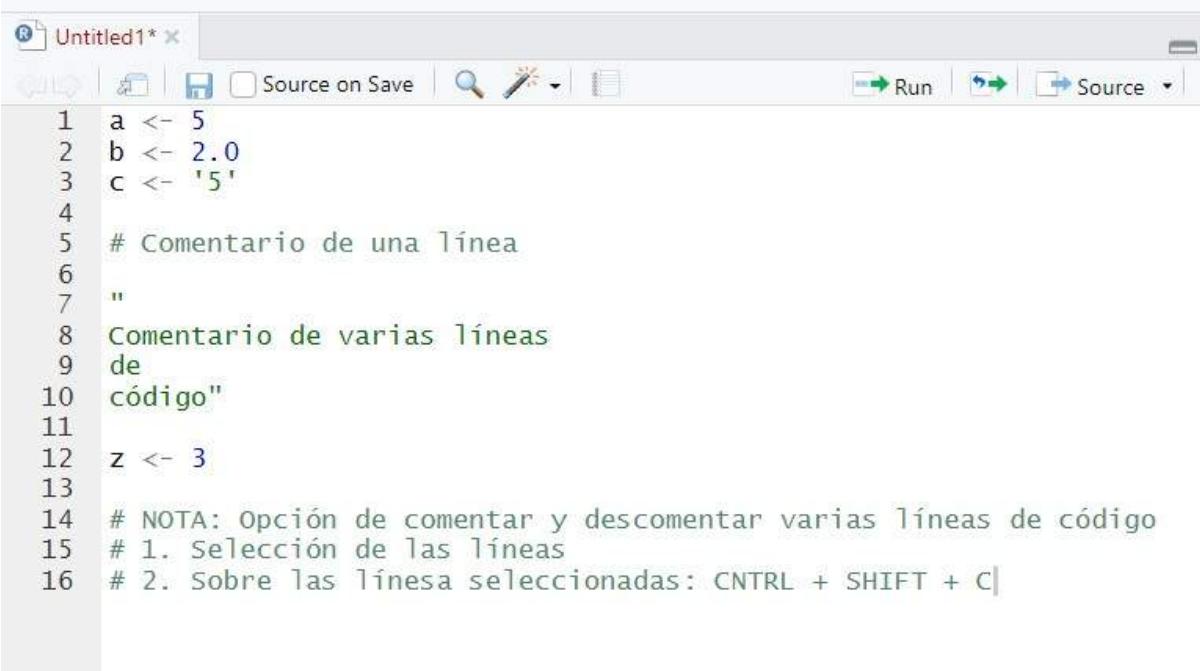
`z <- 3`

#### # NOTA: Opción de comentar y descomentar varias líneas de código

##### # 1. Selección de las líneas

##### # 2. Sobre las líneas seleccionadas: CNTRL + SHIFT + C

## Paso 4



```
R Untitled1* 
Source on Save Run Source
1 a <- 5
2 b <- 2.0
3 c <- '5'
4
5 # Comentario de una línea
6 "
7 Comentario de varias líneas
8 de
9 código"
10
11 z <- 3
12
13
14 # NOTA: Opción de comentar y descomentar varias líneas de código
15 # 1. Selección de las líneas
16 # 2. Sobre las líneas seleccionadas: CNTRL + SHIFT + C|
```

Por lo tanto, se obtendrá el siguiente script:

**Figura 25.** Primer script.

*Fuente:* elaboración propia.

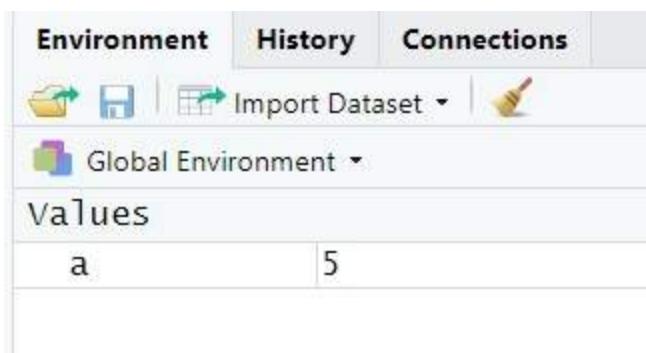
## Paso 5

Para ejecutar línea a línea, simplemente se pulsa sobre esa línea y se ejecutan los siguientes comandos: "CNTRL + INTRO". Aparecerá como resultado dicha línea de código a través de la zona de comandos.

> a <- 5

>

## Paso 6



Además, siempre que se declara una variable, aparecerá en la zona de entorno e historial de comandos.

**Figura 26.** Variables en entornos de programación.

*Fuente:* elaboración propia.

Paso 7



**Figura 27.** Historial de comandos.

**Fuente:** elaboración propia.

## Paso 8

Si en lugar de ir línea a línea se quiere ejecutar todo el *script*, se pulsará el botón “**Source**” y, automáticamente, se ejecutará todo el *script*. Entonces aparecerán las variables “a”, “b”, “c” y “z” en nuestro entorno.

Finalmente, para guardar un *script* se pulsa “File” > “Save with encoding” (escogeremos la codificación por defecto).

Si posteriormente aparecen caracteres extraños en comentarios que contengan tildes, se utilizará la codificación que permita nuestro sistema operativo. Daremos el siguiente nombre al archivo :“Primer\_script”.

Puede descargarse el *script* previamente creado desde el siguiente enlace:



**Primer\_script.zip**

301 B



Complete the content above before moving on.

Son la **forma más habitual**

## **Los *scripts***

**de trabajo en un entorno empresarial.**

**Un proyecto en R se compondrá de varios *scripts* “.R”. No obstante,**

**en un entorno académico o**

Se recomienda descargar los siguientes archivos y abrirlos con RStudio para seguir correctamente los primeros pasos a realizar:



**Pimer\_notebook.zip**

219.6 KB

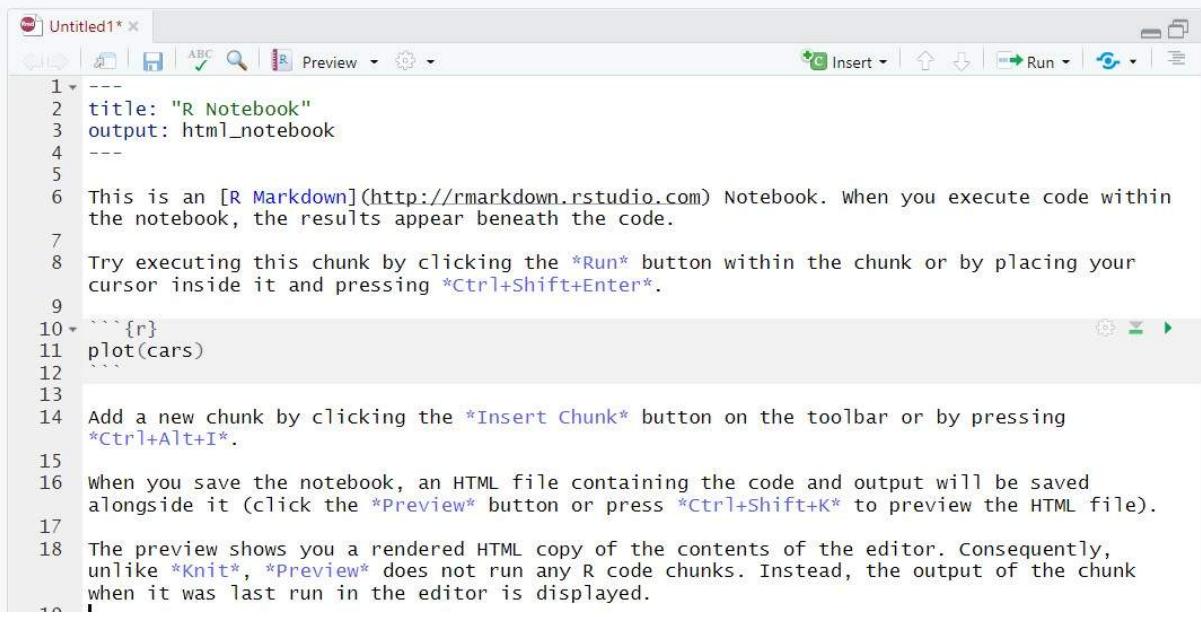


**Pimer\_notebook\_Rmd.zip**

1 KB



## Paso 1



The screenshot shows the RStudio interface with a new notebook titled "Untitled1". The code editor contains the following R Markdown template:

```
1 ---  
2 title: "R Notebook"  
3 output: html_notebook  
4 ---  
5  
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within  
the notebook, the results appear beneath the code.  
7  
8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your  
cursor inside it and pressing *Ctrl+Shift+Enter*.  
9  
10 ````{r}  
11 plot(cars)  
12  
13  
14 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing  
*Ctrl+Alt+I*.  
15  
16 When you save the notebook, an HTML file containing the code and output will be saved  
alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).  
17  
18 The preview shows you a rendered HTML copy of the contents of the editor. Consequently,  
unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk  
when it was last run in the editor is displayed.  
19
```

Para crear el primer *notebook* (a partir de aquí, solo se utilizarán *notebooks*), desde la pestaña “New”, se pulsa en “R Notebook” y, automáticamente, aparecerá un notebook con información por defecto.

**Figura 28.** Notebook por defecto.

Fuente: elaboración propia.

## Paso 2

Para trabajar en el primer notebook, habrá que eliminar todo el contenido, excepto la información del archivo.

---

**title: "R Notebook"**

**output: html\_notebook**

---

Elementos básicos de Markdown en R:

1

2

3

4

Para agregar un encabezado, se utilizará el símbolo de almohadilla o hashtag (#). Hay tres niveles:

1. Encabezado general o título (#).
2. Subtítulo (##).
3. Encabezado de tercer nivel (###).

1	2	3	4	
---	---	---	---	--

Para crear una lista de elementos, se utilizará el asterisco (\*) para comenzar cada elemento.

1	2	3	4	
---	---	---	---	--

Para crear una lista numerada, se utilizará el número seguido de un punto, por ejemplo: 1.

1	2	3	4	
---	---	---	---	--

Para crear diferentes niveles en una lista, se tabulará y agregará un nuevo elemento con el operador más (+).

1	2	3	4	
---	---	---	---	--

Para escribir en cursiva, se escribirá un asterisco o un guion bajo. Por ejemplo: \*cursiva\*, \_cursiva\_.

1	2	3	4	
---	---	---	---	--

Para escribir en negrita, se utilizarán dos guiones bajos o dos asteriscos. Por ejemplo: \*\*negrita\*\*, \_\_negrita\_\_.

Todos estos elementos pueden ser comprobados agregando las siguientes líneas al notebook:

```
# MI PRIMER NOTEBOOK
```

```
## FUNDAMENTOS DE R
```

```
#### Elementos de Markdown
```

Esto, se muestra como una cadena de texto simple.

Ahora agreagamos una lista de elementos:

- \* Primer elemento
- \* Segundo elemento
- \* Tercer elemento

Ahora realizamos una lista numerada:

1. Uno
2. Dos
3. Tres

Lista con sub-elementos:

```
* Ítem 1
  + Sub-ítem 1
* Ítem 2
  + Sub-ítem 2
```

De esta forma añadimos texto en cursiva `*my_function*` , `_my_function_`

De esta otra forma añadimos texto en negrita `**importante**` , `__importante__`

```
9
10 # MI PRIMER NOTEBOOK
11
12 ## FUNDAMENTOS DE R
13
14 ### Elementos de Markdown
15
16 Esto, se muestra como una cadena de texto simple.
17
18 Ahora agreagamos una lista de elementos:
19
20 * Primer elemento
21 * Segundo elemento
22 * Tercer elemento
23
24 Ahora realizamos una lista numerada:
25
26 1. Uno
27 2. Dos
28 3. Tres
29
30 Lista con sub-elementos:
31
32 * Ítem 1
33   + Sub-ítem 1
34 * Ítem 2
35   + Sub-ítem 2
36
37 De esta forma añadimos texto en cursiva *my_function* , _my_function_
38
39 De esta otra forma añadimos texto en negrita **importante** , __importante__
40
```

**Figura 29.** Elementos básicos de Markdown.

Fuente: elaboración propia.

## Paso 1

El siguiente paso será agregar una celda en la que se incluirá código fuente. Para agregar una nueva celda (o chunk) de código, se pulsarán las teclas “CNTRL + SHFT + I” y aparecerá una celda en blanco con el siguiente aspecto.

```
```{r}
```

```
```
```

## Paso 2

Dentro del espacio delimitado en la celda, se incluirá todo el código que se desee ejecutar en ese fragmento (también se podrá ejecutar línea a línea, al igual que un script). Por ejemplo, se pueden definir variables:

# Mis variables

```
a <- 7 # Entero  
b <- 5.33 # Double  
c <- 'Mi cadena de texto' # Sirven tanto comillas simples...  
d <- "Mi otra cadena de texto"# ... como comillas dobles
```

Se ejecuta la celda pulsando en el botón verde “Play”. Aparecerán declaradas las variables en nuestro entorno de sesión.

### Paso 3

Posteriormente, se pueden agregar nuevas celdas de código para imprimir las variables. Para ello, se utilizará la función “print()” y, como parámetro, recibirá el nombre de la variable.

```
```{r}
```

```
print(a)
```

```
```
```

```
```{r}
```

```
print(b)
```

```
```
```

```
```{r}
```

```
print(c)
```

```
```
```

```
```{r}
```

```
print(d)
```

```
```
```

### Resultado

Al ejecutarlas, aparecerán los resultados.

**Figura 30.** Resultados de celdas de código.

*Fuente:* elaboración propia.

```
```{r}
print(a)
```
[1] 7

```{r}
print(b)
```
[1] 5.33

```{r}
print(c)
```
[1] "Mi cadena de texto"

```{r}
print(d)
```
[1] "Mi otra cadena de texto"
```

AYUDA EN R

"SETWD" Y "GET...

VENTAJA DE MARKDOWN

Para desplegar ayuda en R, se puede realizar `help("nombre_func")`, utilizar el operador interrogante con el nombre de la función `?nombre_func` y, también, seleccionar el nombre de la función y pulsar "F1".

```
help("print")
## starting httpd help server ... done
?print
```

AYUDA EN R

"SETWD" Y "GET..."

VENTAJA DE MARKDOWN

Existen dos funciones para establecer un entorno de trabajo (muy útil para cuando existen datasets en una ruta específica): "setwd" y "getwd". A través de la primera, se establece un entorno de trabajo desde la ruta del directorio en el que queramos trabajar; y, con la segunda, obtenemos el entorno de trabajo actual.

**getwd()**

## [1]

"C:/Users/JMMoreno/Desktop/Modulo\_1\_Herramientas\_Para\_Científico\_De\_Datos/Contenido/UD4/Notebooks"

**setwd("C:/Users/JMMoreno/Downloads/") # REALIZAR EL CAMBIO EN CONSOLA**

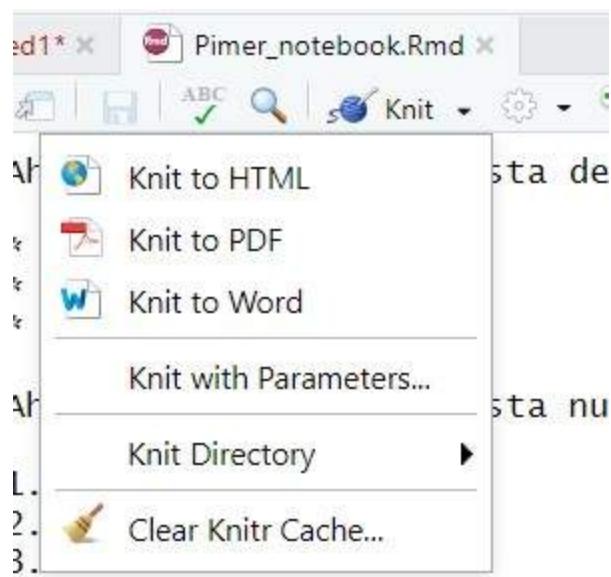
AYUDA EN R

"SETWD" Y "GET..."

VENTAJA DE MARKDOWN

Las ventaja que ofrece Markdown es la posibilidad de guardar en RMD el notebook, además de en HTML o en DOC (también existe la posibilidad de guardar un **notebook** como PDF, pero se deben instalar ciertas librerías antes).

**IMPORTANTE**



Antes de exportar un notebook a HTML o DOC, debe guardarse como RMD. Tras ello, se pulsará el botón “Knit” y “Knit to HTML” o “Knit to Word”.

**Figura 31.** Opciones para guardar un notebook.

*Fuente:* elaboración propia.

## Step 2 Title

```
# Mis variables

a <- 7 # Entero
b <- 5.33 # Double
c <- 'Mi cadena de texto' # Sirven tanto comillas simples...
d <- "Mi otra cadena de texto"# ... como comillas dobles
```

```
print(a)
```

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

```
print(b)
```

```
## [1] 5.33
```

```
print(c)
```

```
## [1] "Mi cadena de texto"
```

```
print(d)
```

```
## [1] "Mi otra cadena de texto"
```

Para desplegar ayuda en R se puede realizar `help("nombre_func")`, utilizar el operador inte  
también seleccionar el nombre de la función y pulsar **F1**

Al guardar un notebook como DOC o HTML, se renderizarán todas las celdas de código y,  
automáticamente, aparecerá el archivo.

**Figura 32.** Vista de un notebook en HTML.

**Fuente:** elaboración propia.

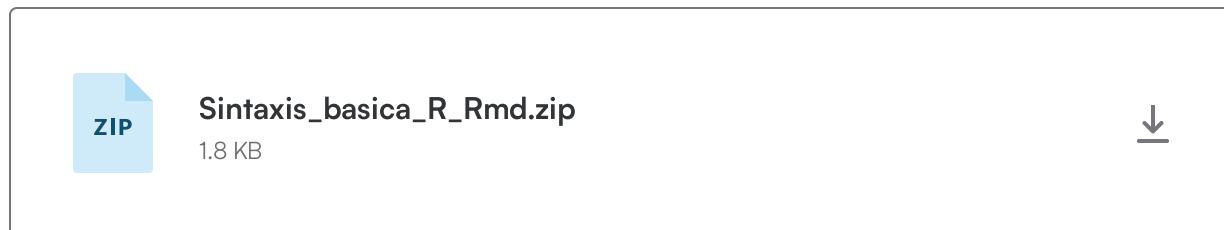
## IV. Sintaxis básica de R

---

Una vez que conocemos los básicos de Markdown y cómo ejecutar las cedas de un notebook en R, se explicará, a lo largo de esta sección, la sintaxis básica de R. Se profundizará sobre los siguientes elementos de R:

- Objetos.
- Comentarios.
- Operadores.
- Tipos de variables.

Se recomienda seguir esta sección a través del siguiente *notebook*:



**CONTINUAR**

## 4.1. Objetos

### Definición

Cualquier variable, función, parámetro, etc. que se declare en R se denomina objeto.

Para crear un objeto en R, a diferencia de otros lenguajes de programación, **no se utiliza el operador de asignación “=”**

Se utiliza la **asignación de forma direccional**, es decir, con los siguientes elementos:

- <-
- ->

### Propiedad mutable

Los objetos son **mutables**, es decir: podemos cambiar su valor en cualquier momento, realizando una nueva asignación.

**Todos los objetos de R se almacenan en memoria RAM hasta que se cierra el programa o se limpia el entorno de variables.**

## Almacenamiento 1

Mediante el comando “ls” se pueden ver todos los objetos que tenemos almacenados en memoria.

**ls()**

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

## Almacenamiento 2

Si, una vez realizada una operación, se han almacenado objetos demasiado grandes en la memoria, se pueden eliminar mediante la función “rm”, a la cual habrá que pasar como parámetro el nombre de la variable a eliminar.

**rm(a)**

```
## Warning in rm(a): object 'a' not found
```

```
print(ls())
```

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

### IMPORTANTE

No obstante, **para ahorrar ciertos pasos**, como carga y creación de archivos y objetos, si se desea, **se puede guardar**, además del notebook o script, **todo el entorno de variables y objetos**. Al guardar, se le asignará un nombre y se guardará con extensión “.rdata”. Cuando se vuelva a abrir la sesión, se restaurarán todos los objetos que había almacenados.

**CONTINUAR**

## 4.2. Comentarios

En R, se pueden utilizar dos tipos de comentario:

### De una línea

Se utiliza el operador "#" y se agrega el comentario.

### Multilínea

Se abre y se cierra el comentario con comillas dobles ("").

# Comentario de una línea

# Otra línea

"

Comentario

de

varias

líneas

"

**Para facilitar el trabajo, en R, se pueden comentar varias líneas a la vez. Para ello:**

Selección de las líneas a comentar.

Sobre las líneas seleccionadas "CTRL + SHIFT + C"

CONTINUAR

### 4.3. Operadores

En R existen operadores de diferentes tipos:

- Aritméticos
- Comparación
- Lógicos

Se definen algunas variables para mostrar operadores

a <- 7  
b <- 2

#### OPERADORES ARITMÉTICOS

```
# Suma  
a + b  
## [1] 9  
  
# Resta  
a - b  
## [1] 5  
  
# Multiplicación  
a * b  
## [1] 14
```

```
# División  
a / b  
## [1] 3.5
```

```
# Exponencial  
a ^ b  
## [1] 49
```

```
# Módulo o resto de una división  
a %% b  
## [1] 1
```

```
# División entera  
a %/% b  
## [1] 3
```

#### OPERADORES DE COMPARACIÓN

#### OPERADORES LOGÍSTICOS

## OPERADORES ARITMÉTICOS

## OPERADORES DE COMPARACIÓN

## OPERADORES LOGÍSTICOS

# Menor que

a < b

## [1] FALSE

# Mayor que

a > b

## [1] TRUE

# Menos o igual que

a <= b

## [1] FALSE

# Mayor o igual que

a >= b

## [1] TRUE

# Igualdad

a == b

## [1] FALSE

# No es igual que

a != b

## [1] TRUE

## OPERADORES ARITMÉTICOS

## OPERADORES DE COMPARACIÓN

## OPERADORES LOGÍSTICOS

# Negación

!(TRUE) # Equivalente con T

## [1] FALSE

```
# AND Lógico  
TRUE & TRUE  
## [1] TRUE  
TRUE & FALSE  
## [1] FALSE  
# OR Lógico  
TRUE | FALSE  
## [1] TRUE  
FALSE | FALSE  
## [1] FALSE
```

CONTINUAR

## 4.4. Tipos de variable

En R, se pueden distinguir tres tipos de variables. Esto es independiente del tipo de estructura que se utilice para su almacenamiento y procesamiento:

- Numéricas
- Carácter
- Booleanas
- Factor

Para determinar el tipo de una variable, se utiliza la función `class`:

```
class(5)  
## [1] "numeric"
```

```
class(3.9)
## [1] "numeric"
class("Hola")
## [1] "character"
class(TRUE)
## [1] "logical"
class(T)
## [1] "logical"
# Por el momento, no prestaremos mucha atención sobre la construcción
# de tipos factor, esto se explicará en detalle en próximas unidades
class(as.factor('A'))
## [1] "factor"
```

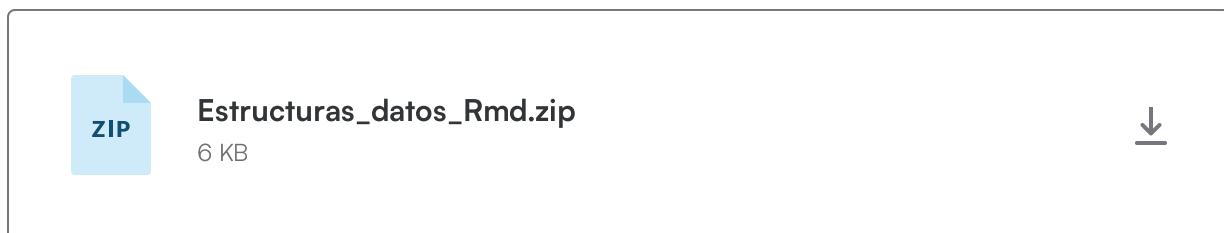
## V. Estructura de datos

---

Dentro de R, existen **diferentes tipos de estructuras de datos** con las que se puede trabajar. Dentro de esta sección, se explicará en qué consisten y se mostrarán algunas de sus principales funciones:

- Vectores
- *Arrays*
- Listas
- Factores
- Matrices
- *Dataframes*

Se recomienda seguir esta sección con el siguiente notebook:



## 5.1. Vectores

En R, un vector es una sucesión de elementos de una dimensión. Un vector puede soportar diferentes tipos de variables.

Para su construcción, se utiliza la función “c”.

```
a <- c(1, 3.0, 7, 'hola')  
  
a  
## [1] "1"  "3"  "7"  "hola"
```

Podemos indexar un vector accediendo a sus posiciones:

```
# Primera posición  
a[1]  
## [1] "1"  
# Posición n  
a[4]  
## [1] "hola"  
# Rango de posiciones  
a[2:4]  
## [1] "3"  "7"  "hola"  
# Excluir elementos  
a[-2] # Sin la posición 2  
## [1] "1"  "7"  "hola"  
# A través de máscara booleana o vector lógico  
a[c(T, F, F, T)]  
## [1] "1"  "hola"
```

**i** Es muy importante resaltar que, en los vectores, cuando se aplica una operación, se hace en todos los elementos del vector, a no ser que se especifique lo contrario.

Para que se puedan aplicar operaciones sobre un vector, todos sus elementos deben ser del mismo tipo. A continuación se muestran dos ejemplos:

1º

2º

Sobre el vector actual no se podría operar:

```
# a+4  
# Error in a + 4 : non-numeric argument to binary operator  
# Asignación de un nuevo vector  
b <- c(1, 30, 4, 56, 9)  
  
# Suma a todos los elementos de 4  
b + 4  
## [1] 5 34 8 60 13  
# Resta a todos los elementos de 3  
b - 3  
## [1] -2 27 1 53 6  
# Todos los elementos al cuadrado  
b ^ 2  
## [1] 1 900 16 3136 81
```

1º

2º

Si hay más de un vector del mismo tipo y longitud, se pueden aplicar operaciones sobre ellos:

```
# Resta de vectores  
c <- c(3, 50, 4, 3, 0)  
  
b - c  
## [1] -2 -20  0  53  9  
# Producto de vectores, elemento a elemento  
c * b  
## [1] 3 1500 16 168 0
```

Existen algunas **funciones que son propias de los vectores y que permiten realizar aritmética vectorial**:

### "length"

"length": longitud de un vector.

```
length(b)  
## [1] 5  
length(a)  
## [1] 4  
length(c)
```

### "sum"

"sum": realiza la suma de todos los elementos del vector.

```
sum(b)  
## [1] 100
```

## "prod"

"prod": realiza el producto de todos los elementos del vector.

```
prod(b)  
## [1] 60480
```

## "min"

"min": devuelve el elemento mínimo de un vector.

```
min(b)  
## [1] 1
```

## "max"

"max": devuelve el elemento máximo de un vector.

```
max(b)  
## [1] 56
```

## "range"

"range": devuelve el rango del vector, es decir, de su elemento mínimo a máximo.

```
range(b)  
## [1] 1 56
```

## "mean"

"mean": devuelve la media del vector.

```
mean(b)  
## [1] 20
```

## "var"

"var": devuelve la varianza del vector.

```
var(b)  
## [1] 533.5
```

## "sd"

"sd": devuelve la desviación estándar del vector.

```
sd(b)  
## [1] 23.09762
```

## "cumsum"

"cumsum": devuelve la suma acumulada elemento a elemento del vector.

```
cumsum(b)  
## [1] 1 31 35 91 100
```

## "cumprod"

"cumprod": devuelve el producto acumulado del vector elemento a elemento.

```
cumprod(b)
## [1] 1 30 120 6720 60480
```

## "sqrt"

"sqrt": raíz de todos los elementos del vector.

```
sqrt(b)
## [1] 1.000000 5.477226 2.000000 7.483315 3.000000
```

A través de vectores, se pueden crear **sucesiones**. En algunas ocasiones, se puede hacer **partiendo de un vector de entrada o creándolo de cero**:

RANGO DEFINIDO

"SEQ"

"REP"

SAMPLE

EX  
B(

Generar un vector a partir de un **rango definido** de n a m. A través del operador:

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

| RANGO DEFINIDO | "SEQ" | "REP" | SAMPLE | EX<br>B |
|----------------|-------|-------|--------|---------|
|----------------|-------|-------|--------|---------|

Generar secuencias a través de la función "seq". Esta función toma como parámetros origen de la secuencia, máximo de la secuencia y elemento de división.

```
seq(-10, 20, 2) # -10 a 20 de dos en dos
## [1] -10 -8 -6 -4 -2  0  2  4  6  8 10 12 14 16 18 20
seq(0, 1, 0.1)
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

| RANGO DEFINIDO | "SEQ" | "REP" | SAMPLE | EX<br>B |
|----------------|-------|-------|--------|---------|
|----------------|-------|-------|--------|---------|

A través de la función "rep" podemos generar vectores de números repetidos. Esta función toma como parámetros el número a repetir (o elemento) y el número de veces.

```
# Un número
rep(2, 200)
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [186] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
rep(0.5, 10)
## [1] 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
# Secuencias de números
rep(1:10, 3)
## [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5
## [26] 6 7 8 9 10
```

| RANGO DEFINIDO | "SEQ" | "REP" | SAMPLE | EX<br>B( |
|----------------|-------|-------|--------|----------|
|----------------|-------|-------|--------|----------|

Mediante la función “sample”, que toma como parámetros el rango de valores, el número de elementos y si los valores se generaran con reemplazamiento o no.

```
no.rep <- sample(1:10, 10, replace = F)
si.rep <- sample(30:45, 10, replace = T)
```

```
no.rep
## [1] 1 7 5 9 2 6 8 10 4 3
si.rep
## [1] 45 42 36 33 44 36 40 45 37 43
```

| RANGO DEFINIDO | "SEQ" | "REP" | SAMPLE | EX<br>B( |
|----------------|-------|-------|--------|----------|
|----------------|-------|-------|--------|----------|

Otra forma de seleccionar subvectores dentro de un vector ya existente es a través de operaciones de comparación (expresiones booleanas):

```
# Creamos un vector nuevo
z <- rep(1:20, 3)

# Vamos a asignar a todos los elementos del vector que sean menores de 7 el valor 1

z[z < 7] <- 1
z
## [1] 1 1 1 1 1 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 1 1 1 1
## [26] 1 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 1 1 1 1 1 7 8 9 10
## [51] 11 12 13 14 15 16 17 18 19 20
# Seleccionar todos los elementos del array que son diferentes de 19
z[z != 19]
```

```
## [1] 1 1 1 1 1 1 7 8 9 10 11 12 13 14 15 16 17 18 20 1 1 1 1 1
## [26] 7 8 9 10 11 12 13 14 15 16 17 18 20 1 1 1 1 1 1 7 8 9 10 11 12
## [51] 13 14 15 16 17 18 20
```

| RANGO DEFINIDO | "SEQ" | "REP" | SAMPLE | EX<br>B) |
|----------------|-------|-------|--------|----------|
|                |       |       |        |          |

Finalmente, se puede ordenar un vector a través de la función "sort":

```
# Generamos vector
no.rep.dos <- sample(30:60, 10, replace = F)

# Mostramos el vector ordenado
sort(no.rep.dos, decreasing = T)
## [1] 60 59 58 57 55 50 45 34 33 30
```

**CONTINUAR**

## 5.2. Arrays

A diferencia de los vectores, en un *array*, los elementos deben ser del mismo tipo. También son indexables.

A continuación, se muestran las diferentes formas de crear un *array*.

| FUNCIÓN ARRAY | FUNCIÓN "DIM" | VECTOR EXISTENTE |
|---------------|---------------|------------------|
|               |               |                  |

Para crear un array, haremos uso de la función array.

```
# Definimos un nuevo array  
my.arr <- array(data = c(1,2,3,4), dim = c(2,2))  
  
my.arr  
## [1] [2]  
## [1,] 1 3  
## [2,] 2 4
```

FUNCTION ARRAY

FUNCTION "DIM"

VECTOR EXISTENTE

Podemos ver la dimensión de un array a través de la función "dim":

```
dim(my.arr) # Dos dimensiones de dos elementos  
## [1] 2 2
```

FUNCTION ARRAY

FUNCTION "DIM"

VECTOR EXISTENTE

También es posible crear un array a través de un vector ya existente:

```
no.rep.dos  
## [1] 50 59 33 60 34 55 57 45 58 30  
length(no.rep.dos) # Divisible entre 2 y 5  
## [1] 10  
dim(no.rep.dos) <- c(5,2)  
no.rep.dos # Vector transformado en array
```

```
## [1] [2]
## [1] 50 55
## [2] 59 57
## [3] 33 45
## [4] 60 58
## [5] 34 30
```

## Ejemplo

A continuación, se muestran diferentes maneras de seleccionar elementos de un array:

```
new.arr <- array(1:40, c(5, 4, 2))
```

```
new.arr
## , , 1
##
## [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
##
## , , 2
##
## [1] [2] [3] [4]
## [1] 21 26 31 36
## [2] 22 27 32 37
## [3] 23 28 33 38
## [4] 24 29 34 39
## [5] 25 30 35 40
print(dim(new.arr))
## [1] 5 4 2
# Sólo primera dimensión.
new.arr[,1]
## [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
# Columna 2 de las dos dimensiones
new.arr[2,1:2]
## [1] [2]
## [1] 6 26
## [2] 7 27
## [3] 8 28
## [4] 9 29
## [5] 10 30
# Fila 4 de la segunda dimensión.
new.arr[4,2]
## [1] 24 29 34 39
# Elementos en tercera y cuarta posición de la quinta fila, de ambas dimensiones
new.arr[5,c(3,4),]
## [1] [2]
## [1] 15 35
## [2] 20 40
# Todas las filas de la segunda columna en adelante de la primera dimensión
new.arr[,2:length(new.arr[1,,1]),1]
## [1] [2] [3]
## [1] 6 11 16
## [2] 7 12 17
## [3] 8 13 18
## [4] 9 14 19
## [5] 10 15 20

```

Análogamente, también es posible indexar un array a través de otro array.

```

index <- array(c(2, 1:3, 1:2), c(2,3))

index
## [1] [2] [3]
## [1] 2 2 1
## [2] 1 3 2
new.arr[index]
## [1] 7 31

```

## Operaciones aritméticas

Del mismo modo que los vectores, sobre los arrays también podemos aplicar operaciones aritméticas, como las descritas anteriormente, mostramos algunos ejemplos:

```
# Suma  
sum(new.arr)  
## [1] 820  
# Media  
mean(new.arr)  
## [1] 20.5  
# Rango  
range(new.arr)  
## [1] 1 40  
# Máximo  
max(new.arr)  
## [1] 40  
# ...
```

## A nivel de dimensión, fila y columna de los arrays.

Estas operaciones también funcionan a nivel de dimensión, fila y columna de los arrays.

```
sum(new.arr[2, 1]) #Suma de los elementos de la segunda fila de la primera dimensión.  
## [1] 38
```

## Operaciones especiales para los arrays

No obstante, existen operaciones especiales para los arrays que funcionan a nivel de eje (fila x columna), que son las que se muestran a continuación:

```
sum(new.arr[2, 1]) #Suma de los elementos de la segunda fila de la primera dimensión.  
## [1] 38  
# Suma de columnas  
colSums(new.arr, dims = 1) # Suma de todas las columnas por fila
```

```

## [1] [2]
## [1] 15 115
## [2] 40 140
## [3] 65 165
## [4] 90 190
colSums(new.arr, dims = 2) # Suma de todas las columnas por columna
## [1] 210 610
# Suma de filas
rowSums(new.arr, dims = 1) # Suma de todas las filas por fila
## [1] 148 156 164 172 180
rowSums(new.arr, dims = 2) # Suma de todas las filas por columna
## [1] [2] [3] [4]
## [1] 22 32 42 52
## [2] 24 34 44 54
## [3] 26 36 46 56
## [4] 28 38 48 58
## [5] 30 40 50 60
# Media de columnas
colMeans(new.arr, dims = 1) # Media de columnas por columna
## [1] [2]
## [1] 3 23
## [2] 8 28
## [3] 13 33
## [4] 18 38
colMeans(new.arr, dims = 2) # Media de columnas por dimensión
## [1] 10.5 30.5
rowMeans(new.arr, dims = 1) # Media de filas por fila
## [1] 18.5 19.5 20.5 21.5 22.5
rowMeans(new.arr, dims = 2) # Media de filas por dimensión
## [1] [2] [3] [4]
## [1] 11 16 21 26
## [2] 12 17 22 27
## [3] 13 18 23 28
## [4] 14 19 24 29
## [5] 15 20 25 30

```

## Función "aperm"

Si se quieren permutar las dimensiones de un array, se puede hacer uso de la función "aperm":

```
aperm(new.arr)
```

```
## ,1
##
## [1] [2] [3] [4]
## [1] 1 6 11 16
## [2] 21 26 31 36
##
## ,2
##
## [1] [2] [3] [4]
## [1] 2 7 12 17
## [2] 22 27 32 37
##
## ,3
##
## [1] [2] [3] [4]
## [1] 3 8 13 18
## [2] 23 28 33 38
##
## ,4
##
## [1] [2] [3] [4]
## [1] 4 9 14 19
## [2] 24 29 34 39
##
## ,5
##
## [1] [2] [3] [4]
## [1] 5 10 15 20
## [2] 25 30 35 40
```

Los arrays también tienen una clase única.

```
class(new.arr)
## [1] "array"
```

CONTINUAR

## 5.3. Listas

Con cierta similitud a un vector, una **lista**, es una **colección de objetos, indexable a través del operador "[[ ]]" (doble corchete)**. Soporta desde el mismo tipo de dato a diferentes tipos de datos. La declaración de una lista se realiza con la función "list".

### "list"

```
my.list <- list(c(1:40))
my.list
## [[1]]
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
my.list[1] # Solamente tiene un elemento la lista, que es un vector
## [[1]]
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
my.list[[1]][5] # Accedemos al primer elemento de la lista, posición 5 del vector.
## [1] 5
```

### Nombres

Algo que es siempre aconsejable para poder identificar los diferentes campos o elementos de una lista, es utilizar nombres.

```
new.list <- list(alumnos=c("Juan", "María", "Alfredo"),
calificaciones=c(7, 4, 5, 9))
```

## Diferentes formas de acceder y "\$"

Podemos acceder a sus elementos además de con doble corchetes con un operador especial de R y, como es habitual en dataframes, el operador "\$":

```
new.list$alumnos  
## [1] "Juan" "María" "Alfredo"  
new.list[["alumnos"]]  
## [1] "Juan" "María" "Alfredo"
```

## Agregar elementos

Si un elemento no existe en un componente de la lista, simplemente se agrega:

```
new.list$alumnos <- c(new.list$alumnos, "Nuevo alumno")  
new.list$alumnos  
## [1] "Juan"      "María"     "Alfredo"    "Nuevo alumno"  
new.list$edad <- c(15, 16, 17, 15)  
  
new.list  
## $alumnos  
## [1] "Juan" "María" "Alfredo" "Nuevo alumno"  
##  
## $calificaciones  
## [1] 7 4 5 9  
##  
## $edad  
## [1] 15 16 17 15
```

## Clase propia

```
class(new.list)  
## [1] "list"
```

**CONTINUAR**

## **5.4. Factores**

Los **factores** son **vectores especiales** que se utilizan únicamente para la representación de variables **categóricas**, que son las que indican una **cualidad o característica**, como, por ejemplo, alto, bajo, medio, etc.

## **Crear una variable categórica**

Para crear una variable categórica, tenemos que pasar un vector de caracteres a la función `factor`.

```
var.cat <- factor(c("ENERO", "FEBRERO", "MARZO", "ABRIL"))  
var.cat  
  
## [1] ENERO FEBRERO MARZO ABRIL  
## Levels: ABRIL ENERO FEBRERO MARZO
```

Una variable categórica se caracteriza porque cada categoría en R se denomina nivel (*level*). Se puede acceder a sus niveles a través de la función “levels”:

```
levels(var.cat)
```

```
## [1] "ABRIL"  "ENERO"  "FEBRERO" "MARZO"
```

Este tipo de vectores especiales, tienen su propia clase:

```
class(var.cat)
```

```
## [1] "factor"
```

### Eliminar un nivel de una variable categórica

Para eliminar un nivel de una variable categórica, se hará uso de la función “*droplevels*”, que recibirá como parámetro la variable categórica y la categoría a eliminar:

```
var.cat = droplevels(x = var.cat, "MARZO")
```

```
var.cat  
## [1] ENERO FEBRERO <NA> ABRIL  
## Levels: ABRIL ENERO FEBRERO
```

CONTINUAR

## 5.5. Matrices

Previamente, se han tratado los arrays multidimensionales. **Una matriz no es ni más ni menos que un array de dos dimensiones.** No obstante, las matrices **tienen algunos tipos de funciones especiales con las que se pueden trabajar.** Se describirán posteriormente.

### Función "matrix"

Para crear una matriz, se utiliza la función “**matrix**”, que recibe los datos de la propia matriz y los parámetros “**nrow**” y “**ncol**”. Además, se puede especificar cómo se rellena la matriz a partir de los

Además de todas las funciones ya vistas que pueden realizarse en arrays y las formas disponibles de indexación, **en matrices existen siguientes funciones propias:**

#### "ncol"

“ncol”: devuelve el número de columnas.

```
# Número de columnas  
ncol(my.mat)  
## [1] 6
```

## "nrow"

"nrow": devuelve el número de filas.

```
# Número de filas  
nrow(my.mat)  
## [1] 5
```

## "diag"

"diag": devuelve la diagonal de la matriz.

```
# Diagonal  
diag(my.mat)  
## [1] 1 7 13 19 25
```

## "crossprod"

"crossprod": realiza el producto entre dos matrices.

```
crossprod(x = my.mat, y = my.mat)  
## [1] [2] [3] [4] [5] [6]  
## [1] 55 130 205 280 355 430  
## [2] 130 330 530 730 930 1130  
## [3] 205 530 855 1180 1505 1830  
## [4] 280 730 1180 1630 2080 2530  
## [5] 355 930 1505 2080 2655 3230  
## [6] 430 1130 1830 2530 3230 3930
```

"t"

"t": devuelve la traspuesta de una matriz.

### # Traspuesta

```
t(my.mat)
## [1] [2] [3] [4] [5]
## [1] 1 2 3 4 5
## [2] 6 7 8 9 10
## [3] 11 12 13 14 15
## [4] 16 17 18 19 20
## [5] 21 22 23 24 25
## [6] 26 27 28 29 30
```

"svd"

"svd": valores singulares de una matriz.

### # Valores singulares

```
svd(my.mat)
## $d
## [1] 9.716531e+01 3.728537e+00 2.538160e-15 1.947191e-15 6.089851e-16
## 
## $u
## [1] [2] [3] [4] [5]
## [1] -0.4018926 -0.66217998 0.48765261 0.2233859 -0.3351025
## [2] -0.4240057 -0.34672632 -0.80457360 -0.2008214 -0.1110499
## [3] -0.4461188 -0.03127266 0.07019089 0.2397858 0.8588224
## [4] -0.4682320 0.28418100 0.32272856 -0.7706509 -0.0440853
## [5] -0.4903451 0.59963465 -0.07599847 0.5083006 -0.3685848
## 
## $v
## [1] [2] [3] [4] [5]
## [1] -0.0711459 0.7202415 -0.55959544 -0.09385682 0.1469272
## [2] -0.1859294 0.5105569 0.24248261 0.08217043 0.2585642
## [3] -0.3007128 0.3008724 0.22067008 0.22952872 -0.8429267
## [4] -0.4154963 0.0911878 0.54034856 -0.61659563 0.1878316
## [5] -0.5302798 -0.1184968 0.08533938 0.68520749 0.3842241
## [6] -0.6450632 -0.3281813 -0.52924519 -0.28645419 -0.1346204
```

## "eigen"

"eigen": realiza el cálculo de autovalores y autovectores (la matriz tiene que ser regular).

**# IMPORTANTE:** Necesita una matriz regular

```
regular.mat <- matrix(1:9, ncol = 3, nrow = 3, byrow = F)
```

```
eigen(regular.mat)
## eigen() decomposition
## $values
## [1] 1.611684e+01 -1.116844e+00 -5.700691e-16
##
## $vectors
## [1] [2] [3]
## [1] -0.4645473 -0.8829060 0.4082483
## [2] -0.5707955 -0.2395204 -0.8164966
## [3] -0.6770438 0.4038651 0.4082483
```

## "as.matrix"

Puede transformarse un vector como matriz a través de la función "as.matrix".

```
new.mat <- new.arr[, 1]
```

```
as.matrix(new.mat)
## [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
class(new.mat)
## [1] "matrix"
```

Las matrices, al igual que el resto de estructuras de datos que se trabajan en esta lección, tienen una clase propia:

```
class(my.mat)
## [1] "matrix"
```

**CONTINUAR**

## 5.6. *Dataframes*

Para finalizar esta sección, se hablará sobre *dataframes*.

**¿Cómo funciona?**

De forma resumida, un *dataframe* funciona de forma similar a una matriz, a una hoja de Excel o a una base de datos. Todos sus elementos tienen un índice común a cada observación. En *dataframes* se habla de

Para aprender las características y diferentes funcionalidades que ofrece un *dataframe*, **se trabajará directamente con un archivo CSV**. La función encargada de leer, cargar y transformar un archivo CSV como *dataframe* es “read.csv”.

- i Es importante revisar otros parámetros, como “sep”, “header” o “na.strings” en el caso de que se sepa de antemano que el *dataset* tiene valores nulos que son representados por otros valores o caracteres.

**"sep", "header" y "na.strings"**

## "sep"

El parámetro "sep" puede tomar diferentes valores en función del separador del CSV, ya sea en coma, punto y coma, barra, etc.

## "header"

El parámetro "header" se utilizará para que la primera fila sea tomada como cabecera del *dataframe*, es decir, los nombres de las columnas.

## "na.strings"

Es importante que, si se conoce de antemano en el CSV que un valor debe ser tratado como nulo, y para que R lo reconozca como tal, debe utilizarse el parámetro "na.strings" especificando el valor que debe ser tratado como nulo.

Para este apartado, debe descargarse el siguiente archivo<sup>1</sup>:

```
df <- read.csv("FB.csv")
```



---

<sup>1</sup>Este archivo contiene información bursátil sobre las acciones de la empresa Facebook, puede descargarse el *dataset* original desde [este enlace](#).

Del mismo modo, se aconseja al alumno probar otro tipo de *datasets* sobre información bursátil de diferentes compañías, pueden tomarse como referencia los *datasets* disponibles en [este enlace](#).

## Columnas

De un modo similar a las listas, puede llamarse a las columnas a través de dobles corchetes, con el operador “\$” o a través de indexación:

```
df$Open
## [1] 42.05 36.53 32.61 31.37 32.95 32.90 31.48 28.70 28.55 28.89 27.20 26.70
## [13] 26.07 27.00 26.55 27.18 27.48 27.66 27.65 28.51 29.96 31.54 31.92 31.67
## [25] 32.41 32.86 32.69 32.46 31.96 31.92 31.25 30.91 31.32 31.44 32.10 32.43
## [37] 31.48 30.70 31.04 30.50 28.48 28.31 29.41 29.00 28.12 28.82 28.39 27.75

df[[2]]
## [1] 42.05 36.53 32.61 31.37 32.95 32.90 31.48 28.70 28.55 28.89 27.20 26.70
## [13] 26.07 27.00 26.55 27.18 27.48 27.66 27.65 28.51 29.96 31.54 31.92 31.67
## [25] 32.41 32.86 32.69 32.46 31.96 31.92 31.25 30.91 31.32 31.44 32.10 32.43
## [37] 31.48 30.70 31.04 30.50 28.48 28.31 29.41 29.00 28.12 28.82 28.39 27.75

df[["Open"]]
## [1] 42.05 36.53 32.61 31.37 32.95 32.90 31.48 28.70 28.55 28.89 27.20 26.70
## [13] 26.07 27.00 26.55 27.18 27.48 27.66 27.65 28.51 29.96 31.54 31.92 31.67
## [25] 32.41 32.86 32.69 32.46 31.96 31.92 31.25 30.91 31.32 31.44 32.10 32.43
## [37] 31.48 30.70 31.04 30.50 28.48 28.31 29.41 29.00 28.12 28.82 28.39 27.75
```

## Resultado

El resultado, en todos los casos, es un vector sobre el que podemos aplicar las mismas funciones de indexación y operaciones ya vistas en la sección de vectores.

```
# Suma de la variable Open
sum(df$Open)
## [1] 5041.93

# Mostramos las posiciones 3 a 8 de la variable Open
df$Open[3:8]
```

```

## [1] 32.61 31.37 32.95 32.90 31.48 28.70
# Filtramos la variable open por todos los valores superiores o iguales que 40
df[df$Open >= 40,]
##   Date Open High Low Close Volume Adj.Close
## 1 2012-05-18 42.05 45 38 38.23 573576400 38.23
# Seleccionamos las filas 100 a 110 de todas las columnas a excepción de la segunda y tercera
df[100:110, -c(2,3)]
##   Date Low Close Volume Adj.Close
## 100 2012-10-09 19.97 20.23 27161800 20.23
## 101 2012-10-10 19.45 19.64 39321800 19.64
## 102 2012-10-11 19.61 19.75 21817300 19.75
## 103 2012-10-12 19.48 19.52 18809400 19.52
## 104 2012-10-15 19.49 19.52 20189700 19.52
## 105 2012-10-16 19.30 19.48 21834700 19.48
## 106 2012-10-17 19.37 19.88 44074500 19.88
## 107 2012-10-18 18.89 18.98 52157400 18.98
## 108 2012-10-19 18.80 19.00 34835000 19.00
## 109 2012-10-22 19.05 19.32 32447300 19.32
## 110 2012-10-23 19.10 19.50 78381200 19.50

```

**CLASE**

"ATTACH"

"DETACH"

Los **dataframes** también tienen un tipo especial de clase:

```

class(df)
## [1] "data.frame"

```

**CLASE**

"ATTACH"

"DETACH"

Para ahorrar trabajo y no tener que escribir siempre el nombre del **dataframe** cuando se quiera trabajar con una variable, puede utilizarse la función “attach” y cargará todas las columnas como variables:

```
attach(df)
```

```
Date
## [1] 2012-05-18 2012-05-21 2012-05-22 2012-05-23 2012-05-24 2012-05-25
## [7] 2012-05-29 2012-05-30 2012-05-31 2012-06-01 2012-06-04 2012-06-05
## [13] 2012-06-06 2012-06-07 2012-06-08 2012-06-11 2012-06-12 2012-06-13
```

CLASE

"ATTACH"

"DETACH"

Cuando ha finalizado el trabajo con el **dataframe**, se utiliza “detach” para que sea necesario declarar el nombre del **dataframe** para trabajar con él:

```
detach(df)
```

Algunas funciones principales para realizar con **dataframes** son las siguientes:

**"summary"**

“summary”: devuelve un resumen estadístico de un **dataframe**.

```
# Resumen estadístico de un dataframe
summary(df)
##      Date      Open      High       Low
## 2012-05-18: 1 Min. :18.08 Min. :18.27
##                                         Max. :18.55
```

```
## 2012-05-21: 1 1st Qu.:21.12 1st Qu.:21.59 1st Qu.:20.61
## 2012-05-22: 1 Median :27.02 Median :27.56 Median :26.44
## 2012-05-23: 1 Mean :25.72 Mean :26.23 Mean :25.15
## 2012-05-24: 1 3rd Qu.:28.94 3rd Qu.:29.51 3rd Qu.:28.39
## 2012-05-25: 1 Max. :42.05 Max. :45.00 Max. :38.00
## (Other) :190
## Close Volume Adj.Close
## Min. :17.73 Min. : 8108300 Min. :17.73
## 1st Qu.:21.11 1st Qu.: 30382925 1st Qu.:21.11
## Median :26.91 Median : 46179100 Median :26.91
## Mean :25.63 Mean : 56928836 Mean :25.63
## 3rd Qu.:29.01 3rd Qu.: 72741500 3rd Qu.:29.01
## Max. :38.23 Max. :573576400 Max. :38.23
##
```

## "str"

"str": devuelve el tipo de variable del **dataframe**.

```
# Estructura columnar de un dataframe
str(df)
## 'data.frame': 196 obs. of 7 variables:
## $ Date : Factor w/ 196 levels "2012-05-18","2012-05-21",..: 1 2 3 4 5 6 7 8 9 10 ...
## $ Open : num 42 36.5 32.6 31.4 33 ...
## $ High : num 45 36.7 33.6 32.5 33.2 ...
## $ Low : num 38 33 30.9 31.4 31.8 ...
## $ Close : num 38.2 34 31 32 33 ...
## $ Volume : int 573576400 168192700 101786600 73600000 50237200 37149800 78063400 57267900
## $ Adj.Close: num 38.2 34 31 32 33 ...
```

## attributes

**attributes**: devuelve el nombre de las filas y columnas.

```
# Nombre de filas y columnas
attributes(df)
```

```
## $names
## [1] "Date" "Open" "High" "Low" "Close" "Volume"
## [7] "Adj.Close"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
## [19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
## [37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
## [55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
## [73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
## [91] 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
## [109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
## [127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
## [145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
## [163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
## [181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196
```

## dim

dim: dimensiones del **dataframe** filas x columnas

```
# Dimensiones de un dataframe
dim(df)
## [1] 196 7
```

## "colnames"

"colnames": nombre de las columnas del **dataframe**.

```
# Nombre de las columnas de un dataframe
colnames(df)
## [1] "Date"     "Open"    "High"    "Low"     "Close"   "Volume"
## [7] "Adj.Close"
```

## "rownames"

# Nombre de las filas

```
rownames(df)
## [1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12"
## [13] "13" "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24"
## [25] "25" "26" "27" "28" "29" "30" "31" "32" "33" "34" "35" "36"
## [37] "37" "38" "39" "40" "41" "42" "43" "44" "45" "46" "47" "48"
## [49] "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"
## [61] "61" "62" "63" "64" "65" "66" "67" "68" "69" "70" "71" "72"
## [73] "73" "74" "75" "76" "77" "78" "79" "80" "81" "82" "83" "84"
## [85] "85" "86" "87" "88" "89" "90" "91" "92" "93" "94" "95" "96"
## [97] "97" "98" "99" "100" "101" "102" "103" "104" "105" "106" "107" "108"
## [109] "109" "110" "111" "112" "113" "114" "115" "116" "117" "118" "119" "120"
## [121] "121" "122" "123" "124" "125" "126" "127" "128" "129" "130" "131" "132"
## [133] "133" "134" "135" "136" "137" "138" "139" "140" "141" "142" "143" "144"
## [145] "145" "146" "147" "148" "149" "150" "151" "152" "153" "154" "155" "156"
## [157] "157" "158" "159" "160" "161" "162" "163" "164" "165" "166" "167" "168"
## [169] "169" "170" "171" "172" "173" "174" "175" "176" "177" "178" "179" "180"
## [181] "181" "182" "183" "184" "185" "186" "187" "188" "189" "190" "191" "192"
## [193] "193" "194" "195" "196"
```

## "nrow"

"nrow": número de filas.

# Número de filas

```
nrow(df)
## [1] 196
```

## "ncol"

"ncol": número de columnas.

```
# Número de columnas
```

```
ncol(df)
```

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

## "head"

"head": primeras n posiciones del *dataframe*.

```
# Primeras posiciones de un dataframe, por defecto
```

```
head(df)
```

```
##   Date Open High Low Close Volume Adj.Close
## 1 2012-05-18 42.05 45.00 38.00 38.23 573576400 38.23
## 2 2012-05-21 36.53 36.66 33.00 34.03 168192700 34.03
## 3 2012-05-22 32.61 33.59 30.94 31.00 101786600 31.00
## 4 2012-05-23 31.37 32.50 31.36 32.00 73600000 32.00
## 5 2012-05-24 32.95 33.21 31.77 33.03 50237200 33.03
## 6 2012-05-25 32.90 32.95 31.11 31.91 37149800 31.91
```

```
# Primeras 10 posiciones
```

```
head(df, 10)
```

```
##   Date Open High Low Close Volume Adj.Close
## 1 2012-05-18 42.05 45.00 38.00 38.23 573576400 38.23
## 2 2012-05-21 36.53 36.66 33.00 34.03 168192700 34.03
## 3 2012-05-22 32.61 33.59 30.94 31.00 101786600 31.00
## 4 2012-05-23 31.37 32.50 31.36 32.00 73600000 32.00
## 5 2012-05-24 32.95 33.21 31.77 33.03 50237200 33.03
## 6 2012-05-25 32.90 32.95 31.11 31.91 37149800 31.91
## 7 2012-05-29 31.48 31.69 28.65 28.84 78063400 28.84
## 8 2012-05-30 28.70 29.55 27.86 28.19 57267900 28.19
## 9 2012-05-31 28.55 29.67 26.83 29.60 111639200 29.60
## 10 2012-06-01 28.89 29.15 27.39 27.72 41855500 27.72
```

## "tail"

"tail": últimas n posiciones del *dataframe*.

*# Últimas posiciones de un dataframe, por defecto.*

**tail(df)**

```
##      Date Open High Low Close Volume Adj.Close
## 191 2013-02-22 27.62 27.63 26.82 27.13 36350200 27.13
## 192 2013-02-25 27.16 27.64 27.15 27.27 34652000 27.27
## 193 2013-02-26 27.36 27.46 26.70 27.39 31611700 27.39
## 194 2013-02-27 27.34 27.34 26.63 26.87 44319700 26.87
## 195 2013-02-28 26.84 27.30 26.34 27.25 83027800 27.25
## 196 2013-03-01 27.05 28.12 26.81 27.78 54064800 27.78
```

*# Últimas 4 posiciones*

**tail(df, 4)**

```
##      Date Open High Low Close Volume Adj.Close
## 193 2013-02-26 27.36 27.46 26.70 27.39 31611700 27.39
## 194 2013-02-27 27.34 27.34 26.63 26.87 44319700 26.87
## 195 2013-02-28 26.84 27.30 26.34 27.25 83027800 27.25
## 196 2013-03-01 27.05 28.12 26.81 27.78 54064800 27.78
```

**Finalmente, se muestran algunas de las principales transformaciones que pueden aplicarse a un *dataframe*.**

## "rbind" y "cbind"

Mediante "rbind" se añade una nueva fila. Por su parte, "cbind" añade una nueva columna.

```
df <- rbind(df, c("2013-03-01", 35.12, 39.32, 32.75, 34.65, 57052800, 34.65), stringsAsFactors=T)
```

```
tail(df)
```

```
##      Date Open High Low Close Volume Adj.Close
## 192 2013-02-25 27.16 27.64 27.15 27.27 34652000 27.27
## 193 2013-02-26 27.36 27.46 26.7 27.39 31611700 27.39
## 194 2013-02-27 27.34 27.34 26.63 26.87 44319700 26.87
## 195 2013-02-28 26.84 27.3 26.34 27.25 83027800 27.25
## 196 2013-03-01 27.05 28.12 26.81 27.78 54064800 27.78
## 197 2013-03-01 35.12 39.32 32.75 34.65 57052800 34.65
```

```
index.list <- list(index=1:nrow(df))
```

```
df <- cbind(df, index.list)
```

```
head(df)
```

```
##      Date Open High Low Close Volume Adj.Close index
## 1 2012-05-18 42.05 45 38 38.23 573576400 38.23 1
## 2 2012-05-21 36.53 36.66 33 34.03 168192700 34.03 2
## 3 2012-05-22 32.61 33.59 30.94 31 101786600 31 3
## 4 2012-05-23 31.37 32.5 31.36 32 73600000 32 4
## 5 2012-05-24 32.95 33.21 31.77 33.03 50237200 33.03 5
## 6 2012-05-25 32.9 32.95 31.11 31.91 37149800 31.91 6
```



"\$"

Otra forma de añadir una columna al *dataframe* es asignar el nombre a través del operador “\$” y un vector de la misma longitud.

```
df$Nueva_Columna <- '^'(as.numeric(df$High),2)
```

```
head(df)
```

```
##      Date Open High  Low Close  Volume Adj.Close index Nueva_Columna
## 1 2012-05-18 42.05 45 38 38.23 573576400 38.23 1 2025.000
## 2 2012-05-21 36.53 36.66 33 34.03 168192700 34.03 2 1343.956
## 3 2012-05-22 32.61 33.59 30.94 31 101786600 31 3 1128.288
## 4 2012-05-23 31.37 32.5 31.36 32 73600000 32 4 1056.250
## 5 2012-05-24 32.95 33.21 31.77 33.03 50237200 33.03 5 1102.904
## 6 2012-05-25 32.9 32.95 31.11 31.91 37149800 31.91 6 1085.703
```

## "NULL"

Si se desea eliminar una variable, se aplicará "NULL" a esta:

*# Borramos la columna*

df\$index <- **NULL**

## "subset" y "select"

Para obtener un subconjunto del dataframe, empleamos “subset”. Para seleccionar las columnas a tener en cuenta, se aplicará el parámetro “select”.

```
muestra <- subset(df, select = c("High", "Low"))
```

```
muestra
```

```
##   High Low
## 1 45 38
## 2 36.66 33
## 3 33.59 30.94
## 4 32.5 31.36
## 5 33.21 31.77
## 6 32.95 31.11
## 7 31.69 28.65
## 8 29.55 27.86
## 9 29.67 26.83
## 10 29.15 27.39
## 11 27.65 26.44
## 12 27.76 25.75
## 13 27.17 25.52
## 14 27.35 26.15
```

## "order"

Para ordenar un *dataframe*, se usa función "order".

```
ordered <- df[order(df$Open, decreasing = T), ]
```

```
head(ordered)
```

```
##      Date Open High Low Close Volume Adj.Close Nueva_Columna
## 1 2012-05-18 42.05 45 38 38.23 573576400 38.23 2025.000
## 2 2012-05-21 36.53 36.66 33 34.03 168192700 34.03 1343.956
## 197 2013-03-01 35.12 39.32 32.75 34.65 57052800 34.65 1546.062
## 5 2012-05-24 32.95 33.21 31.77 33.03 50237200 33.03 1102.904
## 6 2012-05-25 32.9 32.95 31.11 31.91 37149800 31.91 1085.703
## 26 2012-06-25 32.86 33.02 31.55 32.06 24352900 32.06 1090.320
```

## "names"

Si mediante la función “names” accedemos a los nombres de las variables, es posible renombrar las columnas

# Renombramos una columna

```
names(df)[2] <- "Precio-Aertura"
```

```
names(df)
```

```
## [1] "Date"      "Precio-Aertura" "High"       "Low"  
## [5] "Close"     "Volume"     "Adj.Close"   "Nueva_Columna"
```

## as.data.frame.

Con la función “`as.data.frame`” se pueden convertir vectores, matrices, listas y *arrays* como *dataframes*.

*# Tomamos un array anterior, importante, estudiar cómo se distribuyen las dimensiones.*

```
new.df <- as.data.frame(new.arr)
```

new.df

```
## V1 V2 V3 V4 V5 V6 V7 V8
## 1 1 6 11 16 21 26 31 36
## 2 2 7 12 17 22 27 32 37
## 3 3 8 13 18 23 28 33 38
## 4 4 9 14 19 24 29 34 39
## 5 5 10 15 20 25 30 35 40
```

Finalmente, es importante cómo gestionar los nulos en un *dataframe*.

| VALOR NULO | "SUMMARY" | "IS.NA" | "COMPLETE.CASES"<br>" | BOF |
|------------|-----------|---------|-----------------------|-----|
|            |           |         |                       |     |

Primero, se tomará una observación como valor nulo.

```
new.df[3, 1] <- NA
```

| VALOR NULO | "SUMMARY" | "IS.NA" | "COMPLETE.CASES" | BOR |
|------------|-----------|---------|------------------|-----|
|            |           |         | ""               |     |

Una forma inicial de comprobar si hay nulos en un *dataframe*, es a través del comando "summary".

```
summary(new.df) # Aparece un nulo en V1
## V1 V2 V3 V4 V5
## Min. :1.00 Min. : 6 Min. :11 Min. :16 Min. :21
## 1st Qu.:1.75 1st Qu.: 7 1st Qu.:12 1st Qu.:17 1st Qu.:22
## Median :3.00 Median : 8 Median :13 Median :18 Median :23
## Mean :3.00 Mean : 8 Mean :13 Mean :18 Mean :23
## 3rd Qu.:4.25 3rd Qu.: 9 3rd Qu.:14 3rd Qu.:19 3rd Qu.:24
## Max. :5.00 Max. :10 Max. :15 Max. :20 Max. :25
## NA's :1
## V6 V7 V8
## Min. :26 Min. :31 Min. :36
## 1st Qu.:27 1st Qu.:32 1st Qu.:37
## Median :28 Median :33 Median :38
## Mean :28 Mean :33 Mean :38
## 3rd Qu.:29 3rd Qu.:34 3rd Qu.:39
## Max. :30 Max. :35 Max. :40
```

| VALOR NULO | "SUMMARY" | "IS.NA" | "COMPLETE.CASES" | BOR |
|------------|-----------|---------|------------------|-----|
|            |           |         | ""               |     |

Otra manera es utilizar la función "is.na".

```
is.na(new.df)
```

```

##   V1 V2 V3 V4 V5 V6 V7 V8
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [2] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [3] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [4] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [5] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

```

|            |           |         |                       |     |
|------------|-----------|---------|-----------------------|-----|
| VALOR NULO | "SUMMARY" | "IS.NA" | "COMPLETE.CASES"<br>" | BOR |
|------------|-----------|---------|-----------------------|-----|

Pueden detectarse las filas en las que hay nulos a través de "complete.cases".

```

complete.cases(new.df)
## [1] TRUE TRUE FALSE TRUE TRUE

```

|            |           |         |                       |     |
|------------|-----------|---------|-----------------------|-----|
| VALOR NULO | "SUMMARY" | "IS.NA" | "COMPLETE.CASES"<br>" | BOR |
|------------|-----------|---------|-----------------------|-----|

Para borrar los casos nulos:

```

new.df <- new.df[complete.cases(new.df),]

new.df
##  V1 V2 V3 V4 V5 V6 V7 V8
## 1 16 11 16 21 26 31 36
## 2 27 12 17 22 27 32 37
## 4 49 14 19 24 29 34 39
## 5 10 15 20 25 30 35 40

```

| VALOR NULO | "SUMMARY" | "IS.NA" | "COMPLETE.CASES"<br>" | BOR |
|------------|-----------|---------|-----------------------|-----|
|            |           |         |                       |     |

Si, por el contrario, interesa reemplazar el valor nulo por otro valor:

```
new.df <- as.data.frame(new.arr)
new.df[3, 2] <- NA

new.df$V2[which(is.na(new.df$V2))] <- 1000

new.df
## V1 V2 V3 V4 V5 V6 V7 V8
## 1 16 11 16 21 26 31 36
## 2 27 12 17 22 27 32 37
## 3 3 1000 13 18 23 28 33 38
## 4 4 9 14 19 24 29 34 39
## 5 5 10 15 20 25 30 35 40
```

## VI. Herramientas de control de flujo

---

En cualquier lenguaje de programación, es básico **garantizar un correcto funcionamiento de un programa**; concretamente, cuando se trabaja en ciencia de datos, hay que **ejecutar scripts que no contengan errores y que garanticen el correcto funcionamiento de todos los procesos que estén involucrados en cuanto al dato**.

Para el correcto funcionamiento de un programa, se utilizan herramientas de control de flujo. A lo largo de esta sección, se expondrán algunas de las más importantes, que son:

- Condicionales
- Bucles

Se recomienda seguir esta sección junto con el notebook:



**Herramientas\_control.zip**

220.7 KB



**Herramientas\_control\_Rmd.zip**

2.1 KB



**CONTINUAR**

## 6.1. Condicionales

Las sentencias condicionales **se emplean para derivar la ejecución de un programa hacia una acción u otra diferente** en función de una expresión booleana. Esto, traducido a un lenguaje de programación, se basa en lo siguiente:

Si (expresión booleana)

Realizar una acción

Si no (expresión booleana):

Realizar otra acción

Para aplicar sentencias condicionales, R dispone de los comandos “if” y “else”. A continuación se muestran dos ejemplos:

## Primer ejemplo

Se propondrá un primer ejemplo simple: si un número es mayor que cinco lo multiplicamos por dos; si no, lo dividimos.

```
number <- 6

if (number > 5){
  print(number * 2)
} else {
  print(number / 2)
}
## [1] 12
```

## Segundo ejemplo

Otro ejemplo: dado un vector, que recoja su ítem máximo, si su valor máximo es el doble o más grande que el valor mínimo, mostramos el valor mínimo al cuadrado; si no, elevamos al cuadrado el valor máximo.

```
# Creamos un vector
my.vec <- c(3, 4, 9, 10, 2, 20)

if (max(my.vec) >= (2 * min(my.vec))) {

# Elevamos al cuadrado el valor mínimo
print(min(my.vec)^2)
```

```
 } else {  
  
 # Elevamos al cuadrado el valor máximo  
 print(max(my.vec)^2)  
  
}  
## [1] 4
```

En muchas ocasiones, **es probable que existan múltiples restricciones para la realización de un programa**; esto nos obligará a subdividir los bloques de condicionales en “if-else” anidados; en otros casos “if-else apilados”, se mostrará primero un ejemplo de condicional anidado.

## **Ejemplos de "if-else" anidado y apilado**

## "if–else" anidado

Dado un número, si es mayor que 10, si es par, devolver "True"; si no, "False"; y, si no es mayor que 10, dividir el número entre dos.

# Tomamos un número

a <- 13

**if** (a > 10) {

**if** (a %% 2 == 0) {

**print**(TRUE)

} **else** {

**print**(FALSE)

}

} **else** {

**print**(a/2)

}

## [1] FALSE

a <- 12

**if** (a > 10) {

**if** (a %% 2 == 0) {

```
print(TRUE)
```

```
} else {
```

```
print(FALSE)
```

```
}
```

```
} else {
```

```
print(a/2)
```

```
}
```

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

## "if–else" apilado

Dado un vector, tomar el valor máximo. Si es impar, elevar el valor al cubo; si no, si el valor mínimo del vector es mayor a 10, devolvemos la suma del vector; si no, mostramos el producto del vector.

```
vec <- c(3, 5, 13, 9, 5, 7, 8)
```

```
if (max(vec) %% 2 == 0) {
```

```
  print(max(vec)^3)
```

```
} else if (min(vec) > 10) {
```

```
  print(sum(vec))
```

```
} else {
```

```
  print(prod(vec))
```

```
}
```

```
## [1] 491400
```

CONTINUAR

## 6.2. Bucles

Algo muy habitual cuando se trabaja con estructuras de datos es tener que recorrer iterativamente sus elementos. Para realizar esta acción, en R podemos elegir los siguientes tipos de bucles:

### "for"

"for": recorre una secuencia y realiza una acción en cada elemento de esta.

```
for (i in 1:10){  
  # Elevamos todos los valores al cubo  
  print(i^3)  
}  
## [1] 1  
## [1] 8  
## [1] 27  
## [1] 64  
## [1] 125  
## [1] 216  
## [1] 343  
## [1] 512  
## [1] 729  
## [1] 1000  
my.mat <- matrix(1:30, nrow=3, ncol=3)  
  
# Multiplicamos cada valor de fila por el valor de columna  
  
# Recorremos filas  
for (fila in 1:dim(my.mat)[1]){  
  # Recorremos columnas  
  for (columna in 1:dim(my.mat)[2]){  
    cat('Elemento: ', fila, ', ', columna, ' -> ', fila*columna, '\n')  
  }  
}  
## Elemento: 1 1 -> 1  
## Elemento: 1 2 -> 2  
## Elemento: 1 3 -> 3  
## Elemento: 2 1 -> 2  
## Elemento: 2 2 -> 4  
## Elemento: 2 3 -> 6
```

```
## Elemento: 3 1 --> 3  
## Elemento: 3 2 --> 6  
## Elemento: 3 3 --> 9
```

## "while"

"while": se necesita una expresión booleana. Mientras dicha expresión se cumpla, se realiza la misma acción. Hasta que no se cumpla dicha expresión booleana, esto supondrá una acción de parada.

```
i <- 1
```

*# Mientras el valor sea menor que 6, sumamos un valor*

```
while (i < 6) {  
  print(i)  
  i <- i+1  
}  
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
x <- c(100, 2, 3, 900, 4)
```

*# Recorremos ítem a ítem un vector*

```
index <- 1  
while (index <= length(x)){  
  # Multiplicamos cada valor del vector por el valor de índice  
  print(x[index] * index)  
  index <- index + 1  
}  
## [1] 100  
## [1] 4  
## [1] 9  
## [1] 3600  
## [1] 20
```

## "repeat"

"repeat": repite constantemente una acción hasta que se cumple una condición de parada. Esta condición de parada debe ser una expresión booleana. Cuando esta condición se cumpla, se finalizará la ejecución del bucle con "break", que es aplicable al resto de bucles.

```
x <- 1

repeat {
  print(x)
  x <- x+1
  if (x == 6){
    break
  }
}
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

## "next"

En algunas ocasiones, cuando una expresión booleana se cumpla, simplemente no habrá que hacer nada más que pasar a la siguiente iteración. Para este propósito existe la operación "next".

```
v <- c(1, 20, 3, 4, 7, 9, 100, 133, 1000)
```

```
for (item in 1:length(v)){
  if (v[item] %% 2 == 0){
    next
  } else {
    print(item * v[item])
  }
}
## [1] 1
## [1] 9
## [1] 35
```

```
## [1] 54  
## [1] 1064
```

## VII. Funciones

---

Cuando una compañía realiza procesos que serán llevados a producción, uno de los elementos más importantes es que sus programas estén modularizados con funciones; de esta manera, se mejora la legibilidad del código, la velocidad, la reusabilidad y aumenta la tolerancia a fallos del mismo.

Se recomienda seguir esta sección a través del siguiente *notebook*:



**Funciones.zip**

221.8 KB



**Funciones\_Rmd.zip**

2.5 KB



En R, para declarar una función se utiliza el comando **"function"**. Como en cualquier lenguaje de programación, las funciones pueden recibir parámetros de entrada y parámetros de salida. Se describirán algunos ejemplos para comprender estas dos metodologías.

**"function"**

En primer lugar, se realizará una función simple que reciba como parámetro de entrada un número y lo devuelva elevado al cubo. Para devolver cualquier valor de una función, será necesario utilizar la palabra reservada “return”:

```
# Creamos la función
exp.test <- function(number){
  result <- number ^ 3
  return (result)
}
```

```
# Llamamos a la función
a <- exp.test(number = 3)
a
## [1] 27
```

Si se realiza una modificación en el ejemplo anterior, es posible codificarlo para que el programa reciba dos números y devuelva como resultado el primer número elevado al segundo.

```
# Creamos la función
exp.test <- function(number, pow){
  result <- number ^ pow
  return (result)
}
```

```
# Llamamos a la función
a <- exp.test(number = 3, pow = 4)
a
## [1] 81
```

Por lo general, R está preparado para devolver un solo valor en una función, no obstante, si se devuelve una estructura de datos como, por ejemplo, una lista, posteriormente se podrá acceder por separado a sus ítems.

## Ejemplo

En el siguiente ejemplo, se desarrollará una función que reciba como parámetro de entrada un vector y devuelva su valor máximo y mínimo:

```
min.max <- function(v){  
  # Inicializamos una lista vacía.  
  my.list <- list()  
  
  my.list$min <- min(v)  
  my.list$max <- max(v)  
  
  return (my.list)  
}  
  
results <- min.max(v = c(100, 20, 34, 8))  
  
results$min  
## [1] 8  
results$max  
## [1] 100
```

Existe una familia de funciones que sirven para automatizar una función sobre una secuencia de valores. De esta manera, y omitiendo el uso de bucles, estas funciones son **todas derivadas de la función principal "apply"**. Se mostrará un ejemplo de cada tipo de función:

**Derivadas de la función principal "apply"**

## "apply"

"apply": realiza una misma operación sobre los ejes, por ejemplo: supongamos que se quiere obtener la suma de las columnas; la función "apply" recibirá como parámetros, los datos, la dirección (filas o columnas) y la función a aplicar.

```
m <- matrix(c(1:10), nrow = 5, ncol = 2)
```

```
m
```

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

```
## [1,] 1 6
```

```
## [2,] 2 7
```

```
## [3,] 3 8
```

```
## [4,] 4 9
```

```
## [5,] 5 10
```

```
apply(m, 2, sum)
```

```
## [1] 15 40
```

## "lapply"

"lapply": misma función que "apply", pero su funcionamiento está optimizado en listas. Devuelve el resultado también como una lista.

```
v <- matrix(1:10, ncol = 2, nrow = 5)
```

```
v.2 <- matrix(5:15, ncol = 2, nrow = 5)
```

```
## Warning in matrix(5:15, ncol = 2, nrow = 5): data length [11] is not a sub-
## multiple or multiple of the number of rows [5]
```

```
lista = list(v, v.2)
```

```
lista
```

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

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

```
## [1,] 1 6
```

```
## [2,] 2 7
```

```
## [3,] 3 8
```

```
## [4,] 4 9
```

```
## [5,] 5 10
```

```
##
```

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

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

```
## [1,] 5 10
```

```
## [2,] 6 11
```

```
## [3,] 7 12
```

```
## [4,] 8 13
```

```
## [5,] 9 14
```

*# Obtenemos la multiplicación*

```
lapply(lista, FUN = prod)
```

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

```
## [1] 3628800
```

```
##
```

```
## [[2]]  
## [1] 3632428800
```

## "sapply"

"sapply": recibe una lista, aplica una función y devuelve un vector.

```
sapply(lista, sum)
```

```
## [1] 55 95
```

## "tapply"

"tapply": realiza una operación en función de un vector de factores, operación aconsejable para *dataframes*.

```
v.num <- c(10:20)
v факт <- c("PAR", "IMPAR", "PAR", "IMPAR", "PAR", "IMPAR",
"PAR", "IMPAR", "PAR", "IMPAR", "PAR")

tapply(v.num, v. факт, sqrt)

## $IMPAR
## [1] 3.316625 3.605551 3.872983 4.123106 4.358899
##
## $PAR
## [1] 3.162278 3.464102 3.741657 4.000000 4.242641 4.472136

tapply(v.num, v. факт, sum)

## IMPAR PAR
## 75 90
```

## "mapply"

"mapply": opera entre matrices o vectores y devuelve el resultado como una lista o vector. Esto depende del número de iteraciones necesarias.

```
mapply(sum, v, v.2, 1)  
## [1] 7 9 11 13 15 17 19 21 23 25  
  
mapply(rep, 1:5, 5:1)  
  
## [[1]]  
## [1] 1 1 1 1 1  
##  
## [[2]]  
## [1] 2 2 2 2  
##  
## [[3]]  
## [1] 3 3 3  
##  
## [[4]]  
## [1] 4 4  
##  
## [[5]]  
## [1] 5
```

Se ha visto en secciones anteriores que R dispone de una amplia variedad de funciones ya predefinidas que ahorran trabajo a la hora de procesar datos. En este punto, se mostrarán algunas de ellas. Es muy importante que, siempre que quiera realizarse un tipo de acción específica, se investigue si R ya posee una función propia, ya que estará optimizada y, por supuesto, ahorrará trabajo.



## "is.character"

"is.character": devuelve un booleano en función de si la variable es o no carácter.

```
a <- "holá"
```

```
is.character(a)
```

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

```
b <- 5
```

```
is.character(b)
```

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

## "is.factor"

"is.factor": devuelve un booleano en función de si la variable es o no categórica.

```
a <- "hola"  
b <- as.factor("hola")  
  
is.factor(a)  
  
## [1] FALSE  
  
is.factor(b)  
  
## [1] TRUE
```

## "is.numeric"

"is.numeric": devuelve un booleano en función de si la variable es numérica o no.

```
a <- 5
```

```
b <- 3.0
```

```
c <- "a"
```

```
is.numeric(a)
```

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

```
is.numeric(b)
```

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

```
is.numeric(c)
```

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

## "as.character"

"as.character": transforma una variable categórica o numérica a carácter.

```
a <- factor(c("UNO", "DOS"))
```

```
levels(a)
```

```
## [1] "DOS" "UNO"
```

```
class(a)
```

```
## [1] "factor"
```

```
new.a <- as.character(a)
```

```
class(new.a)
```

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

```
as.character(3.23)
```

```
## [1] "3.23"
```

```
as.character(5)
```

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

## "as.numeric"

"as.numeric": transforma en variable numérica una variable categórica.

```
a <- factor(c("POSITIVE", "NEGATIVE"))
levels(a)

## [1] "NEGATIVE" "POSITIVE"

as.numeric(a)

## [1] 2 1
```

También es posible transformar en numéricas cadenas de texto si son explícitamente números.

```
as.numeric("58.45")

## [1] 58.45
```

## "as.factor"

"as.factor": permite transformar a factor un vector, ya sea numérico o de caracteres.

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

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

```
as.factor(a)
```

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

```
## Levels: 1 2
```

```
as.factor(b)
```

```
## [1] hola adios hola hola adios
```

```
## Levels: adios hola
```

## "table"

"table": muestra la frecuencia de los valores únicos de una variable.

**table(b)**

```
## b  
## adios hola  
## 2 3
```

**table(a)**

```
## a  
## 1 2  
## 2 2
```

## "unique"

"unique": muestra los valores únicos de una variable.

```
y <- c(1, 2, 3, 5, 9, 1, 2, 3, 9, 0)
```

```
unique(y)
```

```
## [1] 1 2 3 5 9 0
```

## "set.seed"

"set.seed": recibe como parámetro un número entero. Replica siempre el mismo resultado (muy recomendable cuando trabajamos con aleatorios).

```
set.seed(777)
```

```
sample(1:10, replace=T)
```

```
## [1] 8 1 10 3 10 9 9 4 4 10
```

```
sample(1:10, replace=T)
```

```
## [1] 7 6 5 7 8 10 2 7 5 6
```

```
sample(1:10, replace=T)
```

```
## [1] 8 2 9 1 7 2 10 4 1 1
```

## "data.frame"

"data.frame": permite crear de cero un *dataframe* a través de vectores o matrices.

```
df <- data.frame("ventas_q1" = v[,1], "ventas_q2" = v[,2],  
"ventas_q3" = v[,3], "ventas_q4" = v[,4])
```

```
df
```

```
## ventas_q1 ventas_q2 ventas_q3 ventas_q4  
## 1 1 6 5 10  
## 2 2 7 6 11  
## 3 3 8 7 12  
## 4 4 9 8 13  
## 5 5 10 9 14
```

```
str(df)
```

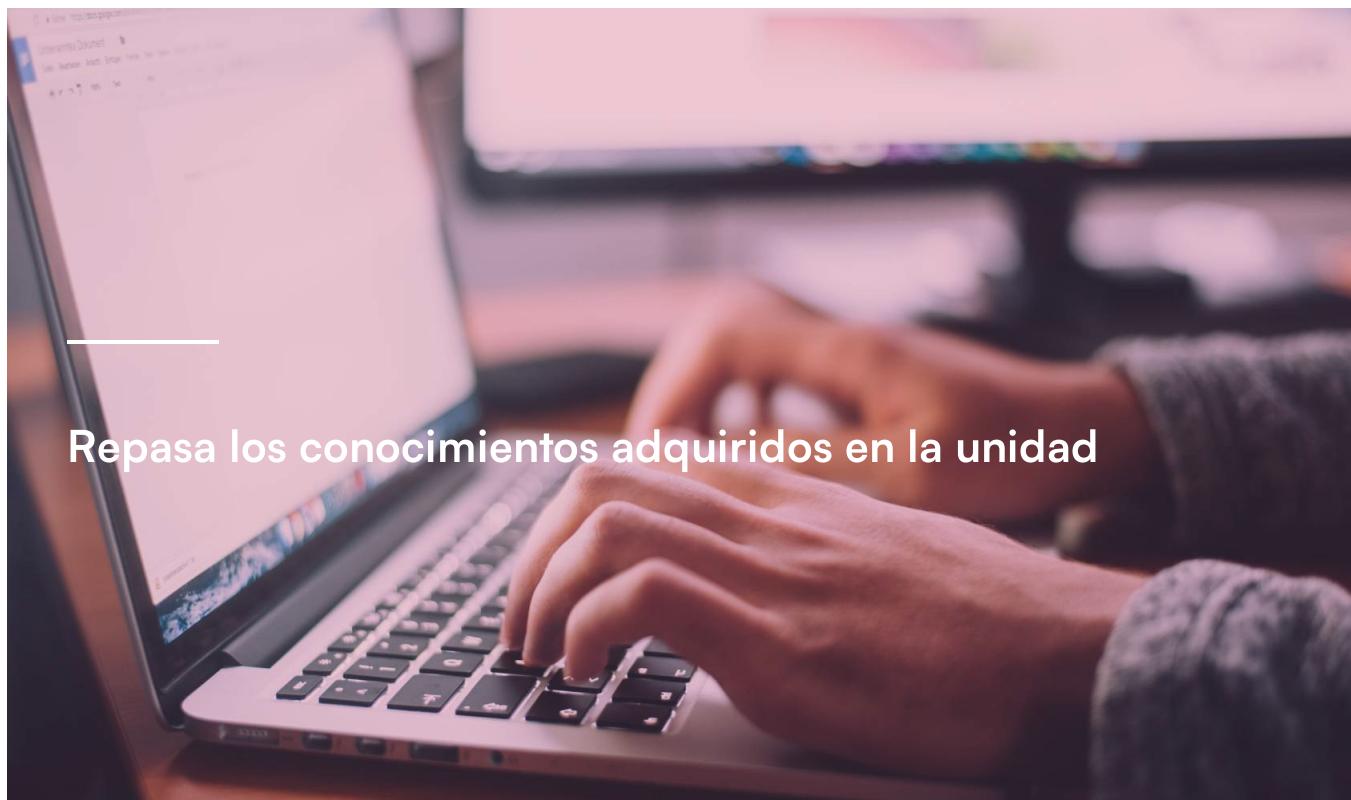
```
## 'data.frame': 5 obs. of 4 variables:  
## $ ventas_q1: int 1 2 3 4 5  
## $ ventas_q2: int 6 7 8 9 10  
## $ ventas_q3: int 5 6 7 8 9  
## $ ventas_q4: int 10 11 12 13 14
```

```
names(df)
```

```
## [1] "ventas_q1" "ventas_q2" "ventas_q3" "ventas_q4"
```

## VIII. Resumen

---



Repasa los conocimientos adquiridos en la unidad

### R

En esta primera unidad de R, el alumno ha estudiado sus **principales características** y descubierto su **comunidad de desarrolladores y su repositorio de paquetes**. También, ha sabido descargar y configurar el IDE R Studio, previa instalación de R base, para comenzar sus tareas en programación estadística.

## Sintaxis de R

El alumno ha aprendido **los principales elementos que intervienen en la sintaxis de R**, como son la declaración de variables, qué tipo de variables y datos existen, cómo realizar anotaciones en Markdown, cómo agregar comentarios y los principales tipos de operadores en R.

## Funciones

También ha aprendido a **definir nuevas funciones**, cómo utilizar las funciones de los paquetes y las diferentes funciones de “apply” en función del tipo de datos que se utilice, con el objetivo principal de optimizar operaciones en filas y columnas en las diferentes estructuras de datos, lo que ahorra en uso de bucles.

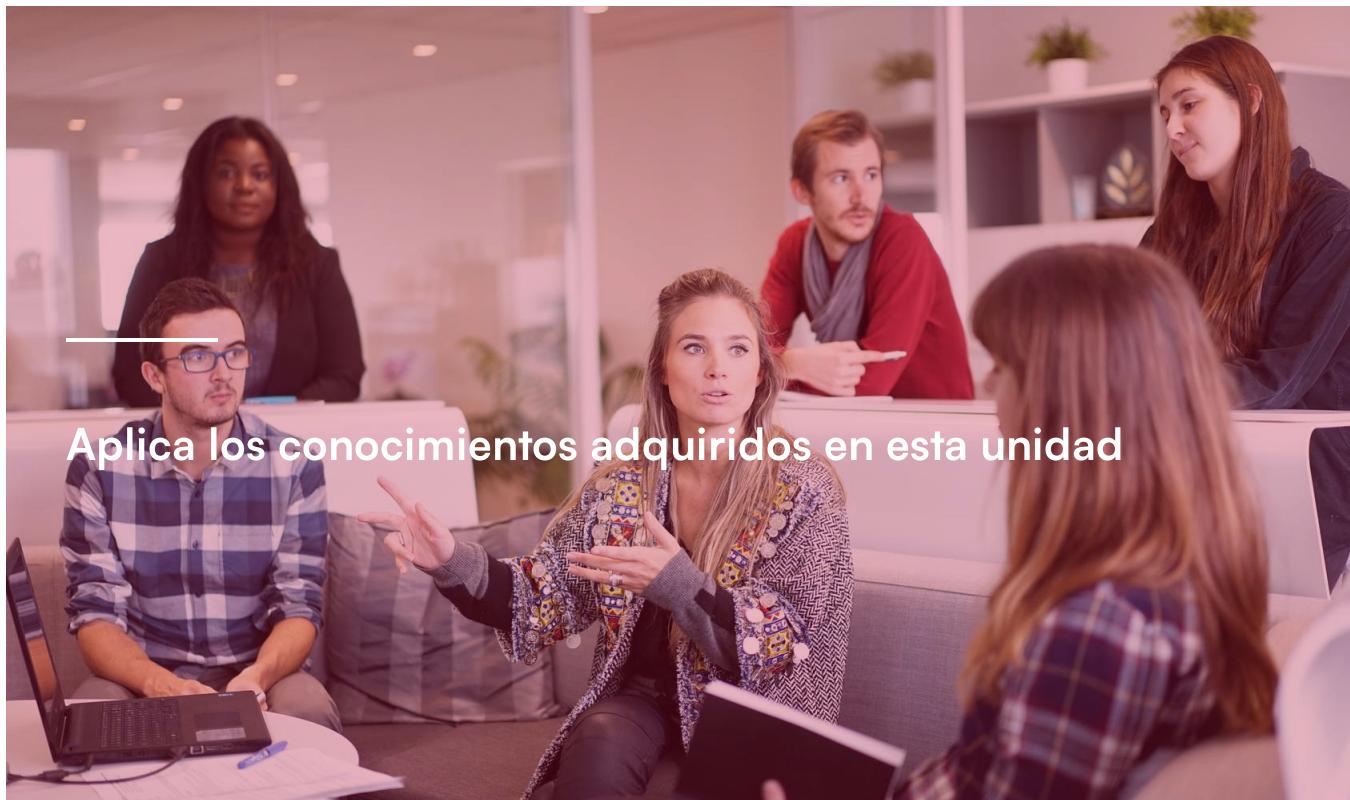
## Flujo de un programa

Por último, se ha entrenado en el control de flujo de un programa a través de estructuras condicionales “if-else” y los **elementos iterativos** como el bucle “for”, “while” y “repeat”.



## IX. Caso práctico con solución

---



### ENUNCIADO

Resuelve los siguientes ejercicios.

## CASO PRÁCTICO 1

1. Crea un vector con la secuencia 1, 2, 3, 4, 5 de longitud 10.
2. Crea un array de dos dimensiones, de 10 elementos cada una, con los 20 primeros números pares del 0 al 40.
3. Crea una muestra de números aleatorios del 1 a 100 con repetición de longitud 10.
4. Crea un nuevo vector que sea el resultado de la suma fila a fila de cada dimensión del array.
5. Finalmente, crea un dataframe con todas las estructuras de datos creadas anteriormente y muestra el resumen estadístico del dataframe.

## CASO PRÁCTICO 2

Mediante los siguientes datos, sobre los que se simulará información aleatoria para los años comprendidos entre 2016 y 2020:

```
vec.1 <- sample(1:100, 100, replace = T)
vec.2 <- sample(50:150, 100, replace = T)
vec.3 <- rep(c("2020", "2019", "2018", "2017", "2016"), 20)
```

1. Crea una matriz de dos columnas con los vectores “vec.1” y “vec.2”.
2. Crea un nuevo vector que sea la suma de las filas de la matriz al cuadrado.
3. Crea un dataframe con todos los vectores con las siguientes columnas:

- vec.1: Low.
  - vec.2: High.
  - Vector del apartado 2: Volume.
  - vec.3: Year (tiene que ser una variable factor).
4. Muestra el resumen estadístico del dataframe.
5. Muestra la estructura de variables del dataframe.
6. Asigna valor nulo (NA) a todos los valores de la columna “Low” por debajo de 20.
7. Muestra el número de valores nulos de cada columna.
8. Muestra la media de la columna “Volume” en función de cada factor de “Year”.
9. Asigna a cada valor “NaN” la media de la columna “Low” (pista: investigar en la función “mean” cómo trabajar con nulos).
10. Obtén un subconjunto de los datos que sean todos los valores de 2020 y que tengan valor superior a 90 en la columna “High”.
11. Investiga sobre la función “write.csv” y crea un archivo CSV con nombre “2020\_Values.csv”.

**VER SOLUCIÓN**

## SOLUCIÓN

Para visualizar la solución de los casos prácticos hay que descargar los siguientes archivos.



**EJERCICIO\_1\_SOLUCION.zip**

219.6 KB



**EJERCICIO\_1\_SOLUCION\_Rmd.zip**

860 B



**EJERCICIO\_2\_SOLUCION.zip**

220.7 KB



**EJERCICIO\_2\_SOLUCION\_Rmd.zip**

1.2 KB





## X. Enlaces de interés

---

Libros gratuitos sobre R disponibles en el CRAN, todos ellos de un nivel básico para que el alumno pueda completar los conocimientos vistos en la asignatura con bibliografía externa.

Venables, W. N. *An introduction to R*; 2020.

[ABRIR ENLACE](#)

Paradis, E. *R for beginners*; 2005.

[ABRIR ENLACE](#)

Manuales oficiales del CRAN de R. Está disponible una versión de manuales en castellano.

Manuales CRAN R.

[ABRIR ENLACE](#)

Recursos adicionales sobre RStudio. En la propia web de RStudio hay una serie de videos que explican el IDE y muestran los principales fundamentos de R. Son perfectos para complementar el estudio de esta unidad.

Vídeos sobre RStudio.

[ABRIR ENLACE](#)

En la página web de RStudio, se recomiendan una serie de libros de todos los niveles, todos gratuitos y que pueden conseguirse online.

Libros RStudio.

[ABRIR ENLACE](#)

Una forma muy interesante de comprender las principales funcionalidades de un paquete es echando un vistazo a sus *cheatsheet*, ya que, de forma muy visual, es posible comprender de manera resumida todas las funciones principales de dicho paquete. RStudio tiene una sección sobre *cheatsheet* sobre muchos de los principales paquetes.

R *Cheatsheets*.

[ABRIR ENLACE](#)



## XI. Glosario

---



**El glosario contiene términos destacados para la comprensión de la unidad**

### Objeto

Un objeto en R es cualquier elemento que se cree en un entorno de trabajo, ya sea una variable, función, etc.

### Matriz

Estructura de datos conformada por dos dimensiones y estructurada en n filas y m columnas

## **Vector**

Estructura de datos más básica de R. Es una sucesión finita de valores. Para declarar un vector se utiliza la función "c()".

## **Factor**

Similar a un vector. Sin embargo, representa a una variable categórica, por lo que todas sus categorías o niveles representan una cualidad de un objeto, por ejemplo: alto, bajo.

## **Nivel**

Es cada una de las categorías de una variable categórica. Podemos acceder a sus valores con la función "levels()".

## **Bucle (repeat)**

Se trata de un bucle en el que, hasta que no se cumple una condición de parada o se encuentra un elemento que fuerce su salida break, realiza constantemente la misma acción.

## Funciones "apply"

Conjunto de funciones destinadas a operar en todo tipo de estructuras de datos, ya sean vectores, listas, matrices, factores, **arrays** o **dataframes**. Sus funciones están optimizadas para iterar sobre los elementos de la estructura de datos que reciba, lo que ahorra el uso de bucles. Pertenecen a esta familia de funciones "Apply", "Sapply", "Tapply", "Lapply" y "Mapply".

## CRAN

**Comprehensive R archive network.** Se trata del repositorio central de la comunidad de R. Dentro de este repositorio se puede encontrar desde el **journal** de R, archivos de instalación de R base y manuales de R, hasta la enorme lista de paquetes desarrollados por la comunidad.

## R Base

Versión más ligera de R que incluye su intérprete. Es un software que cada vez está más en desuso desde la aparición de RStudio.

## R Studio

IDE de programación más utilizado por la comunidad de desarrolladores R. Está estructurado en cuatro zonas principales para la gestión completa y supervisión del desarrollo que realice cualquier usuario.

## CHUNK

Celda de código fuente que podemos implementar en R Notebook y ejecutar su contenido de forma aislada.



## XII. Bibliografía

---

- Daróczi, G. *Mastering data analysis with R*. Pack Publishing; 2015.
- Santana, J. S. *El arte de programar en R. Un lenguaje para la estadística*; 2014.
- Crawley, M. J. *The R book*. John Wiley & Sons, Ltd; 2013.

### Enlaces de interés disponibles en la biblioteca virtual de IMF

Publicación para ampliar sobre las principales estructuras de datos y conocer cómo R trabaja los datos.  
Adquirir conocimientos sobre manipulación de datasets, así como visualización de resultados.

- Johnson, R. *Fundamentals of R programming and statistical analysis*. Pack Publishing; 2017.