



# Módulo 4: Introducción al Machine Learning con R

## Parte I: Introducción al R con RStudio

Autor: Raquel Dormido Canto

Actualizado Enero 2023

## Contenido

1. INTRODUCCIÓN.....	5
1.1. ¿Qué son R y RStudio? .....	5
1.2 Instalación de R y RStudio.....	7
1.2.1. Instalando R en Windows .....	7
1.2.2. Instalando R en GNU/Linux .....	19
1.2.3. Instalando R en Mac OSX.....	19
1.3. Entorno de RStudio.....	20
1.3.1. Pantalla de RStudio.....	20
1.3.2. Primeros pasos con R y RStudio .....	21
1.3.2.1. CONSOLA DE COMANDOS.....	21
1.3.2.2. ALGUNOS DETALLES DE LA INTERFAZ DE RSTUDIO .....	24
1.3.2.3. FUNCIONES DE AYUDA.....	28
1.3.2.4. EN R TRABAJAMOS CON SCRIPTS.....	29
1.3.3. El espacio de trabajo y el directorio de trabajo.....	31
1.3.3.1. EL ESPACIO DE TRABAJO .....	32
1.3.3.2. LA PESTAÑA ENVIRONMENT.....	33
1.3.3.3. EL DIRECTORIO DE TRABAJO.....	33
1.3.4. Barra del Menú Principal .....	35
1.3.4.1. EL MENÚ File.....	35
1.3.4.2. EL MENÚ Edit .....	39
1.3.4.3. EL MENÚ Code .....	41
1.3.4.4. EL MENÚ View .....	41
1.3.4.5. EL MENÚ Plots.....	43
1.3.4.6. EL MENÚ Session .....	44
1.3.4.7. EL MENÚ Build .....	45
1.3.4.8. EL MENÚ Debug .....	45
1.3.4.9. EL MENÚ Profile .....	46
1.3.4.10. EL MENÚ Tools.....	46
1.3.4.11. EL MENÚ Help.....	48

2. INTRODUCCIÓN AL LENGUAJE R .....	51
2.1. Tipos de datos en R.....	51
2.1.1. Definición de tipos.....	51
2.1.2. Conversión de tipos de datos .....	51
2.2. Estructuras de datos en R.....	52
2.2.1. Variables en R.....	52
2.2.1.1. ASIGNACIONES.....	53
2.2.1.2. VISUALIZAR LOS RESULTADOS DE UN CÁLCULO .....	54
2.2.1.3. NOMBRES DE LAS VARIABLES .....	54
2.2.2. Vectores en R.....	55
2.2.2.1. CONSTRUYENDO VECTORES .....	55
2.2.2.2. MANEJO DE VECTORES .....	56
2.2.2.3. ALGUNAS FUNCIONES SOBRE VECTORES .....	59
2.2.2.4. VECTORES DE VARIABLES CUALITATIVAS.....	61
2.2.3. Matrices en R.....	64
2.2.3.1. construyendo MATRICES .....	64
2.2.3.2. MANEJO DE MATRICES .....	66
2.2.3.3. ALGUNAS FUNCIONES SOBRE MATRICES.....	69
2.2.4. Arrays en R.....	73
2.2.4.1. DEFINICIÓN DE ARRAYS .....	73
2.2.4.2. MANEJO DE ARRAYS .....	73
2.2.4.3. ALGUNAS FUNCIONES SOBRE ARRAYS.....	75
2.2.5. Data Frames en R.....	75
2.2.5.1. DEFINICIÓN DE DATA FRAMES.....	75
2.2.5.2. CREACIÓN Y MANEJO DE DATA FRAMES .....	76
2.2.5.3. ALGUNAS FUNCIONES SOBRE DATA FRAMES.....	77
2.2.6. Listas en R .....	79
2.2.6.1. CREACIÓN DE LISTAS.....	79
2.2.6.2. MANEJO DE LISTAS .....	79
2.3. Selección de variables.....	80
2.4. Sintaxis y estructuras de control en R.....	82
2.4.1. Sintaxis.....	82
2.4.2. Estructuras de control .....	83

2.4.2.1. CONDICIONAL: LA ORDEN <b>if</b> .....	83
2.4.2.2. BUCLES: LAS ÓRDENES <b>for</b> , <b>repeat</b> Y <b>while</b> .....	84
2.5. Funciones y paquetes en R .....	85
2.5.1. Funciones.....	85
2.5.1.1. FUNCIONES INCORPORADAS EN R.....	85
2.5.1.2. CREACIÓN DE FUNCIONES .....	86
2.5.2. Packages en R .....	90
2.5.2.1. PACKAGES ESTÁNDAR.....	90
2.5.2.2. INSTALACIÓN DE PACKAGES .....	92
2.6. E/S. Lectura de datos desde distintos tipos de fuentes.....	94
2.6.1. Importando datos de ficheros de texto: .csv y .txt.....	94
2.6.1.1. IMPORTANDO .csv .....	94
2.6.1.2. IMPORTANDO .txt.....	97
2.6.2. Importando datos de ficheros Excel .....	97
2.6.3. Importando datos de ficheros con otros formatos .....	98
2.6.4. Exportando distintos tipos de datos.....	98
2.6.4.1. EXPORTANDO A .CSV .....	99
2.6.4.2. EXPORTANDO A .txt .....	99
2.6.4.3. EXPORTANDO A Excel .....	99
2.6.4.4. EXPORTANDO A OTROS FORMATOS.....	99

# 1. INTRODUCCIÓN

## 1.1. ¿Qué son R y RStudio?

**R** es un proyecto de software libre lo que permite su uso libre y gratuito. Es un potente lenguaje de programación destinado al análisis estadístico y gráfico (representación de datos). Es probablemente el lenguaje más utilizado en la comunidad científica estando su uso extendido, por ejemplo, a la comunidad estadística, la investigación biomédica, la bioinformática o las matemáticas financieras por mencionar algunos campos. Además de estar ampliamente implantado en las universidades, cada vez es más extendido su uso en el mundo empresarial. Se distribuye bajo licencia GNU GPL v2 y está disponible para los sistemas operativos Windows, Mac OS X y Linux.

Entre las principales ventajas de R destacamos las siguientes:

- El ser libre y distribuido bajo licencia GNU implica que lo podemos utilizar y si nos animamos...mejorar.
- Es multiplataforma: hay versiones para Windows, Linux, Mac,....
- Permite analizar cualquier tipo de datos. Es compatible con infinidad de formatos de datos (.csv, .xls, .sav, .sas,...).
- Dispone de un sistema de manejo y almacenamiento de datos eficiente y muy potente.
- Dispone de una gran potencialidad gráfica orientada al análisis de datos. Probablemente no hay otro paquete estadístico que supere la capacidad gráfica de R.
- Dispone de una gran cantidad de funciones integradas en el sistema. Al tratarse de un software libre colaboran muchos usuarios para ampliar estas funciones.
- Dispone de una amplia variedad de librerías o paquetes especializados para estadística y análisis de datos.

En el sitio web <https://www.r-project.org/> (Figura 1.1-1) se encuentra todo el proyecto R.



[Home]

Download

CRAN

R Project

About R

Logo

Contributors

What's New?

Reporting Bugs

Conferences

Search

Get Involved: Mailing Lists

Get Involved: Contributing

Developer Pages

R Blog

R Foundation

Foundation

Board

# The R Project for Statistical Computing

## Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

## News

- [R version 4.2.2 \(Innocent and Trusting\)](#) has been released on 2022-10-31.
- [R version 4.1.3 \(One Push-Up\)](#) was released on 2022-03-10.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the [R Consortium YouTube channel](#).
- You can support the R Foundation with a renewable subscription as a [supporting member](#)

## News via Twitter



Figura 1.1-1: Página oficial de R.

**RStudio** es un IDE (Integrated Development Environment) que permite trabajar con R en multiplataforma (Windows, Linux y Mac). Lo podemos definir como una interfaz que permite acceder de manera sencilla a toda la potencia de R. También se suele definir como un entorno libre y de código abierto para el desarrollo integrado (IDE) de R. Para utilizar RStudio con R se requiere haber instalado R previamente.

Entre las principales características de RStudio citamos las siguientes:

- RStudio, al igual que R, es software libre con licencia GPLv3 y se puede ejecutar sobre distintas plataformas (Windows, Mac, o Linux) o incluso desde la web usando RStudio Server.
- Permite ejecutar código R directamente desde el editor de código fuente. Está pensado como herramienta que permite análisis y desarrollo de datos con R.
- Permite abrir varios *scripts* (ficheros con instrucciones) a la vez y ejecutar trozos de código con solo marcarlos en los *scripts*.
- Nos muestra el *workspace*, el historial, los objetos del *workspace*,...
- Integra la ayuda y la gestión de librerías.

- Proporciona un entorno amigable, tanto para los ya experimentados como para los nuevos usuarios.

La descarga de RStudio se puede realizar desde la página <https://posit.co/downloads/> (ver Figura 1.1-2), desde el enlace de Rstudio Desktop “**DOWNLOAD RSTUDIO**”.

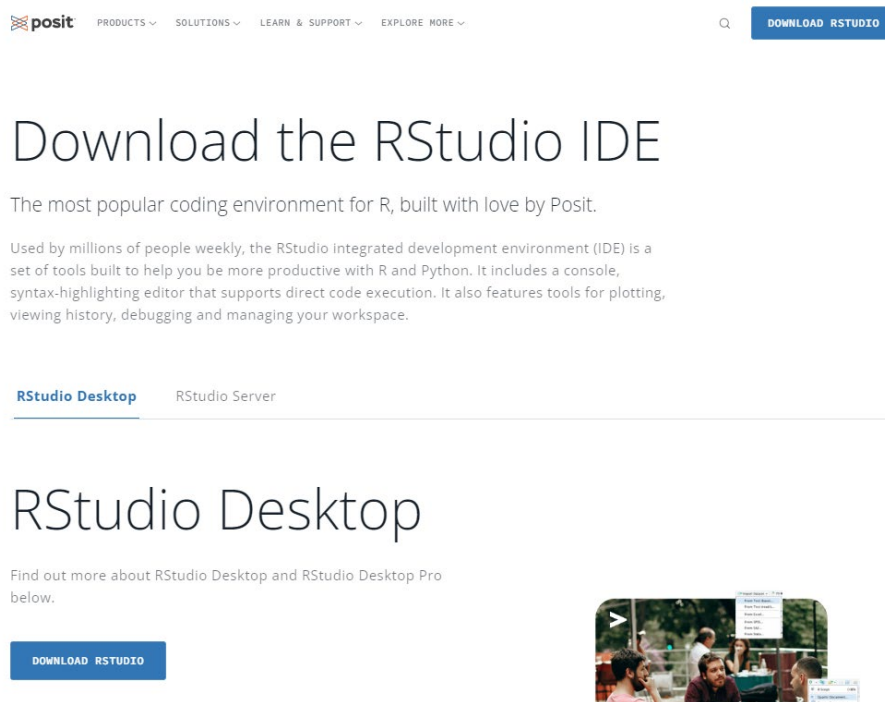


Figura 1.1-2: Página oficial de RStudio.

## 1.2 Instalación de R y RStudio

En Windows, como veremos a continuación se trata de un fichero ejecutable siendo una instalación típica de Windows del tipo siguiente, siguiente, siguiente. En Linux se hace habitualmente ejecutando algunos comandos desde la consola.

Para completar la instalación y configuración de RStudio en nuestro PC hay que instalar R y a continuación RStudio.

### 1.2.1. Instalando R en Windows

R se puede conseguir gratuitamente del Comprehensive R Archive Network (CRAN) que está ubicado en <https://cran.r-project.org/>. Ahí nos podemos descargar la última revisión estable, la R-4.2.2 (<https://cran.r-project.org/bin/windows/base/>). Para instalarlo se pulsa en “Download R 4.2.2 for Windows” y de ahí nos bajamos **R-4.2.2-win.exe**. Haciendo doble clic sobre este fichero comenzamos

la instalación. Saldrá el típico aviso de Windows para permitir que la aplicación haga cambios en el dispositivo. Se selecciona Si.

Después nos solicitará el idioma de instalación (ver Figura 1.2-1).

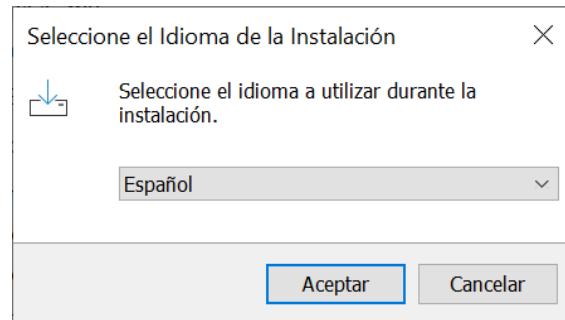


Figura 1.2-1: Selección de idioma en la instalación de R.

A continuación, arranca el Asistente de instalación del R-4.1.2 para windows. El proceso nos informa sobre el tipo de licencia instalación (ver Figura 1.2-2), y pulsamos Siguiente.

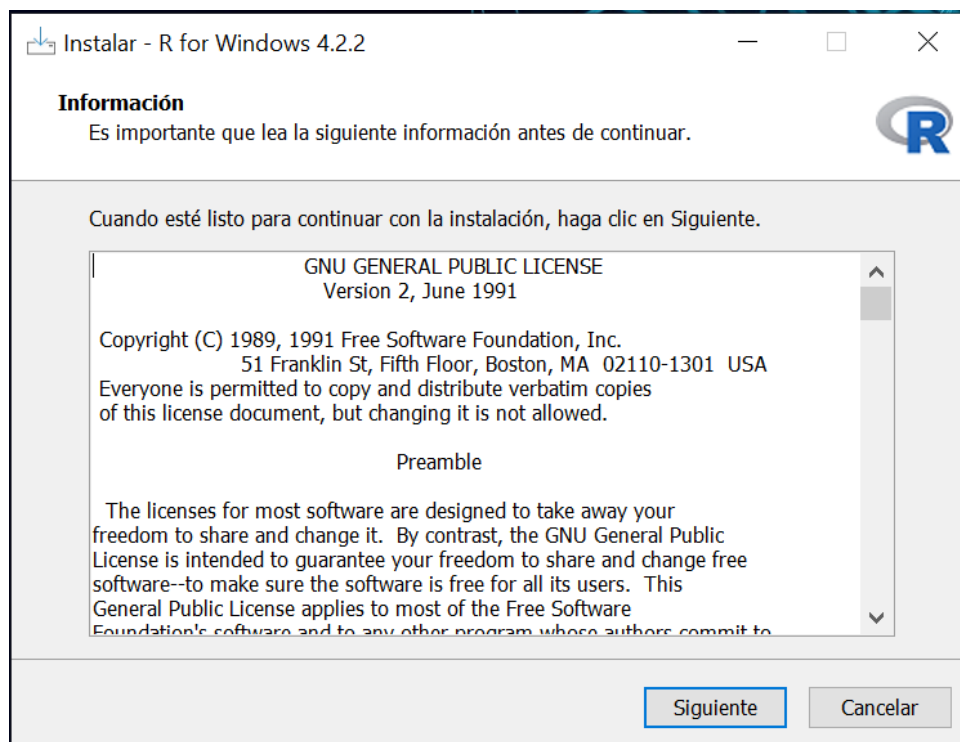


Figura 1.2-2: Licencia de la instalación de R.

A continuación, el proceso de instalación nos indica la ruta de instalación (ver Figura 1.2-3), y pulsamos siguiente.



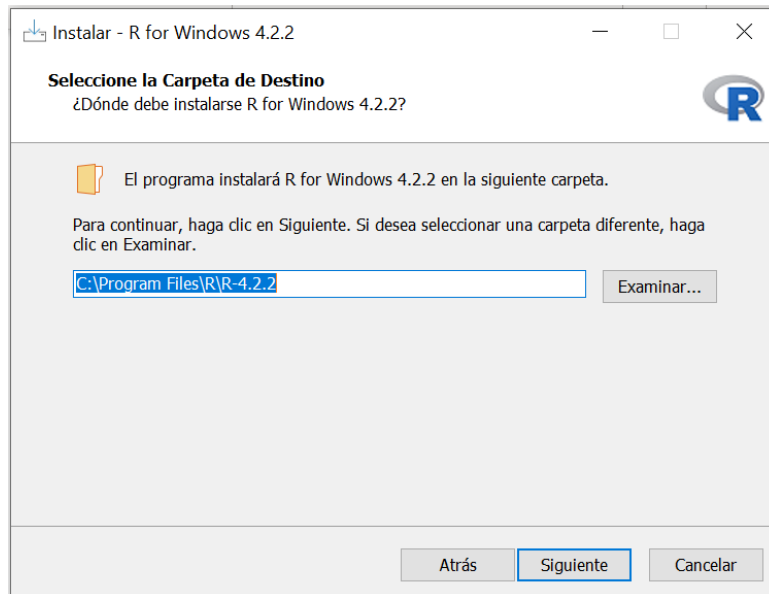


Figura 1.2-3: Selección de ruta de instalación de R.

Después tenemos que seleccionar los paquetes a instalar (ver Figura 1.2-4) y pulsamos siguiente

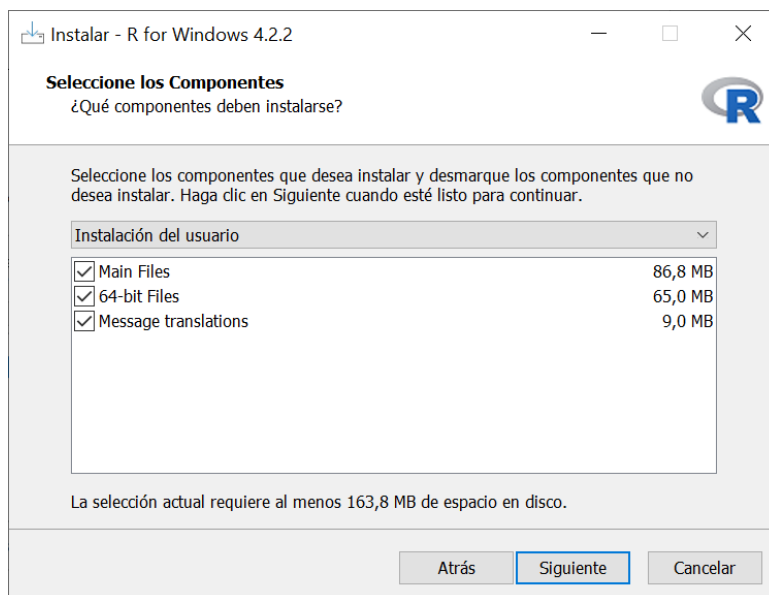


Figura 1.2-4: Selección de idioma en la instalación de R.

Después la instalación nos pregunta si queremos usar las opciones de configuración o no (ver Figura 1.2-5). Por defecto aparece no. Pulsamos siguiente.

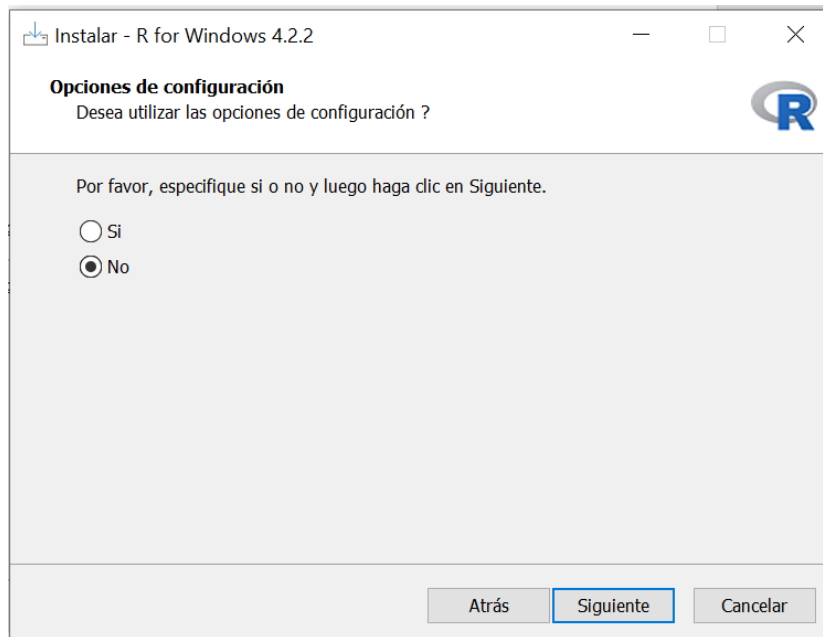


Figura 1.2-5: Selección de Opciones de configuración en la instalación de R.

A continuación, hay que elegir la carpeta del menú inicio donde colocar los accesos directos a los elementos del paquete (ver Figura 1.2-6) y pulsamos siguiente.

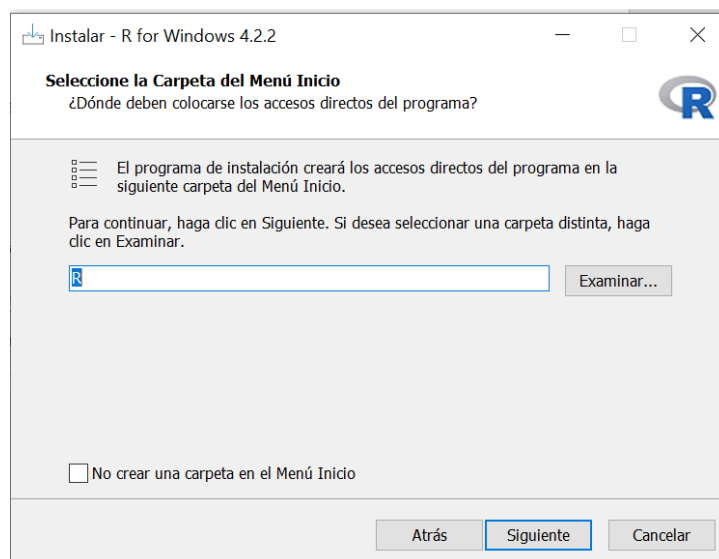


Figura 1.2-6: Selección de la carpeta del Menú Inicio.

Después la instalación pasa por la selección de las tareas adicionales: crear icono en el escritorio, ... se pueden seleccionar por ejemplo las opciones que aparecen seleccionadas en la Figura 1.2-7 y pulsamos siguiente.

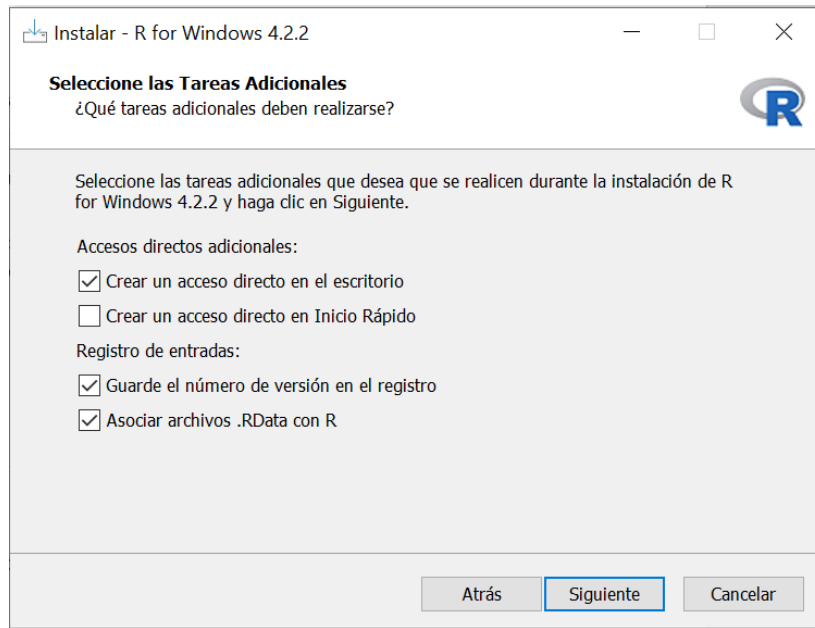


Figura 1.2-7: Selección de Tareas Adicionales en la instalación de R.

En este instante se inicia el desempaquetado e instalación de la aplicación (ver Figura 1.2-8).

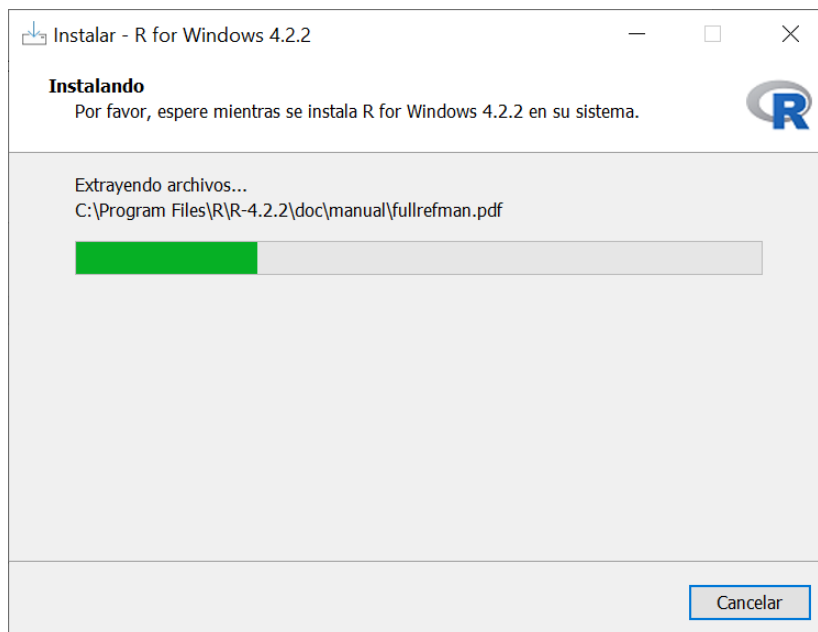


Figura 1.2-8: Desempaquetado e instalación de R.

Una vez finalizado el proceso de instalación sale una ventana indicándolo (ver Figura 1.2-9). Pulsar Finalizar.

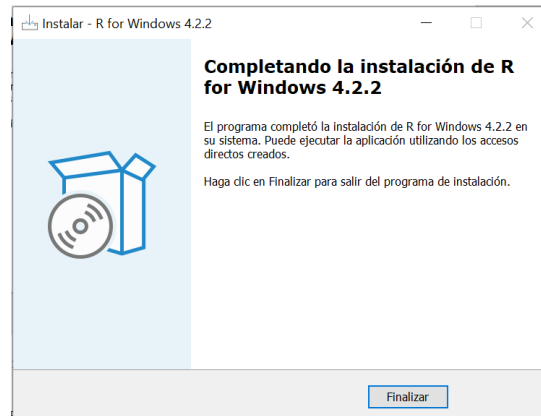


Figura 1.2-9: Finalizando la instalación de R.

Si vamos al escritorio aparecerá el icono de R (ver Figura 1.2-10).



Figura 1.2-10: Icono de acceso directo en el escritorio a R.

Arrancando R desde este icono se despliega la pantalla que se muestra en la Figura 1.2-11.

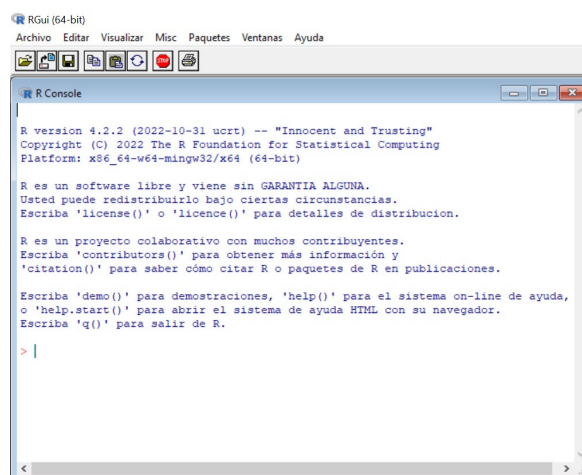


Figura 1.2-11: Consola de R.

Una vez instalado R hay que instalar RStudio. Para instalar **RStudio** hay que descargar la última versión estable de RStudio desde el [sitio oficial de posit](https://posit.co/) (<https://posit.co/>) que sea compatible con el sistema operativo de nuestra computadora. En este caso desde el enlace de [download](https://posit.co/download/rstudio-desktop/) (<https://posit.co/download/rstudio-desktop/>) seleccionamos la versión gratuita de RStudio Desktop for Windows (si es este nuestro sistema operativo). Se descarga **RStudio-2022.12.0-353.exe**.

A continuación, ejecutamos el instalador descargado y, al igual que en la instalación de R, nos volverá a salir aviso de Windows de que no se puede comprobar el editor, no le hacemos caso y pulsamos en Ejecutar. A veces también sale otra ventana indicando que hay que permitir la ejecución del paquete ya que requiere privilegios de administrador.

A continuación, aparece el Asistente de instalación de RStudio (ver Figura 1.2-12) y pulsamos Siguiente.

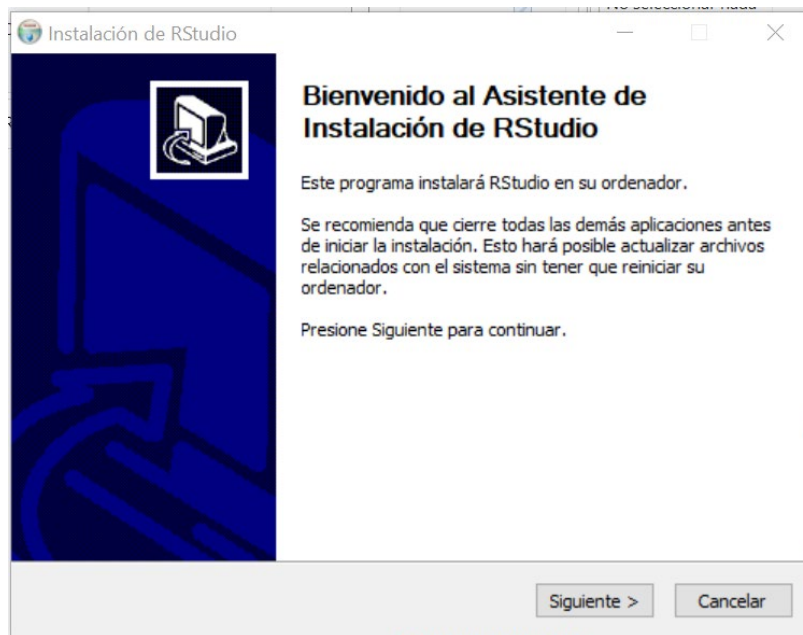


Figura 1.2-12: Asistente de la instalación de RStudio.

Después, tal y como se muestra en la Figura 1.2-13 nos pide el lugar de instalación del paquete. Pulsamos siguiente.

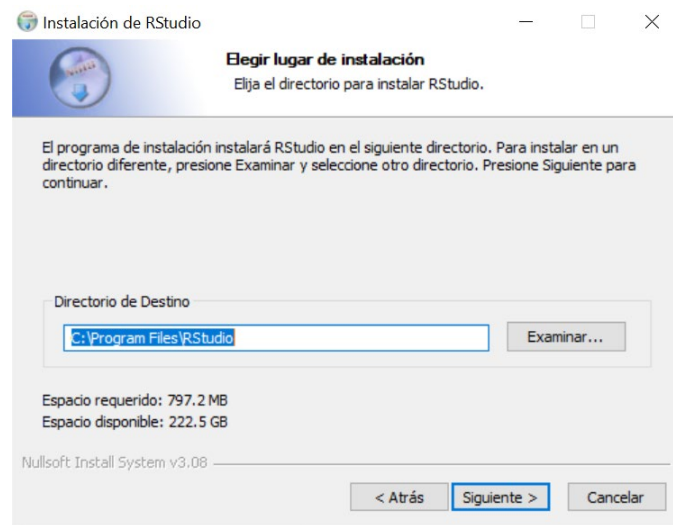


Figura 1.2-13: Selección del directorio de instalación de RStudio.

La elección de la carpeta del menú inicio donde colocar los accesos directos a los elementos del paquete se realiza desde la pantalla que se muestra en la Figura 1.2-14. Y se pulsa Instalar.

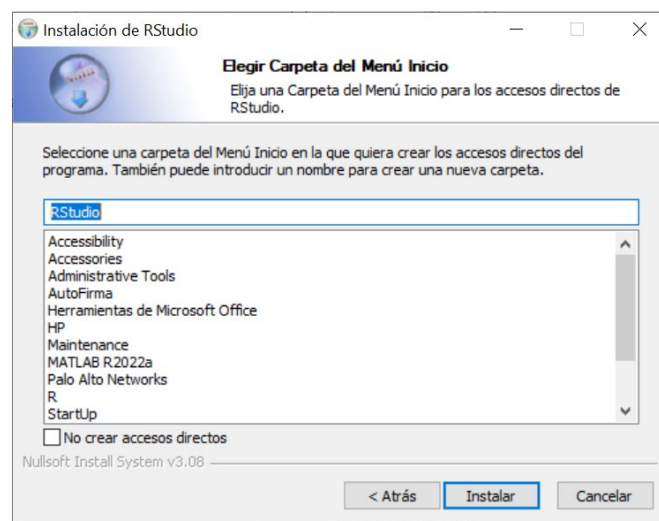


Figura 1.2-14: Elección de la Carpeta del Menú Inicio en la instalación de RStudio.

En este momento comienza la instalación (ver Figura 1.2-15).

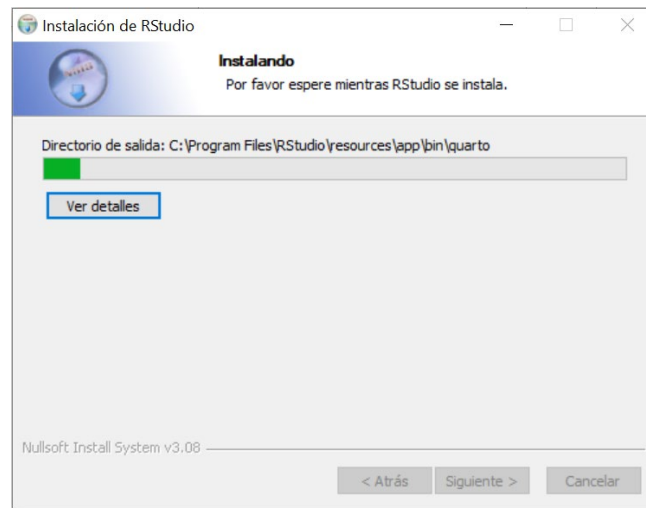


Figura 1.2-15: Instalando RStudio.

Cuando termina de instalar aparece el asistente indicando que ha finalizado la instalación (ver Figura 1.2-16). Pulsamos Terminar.

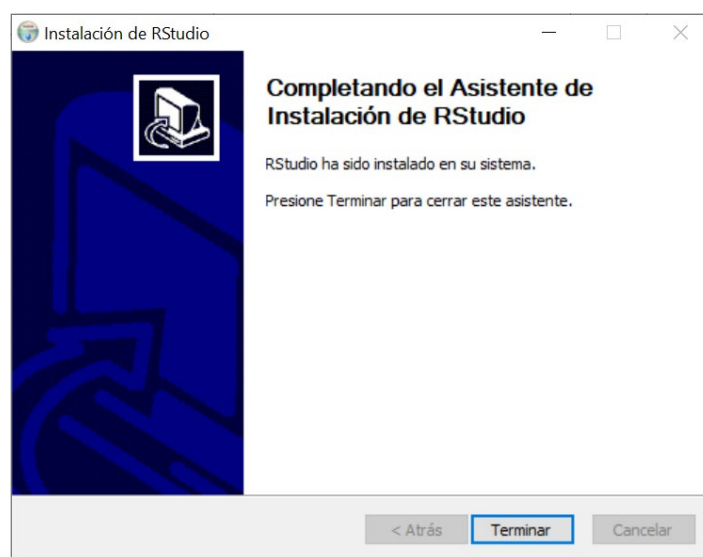
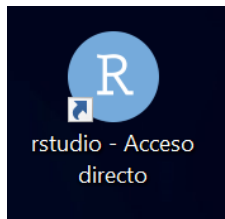
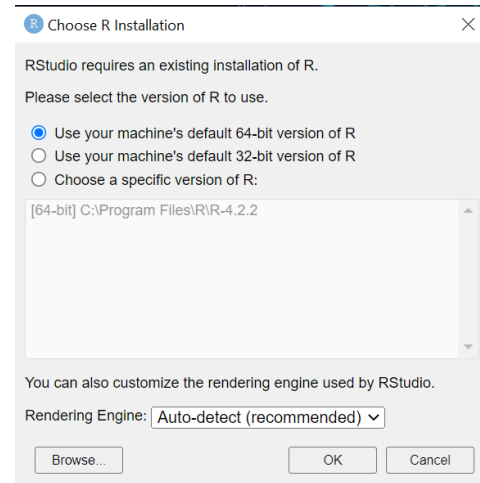


Figura 1.2-16: Terminando instalación de RStudio.

Si se nos ha olvidado marcar la opción de Accesos directos (en la Figura 1.2-14) podemos crear a mano el acceso directo en el escritorio. Para ello vamos en el explorador de Windows a la carpeta “Archivos de Programa” y en el directorio en el que hemos instalado RStudio (Figura 1.2-13). Buscamos el fichero de la aplicación rstudio y una vez seleccionado con el botón derecho del ratón elegimos la opción de “crear acceso directo” para crear el acceso directo en el escritorio. El acceso directo creado queda como se muestra en la Figura 1.2-17a. Al arrancar hacer doble clic sobre este icono se mostrará la Figura 1.2-17b en la que se selecciona la instalación de R con la que se trabajará.



(a)



(b)

Figura 1.2-17: (a) Icono de acceso directo en el escritorio a RStudio. (b) Selección de la Instalación de R

El interfaz de RStudio que arranca al dar OK en la Figura 1.2-17b es el que se muestra en la Figura 1.2-18.

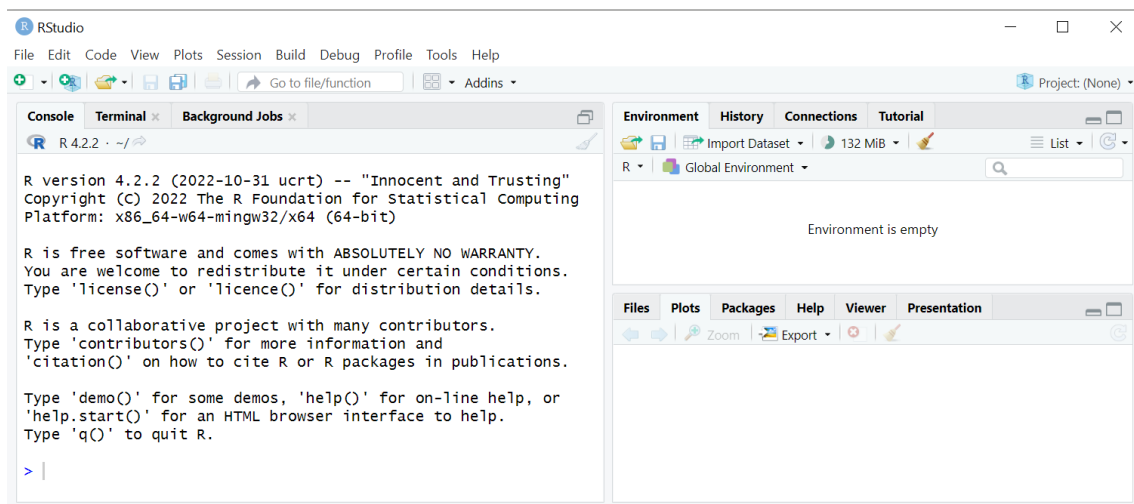


Figura 1.2-18: Interfaz de RStudio.

Una vez instalado el RStudio ya podemos empezar a trabajar. Sin embargo, conviene mencionar que hay determinados paquetes R que no se instalan correctamente en Windows por necesitar algunos requisitos adicionales para su compilación. Por ello se recomienda que se instale también la última versión de **Rtools** ( <https://cran.r-project.org/bin/windows/Rtools/>). Una vez descargado el fichero



(versión 4.2 en el momento de escribir estas líneas) es una instalación típica. Las pantallas correspondientes se muestran en las Figuras 1.2-19/1.2-23.

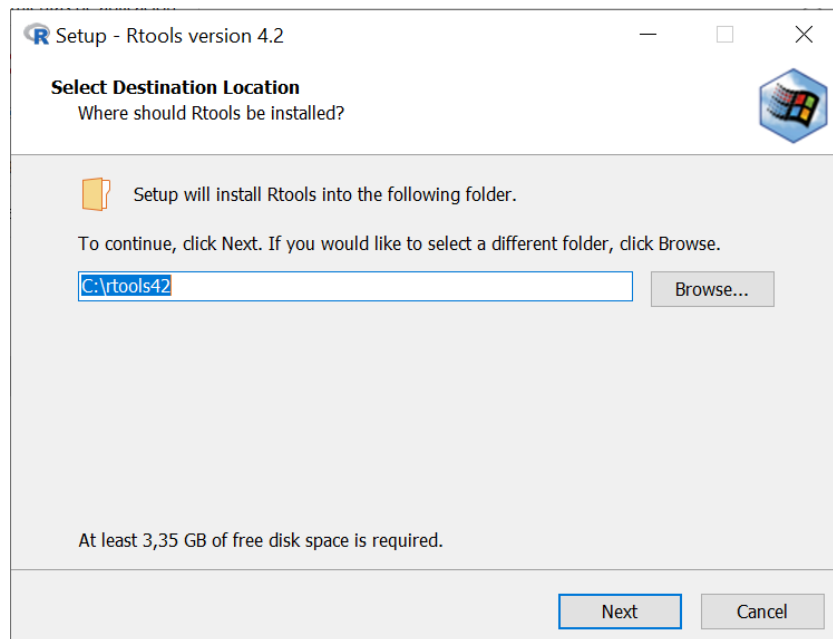


Figura 1.2-19: Instalación Rtools.

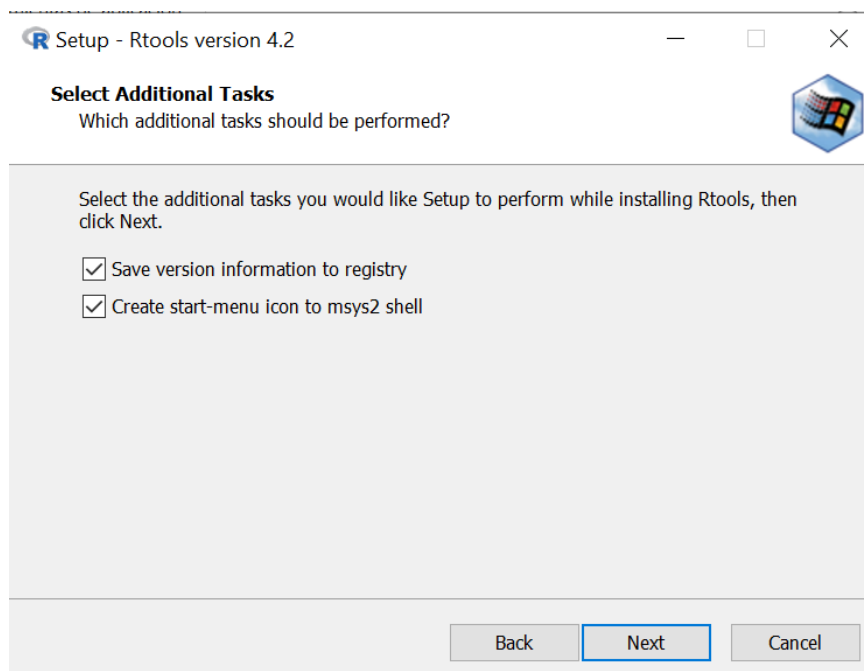


Figura 1.2-20: Instalación Rtools.

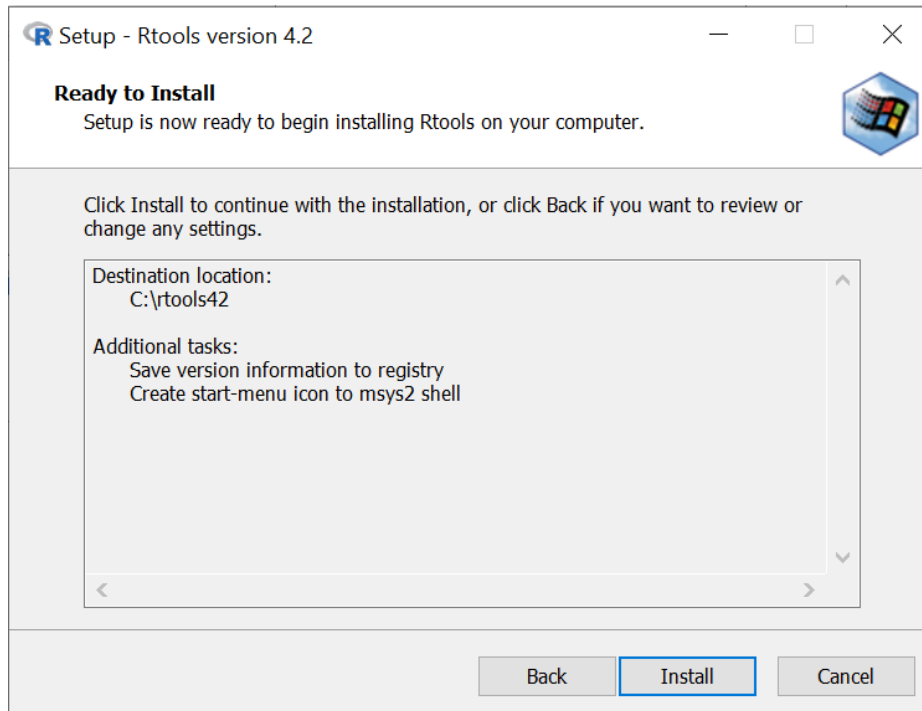


Figura 1.2-21: Instalación Rtools.

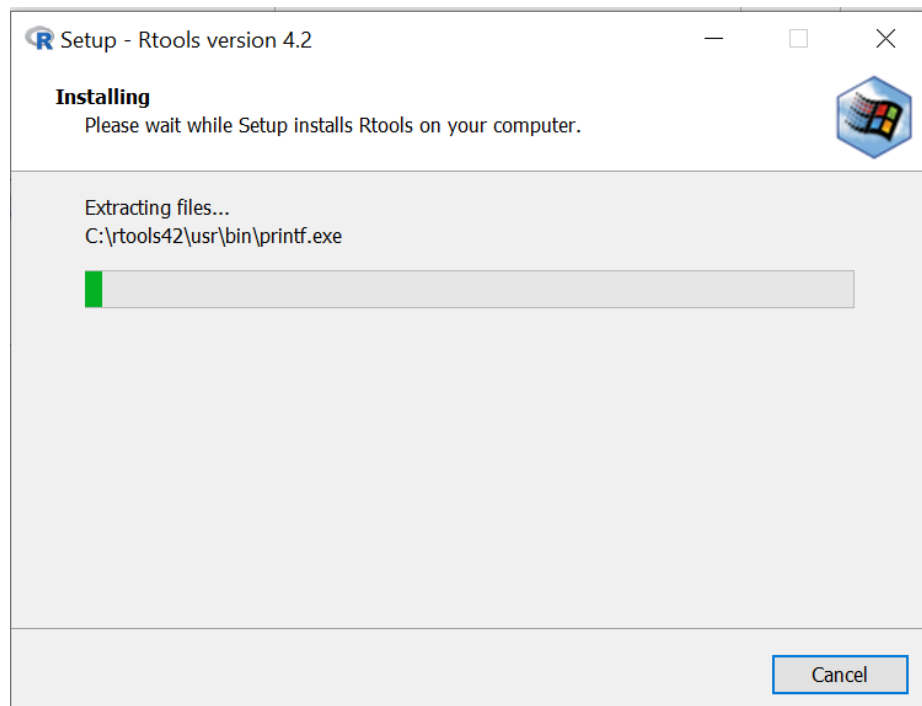


Figura 1.2-22: Instalación Rtools.

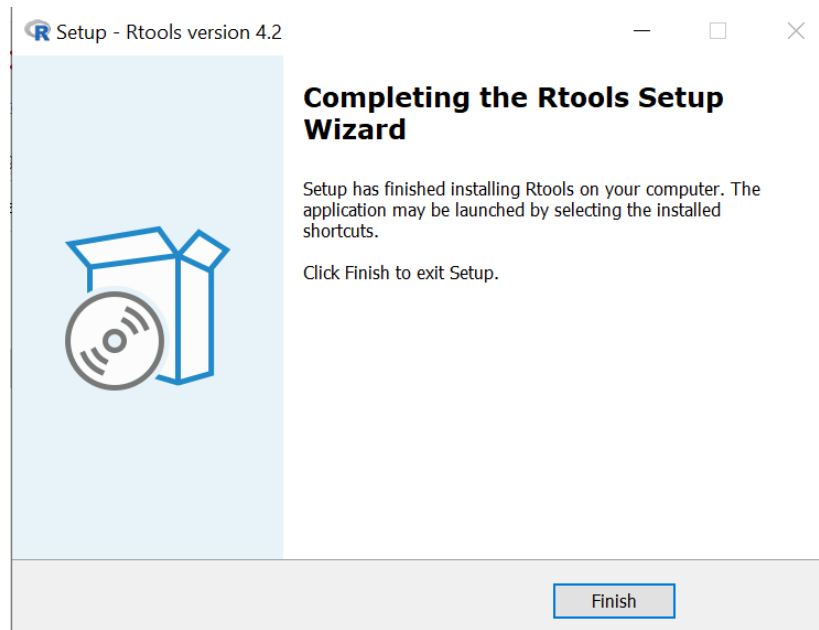


Figura 1.2-23: Instalación Rtools.

### 1.2.2. Instalando R en GNU/Linux

Dependiendo de cuál sea la distribución el procedimiento puede variar. Por ejemplo, si la distribución es Ubuntu el procedimiento de la instalación sería:

```
sudo apt-get update  
sudo apt-get install r-base
```

Para más detalles sobre el procedimiento de instalación ver CRAN o este enlace: <http://numerorojo.wordpress.com/2008/04/27/instalar-r-en-ubuntu/>

### 1.2.3. Instalando R en Mac OSX

Tanto la información para la instalación como el fichero necesario están disponibles aquí: <http://cran.es.r-project.org/bin/macosx/>

Un vídeo con los pasos para realizar la instalación es el siguiente:

<https://www.youtube.com/watch?v=GFI mMj1IMRI&feature=youtu.be>

## 1.3. Entorno de RStudio

### 1.3.1. Pantalla de RStudio

Si ejecutamos por primera vez el programa RStudio una vez instalado se mostrará la pantalla que se muestra en la Figura 1.3-1.

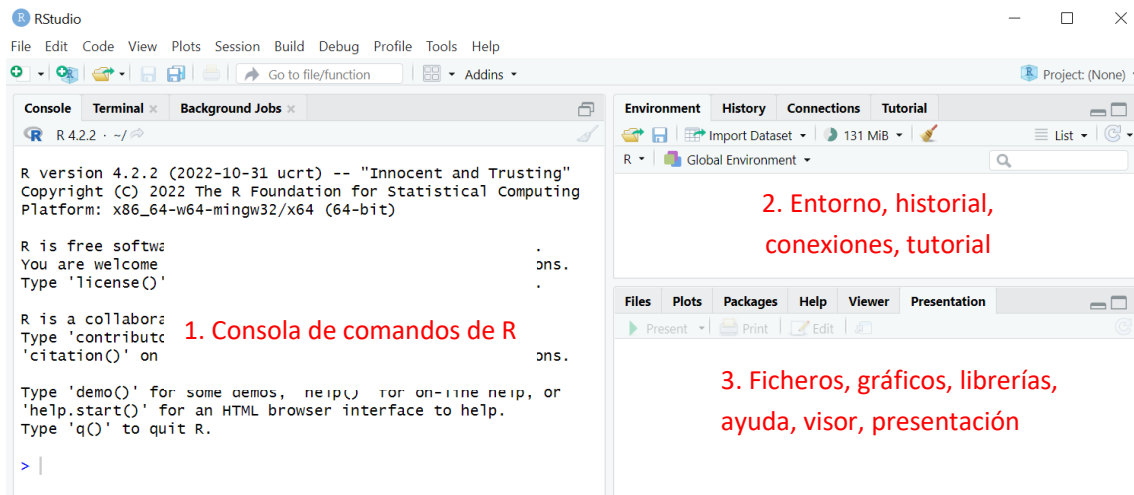


Figura 1.3-1: Pantalla inicial de RStudio.

Esta pantalla está dividida en tres partes fundamentales o paneles (en rojo en la Figura 1.3-1):

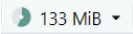



1. **Ventana situada a la izquierda.** Es la denominada **Consola de Comandos de R** (donde está el prompt ">"). En esta consola empezaremos nuestro trabajo. En ella podremos escribir instrucciones y ver las salidas. R es un lenguaje interpretado, en el cual se distingue entre las letras mayúsculas y minúsculas. Es posible ir introduciendo los comandos uno a uno en la línea de comandos de la Consola, tras el símbolo del sistema (>), o bien ejecutar un conjunto de comandos escritos en un fichero.

Al lado de la pestaña **Console** hay otras dos pestañas, **Terminal** y **Background Jobs** que no vamos a utilizar en esta primera aproximación a R. La pestaña **Terminal** se utiliza para crear sesiones independientes. La pestaña **Jobs** permite poder ejecutar los scripts que creemos en segundo plano mientras continuamos usando el interfaz de RStudio.

2. **Ventana situada en la parte superior derecha.** En la pestaña **Environment (Entorno)** se muestran todos los objetos activos. Desde esta pestaña también podemos entre otras cosas realizar las acciones siguientes:

- a. Limpiar nuestro entorno



- b. Memoria en uso  133 MiB
- c. Importar datos 
- d. Abrir algún fichero 
- e. o guardar el espacio de trabajo .

En la pestaña **History (Historial)** se encuentra el historial de objetos y comandos utilizados. La pestaña **Connections** situada a continuación de la pestaña **Environment** se utiliza para conectar con distintas fuentes de datos, aunque en este módulo de introducción a R no haremos uso de ella. Tampoco utilizaremos la pestaña **Tutorial** que sirve para poder ejecutar tutoriales de RStudio una vez que se instalan los paquetes shiny y learnr.

3. **Ventana situada en la parte inferior derecha.** Tiene 5 pestañas. La pestaña **Files (Ficheros)** muestra como si estuviéramos en un PC/Mac Windows los ficheros y carpetas del directorio de trabajo. La pestaña **Plots (Gráficos)** muestra los gráficos que se van generando. En la pestaña **Packages (Librerías)** están listados paquetes para cargarlos e instalarlos directamente cuando sea preciso. En la pestaña **Help (Ayuda)** encontramos la ayuda, información adicional. Además hay un visor HTML disponible en la pestaña **Viewer (Visor)**.

### 1.3.2. Primeros pasos con R y RStudio

Según vayamos practicando con RStudio iremos conociendo la finalidad de cada una de las tres zonas que comentamos en la Figura 1.3-1. Comenzamos por la Consola de Comandos.

#### 1.3.2.1. CONSOLA DE COMANDOS

En ella aparecen en primer lugar una serie de mensajes con información sobre la versión de R que estamos ejecutando (la 4.1.2 en la Figura 1.3-1). Debajo de estos mensajes está el símbolo `>` que es el prompt de R, junto al que parpadea el cursor. Ese símbolo, y el cursor parpadeando, nos indican que ese es el prompt activo, y que R está esperando una orden.

#### **EJEMPLO: RStudio como Calculadora**

Para empezar a familiarizarnos con la sintaxis de R y con las características de la interfaz RStudio vamos, en primer lugar, a aprender a usar R (a través de RStudio) como una calculadora.

Vamos a calcular  $4^2$ . Nos situamos en la línea del prompt de R y tecleamos `4^2`. Al pulsar Intro aparecerá una línea como esta:

[1] 16

y, justo debajo, aparece un nuevo prompt, que ahora pasa a ser el prompt activo (ver Figura 1.3-2).

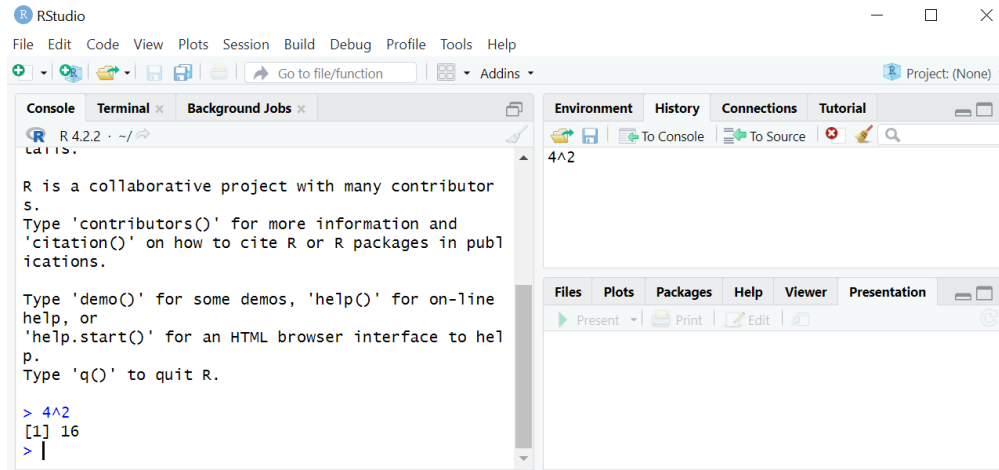


Figura 1.3-2: RStudio como calculadora.

El uno entre corchetes es la forma en la que R nos indica que esta es la primera línea de su respuesta. El número 16, naturalmente, es el valor  $4^2$  que hemos calculado. Como vemos, en R se usa ^ para las potencias. En R los comentarios son precedidos por el símbolo #. Esto es, el intérprete de R ignora todo el texto que aparezca tras este símbolo.

En R las distintas operaciones aritméticas se representan mediante los siguientes símbolos:

suma	+
resta	-
producto	*
división	/
potencia	^

En R la mayor parte de su funcionalidad se consigue mediante el uso de funciones, tanto las proporcionadas por el lenguaje como las definidas por el propio usuario. La raíz cuadrada es una función en R. Se representa con `sqrt` (del inglés, square root). Para utilizarla hay que poner el argumento entre paréntesis. En la Figura 1.3-3 se muestra el cálculo de la raíz cuadrada de 25. Recordamos de nuevo en este punto que R distingue mayúsculas y minúsculas; así que si tecleamos `Sqrt` o `QRT` no funcionarán

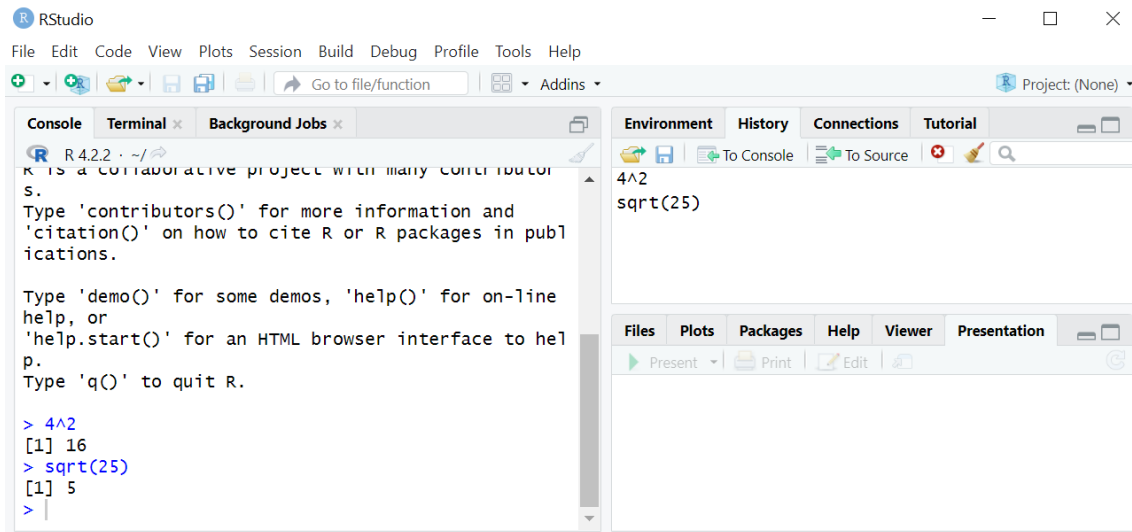


Figura 1.3-3: RStudio como calculadora.

Como os podéis imaginar `sqrt` es solo una función de las muchas disponibles en R, como iremos viendo. Muchas funciones básicas están disponibles por defecto. Otras se encuentran en paquetes, que deben ser abiertos (esta acción se denomina *attach*) para poder usar las funciones que contienen. Este tema lo volveremos a tratar al presentar las funciones y paquetes.

\*\*\*\*\*

### EJERCICIO 1.

Copia o teclea las líneas de código que aparecen a continuación. Cada una de las líneas en el prompt, una por una, y ejecútalas, pulsando Entrar. Intenta adivinar el resultado de cada operación antes de ejecutar.

```
3+4
13-6
5*4
13/5
13%/%5
13%%5
1/3+1/5
sqrt(9)
sin(pi)
sin(3.14)
```

Guarda un pantallazo de las operaciones realizadas con sus resultados. A la vista de los resultados ¿trabaja R con representación mediante decimales? ¿Puede haber pérdida de precisión en determinadas operaciones de R? Comenta como interpreta R la operación  $1/3+1/5$ . ¿La interpreta como  $\frac{1}{3} + \frac{1}{5}$  o como  $\frac{1}{\left(\frac{3+1}{5}\right)}$ ? ¿El valor obtenido para `sin(pi)` es el esperado? ¿Qué diferencia encuentra entre `sin(pi)` y `sin(3.14)`?

**Comentarios al Ejercicio 1:** Los operadores `%/%` y `%%` representan, respectivamente, el cociente y el resto de la división entera. La interpretación que realiza R de una expresión es debida a la prioridad de operadores que tiene definida el lenguaje. La función `sin` es la función seno de trigonometría. Por `pi` R denota la constante matemática  $\pi \sim 3.141593$ .

\*\*\*\*\*

### 1.3.2.2. ALGUNOS DETALLES DE LA INTERFAZ DE RSTUDIO

#### Comandos incompletos

¿Qué sucede si en el prompt activo de la consola escribimos una operación incompleta? Probemos a escribir en la siguiente operación incompleta:

```
3*
```

y pulsar Intro. La respuesta de R es un símbolo +:

```
> 3*
+
```

que en este caso no tiene nada que ver con la suma. Es la forma que tiene R de decirnos que “necesita algo más”. Cuando R se encuentra con una expresión que considera no errónea, sino incompleta (por ejemplo, con un paréntesis abierto al que corresponde un paréntesis cerrado), entonces nos lo indica con este símbolo +. Y al hacerlo, nos da la oportunidad, en casos como este, de corregir el error, completando la expresión incompleta. Así que si, ahora, junto al símbolo + escribimos 2, de esta forma:

```
> 3*
+ 2
```

y pulsamos Intro, verás que R ha entendido la expresión completa `3*2` y nos devuelve la respuesta:

```
> 3*
```



```
+ 2  
[1] 6
```

En algún momento, no obstante, ese símbolo + de los comandos incompletos se convertirá en un inconveniente. Imaginemos, por ejemplo, que queremos multiplicar  $5 \times 7$ , pero que por error hemos tecleado la expresión

```
5*7-
```

y hemos pulsado Intro, antes de darnos cuenta del error. En tal caso, R piensa que esa expresión está incompleta, y vuelve a aparecer el signo +

```
> 5*7-  
+
```

¡Pero ahora no queremos añadir nada! En este caso podríamos añadir un 0, sin que eso afectara el resultado. Pero habrá, más adelante, muchos casos en los que no hay una solución sencilla evidente.

Para solucionar esto basta con que pulsemos la tecla `ESC` (escape), con lo que R interrumpe el comando y nos devuelve a un nuevo prompt activo, sin calcular nada.

### Navegando por el historial de comandos

En un caso como el que hemos descrito, no supone un problema volver a teclear la expresión, evitando el error que habíamos cometido. Pero en ocasiones, si hemos escrito una expresión realmente complicada, y, por ejemplo, hemos tecleado un paréntesis de más en medio de la expresión, seguramente preferiríamos no tener que volver a teclear la expresión completa. Un remedio casero para solucionarlo consiste en seleccionar y copiar la expresión errónea, pegarla en el prompt activo, y editarla ahí para corregir el error.

Hay una segunda manera de conseguir el mismo efecto. Podemos hacer uso de otra característica interesante de la Consola de Comandos. Si nos situamos en el prompt activo, y pulsas las teclas con flechas arriba ↑ o abajo, pasará la lista completa de comandos que hemos ejecutado, desde el principio de la sesión de trabajo. Puedes probar a hacerlo para ver cómo funciona. Esa lista de comandos es el **Historial de Comandos**. Usando las flechas arriba y abajo podemos recorrer el Historial de Comandos, detenernos en cualquier punto, modificar ese comando si lo deseamos, y después ejecutarlo (con o sin modificaciones).

El Historial de Comandos que podemos recorrer con las flechas arriba y abajo se limita a la sesión de trabajo actual. Pero si miras en el panel superior derecho de RStudio, verás una pestaña llamada **History**, que se muestra en la Figura 1.3-4.

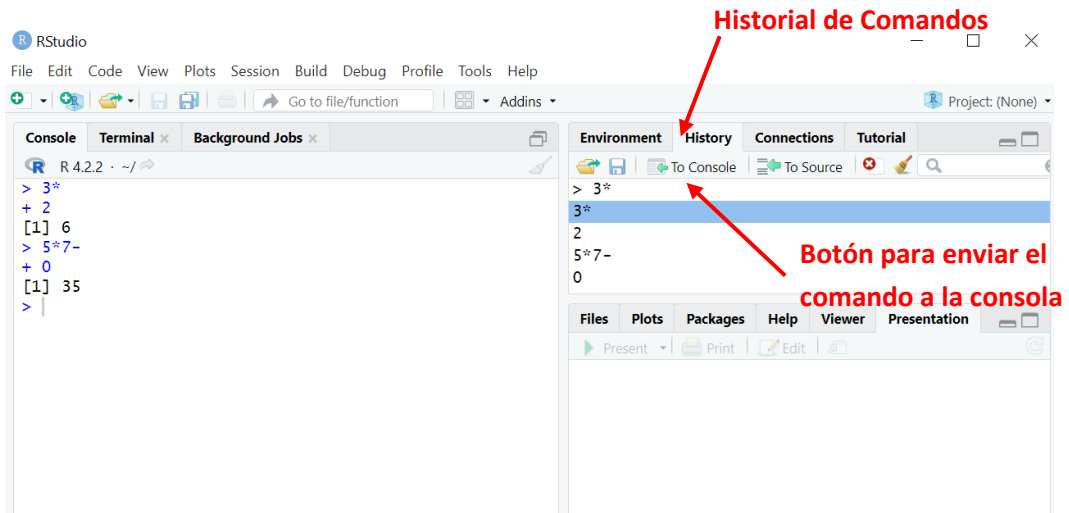


Figura 1.3-4: Historial de Comandos.

El Historial de Comandos que aparece en ese panel refleja los comandos de la sesión actual y de anteriores sesiones. Si esta es tu primera sesión de trabajo con R, ambos coincidirán, claro está. Pero a medida que vayas abriendo RStudio en días sucesivos, te puede resultar muy útil localizar, por ejemplo, un comando que usaste hace dos sesiones, pero que no recuerdas exactamente. Los comandos que aparecen en ese panel se pueden copiar y pegar en la Consola de Comandos o, de forma más directa, puedes seleccionar uno de ellos con el ratón, y pulsar en el botón **To Console** que hemos destacado en la Figura 1.3-4.

## Errores

R nos informará con distintos tipos de mensajes dependiendo del tipo de error que cometamos. Comentamos aquí algunos de ellos.

Si ejecutamos una expresión sin sentido, por ejemplo:

```
> 5/*3
```

En la consola de RStudio nos aparece el siguiente mensaje en rojo

```
Error: unexpected '*' in "5/*"
```

Con este mensaje el R “se está quejando” de que esta expresión es defectuosa. No se trata en este caso de una expresión incompleta como la que hemos visto antes. Es imposible completar esta expresión de forma correcta.

Otro mensaje de error nos puede dar, por ejemplo, al dividir por 0. Si escribimos:

```
> 2/0  
[1] Inf
```

el símbolo `Inf` (de infinito) nos avisa del problema. Ese símbolo también puede aparecer con signo menos, si por ejemplo tratas de calcular el logaritmo neperiano de 0. En R el logaritmo neperiano es la función `log`, así que sería:

```
> log(0)  
[1] -Inf
```

En ambos casos R no lanza un mensaje de error, pero utiliza el símbolo de infinito, porque así nos proporciona más información matemática sobre lo que ha ocurrido.

En otros casos, el problema es diferente. Por ejemplo, si tratas de calcular la raíz cuadrada de un número negativo:

```
> sqrt(-4)  
[1] NaN  
warning message:  
In sqrt(-4) : NaNs produced
```

En este caso obtenemos una respuesta `NaN`, y además una advertencia (*warning*) sobre la aparición de esa respuesta. El símbolo `NaN` es la abreviatura de *Not a Number*; es decir “no es un número”. Esta es la respuesta que obtendremos cuando tratamos de hacer operaciones como esta, que producen como resultado un número complejo.

### Haciendo limpieza en la Consola de Comandos.

Si has seguido todas las instrucciones hasta aquí, tu Consola de Comandos seguramente empieza a estar llena de comandos. ¿Qué podemos hacer para limpiar la Consola?

Esto se consigue con la combinación de teclas `Ctrl + L`. Hay que tener cuidado porque es un proceso no reversible. Podremos seguir navegando por el Historial de Comandos con las teclas arriba y abajo, pero para recuperar los resultados tendremos que ejecutar de nuevo los comandos.

### 1.3.2.3. FUNCIONES DE AYUDA

R es un software funcional, esto es, realiza las tareas a través de funciones. La primera función que proporciona ayuda sobre cualquier función de R es `help`. La sintaxis de esta función es:

`help (nombre de la funcion)`

Para acceder a la página de ayuda de la función `mean` debemos teclear en la consola:

```
> help (mean)
```

Veremos entonces en la parte inferior derecha de la interfaz de Rstudio la pantalla de ayuda con la descripción de la media aritmética, su uso y ejemplos (ver Figura 1.3-5).

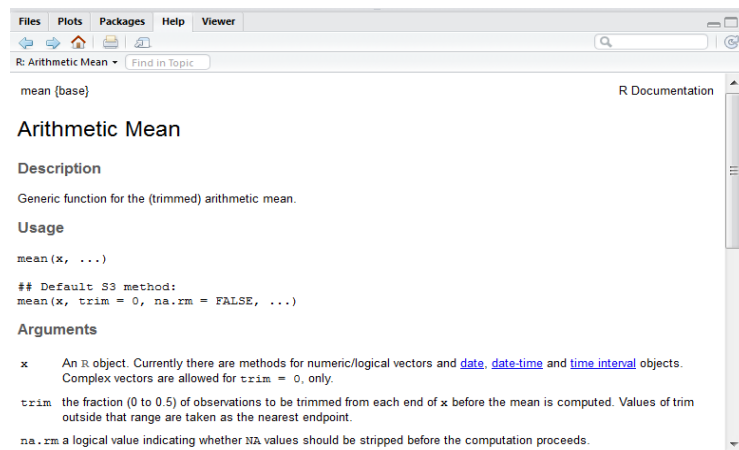


Figura 1.3-5: Pantalla de ayuda.

Otra forma alternativa de obtener ayuda sobre una función es escribir el símbolo `?` delante de su nombre.

```
> ?mean
```

Haciendo clic en el icono marcado en la Figura 1.3-6 se accede a un repositorio de distintos temas de ayuda de R.

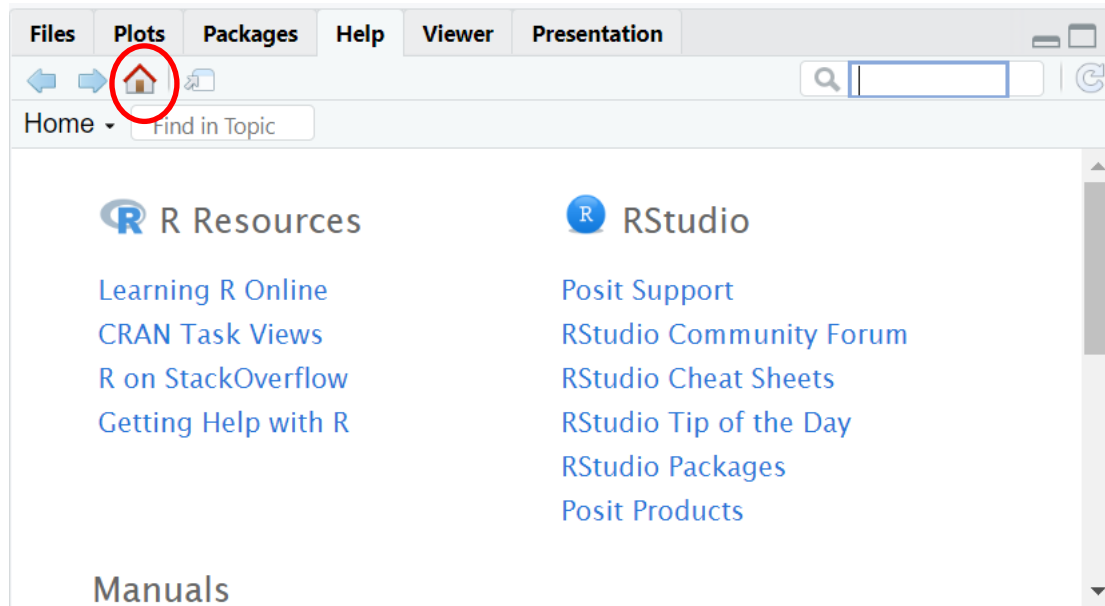


Figura 1.3-6: Repositorio de ayuda.


Cuando no se conoce el nombre exacto de la función sino una o varias palabras claves, se puede utilizar la función `help.search` para la búsqueda de ayuda. Si se escribe en la consola

```
> help.search("median")
```

R devuelve todas las funciones relacionadas con el cálculo de la mediana. RStudio dispone también del acceso a las ayudas a través del menú Help, como veremos más adelante.

#### 1.3.2.4. EN R TRABAJAMOS CON SCRIPTS

Aunque volveremos sobre ello en la sección dedicada a la barra del menú principal, conviene mencionar ya que en R se trabaja con scripts. Un script en R es un fichero en el que guardamos nuestro trabajo.

Si seleccionamos **File → New File → R Script** en el interfaz de Rstudio se abre una nueva ventana denominada Editor de R (ver Figura 1.3-7). En esa ventana podremos escribir las sentencias de código R que queramos y ejecutarlas haciendo clic en el icono  **Run**.

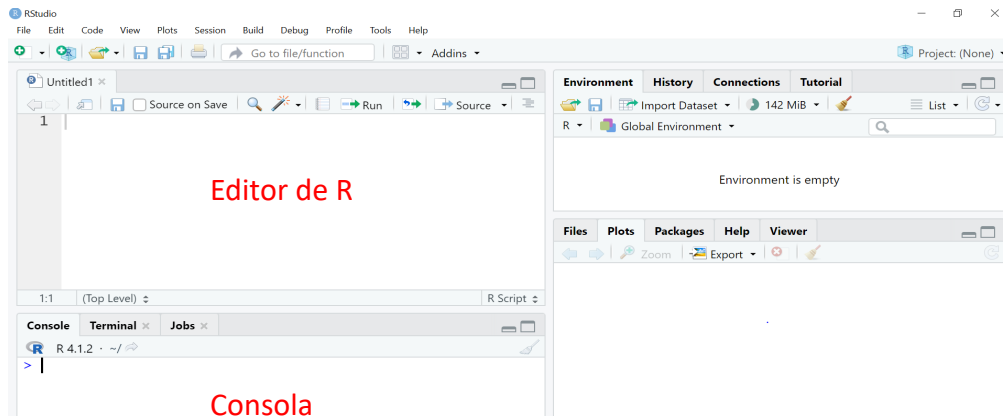


Figura 1.3-7: Interfaz de RStudio.

### **EJEMPLO: Creando un script con RStudio**

Para crear un R script nuevo o se accede a **File → New File → R Script** o se selecciona en el icono



R Script o se presiona Ctrl+Shift+N.

En la Figura 1.3-8 se muestra un nuevo script en el que realizaremos unos cálculos con R.

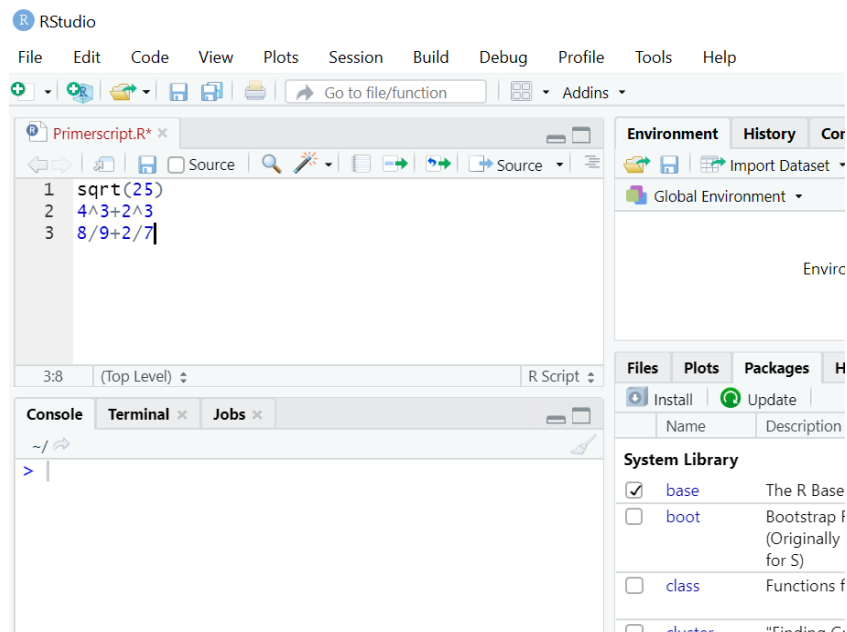



Figura 1.3-8: Escribiendo script en R.

Una vez escrito el script tenemos que guardarlo. Esto se hace desde la opción **File → Save**. La extensión con la que se guarda es **.R**

Para ejecutarlo basta con dejar el cursor en la línea que queremos que se ejecute y seleccionar . En la consola veremos el resultado de la ejecución. Si queremos ejecutar varias líneas a la vez basta con tener todas seleccionadas (ver Figura 1.3-9).

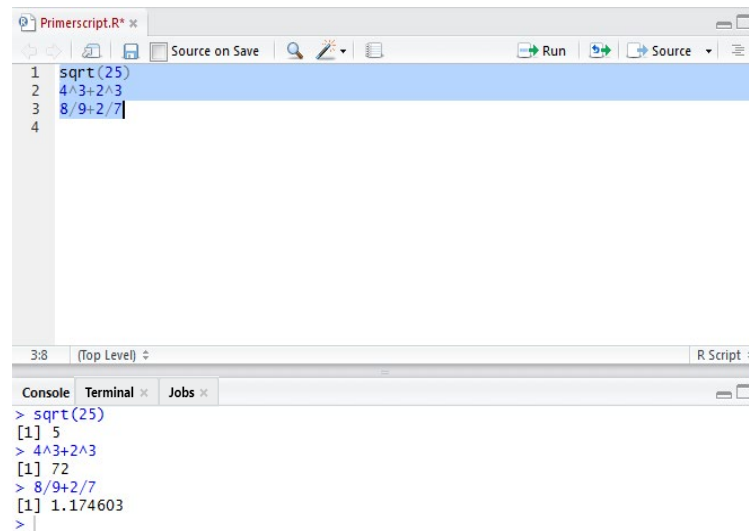


Figura 1.3-9: Ejecutando script en R.

Trataremos más el tema de la ejecución de scripts cuando veamos cómo se definen en R funciones de usuario. Una vez cerremos el script podremos volver a abrirlo desde **File → Open File**

\*\*\*\*\*

### 1.3.3. El espacio de trabajo y el directorio de trabajo

Adelantándonos un poco a lo que veremos en secciones siguientes, conviene tener en mente desde este momento que R incluye la posibilidad de manejar una gran variedad de estructuras de datos, incluyendo vectores, matrices, dataframes y listas. Además, como recalcaremos más adelante, la mayor parte de la funcionalidad de este lenguaje se consigue mediante el uso de funciones, tanto las proporcionadas por el lenguaje como las definidas por el propio usuario. En el siguiente ejemplo vamos a definir dos vectores en RStudio con el objetivo de explicar utilizando estos vectores lo que se entiende por espacio de trabajo.

#### **EJEMPLO: Definiendo los primeros objetos en RStudio**

Las sentencias del lenguaje R consisten en funciones y asignaciones. R usa normalmente el símbolo `<-` para las asignaciones. También puede utilizar el símbolo `=`. Por ejemplo, las sentencias:

```
> a <- 3
> x <- rnorm(10)
```

Asigna un valor 3 al símbolo `a` y crea un objeto de tipo vector llamado `x`, que contiene 10 observaciones de la distribución normal estándar. \*\*\*\*\*

### 1.3.3.1. EL ESPACIO DE TRABAJO

El **espacio de trabajo** (*workspace*) es el espacio de memoria en el cual se guardan todos los objetos creados durante la sesión. En él se almacena cualquier objeto, variable o función que creemos durante una sesión de R.

Para finalizar una sesión de trabajo debemos ejecutar la función `q()`. Es posible salvar el espacio de trabajo al finalizar la sesión, cuando R pregunta antes de cerrarse si debe hacerlo (ver Figura 1.3-10), de modo que automáticamente vuelve a cargarse la siguiente vez que se arranque R.

```
> a<-3
> x<-rnorm(10)
> q()
Save workspace image to ~/.RData? [y/n]: |
```

Figura 1.3-10: Salvando el espacio de trabajo.

Puede guardarse también el espacio de trabajo donde se desee desde el menú `Session>>Save Workspace As...` y volverlo a cargar desde la opción `Session>>Load Workspace...`

Puede obtenerse una lista de todos los objetos del espacio de trabajo mediante el comando `> ls()` (ver Figura 1.3-11 ).

```
> ls()
[1] "a" "x"
```

Figura 1.3-11: Listando objetos del espacio de trabajo.

La función `rm()` elimina del espacio de trabajo los objetos que se le pasan como argumento. Por ejemplo,

```
> rm(x)
```

elimina del espacio de trabajo el objeto llamado `x`.

En la pestaña **Environment** se muestran los elementos del espacio de trabajo que creemos durante una sesión de R.



### 1.3.3.2. LA PESTAÑA ENVIRONMENT

Supongamos a modo de ejemplo que en la Consola hemos escrito lo que aparece en la Figura 1.3-12.

En la pestaña Environment se muestran los dos objetos (vectores `a` y `x` en este caso) que hemos creado.

Desde esta pestaña Environment podemos salvar el espacio de trabajo (📁), cargar en la sesión actual un espacio de trabajo que tengamos almacenado (📁), o eliminar los objetos de nuestro espacio de trabajo (🧻). Como ya hemos comentado desde el menú **Session** también están disponibles algunas de estas opciones.

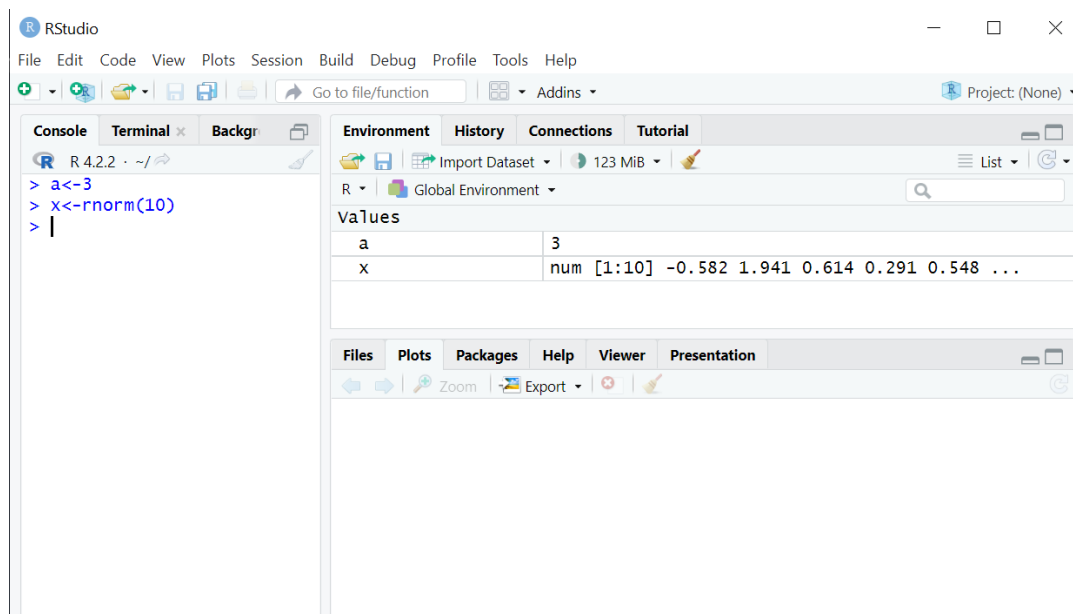


Figura 1.3-12: Pestaña Environment de RStudio.

### 1.3.3.3. EL DIRECTORIO DE TRABAJO

El **directorio de trabajo** es donde R salva por defecto los resultados y también de donde intenta por defecto leer los ficheros. Para saber cuál es el directorio de trabajo podemos ejecutar

```
> getwd()
```

Es recomendable separar en diferentes directorios los diferentes proyectos. La función `setwd()` permite modificar el directorio de trabajo, pasando como argumento a la función la ruta del nuevo directorio escrito entre comillas. Por ejemplo:

```
> setwd("C:/Trabajo/R")
```

Crea como directorio de trabajo la carpeta R dentro de la carpeta Trabajo.

Para listar los archivos que están en una determinada carpeta podemos utilizar el comando:

```
> dir()
```

Si tenemos varios proyectos, cada uno almacenado en un directorio, utilizando RStudio se puede cambiar directamente el directorio de trabajo desde el menú `Session >> Set Working Directory` para una determinada sesión (ver Figura 1.3-13), sin necesidad de ejecutar desde la Consola la función `setwd`.

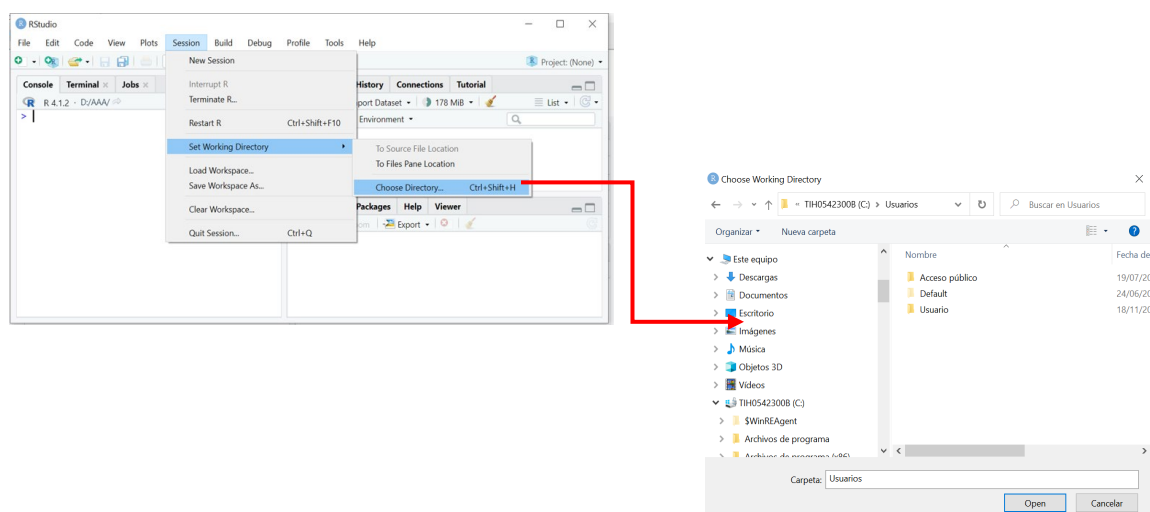


Figura 1.3-13: Cambiando el directorio de trabajo con RStudio.

Cada vez que se abre RStudio, va al directorio de trabajo por defecto. Se puede cambiar este directorio por defecto desde el menú `Tools >> Global options` (ver Figura 1.3-14).

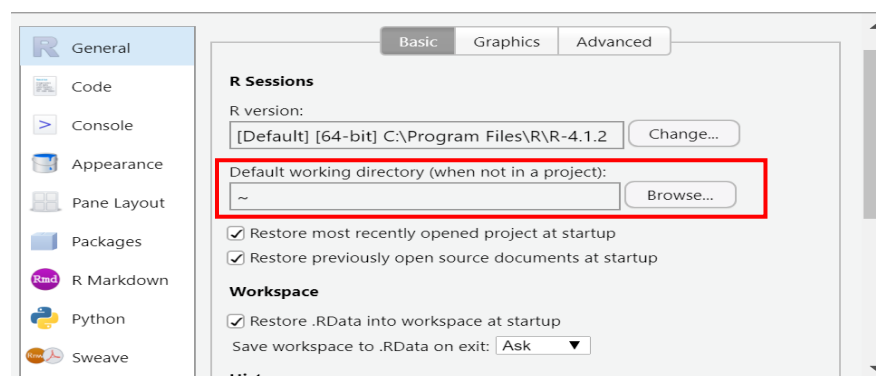


Figura 1.3-14: Directorio de trabajo por defecto.

### 1.3.4. Barra del Menú Principal

RStudio dispone de unos menús accesibles desde la barra del **Menú principal** (ver Figura 1.3-15). Algunas de las acciones que se pueden realizar desde estos menús ya las hemos comentado en apartados anteriores. En esta sección presentamos una revisión bastante exhaustiva de las distintas acciones que se pueden realizar desde la barra del menú principal para que las podamos consultar si las necesitamos.

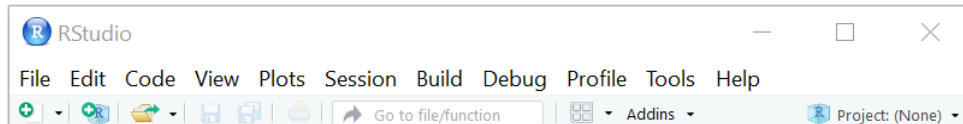


Figura 1.3-15: Barra de Menú principal de RStudio.

Algunos de los menús de esta barra son habituales en programas bajo Windows. Por ejemplo, los menús **File** (Archivo), **Edit** (Edición), **View** (Ver) o el menú **Help** (Ayuda).

Otros menús son específicos de RStudio y permiten realizar cambios en los datos, obtener resultados estadísticos, numéricos, gráficos.....En lo que sigue mostramos las opciones más habituales de estos menús. A lo largo del texto iremos viendo con más detalle aquellos que vayamos utilizando. En esta sección mostraremos únicamente su descripción, no su funcionamiento. Os recomiendo ir aprendiendo las opciones conforme las uséis y que esta descripción la uséis como consulta puntual.

#### 1.3.4.1. EL MENÚ File

Contiene las opciones más generales que suelen tener todos los programas como abrir un archivo, guardar, cerrar... (ver Figura 1.3-16).

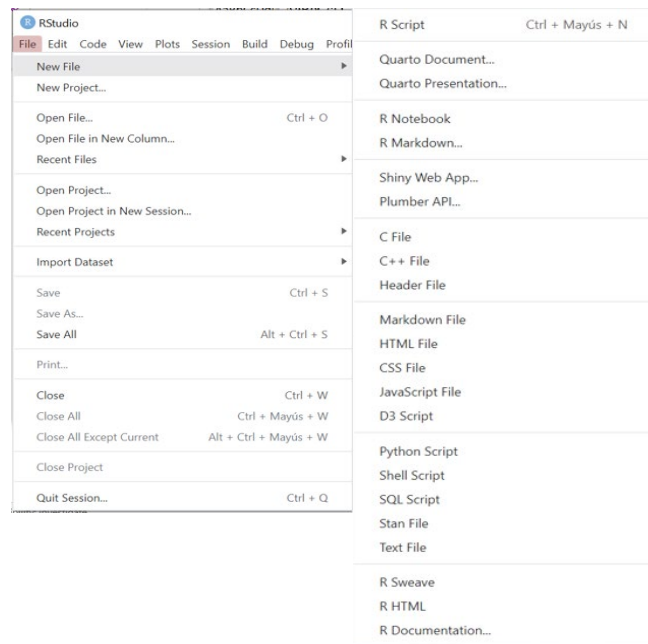

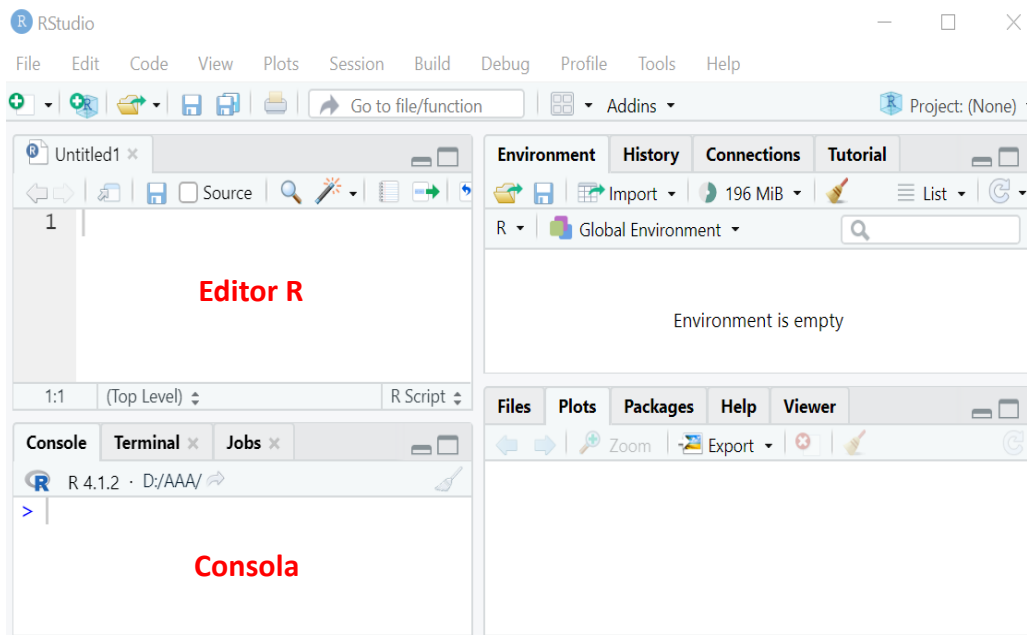


Figura 1.3-16: Menú File.

RStudio ofrece gran flexibilidad para trabajar con distintos archivos que podemos generar desde **New File**. Comentamos a continuación algunas de las posibilidades.

- **New File → R Script:** Permite abrir una nueva ventana denominada Ventana de Edición en la interfaz de RStudio. Aunque realmente se puede desarrollar todo el trabajo en la Consola (en el espacio de trabajo), esta no es la forma más eficiente de trabajar en RStudio. Es muy útil tener un entorno donde manipular (corregir, repetir, guardar, ...) las sentencias de código que solicitemos a R. Este entorno de trabajo es el Editor de R (ver Figura 1.3-17). En la Ventana de Edición se escriben las instrucciones y para ejecutarlas las seleccionamos y pulsamos (**Ctrl+Intro**); (**Ctrl+r**) o pinchamos en el icono Run  (en la parte superior derecha de la Ventana de Edición).
- **New File → Text File:** Permite editar un archivo de texto. En estos archivos no se pueden ejecutar ninguna función a menos que se copien y se peguen en el espacio de trabajo.
- **New File → C++ File:** Permite compilar funciones de C++ en R.

Figura 1.3-17: Menú **New File** → **RScript**

- **New File** → **R Sweave**: Crea un archivo que permite trabajar con LaTeX.
- **New File** → **R Markdown** y **New File** → **R HTML**: Herramientas de RStudio para la creación de informes web. Markdown es un lenguaje simple de marcas diseñado para hacer que el contenido web sea fácil. En lugar de escribir el código HTML y CSS, Markdown permite el uso de una sintaxis mucho más cómoda.
- **New File** → **Quarto document** y **New File** → **Quarto presentation**: permite la creación de publicaciones de documentos científicos y técnicos en diferentes formatos y bajo diferentes tipos de proyectos.
- **New File** → **Shiny Web app**: Para la creación de aplicaciones Shiny.
- **New File** → **R Documentation**. Uno de los requisitos básicos para los paquetes de R es que todas las funciones exportadas, objetos y conjuntos de datos tengan la documentación completa. RStudio también incluye un amplio soporte para la edición de documentación de R (los archivos Rd que se utilizan en la documentación tienen un formato simple de marcas que se asemeja sintácticamente a LaTeX).

**New Project:** Permite crear proyectos que hacen que sea más fácil dividir el trabajo en múltiples contextos, cada uno con su propio directorio de trabajo, espacio de trabajo, historial y los documentos de origen. Como ya hemos comentado anteriormente, los proyectos de RStudio están asociados a los directorios de trabajo.

Al crear un proyecto RStudio:

- Se crea un archivo de proyecto (con una extensión **Rproj.**) dentro del directorio del proyecto. Este archivo contiene diversas opciones de proyecto.
- Se crea un directorio oculto (nombrado **Rproj.user**) donde se almacenan los archivos específicos de un proyecto de carácter temporal (por ejemplo, los documentos originales guardados automáticamente, estado de la ventana, etc.).
- Se carga el proyecto en RStudio y se muestra su nombre en la barra de herramientas de Proyectos.

Seleccionando **File/New → Project** se muestra la ventana de la Figura 1.3-18. Las opciones de esta ventana son las siguientes:

- **New Directory:** Permite comenzar un proyecto en un nuevo directorio de trabajo: bien crearlo en un directorio vacío (*New Project*), bien crear un nuevo paquete de R (**R Package**), bien crear una nueva aplicación Shiny web (**Shiny Web Application**) o crear un proyecto dinámico utilizando herramientas de Quarto.
- **Existing Directory:** Permite asociar un proyecto con un directorio de trabajo ya existente.
- **Version Control:** Permite acceder a distintos sistemas de control de versiones. Para ello es necesario tener previamente instalado el software correspondiente.

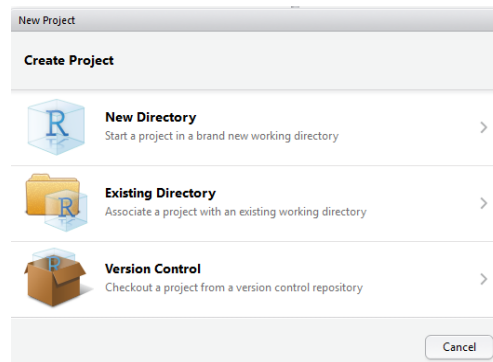


Figura 1.3-18: Menú **File/New→Project**.

**Open File...:** Permite abrir un fichero previamente guardado.

**Open File in a New Column...:** Permite abrir un fichero en una nueva columna que crea para tal efecto en el interfaz.

**Recent Files:** Permite seleccionar un fichero de la lista de los ficheros accedidos más recientemente.

**Open Project...:** Permite abrir un proyecto previamente guardado.

**Open Project in New Session...:** Permite trabajar con varios proyectos RStudio simultáneamente. Esta misma acción también se puede realizar haciendo doble clic en el archivo del proyecto desde la shell del sistema.

**Recent Projects:** Permite seleccionar un proyecto de la lista de proyectos abiertos recientemente.

**Import Dataset:** Permite la importación a R de distintos tipos de datos.

Desde el menú **File** tenemos disponibles también otras opciones como por ejemplo salvar (**Save**), salvar como (**Save As...**), salvar todo (**Save All**), Imprimir (**Print**), Cerrar (**Close**), Cerrar todo (**Close All**), Cerrar todo excepto el actual (**Close All Except the Current**) o Cerrar sesión (**Quit Session...**).

#### 1.3.4.2. EL MENÚ Edit

Las distintas opciones disponibles en este menú (ver Figura 1.3-19) las describimos a continuación.

Edit	Code	View	Plots	Session	Build	Debu
Back					Ctrl + F9	
Forward					Ctrl + F10	
Undo					Ctrl + Z	
Redo					Ctrl + Mayús + Z	
Cut					Ctrl + X	
Copy					Ctrl + C	
Paste					Ctrl + V	
Paste with Indent					Ctrl + Mayús + V	
Select All					Ctrl + A	
Folding						►
Go to Line...					Alt + Mayús + G	
Find...					Ctrl + F	
Find Next					F3	
Find Previous					Mayús + F3	
Use Selection for Find					Ctrl + F3	
Replace and Find					Ctrl + Mayús + J	
Find in Files...					Ctrl + Mayús + F	
Word Count						
Clear Console					Ctrl + L	

Figura 1.3-19: Menú **Edit**.

**Back/Fordward:** Rehacer/Deshacer una acción hecha en el script de trabajo.

**Undo/Redo:** Deshace/Rehace la última acción realizada/rechazada en la ventana del editor.

**Cut/Copy/Paste:** Corta/ copia/pega cualquier conjunto de texto de la ventana del editor.

**Select All:** Selecciona todas las líneas de un script

**Folding:** Permite mostrar y ocultar fácilmente los bloques de código para que sea más fácil navegar por el archivo del código fuente. También se puede desde esta opción poder duplicar una sección arbitraria de código.

**Go to Line...:** Permite ir a una línea determinada

**Find, Find Next, Find Previous, Use Selection for Find, Replace and Find:** Permiten buscar ocurrencias de cadenas, la siguiente ocurrencia, la anterior, utilizar una selección para la búsqueda o sustituir la cadena por otra tras encontrarla.

**Find in Files...** Permite buscar de forma recursiva todos los archivos para cada ocurrencia de una cadena dada en un directorio específico (ver Figura 1.3-20).

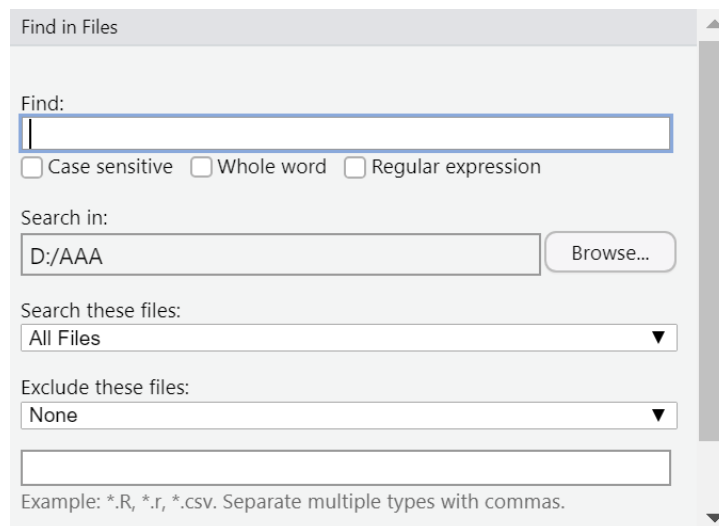


Figura 1.3-20: Menú **Edit**→**Find in Files...**

Se puede personalizar aún más la búsqueda con expresiones regulares y filtros para tipos de archivos específicos. El resultado de la búsqueda se mostrará en el panel junto a la consola (por defecto).



**Word Count:** Contador de palabras.

**Clear Console:** Limpia por completo la consola, pero no borra los objetos que se hayan almacenado anteriormente en la memoria.

#### 1.3.4.3. EL MENÚ Code

Desde el menú **Code** se pueden acceder a muchas opciones que permiten flexibilizar la ejecución de código R directamente desde el editor. La Figura 1.3-21 muestra todas las opciones disponibles. No vamos a nombrar la funcionalidad de estas opciones, únicamente iremos viendo aquellas que vayamos necesitando en cada momento.

Insert Section...	Ctrl+Shift+R
Jump To...	Alt+Shift+J
Go To File/Function...	Ctrl+.
Show Document Outline	Ctrl+Shift+O
Soft Wrap Long Lines	
Rainbow Parentheses	
Show Diagnostics	
Go To Help	
Go To Function Definition	
Extract Function	Ctrl+Alt+X
Extract Variable	Ctrl+Alt+V
Rename in Scope	Ctrl+Alt+Shift+M
Reflow Comment	Ctrl+Shift+ /
Comment/Uncomment Lines	Ctrl+Shift+C
Insert Roxygen Skeleton	Ctrl+Alt+Shift+R
Reindent Lines	Ctrl+I
Reformat Code	Ctrl+Shift+A
Run Selected Line(s)	Ctrl+Enter
Re-Run Previous	Ctrl+Alt+P
Run Region	▶
Run Selection as Local Job	
Terminal	▶
Source	Ctrl+Shift+S
Source with Echo	Ctrl+Shift+Enter
Source File...	Ctrl+Alt+G

Figura 1.3-21: Menú **Code**.

#### 1.3.4.4. EL MENÚ View

La Figura 1.3-22 muestra las opciones disponibles en el menú **View**. Se comentan a continuación algunas de ellas.

Figura 1.3-22: Menú **View**.

**Hide/Show Toolbar:** Oculta/ Muestra la barra de herramientas.

**Panes:** Las distintas opciones contenidas en Panes permiten mostrar las distintas partes de la consola de RStudio o ampliar una determinada ventana. También permite seleccionar la distribución de paneles que se prefiera.

**Zoom In/Zoom Out:** Realiza un zoom sobre cada una de las ventanas aumentando/disminuyendo el tamaño de su contenido.

**Switch To Tab:** Permite cambiar de pestaña para visualizar cualquier hoja de edición.

**Next Tab/Previous Tab:** Permite cambiar a la pestaña siguiente/anterior para visualizar la hoja de edición siguiente/anterior.

**First Tab/Last Tab:** Permite cambiar a la primera/última pestaña para visualizar la primera/última hoja de edición.

**Move Focus To Source/Move Focus To Console:** Mueve el cursor a la ventana de edición/consola de trabajo desde cualquier otra ventana.

**Move Focus To Help:** Mueve el cursor a la pestaña de ayuda.

**Show History:** Muestra todo el código que se ha ejecutado en la consola desde la última vez que se eliminó el historial.

**Show Files/Show Plots/ Show Packages:** Muestra el conjunto de ficheros existentes en el directorio de trabajo/conjunto de gráficos que se han generado/conjunto de paquetes que el programa tiene instalados hasta el momento.

**Show Environment:** Muestra el conjunto de objetos que se han guardado en la memoria del programa.

**Show Viewer:** Muestra el panel Visor (**Viewer**).

#### 1.3.4.5. EL MENÚ Plots

La Figura 1.3-23 muestra las opciones disponibles en el menú **Plots**.

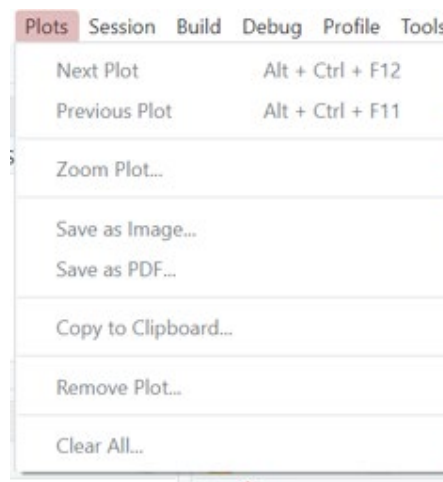


Figura 1.3-23: Menú **Plots**.

**Next Plot/ Previous Plot:** Muestra el gráfico siguiente/anterior.

**Zoom Plot...:** Abre una nueva ventana en la que se muestra el gráfico seleccionado.

**Save Plot as Image.../Save Plot as PDF...:** Guarda el gráfico seleccionado como una imagen (.png, .jpg, .tiff, .bmp, .metafile, .svg, .eps)/en pdf

**Copy Plot to Clipboard:** Copia el gráfico en un portapapeles.

**Remove Plot...:** Elimina el gráfico seleccionado

**Clear All...:** Elimina todos los gráficos creados.

#### 1.3.4.6. EL MENÚ Session

La Figura 1.3-24 muestra las opciones disponibles en el menú **Session**.

**New Session:** Inicia una nueva sesión de RStudio.

**Interrupt R:** Permite interrumpir algún proceso interno que no queremos que finalice.

**Restart R:** Permite actualizar la sesión en la que estemos trabajando.

**Terminate R:** Permite eliminar toda la información creada en una sesión, pero sin eliminar lo escrito en la ventana de edición.

**Set Working Directory:** Permite configurar el directorio de trabajo.

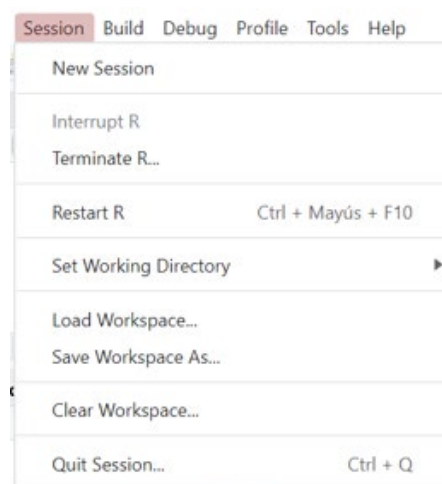


Figura 1.3-24: Menú **Session**.

**Load Workspace/Save Workspace As/Clear Workspace:** Permite cargar/guardar/eliminar un determinado espacio de trabajo.

**Quit Session:** Cierra la sesión actual.

#### 1.3.4.7. EL MENÚ Build

La Figura 1.3-25 muestra las opciones disponibles en el menú **Build**.

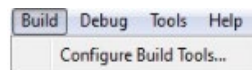


Figura 1.3-25: Menú **Build**.

**Configure Build Tools:** Permite construir paquetes y herramientas dentro de un proyecto creado por el usuario.

#### 1.3.4.8. EL MENÚ Debug

La Figura 1.3-26 muestra las opciones disponibles en el menú **Debug**.

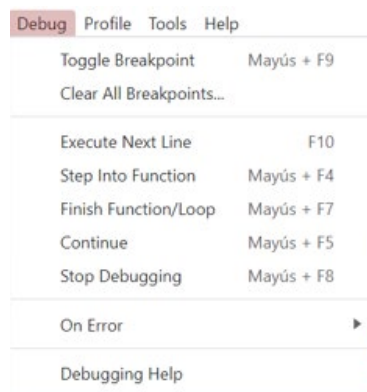


Figura 1.3-26: Menú **Debug**.

**Toggle Breakpoint:** Permite introducir un punto de interrupción en el texto de la ventana de edición con la finalidad de averiguar si, hasta la línea donde se coloca dicho punto, la ejecución del texto es correcta.

**Clear All Breakpoints:** Permite eliminar todos los puntos de interrupción que se hayan creado hasta el momento.

**Execute Next Line:** Permite ejecutar la siguiente línea tras un punto de interrupción. Nos sirve para ejecutar línea a línea.

**Finish Function/Loop** Permite finalizar un bucle.

**Continue:** Permite continuar con la ejecución una vez que se ha detenido dicha ejecución en el punto de interrupción.

**Stop Debugging:** Detiene la depuración.

**On Error:** En caso de error, permite elegir entre que sólo salga un mensaje de aviso, que se inspeccione el error o que no ejecute más código a partir del error.

**Debugging Help:** Muestra la página web del programa con la ayuda sobre la depuración de errores.

#### 1.3.4.9. EL MENÚ Profile

Desde este menú se puede cargar la herramienta Profvis (<https://rstudio.github.io/profvis/>) que nos sirve para analizar el código R, para poder optimizarlo.

#### 1.3.4.10. EL MENÚ Tools

La Figura 1.3-27 muestra las opciones disponibles en el menú **Tools**.

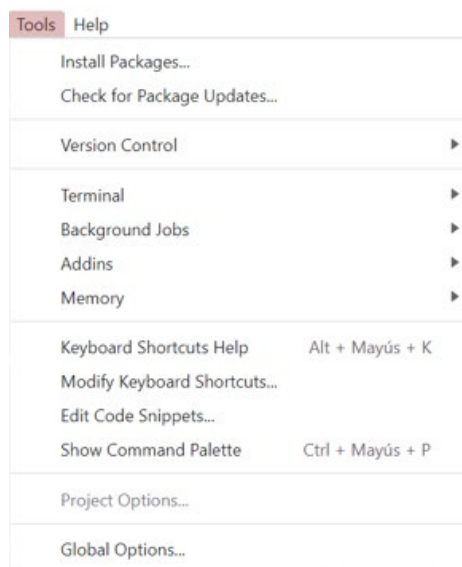


Figura 1.3-27: Menú **Tools**.

**Install Packages....** Permite instalar paquetes. Es importante que dentro de esta opción esté marcada la opción de Install dependencies (ver Figura 1.3-28).

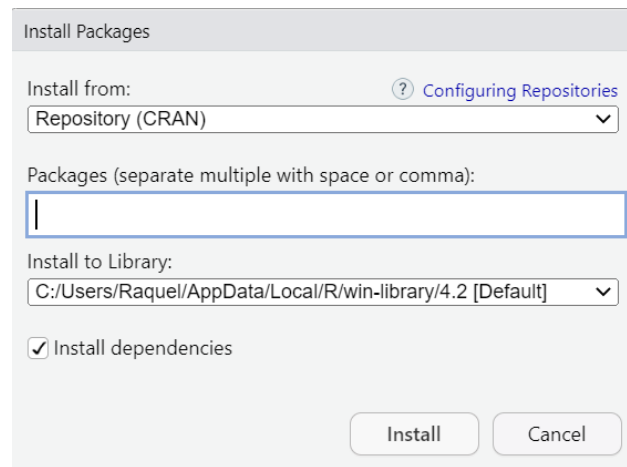


Figura 1.3-28: Menú **Tools**→**Install Packages**.

**Check for Packages Updates....** Permite actualizar los paquetes seleccionados.

**Version Control** (Figura 1.3-29): Permite controlar varios proyectos a la vez, hacer copias de seguridad de los proyectos...

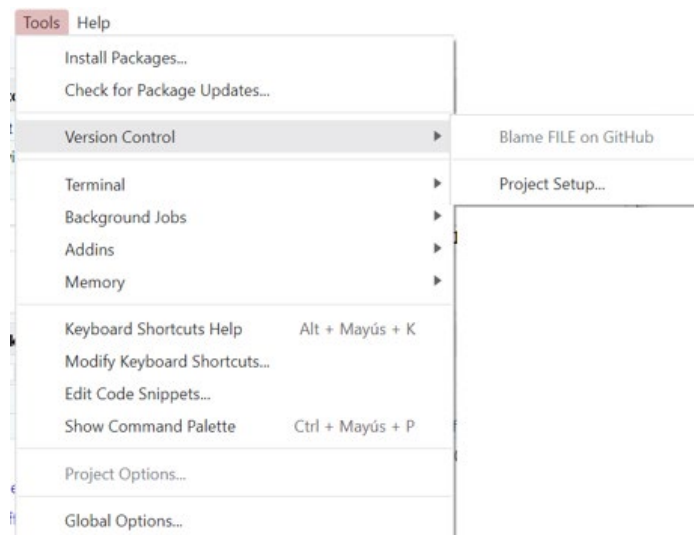


Figura 1.3-29: Menú **Tools**→**Version Control**.

**Background jobs:** Permite ejecutar scripts en segundo plano mientras se continúa usando el IDE (Figura 1.3-30)

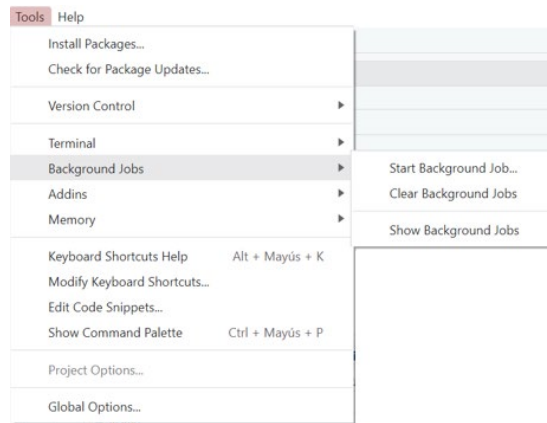


Figura 1.3-30: Menú **Tools** → **Background jobs**.

**Addins:** posibilita añadir “extensiones” a RStudio

**Keyboard Shortcuts Help:** ayuda de métodos abreviados de teclado.

**Modify Keyboard Shorcuts....:** permite modificar los métodos abreviados de teclado.

**Project Options:** Permite establecer opciones sobre un Proyecto en particular.

**Global Options....:** Muestra las opciones generales de RStudio.

#### 1.3.4.11. EL MENÚ Help

La Figura 1.3-31 muestra las opciones disponibles en el menú **Help**.

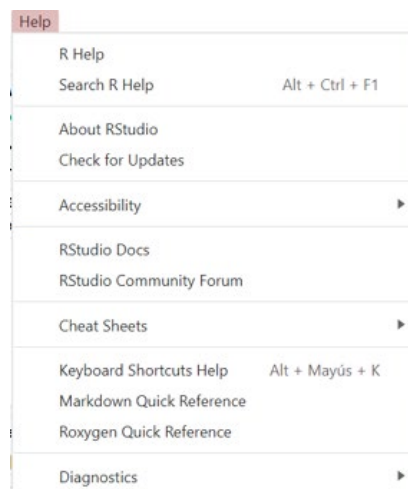


Figura 1.3-31: Menú **Help**.



**R Help:** Muestra la ayuda del programa.

**Search R Help:** Se accede al campo de búsqueda de la pestaña Help.

**About RStudio:** Muestra información sobre RStudio.

**Check For Updates:** Permite realizar una búsqueda en la última versión con la finalidad de obtener la última actualización de dicho programa.

**Accessibility:** Permite acceder a distintas opciones de accesibilidad de la herramienta.

**RStudio Docs:** Muestra la página web del programa en la que se explica la documentación con la que se puede trabajar en RStudio.

**RStudio Community Forum:** Muestra la página web del programa en la que hay un foro sobre distintos temas de RStudio.

**Cheatsheets:** permite acceder a la información de determinados paquetes muy utilizados como por ejemplo ggplot (Figura 1.3-32).

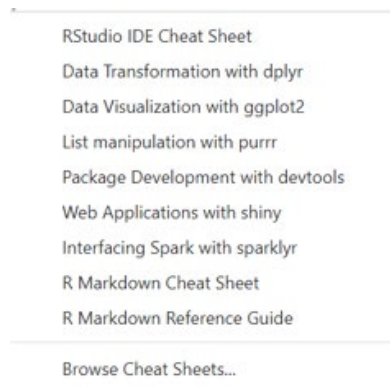


Figura 1.3-32: Menú **Help**→**Cheatsheets**.

**Markdown Quick Reference:** acceso a guía rápida para crear documentos dinámicos utilizando R Markdown.

**Keyboard Shortcuts Help:** guía de referencia de abreviaciones por teclado.

**Roxygen Quick Reference:** acceso a la ayuda de `roxygen2`, paquete de R que permite documentar un paquete de R.

**Diagnostics:** Permite realizar algunas opciones sobre diagnósticos del programa (ver Figura 1.3-33 ).

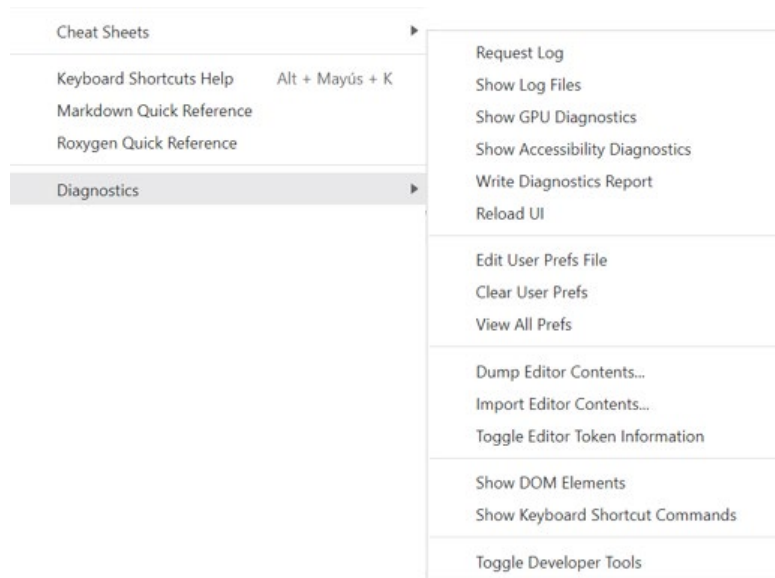


Figura 1.3-33: Menú **Help**→**Diagnostics**.

## 2. INTRODUCCIÓN AL LENGUAJE R

### 2.1. Tipos de datos en R

#### 2.1.1. Definición de tipos

R permite trabajar distintos **tipos de datos**. Los más comunes son:

**Numeric:** datos de tipo numérico. A su vez, dentro del tipo numérico se pueden distinguir dos subtipos de datos: Integer, para los datos de tipo entero y Double para datos de tipo real. Double es el tipo de dato numérico por defecto en R.

**Logical:** Para datos de tipo lógico o binario/booleano. Los valores que pueden tomar las variables de tipo lógico son TRUE o FALSE

**Character:** Para caracteres o cadenas de caracteres.

#### 2.1.2. Conversión de tipos de datos

R proporciona funciones para identificar el tipo de dato de un objeto y convertirlo a un tipo diferente. Las funciones cuyo nombre tiene la forma `is.tipodato()` devuelven TRUE o FALSE dependiendo de que el objeto pasado como argumento sea o no del tipo `tipodato`. Por el contrario, las funciones cuyo nombre es de la forma `as.tipodato()` realizan la conversión del argumento al tipo `tipodato`. En la Tabla 2.1-1 se muestran las funciones disponibles.

Comprobación	Conversión
<code>is.numeric()</code>	<code>as.numeric()</code>
<code>is.character()</code>	<code>as.character()</code>
<code>is.vector()</code>	<code>as.vector()</code>
<code>is.matrix()</code>	<code>as.matrix()</code>
<code>is.data.frame()</code>	<code>as.data.frame()</code>
<code>is.factor()</code>	<code>as.factor()</code>
<code>is.logical()</code>	<code>as.logical</code>

Tabla 2.1-1: Funciones para la comprobación y conversión de tipos de datos

## 2.2. Estructuras de datos en R

Sea cual sea el tipo de datos que manejemos, el primer paso en el análisis de datos es crear una estructura de datos que contenga los datos a estudiar. El término estructura de datos en computación describe las herramientas que nos permiten almacenar y procesar información. En R hay básicamente los siguientes tipos de estructuras de datos: variables, vectores, matrices, arrays, data frames y listas. Resumidamente, lo que diferencia una estructura de otra es lo siguiente:

Hay **estructuras** que pueden tener varias dimensiones, pero **admiten únicamente un tipo de dato**. Dependiendo del número de dimensiones distinguimos:

- **Vectores.** Una dimensión
- **Matrices.** Dos dimensiones
- **Arrays.** Tres o más dimensiones.

**Factores.** Se utilizan para almacenar variables cualitativas en las que no existe un orden específico entre las categorías.

**Factores ordenados.** Almacenan variables cualitativas ordinales, es decir, variables cualitativas con un cierto orden entre sus categorías.

**Data frames.** Admiten variables de distintos tipos, pero con la misma longitud.

**Listas.** La lista es la estructura de datos más flexible puesto que permite incluir variables de tipos y longitudes diferentes.

### 2.2.1. Variables en R

Una variable, en R, es un símbolo que usamos para referirnos a un valor. Por ejemplo, si tecleamos en la Consola de Comandos

```
a = 2
```

y ejecutamos esa instrucción, entonces, aunque aparentemente no ha sucedido nada (porque no hay respuesta), a partir de ese momento R ha asignado el valor 2 al símbolo `a`. Así que si, por ejemplo, tecleamos y ejecutamos

```
a + 1
```

obtendremos como respuesta 3.

\*\*\*\*\*

## EJERCICIO 2.

¿Qué se obtiene al ejecutar estos comandos, uno detrás de otro? ¿Cuánto valen las variables *a*, *b* y *c* al final?

```
a = 2
b = 3
c = a + b
a = b * c
b = (c - a)^2
c = a * b
```

\*\*\*\*\*

Como acabamos de ver, las variables en R pueden almacenar valores numéricos (variables cuantitativas). Además, pueden almacenar valores de variables cualitativas (factores).

Por ejemplo, si ejecutamos el código:

```
estado = "Nuevo"
```

veremos que R lo admite. “Nuevo” es un valor perfectamente aceptable.

En R los valores que representan palabras, frases, o como las llamaremos técnicamente *cadenas alfanuméricas*, se escriben siempre entre comillas. Podemos usar comillas simples o dobles.

### 2.2.1.1. ASIGNACIONES

Las instrucciones como

```
a = 3
```

o como

```
b = (c - a)^2
```

se denominan *asignaciones*. En la primera, por ejemplo, decimos que hemos asignado el valor 3 a la variable `a`.

Como vimos en un ejemplo de secciones anteriores, en R se pueden escribir las asignaciones empleando el símbolo `<-` (el signo `_menor_` seguido, sin espacios, de un signo `_menos_`). Así que, si ejecutamos

```
a <- 3
```

el resultado será el mismo que antes con `=`

#### 2.2.1.2. VISUALIZAR LOS RESULTADOS DE UN CÁLCULO

Al hacer una asignación, R no produce ninguna respuesta, y se limita a guardar el valor en la variable. En una asignación sencilla como `a<-3`, esto no genera ninguna duda. Pero si el miembro derecho de la asignación es una expresión más complicada, puede ser conveniente pedirle a R que nos muestre el resultado de esa expresión a la vez que se hace la asignación.

Por ejemplo, si queremos saber el valor de la variable `a` tras ejecutar las líneas de código siguiente:

```
b = 3
c = -2
d = -2
a = b * c^2 / d + 3
```

Basta con ejecutar una línea adicional en la que aparece el nombre de la variable que queremos inspeccionar. Por ejemplo, si quiero saber el valor de la variable `a` ejecutaríamos la siguiente sentencia y R nos ofrece la respuesta.

```
> a
[1] -3
```

Otra manera de conseguir que R nos muestre el resultado de la asignación es encerrar toda la asignación entre paréntesis.

```
> (a = b * c^2 / d + 3)
[1] -3
```

#### 2.2.1.3. NOMBRES DE LAS VARIABLES

Aunque hasta ahora hemos usado letras como nombres de las variables, podemos utilizar nombres más descriptivos para las variables.

Por ejemplo, si hace tiempo hemos escrito las siguientes instrucciones para resolver un problema:

```
a = 2
b = 3
c = a / b
```

Puede que pasado el tiempo no recordemos qué era lo que estábamos tratando de conseguir al hacer esto. En cambio, al ver estas instrucciones:

```
espacio = 2
tiempo = 3
velocidad = espacio / tiempo
```

es mucho más fácil recordar lo que hicimos.

En R las reglas para los nombres de variables son muy flexibles: esencialmente que empiecen por una letra y no contengan espacios ni caracteres especiales como ?, +, paréntesis, etcétera. Pero cuidado con los excesos. Es cierto que podemos usar nombres de variables arbitrariamente largos. Pero si usamos como nombre:

```
Estavariablenalmacenaelresultadodelaoperacionqueacabamosdehacer
```

nuestro trabajo resultará más ilegible.

## 2.2.2. Vectores en R

### 2.2.2.1. CONSTRUYENDO VECTORES

La estructura vector almacena una colección ordenada de elementos todos del mismo tipo.

Para crear un vector se utiliza la función `c`, que concatena todos los elementos que recibe como argumentos:

```
c(elemento1, elemento2, elemento3, ...)
```

Supongamos que queremos almacenar una colección de números que son las temperaturas medias en meses sucesivos en un lugar:

22, 21.8, 18, 19.2, 17, 21.7, 18, 20, 17, 18.9, 17, 25

Para almacenar en R este conjunto de números tenemos que definir el siguiente vector:

```
temperaturas = c(22, 21.8, 18, 19.2, 17, 21.7, 18, 20, 17, 18.9, 17, 25)
```

En esta expresión primero hemos creado el vector con los datos en la parte derecha de la expresión utilizando para ello la función `c()` que, en este contexto, puede tener un número arbitrario de valores como argumento y cuyo valor es el vector obtenido mediante la concatenación de todos ellos. Una vez creado el vector lo hemos asignado a la variable `temperaturas`. Hasta ahora sólo habíamos usado variables para identificar un único valor (un número o una cadena alfanumérica), pero como vemos una variable puede usarse para identificar un vector o estructuras más complejas.

Se puede construir un vector de tipo numérico, lógico o alfanumérico (caracteres). Ejemplos de creación de vectores son:

<pre>&gt; c(1,5,3,2) [1] 1 5 3 2</pre>	Vector numérico de 4 elementos
<pre>&gt; c(T,F,T,F) [1] TRUE FALSE TRUE FALSE</pre>	Vector lógico de 4 elementos
<pre>&gt; c("madrid","tarragona","lerida") [1] "madrid" "tarragona" "lerida"</pre>	Vector de 3 cadenas de caracteres
<pre>&gt; c(1:10) [1] 1 2 3 4 5 6 7 8 9 10</pre>	Vector numérico de 10 elementos
<pre>&gt; c(1:3,2,7,2:-3) [1] 1 2 3 2 7 2 1 0 -1 -2 -3</pre>	Vector numérico de 11 elementos

El símbolo de dos puntos, situado entre dos números, construye un vector de modo sencillo, tanto en orden ascendente como descendente.

Como hemos comentado, un vector está formado por elementos del mismo tipo. Si se mezclan números y cadenas de caracteres, R no da error. Lo que hace es convertir internamente los datos almacenados en el vector a cadenas alfanuméricas. Si se mezclan números reales y complejos, se obtendrá un vector de números complejos.

#### 2.2.2.2. MANEJO DE VECTORES

##### • Concatenación de vectores

La concatenación de dos vectores se puede realizar utilizando la función `c()`. Por ejemplo:

```
> x = c(1,3,5)
> y = c(2,4,6)
> c(x,y)
[1] 1 3 5 2 4 6
```

Supongamos que después de haber creado nuestra lista de temperaturas medias queremos incorporar unas temperaturas nuevas a nuestro vector de datos que son:

22, 18, 20, 21, 20



Podemos hacerlo mediante concatenación. Empezamos creando un nuevo vector:

```
(temperaturas2 = c(22, 18, 20, 21, 20))
```

Y a continuación concatenamos los vectores, usando de nuevo la misma función `c`, de esta manera:

```
(temperaturas2 = c(temperaturas, temperaturas2))
```

La instrucción `c(temperaturas, temperaturas2)` toma los datos contenidos en `temperaturas` y `temperaturas2` y los concatena, en ese orden, para fabricar un nuevo vector, que todavía no está guardado en ninguna variable. La asignación `temperaturas = c(temperaturas, temperaturas2)` toma ese vector y lo guarda en la propia variable `temperaturas`. Al hacer esto, como era de esperar, el anterior vector `temperaturas` es reemplazado con la nueva lista, y su contenido anterior se pierde.

La Figura 2.2-1 resume lo que hemos venido haciendo con estos datos:

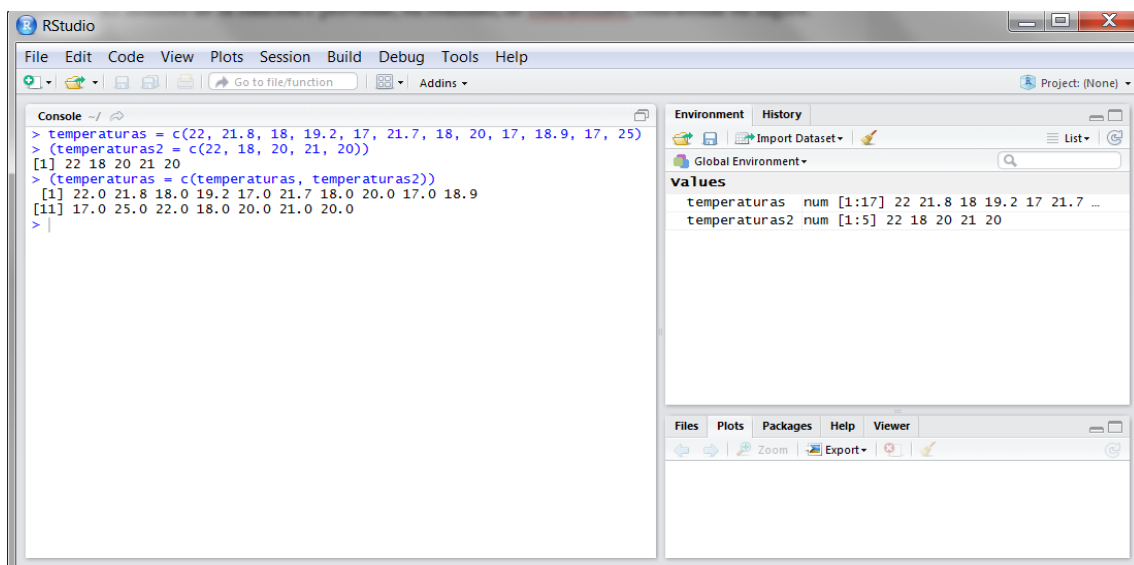


Figura 2.2-1: Concatenando vectores.

Los números entre corchetes que aparecen en la figura al principio de cada fila, como el [1] y el [11] son sólo ayudas para que podamos localizar los elementos del vector. La primera fila en este caso muestra a partir del primer elemento y la segunda fila a partir del undécimo. Dependiendo de la anchura de la ventana de RStudio estos números pueden variar.

### • Manejo de los elementos de un vector

Para *acceder a los elementos* de un vector se utilizan los corchetes. Por ejemplo, si queremos acceder a la segunda componente del vector `temperaturas` ejecutamos:

```
> temperaturas[2]
[1] 21.8
```

Para *extraer determinados elementos* de un vector tenemos que especificar los índices de los elementos a extraer. Por ejemplo, la orden:

```
> x = c(18,11,12,10,7,6,17)
> x[c(1,3,6)]
[1] 18 12 6
```

extrae los elementos 1, 3 y 6 del vector.

Un número negativo precediendo al índice significa **exclusión**. Con el vector `x` creado anteriormente:

```
> x[-3]
[1] 18 11 10 7 6 17
> x[-c(1,2)]
[1] 12 10 7 6 17
```

También es posible *especificar una condición lógica*. En el caso del vector `x` creado antes:

```
> x>10
[1] TRUE TRUE TRUE FALSE FALSE FALSE TRUE
> x[x>10]
[1] 18 11 12 17
```

En el caso de un vector de variables, podemos utilizar los nombres de las variables para extraer los elementos. Por ejemplo, si creamos tres variables `A`, `B` y `C` con los valores 1, 3 y 5 respectivamente, y, a continuación, creamos un vector `y` formado por dichas variables, y después extraemos el valor referenciado por la variable `B` tenemos que escribir el código:

```
> A = 1
> B = 3
> C = 5
> y = c(A,B,C)
> y
[1] 1 3 5
> y[B]
[1] 5
```

### 2.2.2.3. ALGUNAS FUNCIONES SOBRE VECTORES

#### • Creación de patrones: `from:to`, `seq` y `rep`

Hay varias órdenes para crear vectores de modo automático.

`from:to`

```
> 1:5
[1] 1 2 3 4 5
```

Genera números enteros entre 1 y 5.

La función `seq()` permite construir vectores que son sucesiones equiespaciadas. Esta función posee los siguientes argumentos:

**from** Valor inicial de la sucesión

**to** Valor final de la sucesión

**by** Espaciado entre los elementos

**length** Longitud del valor resultante

**along** Un objeto cuya longitud se usará para el objeto a construir.

Si no se indica algún argumento, por defecto vale 1

```
> seq(1, 100, length = 5)
[1] 1.00 25.75 50.50 75.25 100.00
> x <- c(1: 7)
> x [seq(1, 7, by = 2)]
[1] 1 3 5 7
```

Si queremos generar números entre 1 y 6 podemos hacer, por ejemplo:

```
> seq(1,6)
[1] 1 2 3 4 5 6
> seq(1,6,by=0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
> seq(1,6,length=10)
[1] 1.000000 1.555556 2.111111 2.666667 3.222222 3.777778 4.333333 4.888889
[9] 5.444444 6.000000
```

En el segundo caso se va sumando al número anterior 0.5 hasta que se llega a 6. En el tercer caso hemos ordenado generar una secuencia de 10 números entre 1 y 6.

La generación de valores repetidos la podemos hacer utilizando la función `rep`.

```
> rep(1,5)
[1] 1 1 1 1 1
> rep(c(1,2),5)
[1] 1 2 1 2 1 2 1 2 1 2
> rep(1:4,2)
[1] 1 2 3 4 1 2 3 4
> rep(1:3,c(1,4,5))
[1] 1 2 2 2 2 3 3 3 3 3
```

En el primer caso se ha repetido el 1 cinco veces. En el segundo, se ha repetido el patrón (1,2) cinco veces. En el tercer caso, la secuencia 1,2,3,4 se ha repetido 2 veces. En el cuarto caso, la secuencia 1,2,3 ha sido repetida de acuerdo con el vector (1,4,5), esto es, 1 vez el 1, 4 veces el 2 y 5 veces el 3.

- **Asignación de nombre a las componentes de un vector: `names()`**

Mediante la función `names()` podemos asignar nombre a los elementos de un vector. De hecho, esta función permite dar nombre a los elementos de cualquier objeto. Aplicada a un vector se hace de la siguiente forma:

```
> x<-1:5
>x
[1] 1 2 3 4 5
>names(x)<-c("I","II","III","IV","V")
>x
I  II III  IV   V
```

- **Conocer o modificar el tipo de un objeto: `mode()`**

La función `mode()` devuelve o modifica el tipo de un objeto.

```
>x = 1:5
>mode(x)
[1] "numeric"
```

- **Valores faltantes: `NA`**

El símbolo de valor faltante es `NA` (significa Not Available). Cualquier operación aritmética que involucre a un `NA` da por resultado un `NA`. Esto se aplica también a los operadores lógicos tales como `<`, `<=`, `>`, `>=`, `=`, `!=` (`=` es para comprobar si dos objetos son iguales, y `!=` comprueba si dos objetos son distintos). Veamos algunos ejemplos:

```
> x = c(1,2,3,NA,4,5)
> x
[1] 1 2 3 NA 4 5
> is.na(x)
```

```
[1] FALSE FALSE FALSE TRUE FALSE FALSE
> x[x>2]
[1] 3 NA 4 5
> x*2
[1] 2 4 6 NA 8 10
```

`is.na(x)` pregunta qué elementos de `x` son faltantes. Sólo da valor cierto para el cuarto. La siguiente orden pregunta qué valores de `x` superan a 2. La última multiplica cada elemento de `x` por 2.

```
> x
[1] 1 2 3 NA 4 5
> x<-x[!is.na(x)]
> x
[1] 1 2 3 4 5
```

En este caso vemos que `x` contiene un valor faltante. La siguiente instrucción selecciona los valores de `x` no faltantes y asigna el resultado al mismo vector `x`. De esta manera eliminamos los valores faltantes del vector `x`.

#### 2.2.2.4. VECTORES DE VARIABLES CUALITATIVAS

Los vectores de variables cualitativas se conocen como factor y factor ordenado. Vamos a estudiar qué son y cómo se crean.

##### • Factor

Un factor es un vector que se utiliza para almacenar variables cualitativas en las que no existe un orden específico entre las categorías. Se usa para especificar una clasificación discreta de los componentes de otros vectores de la misma longitud

Los factores en R se crean utilizando la función `factor`. Esta función necesita como argumentos el conjunto de datos y los niveles o categorías del factor:

```
factor(x, levels,...)
```

#### **EJEMPLO: Creando vectores de variables cualitativas con `factor`**

Supongamos que disponemos de una muestra de 30 personas de Australia de tal modo que su estado o territorio se especifica mediante un vector de caracteres con las abreviaturas de los mismos:

```
> estado <- c("tas", "sa", "qld", "nsw", "nsw", "nt", "wa", "wa",
              "qld", "vic", "nsw", "vic", "qld", "qld", "sa", "tas",
              "sa", "nt", "wa", "vic", "qld", "nsw", "nsw", "wa",
```

```
"sa", "act", "nsw", "vic", "vic", "act")
```

Un factor se crea utilizando la función `factor()`:

```
> FactorEstado <- factor(estado)
> FactorEstado
[1] tas sa qld nsw nsw nt wa wa qld vic nsw vic qld qld sa
[16] tas sa nt wa vic qld nsw nsw wa sa act nsw vic vic act
Levels: act nsw nt qld sa tas vic wa
```

Podemos utilizar la función `levels()` para ver los niveles de un factor:

```
> levels(FactorEstado)
[1] "act" "nsw" "nt" "qld" "sa" "tas" "vic" "wa"
```

Supongamos que disponemos en otro vector de los ingresos de las mismas personas (medidos con unas unidades apropiadas)

```
> ingresos <- c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42, 56,
61, 61, 61, 58, 51, 48, 65, 49, 49, 41, 48, 52, 46, 59, 46, 58, 43)
```

Para calcular la media para cada estado podemos usar la función `tapply()`:

```
> MediaIngresos <- tapply(ingresos, FactorEstado, mean)
```

que devuelve el vector de medias con las componentes etiquetadas con los niveles:

```
> MediaIngresos
act    nsw    nt    qld    sa    tas    vic    wa
44.500 57.333 55.500 53.600 55.000 60.500 56.000 52.250
```

La función `tapply()` aplica una función, en este ejemplo la función `mean()`, a cada grupo de componentes del primer argumento, en este ejemplo `ingresos`, definidos por los niveles del segundo argumento, en este ejemplo `FactorEstado`, como si cada grupo fuese un vector por sí solo. El resultado es una estructura cuya longitud es el número de niveles del factor. Podemos consultar la ayuda para obtener más detalles.

\*\*\*

### • Factor Ordenado

Un factor ordenado almacena variables cualitativas ordinales, es decir, variables cualitativas con un cierto orden entre sus categorías.

Los factores ordenados en R se crean utilizando la función `ordered`. Esta función necesita como argumentos el conjunto de datos y los niveles o categorías del factor:

```
ordered(x, levels,...)
```

Un detalle muy importante que hay que tener en cuenta a la hora de definir los niveles de un factor ordenado es que R considera que los niveles vienen dados de menor a mayor, es decir, que si dentro de la función `ordered` se especifica que `levels = c(2, 1, 3)`, R entiende que la primera categoría es menor que la segunda y que la segunda es, a su vez, menor que la tercera, esto es, que  $2 < 1 < 3$ .

### **EJEMPLO: Creando vectores de variables cualitativas con `ordered`**

Supongamos que tenemos un vector con el nivel de inglés de 10 estudiantes:

```
> nivel.ingles = c("medio", "medio", "bajo", "medio", "bajo", "medio",
", "alto", "alto", "bajo", "bajo" )
> nivel.ingles
[1] "medio" "medio" "bajo" "medio" "bajo" "medio" "alto" "alto" "bajo" "bajo"
```

Ahora creamos un factor ordenado con el nivel de inglés de los estudiantes:

```
> fnivel.ingles = ordered(nivel.ingles, levels=c("bajo","medio","alto"))
> fnivel.ingles
[1] medio medio bajo medio bajo medio alto alto bajo bajo
Levels: bajo < medio < alto
```

Si ahora queremos saber qué estudiantes tienen un nivel de inglés por debajo de "medio":

```
> fnivel.ingles<"medio"
[1] FALSE FALSE TRUE FALSE TRUE FALSE FALSE FALSE TRUE TRUE
***
```

\*\*\*\*\*

### **EJERCICIO 3.**

Supongamos que tenemos un vector con la población de origen de 15 estudiantes. Este vector debe contener como información la siguiente:

getafe, mostoles, madrid, madrid, mostoles, leganes, getafe, leganes, madrid, mostoles, parla, alcorcon, mostoles, getafe, leganes

Además, disponemos de las estaturas de cada uno de los estudiantes, que es la siguiente:

1.83, 1.71, 1.79, 1.64, 1.74, 1.81, 1.62, 1.84, 1.68, 1.81, 1.82, 1.74, 1.84, 1.61, 1.84

- Mostrar los niveles del factor (las poblaciones de origen), junto con el número de estudiantes correspondientes a tales niveles.
- Calcular la estatura promedio de los estudiantes de cada población a partir de la muestra de la que disponemos.

**Nota:** En el apartado a) hacer uso de la función `factor` y de la función `summary`. En el apartado b) hacer uso de la función `tapply` y de la función `mean`

\*\*\*\*\*

### 2.2.3. Matrices en R

Una matriz en R es un conjunto de objetos indizados por filas y columnas. Un array en R es lo mismo, salvo que puede tener más de dos dimensiones. Los datos que contiene una matriz deben ser todos del mismo tipo (todos numéricos, o de tipo carácter o lógicos, pero no mezclados).

#### 2.2.3.1. construyendo MATRICES

Las matrices se pueden definir utilizando la función `matrix`. Para ello, se ha de indicar qué datos va a tener la matriz, así como el número de filas y columnas.

La sintaxis general de la orden para crear una matriz utilizando la función `matrix` es la siguiente:

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)
```

donde:

<b>data:</b>	vector que contiene los valores que formarán la matriz.
<b>nrow:</b>	número de filas. Si no especifica, se toma <code>nrow = 1</code>
<b>ncol:</b>	número de columnas.
<b>byrow:</b>	variable lógica que indica si la matriz debe construirse por filas o por columnas. El valor predeterminado es F. Por defecto se colocan por columnas.

Modificando estos argumentos se pueden definir distintas matrices según las necesidades que se tengan.



**EJEMPLO: Creando matrices con `matrix`**

```
> matrix(1:6)
      [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```

```
> M = matrix( 1: 9, nrow = 3, byrow = TRUE) # la matriz se rellena por
filas
> M
```

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

```
> N = matrix( 1: 9, nrow = 3, byrow = FALSE) # la matriz se rellena p
or columnas
> N
```

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

```
> matrix(1:15, 3, 5)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15
```

Se puede utilizar la función `dimnames` para definir mediante una lista de longitud 2 los nombres de las filas y las columnas. Por ejemplo:

```
> matrix(1:10,5,2,dimnames=list (c("fila1","fila2","fila3","fila4","fi
la5"),c("columna1","columna2")))
      columna1 columna2
fila1         1         6
fila2         2         7
fila3         3         8
fila4         4         9
fila5         5        10
```

Cuando la longitud del vector no coincide con el número de filas y columnas, los elementos de la matriz se repiten, por lo que R te da un warning advirtiéndote que las salidas pueden no ser las deseadas. Por ejemplo:

```
> matrix( 1:15, 4, 6)
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]     1     5     9    13     2     6
[2,]     2     6    10    14     3     7
[3,]     3     7    11    15     4     8
[4,]     4     8    12     1     5     9
```

Warning message:  
In matrix(1:15, 4, 6) :  
data length [15] is not a sub-multiple or multiple of the number of  
rows [4]

\*\*\*

### 2.2.3.2. MANEJO DE MATRICES

#### • Seleccionar los elementos de una matriz

Supongamos que definimos la siguiente matriz N:

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

Para seleccionar el término que ocupa la posición primera fila, segunda columna, [1, 2] de la matriz N tenemos que escribir:

```
> N[1, 2]
[1] 4
```

Para seleccionar sólo una fila (columna), dejamos en blanco la posición de la columna (fila). Por ejemplo, para seleccionar la segunda fila de la matriz N:

```
> N[2, ]
[1] 2 5 8
```

Si lo que queremos es seleccionar la tercera columna de la matriz N:

```
> N[, 3]
[1] 7 8 9
```

La segunda y tercera fila las seleccionaríamos así:

```
> N[2:3, ]
      [,1] [,2] [,3]
[1,]     2     5     8
[2,]     3     6     9
```

Para seleccionar la segunda y tercera fila de las dos primeras columnas haríamos:

```
> N[2:3, 1:2]
      [,1] [,2]
[1,]    2    5
[2,]    3    6
```

#### • Asignando nombre a las filas y columnas de las matrices

Para asignar nombres a las filas y columnas de una matriz se utilizan las funciones `dimnames`, `colnames` y `rownames`.

#### **EJEMPLO:** Asignando nombres con `colnames` y `rownames`

Vamos a crear una matriz con tres personas y su edad, altura y peso como variables.

Se crea una matrix 3x3 y se muestran sus valores:

```
> datos=matrix(c(20,65,174,22,70,180,19,68,170),nrow=3,byrow=T)

> datos
      [,1] [,2] [,3]
[1,]   20   65  174
[2,]   22   70  180
[3,]   19   68  170
```

Se asignan nombres a las columnas y se vuelve a mostrar la matriz:

```
> colnames(datos) = c("edad","peso","altura")
> datos
      edad peso altura
[1,]   20   65   174
[2,]   22   70   180
[3,]   19   68   170
```

Se asignan nombres a las filas y se vuelve a mostrar la matriz en la que se verán los nombres asignados:

```
> rownames(datos) = c("paco","pepe","kiko")
> datos
      edad peso altura
paco   20   65   174
pepe   22   70   180
kiko   19   68   170
```

\*\*\*

Alternativamente podemos utilizar la función `dimnames` para asignar nombres a las filas y columnas de una matriz. Para ello tenemos que asignar como argumento de la función una lista de dos vectores de caracteres (los nombres de las filas y de las columnas de la matriz).

La sintaxis de la función `dimnames` es la siguiente:

```
dimnames(objeto) = list(vector de nombres de las filas, vector de
nombres de las columnas)
```

**EJEMPLO:** Asignando nombres con `dimnames` y accediendo por nombre a los elementos de una matriz

Se crea una matrix 3x3 y se muestran sus valores:

```
> datos = matrix(c(20,65,174,22,70,180,19,68,170),nrow=3,byrow=T)
> datos
```

```
      [,1] [,2] [,3]
[1,]   20   65  174
[2,]   22   70  180
[3,]   19   68  170
```

Se asignan nombres a las filas y a las columnas y se vuelve a mostrar la matriz en la que se verán los nombres asignados:

```
> dimnames(datos) = list(c("paco","pepe","kiko"),
c("edad","peso","altura"))
```

```
> datos
```

	edad	peso	altura
paco	20	65	174
pepe	22	70	180
kiko	19	68	170

Ahora podemos acceder también a los elementos de la matriz usando los nombres:

```
> datos[, "edad"]           # Edades de todas las personas
paco pepe kiko
20    22    19

> datos["pepe",]           # Variables del individuo "Pepe"
edad  peso altura
22    70    180

> datos[,c("edad","altura")] # Edad y altura de todas las personas
      edad altura
paco   20    174
```

```
pepe  22    180
kiko  19    170
```

```
> dimnames(datos)      # Muestra los nombres de filas y columnas
```

```
[[1]]
```

```
[1] "paco" "pepe" "kiko"
```

```
[[2]]
```

```
[1] "edad"  "peso"  "altura"
```

```
> apply(datos,2,mean)  # Hallamos la media de las variables edad,
                        # peso y altura (el 2 indica que se aplica la función media a las
                        # columnas)
```

```
edad      peso      altura
20.33333  67.66667 174.66667
```

\*\*\*

### 2.2.3.3. ALGUNAS FUNCIONES SOBRE MATRICES

En la siguiente tabla se recogen algunas funciones para trabajar con matrices. Parte de ellas ya han sido comentadas en las secciones anteriores.

dim	devuelve las dimensiones de una matriz
dimnames	devuelve el nombre de las filas y columnas de una matriz
colnames	devuelve el nombre de las columnas de una matriz
rownames	devuelve el nombre de las filas de una matriz
mode	devuelve el tipo de datos de los elementos de una matriz
length	devuelve el número total de elementos de una matriz
is.matrix	devuelve T si el objeto es una matriz, F si no lo es
[ , ]	accede a elementos dentro de la matriz
apply	Aplica una función sobre las filas o columnas de una matriz
cbind	Añade una columna a una matriz dada
rbind	Añade una fila a una matriz dada

**EJEMPLO: Funciones de interés sobre matrices**

```

> x = matrix(1:6,nrow=3)  # Creamos una matriz 3 x 2
> x                        # Se muestra la matriz x
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> length(x)               # Número de elementos de x
[1] 6

> mode(x)                 # Tipo de datos de la matriz x
[1] "numeric"

> dim(x)                  # Dimensiones de la matriz x
[1] 3 2

> dimnames(x)             # Nombre de las dimensiones de la matriz
NULL

> rownames(x)             # Nombre de las filas de la matriz
NULL

> colnames(x)             # Nombre de las columnas de la matriz
NULL

> is.matrix(x)            # El objeto x, ¿es una matriz?
[1] TRUE

> y<-c("blanco","negro")  # Creamos un vector de dos palabras

> is.matrix(y)            # El objeto y, ¿es una matriz?
[1] FALSE

> x[]                     # Se muestran todos los elementos de x
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

```

```

> x[1,2]                # Se muestra el elemento 1,2 de x
[1] 4

> x[1,]                 # Se muestra la primera fila de x
[1] 1 4

> x[,2]                 # Se muestra la segunda columna de x
[1] 4 5 6

> cbind(x,c(0,0,0))     # Se añade una columna de ceros a x
      [,1] [,2] [,3]
[1,]    1    4    0
[2,]    2    5    0
[3,]    3    6    0

> rbind(x,c(0,0))       # Se añade una fila de ceros a x
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
[4,]    0    0

```

\*\*\*\*\*

Se pueden utilizar las funciones `rbind()` y `cbind()` como alternativa a `matrix` para crear matrices “pegando” los vectores por filas y columnas.

Su sintaxis es la siguiente:

```

matriz = rbind (x1, x2,...)
matriz = cbind (x1, x2...)

```

donde en la función `rbind()`, `x1` y `x2`, son las filas de la matriz, y en la función `cbind()` las columnas.

**EJEMPLO: Creando matrices con cbind y rbind**

```
> v1 = c(1,2,3,4)
> v2 = c(5,6,7,8)
> v3= c(9,10,11,12)
> FA = rbind(v1,v2,v3)
> FA
```

	[,1]	[,2]	[,3]	[,4]
v1	1	2	3	4
v2	5	6	7	8
v3	9	10	11	12

```
> QA = cbind (v1, v2, v3)
> QA
```

	v1	v2	v3
[1,]	1	5	9
[2,]	2	6	10
[3,]	3	7	11
[4,]	4	8	12

```
> matriz= rbind (c(1,2,3), c(4,5,6), c(7,8,9))
> matriz
```

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

```
> matriz = cbind (c(1,2,3), c(4,5,6), c(7,8,9))
>matriz
```

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

\*\*\*\*\*

\*\*\*\*\*

**EJERCICIO 4.**

Construir una matriz de 14×3 donde los nombres de las columnas son las variables peso, altura y edad de 14 personas.

Los datos correspondientes al peso son:

77, 58, 89, 55,47,60,54,58,75,65,82,85,75,65

Los correspondientes a la altura:

1.63,1.63,1.85,1.62,1.60,1.63,1.70,1.65,1.78,1.70,1.77,1.83,1.74,1.65

Y los correspondientes a las edades son:



23,23,26,23,26,26,22,23,26,24,28,42,25,26

Realizar las siguientes acciones sobre esa matriz: Seleccionar la primera columna de dos formas diferentes, seleccionar un elemento, seleccionar una fila. Añadir a la matriz la variable sexo, en la última columna de la matriz, que contenga la siguiente información:

"H","M","H","H","M","M","H","M","M","H","H","H","M","M"

\*\*\*\*\*

## 2.2.4. Arrays en R

### 2.2.4.1. DEFINICIÓN DE ARRAYS

Un array es la generalización de una matriz de dos dimensiones al caso multidimensional. Su definición general es de la forma:

`array(datos, dimensiones, dimnames)`

siendo:

<b>data:</b>	vector que contiene los valores de los datos.
<b>dimensiones:</b>	dimensión del array. Vector de enteros de longitud uno o más que indican los índices máximos en cada dimensión
<b>dimnames:</b>	o NULL o los nombres para las dimensiones.

### 2.2.4.2. MANEJO DE ARRAYS

Los comandos para manejar arrays son similares a los que se manejan para las matrices. Por ejemplo, si queremos crear un array tridimensional en el que la primera dimensión tenga 2 niveles, la segunda 3 y la tercera 2 y que contenga los datos del 1 al 12, escribiremos lo siguiente:

```
> array(1:12,c(2,3,2))
```

```
, , 1
```

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

```
, , 2
```

```
  [,1] [,2] [,3]
[1,]   7   9  11
[2,]   8  10  12
```

**EJEMPLO: Creando arrays y accediendo a sus elementos**

Vamos a crear un array con la edad media, el peso medio y la estatura media para hombres y mujeres de dos poblaciones: Mérida y Trujillo

```
> x<-array(c(45,46,65,55,170,167,48,49,68,56,169,165),c(2,3,2))

> dimnames(x) = list(c("hombres","mujeres"),c("edad","peso","altura"),
c("Mérida","Trujillo"))

> x
, , Mérida

      edad peso altura
hombres  45   65   170
mujeres  46   55   167

, , Trujillo

      edad peso altura
hombres  48   68   169
mujeres  49   56   165
```

Para acceder a los elementos del array:

```
# Nombre de las dimensiones del array
> dimnames(x)
[[1]]
[1] "hombres" "mujeres"

[[2]]
[1] "edad" "peso" "altura"

[[3]]
[1] "Mérida" "Trujillo"

# Datos para la población "Mérida"
> x[,,"Mérida"]
      edad peso altura
hombres  45   65   170
mujeres  46   55   167

# Datos de todos los hombres
> x["hombres",,]
      Mérida Trujillo
edad      45      48
peso      65      68
altura   170     169
```

```
# Edades de las personas
x[,"edad",]
      Mérida Trujillo
hombres    45      48
mujeres    46      49
```

\*\*\*\*\*

### 2.2.4.3. ALGUNAS FUNCIONES SOBRE ARRAYS

Vamos a aplicar ahora funciones a los elementos del array, utilizando la función `apply`.

```
# Media de las variables edad, peso y altura para hombres y mujeres,
sin distinguir de qué población son.
```

```
> apply(x,1,mean)
hombres mujeres
94.16667 89.66667
```

Por ejemplo, la primera variable se obtiene de calcular:

$$(45+65+170+48+68+169)/6 = 94.16667$$

```
# Media de la variable edad para toda la población
> apply(x,2,mean)
edad peso altura
47.00 61.00 167.75
```

En este caso la primera variable se obtiene de calcular:

$$(45+46+48+49)/4 = 47$$

```
# Media de todas las variables para las poblaciones de Mérida y Trujillo
> apply(x,3,mean)
      Mérida Trujillo
91.33333 92.50000
```

### 2.2.5. Data Frames en R

#### 2.2.5.1. DEFINICIÓN DE DATA FRAMES

Un data frame es una estructura de datos que generaliza a las matrices. Se trata de una tabla bidimensional en la que los datos almacenados en una columna pueden diferir de los datos almacenados en la otra (que las dos columnas no sean datos numéricos, por ejemplo). Sin embargo, en esta estructura, todos los elementos de una misma columna deben ser del mismo tipo.

La sintaxis para crear un data frame es:

```
> data.frame (datos1, datos2, ...)
```

donde los puntos suspensivos indican que pueden especificarse tantos conjuntos de datos como sea necesario.

## 2.2.5.2. CREACIÓN Y MANEJO DE DATA FRAMES

**EJEMPLO: Creando data frames**

La función `data.frame()` es la que utiliza R para crear los data frames. En este ejemplo aprenderemos a cómo crearlos. Creamos en primer lugar la siguiente matriz que contiene los datos de edad, peso y altura de tres personas:

```
> datos = matrix(c(20,65,174,22,70,180,19,68,170),nrow=3,byrow=T)
> dimnames(datos)<-list(c("paco","pepe","kiko"),c("edad","peso","altura"))
> datos
```

	edad	peso	altura
paco	20	65	174
pepe	22	70	180
kiko	19	68	170

`edad`, `peso` y `altura` son las denominadas variables del data frame.

A continuación, vamos a añadir una columna a la matriz `datos` para que contenga la provincia de origen de cada persona:

```
> provincia = c("madrid","malaga","murcia")
> datos2 = cbind(datos,provincia)
> datos2
```

	edad	peso	altura	provincia
paco	"20"	"65"	"174"	"madrid"
pepe	"22"	"70"	"180"	"malaga"
kiko	"19"	"68"	"170"	"murcia"

Al añadir esta nueva columna todas las variables han sido convertidas a tipo carácter, lo que no nos conviene, porque si intentamos hacer algún tipo de cálculo obtendremos un error.

Por ejemplo, si sobre la matriz `datos` calculamos la media de la variable `edad` obtenemos el siguiente resultado:

```
> mean(datos[, "edad"])
[1] 20.33333
```

Sin embargo, si la media la hacemos utilizando la matriz `datos2` nos da el siguiente error:

```
> mean(datos2[, "edad"])

[1] NA
Warning message:
In mean.default(datos2[, "edad"]) :
  argument is not numeric or logical: returning NA
```

debido a que el tipo de datos de `datos2` no es el adecuado.

Podemos utilizar la función `mode` para ver qué tipo de datos hay en ambas matrices:

```
> mode(datos)
[1] "numeric"
> mode(datos2)           # Los datos han sido convertidos a carácter
[1] "character"
```

Lo que podemos hacer es utilizar una estructura de tipo data frame para poder añadir la columna de tipo carácter sin variar el tipo del resto de columnas de la siguiente manera:

```
> datos2 = data.frame(datos,provincia)
> datos2
  edad peso altura provincia
paco  20  65   174   madrid
pepe  22  70   180   malaga
kiko  19  68   170   murcia
```

La media de la variable edad la podemos calcular ahora de la siguiente forma:

```
> mean(datos2[, "edad"])
[1] 20.33333
```

\*\*\*\*\*

### 2.2.5.3. ALGUNAS FUNCIONES SOBRE DATA FRAMES

A la hora de utilizar ciertas funciones hay que tener presente que no todas las variables de un data frame tienen que ser del mismo tipo. Analicemos el siguiente ejemplo.

```
> apply(datos2,2,mean)
  edad      peso      altura provincia
   NA         NA         NA         NA
Warning messages:
1: In mean.default(newX[, i], ...) :
  argument is not numeric or logical: returning NA
2: In mean.default(newX[, i], ...) :
  argument is not numeric or logical: returning NA
3: In mean.default(newX[, i], ...) :
  argument is not numeric or logical: returning NA
4: In mean.default(newX[, i], ...) :
  argument is not numeric or logical: returning NA
```

No funciona porque hemos intentado hallar la media aritmética de cada columna de `datos2`. El error lo provoca la variable "provincia", que no es numérica.

Para evitar este error podemos aplicar la función `apply` solo sobre las variables numéricas de esta forma:

```
> apply(datos2[,1:3],2,mean)
  edad      peso      altura
```

```
20.33333 67.66667 174.66667
```

Para **acceder a los datos** podemos proceder indistintamente como si de una matriz o de una lista se tratase. Las siguientes instrucciones muestran cómo hacerlo.

```
> datos2[,2]                # Acceso en modo matriz
[1] 65 70 68

> datos2[,"edad"]           # Acceso en modo matriz
[1] 20 22 19

> datos2$edad               # Acceso en modo lista.
[1] 20 22 19
```

Lo mismo para la variable "provincia":

```
> datos2[,4]                # Acceso en modo matriz
[1] madrid malaga murcia
Levels: madrid malaga murcia

> datos2[,"provincia"]      # Acceso en modo matriz
[1] madrid malaga murcia
Levels: madrid malaga murcia

> datos2$provincia         #se refiere a la variable provincia del data frame
[1] madrid malaga murcia
Levels: madrid malaga murcia
```

Como vemos, las variables no numéricas se presentan como factores.

Si intentamos acceder directamente a la variable edad por su nombre, R nos da un error:

```
> edad
Error: object 'edad' not found
```

Si queremos utilizar las variables de un data frame por su nombre, sin hacer referencia a la matriz utilizaremos la función `attach` de la manera siguiente:

```
> rm(provincia)
> attach(datos2)           # Acceso directo a los nombres de datos2
> edad
[1] 20 22 19
> detach(datos2)           # Anula el acceso directo
> edad                     # Ya no se reconoce la variable directamente
Error: Object "edad" not found
```

\*\*\*\*\*

### EJERCICIO 5.

Al añadir a la matriz creada en el Ejercicio 4 la variable sexo R la transforma en tipo numérico asignándole el valor 1 para hombre 2 para mujeres.

Convertir la matriz en un data frame y añadirle una variable de tipo carácter que se corresponda con los nombres de los individuos que son los siguientes:

Juan, Inés, Andrés, Felipe, Pablo, Martina, Germán, Celia, Carmen, Santi, Dani, Antonio, Belinda y Sara

\*\*\*\*\*

#### 2.2.6. Listas en R

Una lista es un objeto formado por objetos que pueden tener estructuras distintas. Para crear una lista se utiliza la función `list()`.

##### 2.2.6.1. CREACIÓN DE LISTAS

Para crear una lista podemos hacer algo como lo siguiente:

```
> milista = list(numeros = 1:5, A = matrix(1:6, nrow =3), B= matrix(1:8, ncol = 2), ciudades = c("Sevilla", "Granada", "Málaga"))
```

```
> milista
```

```
$numeros
[1] 1 2 3 4 5
```

```
$A
```

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

```
$B
```

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

```
$ciudades
```

```
[1] "Sevilla" "Granada" "Málaga"
```

##### 2.2.6.2. MANEJO DE LISTAS

La **selección de los elementos** de una lista se puede realizar de varias formas. Utilizando la lista que acabamos de definir, las siguientes expresiones nos seleccionan el primer elemento de ella:

```
> milista$numeros           # utilizamos el operador$
[1] 1 2 3 4 5
> milista[["numeros"]]      # indicamos entre doble corchetes y comillas
el nombre
[1] 1 2 3 4 5
> milista[[1]]              # indicamos entre doble corchetes la posición
[1] 1 2 3 4 5
```

Para acceder a una sublista utilizamos la sintaxis:

```
NombreDeLaLista[c(Posicion1, Posicion2, ...)]
```

Por ejemplo, para acceder a una sublista formada por los elementos 1, 3 y 4 de `milista` escribimos:

```
> milista[c(1,3,4)]
$numeros
[1] 1 2 3 4 5

$B
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8

$ciudades
[1] "Sevilla" "Granada" "Málaga"
```

Otra forma de seleccionar elementos de una lista consiste en indicar mediante índices negativos los elementos que no se obtienen, siguiendo la sintaxis:

```
NombreDeLaLista[c(-Posicion1, -Posicion2,...)]
```

Por ejemplo, si únicamente queremos seleccionar el elemento 2 de la lista ejecutaremos:

```
> milista[c(-1,-3,-4)]
$A
      [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Para ver los nombres de los objetos dentro de la lista se utiliza la función `names`

```
> names(milista)
[1] "numeros" "A"      "B"      "ciudades"
```

## 2.3. Selección de variables

En este apartado vamos a ver cómo podemos seleccionar un subconjunto de valores en vectores y data frames. Para ello utilizamos la función `subset()`.



La sintaxis de esta función en el caso de vectores es:

```
subset(x, subset, ...)
```

Y para los data frames es:

```
subset(x, subset, select, drop = FALSE, ...)
```

donde los argumentos son:

x:	El objeto al que se aplica la selección
subset:	La condición que se aplica para seleccionar un subconjunto
select:	Columnas que se desea conservar en una hoja de datos
Drop:	Este argumento se pasa al método de indexación de hojas de datos

Recordemos los datos del ejemplo de creación de un data frame presentado anteriormente.

```
> datos = matrix(c(20,65,174,22,70,180,19,68,170),nrow=3,byrow=T)
> dimnames(datos)<-list(c("paco","pepe","kiko"),c("edad","peso","altura"))
> datos
      edad peso altura
paco   20   65   174
pepe   22   70   180
kiko   19   68   170
```

```
> provincia = c("madrid","malaga","murcia")
> datos2 = cbind(datos,provincia)
> datos2
      edad peso altura provincia
paco "20" "65" "174" "madrid"
pepe "22" "70" "180" "malaga"
kiko "19" "68" "170" "murcia"
```

```
> datos2 = data.frame(datos,provincia)
> datos2
      edad peso altura provincia
paco   20   65   174    madrid
pepe   22   70   180    malaga
kiko   19   68   170    murcia
```

#### • Seleccionar variables

```
> subset(datos2,select=c(edad,provincia))
      edad provincia
paco "20" "madrid"
pepe "22" "malaga"
kiko "19" "murcia"
```

#### • Seleccionar un subconjunto de la variable

```
> subset(datos2, peso > 66)
      edad peso altura provincia
pepe   22   70   180    malaga
kiko   19   68   170    murcia
```

```
> subset(datos2, altura > 1.70 & peso > 65)
  edad peso altura provincia
pepe  22  70   180    malaga
kiko  19  68   170    murcia

> subset(datos2, altura > 1.70 & peso > 65, select=provincia)
  provincia
pepe    malaga
kiko    murcia
```

• **Seleccionar variables usando una variable auxiliar filtro.**

```
> filtro = datos2[, "peso"] > 65
> filtro
[1] FALSE  TRUE  TRUE
> datos2[filtro, "peso"] = NA
> datos2
  edad peso altura provincia
paco  20  65   174    madrid
pepe  22  NA   180    malaga
kiko  19  NA   170    murcia
```

• **Creación de variables a partir de otras existentes**

```
> transform(datos2, logPeso=log(peso))
  edad peso altura provincia logPeso
paco  20  65   174    madrid 4.174387
pepe  22  70   180    malaga 4.248495
kiko  19  68   170    murcia 4.219508
```

\*\*\*\*\*

## EJERCICIO 6.

Utilizando el data frame generado en el ejercicio 5 y utilizando la función subset realizar lo siguiente:

- Seleccionar las variables correspondientes a los nombres y su correspondiente sexo.
- Seleccionar las variables correspondientes al peso y a la altura.
- Seleccionar los hombres con altura mayor que 1.70, peso mayor que 65

\*\*\*\*\*

## 2.4. Sintaxis y estructuras de control en R

### 2.4.1. Sintaxis

Como hemos visto a lo largo de los ejemplos presentados en las secciones anteriores, el único tipo de orden que posee R es una expresión o función que devuelve un resultado.

La **sintaxis de R**, esto es, el conjunto de reglas que deben seguirse al escribir código R, la hemos ido presentando y continuaremos haciéndolo a lo largo de los distintos ejemplos que iremos planteando. Por ejemplo, hemos aprendido cómo realizar asignaciones utilizando `<-` o `=`. También hemos visto que la llamada a una función se escribe usualmente como el nombre de la función seguido por sus argumentos colocados entre paréntesis.

Las distintas órdenes de R pueden agruparse entre llaves de la forma:

```
{expr 1;. . .; expr m}
```

En este caso, el valor del grupo es el resultado de la última expresión del grupo que se haya evaluado. Un grupo de órdenes así definido podremos incluirlo entre paréntesis y utilizarlo como parte de una expresión mayor.

### 2.4.2. Estructuras de control

R tiene sentencias para controlar el flujo de ejecución. A estas sentencias las denominamos estructuras de control.

#### 2.4.2.1. CONDICIONAL: LA ORDEN **if**

La sentencia **if-else** permite definir que una o varias sentencias sean ejecutadas sólo si se satisface determinada condición.

La sintaxis es:

```
> if (condición) sentencia
```

```
> if (condición) sentencia1 else sentencia2
```

En el primer caso si el valor lógico de **condición** es verdadero (T) se ejecutará **sentencia**. En el segundo caso se ejecutaría **sentencia1** y si es falso (F) se ejecuta **sentencia2**.

#### **EJEMPLO:** Usando la orden **if-else**

```
cond<-TRUE
> if ( cond==TRUE ) { print( "esta siempre se ejecuta" ) }

[1] "esta siempre se ejecuta"
>
> if ( cond==FALSE ) {
+   print( "esta nunca se ejecuta" )
+   print( "pero nunca, nunca" )
+ } else {
+   print( "¿lo ves?" )
+   print( ";-)" )
+ }
[1] "¿lo ves?"
[1] ";-)"
```

\*\*\*\*\*

### 2.4.2.2. BUCLES: LAS ÓRDENES `for`, `repeat` Y `while`

Las sentencias `for`, `repeat` y `while` permiten definir bucles. Estas órdenes se utilizan habitualmente cuando escribimos funciones.

La sentencia `for` tiene la siguiente sintaxis:

```
> for (var in secuencia) sentencia
```

donde `var` es la variable de control de iteración, `secuencia` es un vector (a menudo de la forma `m:n`), y `sentencia` es una expresión, a menudo agrupada. `sentencia` se evalúa repetidamente conforme `var` recorre los valores del vector `secuencia`.

#### **EJEMPLO: Usando la orden `for`**

La función `length()` devuelve el número de elementos del objeto que se le pasa como argumento.

El siguiente ejemplo ilustra el uso de dicha función y del bucle `for`.

```
x <- c(1,2,3,5,7,11)
> y <- numeric(0)
> k <- numeric(0)
> for (i in 1:length(x)) {
+   y[i] <- i*x[i]
+   k[i] <- y[i]/2
+ }
> y
[1] 1 4 9 20 35 66
> k
[1] 0.5 2.0 4.5 10.0 17.5 33.0
```

\*\*\*\*\*

La sentencia `repeat` y `while` tienen la siguiente sintaxis:

```
> repeat expr
> while (condicion) expr
```

Con `repeat` se repite la evaluación `expr` hasta que se interrumpa mediante la instrucción `break`.

#### **EJEMPLO: Usando la orden `repeat`**

El siguiente ejemplo ilustra el uso de la orden `repeat`, para imprimir los números enteros comprendidos entre el 0 y el 19, ambos inclusive.

```
> i <- 0
> repeat {if (i<20){
+ print(i)
+ i <- i+1
+ }
+ else
+ break;
+ }
[1] 0
[1] 1
```

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

\*\*\*\*\*

Cuando se alcanza una sentencia `while`, se ejecuta `expr` mientras `condicion` sea `True`. En cuanto `condicion` alcance el valor `False`, se dejará de ejecutar `expr`.

La función `break` se utiliza para terminar cualquier ciclo. Esta es la única forma (salvo que se produzca un error) de finalizar un ciclo `repeat`.

## 2.5. Funciones y paquetes en R

### 2.5.1. Funciones

#### 2.5.1.1. FUNCIONES INCORPORADAS EN R

R proporciona muchas funciones que permiten realizar infinidad de acciones. Entre las muchas que hay a continuación nombramos algunas funciones matemáticas y estadísticas.

Funciones matemáticas	Devuelve
<code>abs(x)</code>	Valor absoluto de x, <code>abs(-5)</code> devuelve 5
<code>sqrt()</code>	Raíz cuadrada de x, <code>sqrt(25)</code> devuelve 5
<code>ceiling(x)</code>	Entero más pequeño mayor que x, <code>ceiling(4.6)</code> devuelve 5
<code>floor(x)</code>	Entero más grande no mayor que x, <code>floor(4.6)</code> devuelve 4
<code>trunc(x)</code>	Truncamiento de x
<code>round(x, digits=n)</code>	Redondea x a un número específico de decimales, por defecto <code>n=0</code>
<code>log(x, base=n)</code>	Logaritmos ( <code>log</code> devuelve el logaritmo natural, <code>log(exp(1))</code> devuelve 1)

Funciones estadísticas	Devuelve
<code>mean(x)</code>	Media
<code>median(x)</code>	Mediana
<code>sd(x)</code>	Desviación estándar
<code>var(x)</code>	Varianza
<code>sum(x)</code>	Suma
<code>cumsum(x)</code>	Suma acumulada
<code>range(x)</code>	Rango

Hay muchas otras funciones que se utilizan habitualmente. Algunas de ellas ya las hemos comentado y otras las iremos viendo según las vayamos necesitando. Por ejemplo la función `apply()` que aplica funciones a los elementos de los objetos. Recordamos que es conveniente hacer uso de la ayuda de R. Para saber más sobre la función `apply()` podemos ejecutar `?apply`. La función `demo()` muestra la lista de las demos disponibles en R que ilustran muchas de las capacidades del lenguaje. Por ejemplo, `demo(graphics)`, `demo(persp)` y `demo(image)` están relacionadas con las capacidades gráficas. La lista completa de demos se puede obtener tecleando `> demo()`.

#### 2.5.1.2. CREACIÓN DE FUNCIONES

R permite al usuario definir sus propias funciones. Una función se define asignando a un objeto la palabra `function` seguida de los argumentos, escritos entre paréntesis y separados por comas y seguida de la orden, entre llaves si son varias órdenes, las que se quiera asignar. La sintaxis es la siguiente:

```
NombreFuncion = function (arg1, arg2, ...) {órdenes de la función return(Valor que ha de devolver la función) }
```

<b>NombreFuncion:</b> Nombre que le daremos a la función
<b>arg1, arg2, ...:</b> Parámetros que debe de tener la función
<b>órdenes de la función:</b> Las líneas de órdenes que van a llevar a cabo que la función dé el resultado deseado. Pueden ser condicionales, operadores lógicos, bucles, etc.
<b>return( ):</b> Es el valor que devuelve la función. Normalmente suele ser una lista.

Los objetos creados dentro del cuerpo de la función son locales a la función. El objeto devuelto por la función puede ser cualquier tipo de dato. La función se invoca de la misma forma que las demás funciones definidas en R: especificando su nombre y escribiendo a continuación los argumentos entre paréntesis, separados por coma.

Una vez definida la función en el script (que habremos guardado con extensión `.R`), la función `source` permite cargar y ejecutar el script.

Supongamos que nuestro script se llama `Primerscript.R`. Si éste se encuentra en el directorio de trabajo para cargar nuestro script bastaría con ejecutar:

```
> source('Primerscript.R')
```

Si tenemos el script abierto podemos utilizar una de las siguientes combinaciones de teclas:

Ctrl + Shift + S	ejecución directa del script
Ctrl + Shift + Enter	imprime todas las líneas en la consola al ejecutarlas
Ctrl + Shift + O	abre una ventana para elegir el script
Ctrl + Enter	ejecuta una línea o una selección previa del usuario

Si el script no se encuentra en el directorio de trabajo, o bien cambiamos de directorio, se debe incluir el camino relativo donde se encuentra el script respecto al directorio actual en el comando `source`.

### **EJEMPLO: Escribiendo y ejecutando una función**

Cuando vamos a escribir una función escrita por nosotros mismos deberemos cargarla previamente en R. Para ello utilizaremos los scripts, donde podremos almacenar varias funciones y cargarlas en cualquier momento para su uso posterior.

Supongamos que queremos diseñar una función que devuelva el área de un círculo pasando por parámetro su radio. Vamos a llamar a la función `AreaCirculo`, el parámetro de la función será el radio y la función deberá devolver el área del círculo.

Y la función es:

```
> AreaCirculo <- function(radio){
  area <- pi*radio^2
  return(area)}

```

Para diseñar un script donde podamos almacenar esta función y otras para su posterior uso realizamos lo siguiente:

En la barra de herramientas seleccionamos **File → New File → R script**. Se nos abrirá una ventana (ver Figura 2.5-1) que es la que utilizaremos para almacenar las funciones.

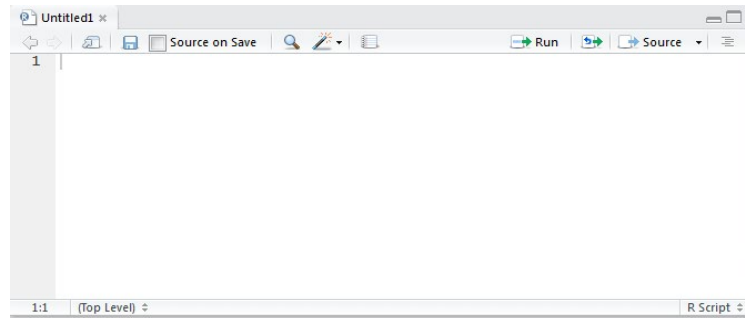


Figura 2.5-1: Nuevo R script.

El siguiente paso es escribir la función o funciones en dicho editor del script (ver Figura 2.5-2)

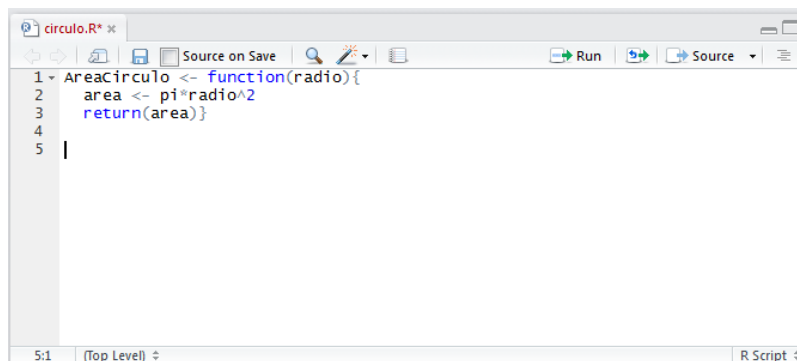



Figura 2.5-2: Escritura de la función en el Nuevo R script.

Y el último paso es, guardar el script: En la barra de herramientas del script, **File → Save As...**, se nos abrirá la ventana para elegir carpeta o ubicación para guardar nuestro script. Podemos por ejemplo tener una carpeta llamada R Script en el directorio principal, en C, para guardar todos nuestros scripts, y le damos nombre, por ejemplo: Circulo.R.

Y en estos momentos ya tenemos un script con una función lista para ser cargada en R y usada cuando queramos.

¿Cómo cargamos en R el script para poder usarlo?

Vamos a borrar en primer lugar todos los objetos cargados en R, para ello basta con que en la pestaña Environment seleccionamos .

Para comprobar que efectivamente hemos eliminado todos los objetos podemos hacer:

```
> ls()  
character(0)
```

efectivamente, `character(0)` nos indica que no hay ninguno.

Para cargar nuestro script hacemos lo siguiente:



En el menú de Barra de Herramientas seleccionamos **File -> Open File...**, nos saldrá la ventana de búsqueda, seleccionamos nuestro script, Circulo.R, en la ruta que haya sido guardado y se nos abrirá el editor del script con las funciones que contiene. En la barra de herramientas del script seleccionamos Source (ver Figura 2.5-3) y automáticamente se nos cargará en R nuestro script con todas las funciones diseñadas en él.

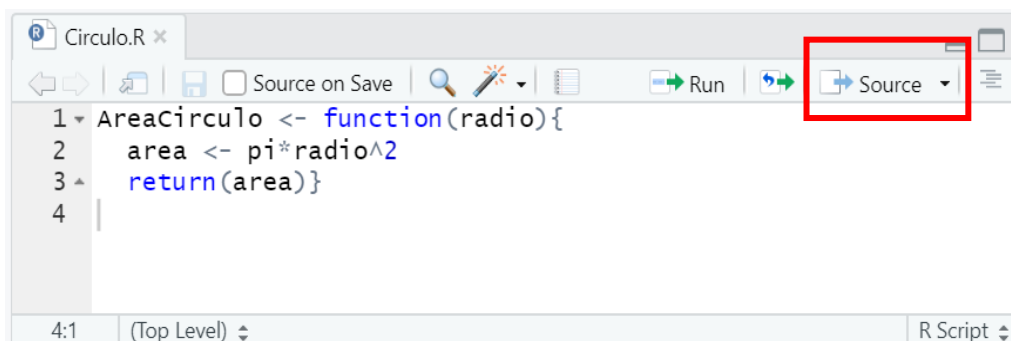


Figura 2.5-3: Escritura de la función en el Nuevo R script.

Observamos que en la pantalla Environment (Figura 2.5-4) aparece la función AreaCirculo y en la consola aparece la línea:

```
> source('C:/RScript/circulo.R')
```

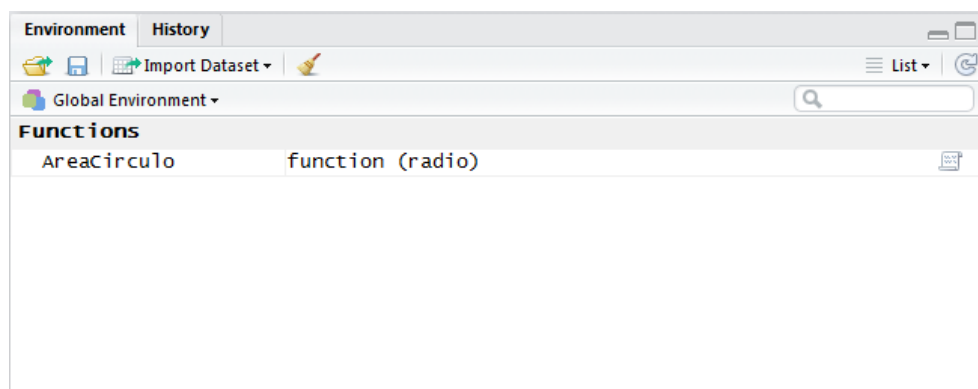


Figura 2.5-4: Objeto AreaCirculo en Environment.

Ya tendríamos nuestro script cargado, para verificarlo, comprobamos los objetos en R:

```
> ls()
```

```
[1] "AreaCirculo"
```

Como podemos comprobar, está cargada nuestra función, y ya sólo nos queda usarla. Por ejemplo, calcular el área de un círculo de radio 3 cm sería:

```
> AreaCirculo(0.03)
[1] 0.002827433
```

El resultado del área es: 0.002827433 m<sup>2</sup>.

\*\*\*\*\*

\*\*\*\*\*

### EJERCICIO 7.

Escribir una función que devuelva el valor lógico TRUE si la suma de los elementos del vector es menor que 100 y FALSE en caso contrario. Realizar dos llamadas a la función, una con un vector cuyas componentes son los primeros 50 números enteros y otra con un vector cuyas componentes son los primeros 10 números enteros.

\*\*\*\*\*

## 2.5.2. Packages en R

### 2.5.2.1. PACKAGES ESTÁNDAR

Todas las funciones y conjuntos de datos de R están almacenadas en packages (librerías). Tras realizar una instalación de R se cargan automáticamente una serie de packages básicos. Estas librerías contienen funciones básicas, conjuntos de datos, funciones estadísticas y gráficas que incluyen gran cantidad de funcionalidades. En la pestaña packages en la parte inferior derecha del interfaz de RStudio se muestra una lista con todas las librerías incluidas en la instalación (ver Figura 2.5-5).

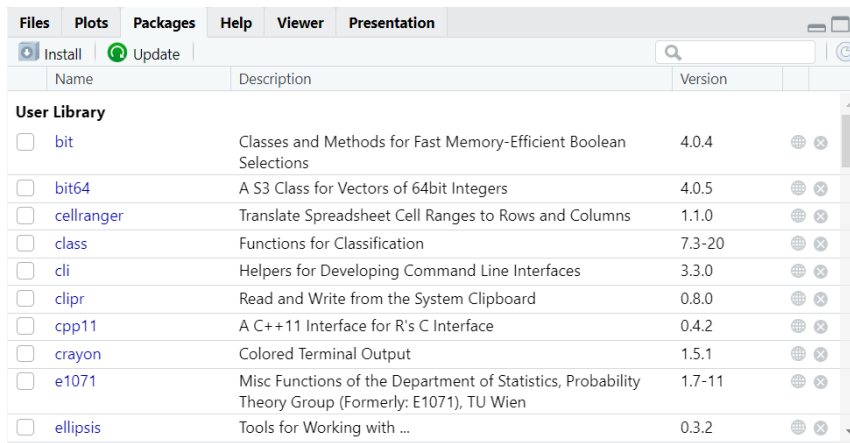


Figura 2.5-5: Librerías incluidas en la instalación de R.

Para que el contenido de un package esté disponible debe estar previamente cargada o activada. Para ello deberemos marcar el paquete correspondiente en esta pestaña (ver Figura 2.5-6). Si no lo marcamos, cualquier orden relacionada con el paquete no funcionará y necesitaremos seleccionarlo.

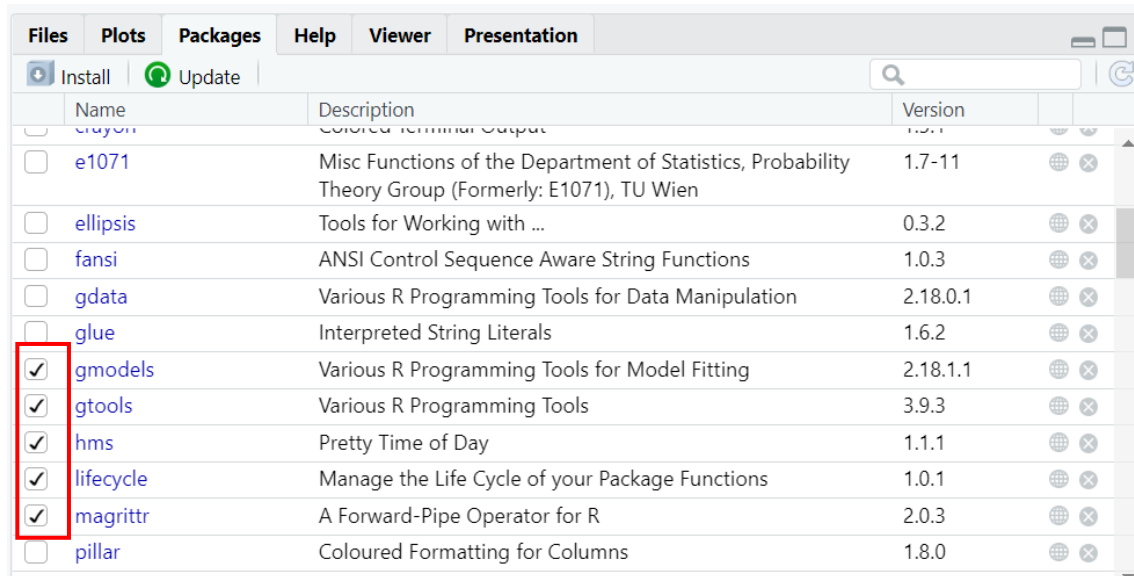


Figura 2.5-6: Activación de packages.

Otra forma de activar un package es utilizar el comando `library()`. Por ejemplo, para cargar el package `grid` tendríamos que escribir:

```
> library(grid)
```

Podemos ver cómo tras este comando en la pestaña Packages aparece la librería `grid` seleccionada.

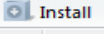
### 2.5.2.2. INSTALACIÓN DE PACKAGES

En muchas ocasiones es necesario cargar librerías específicas que tengan implementadas determinadas funcionalidades.

En el repositorio CRAN, disponemos de un conjunto de librerías listas para ser instaladas y usadas. En el siguiente enlace, podemos comprobar las distintas librerías disponibles: <https://cran.r-project.org/web/views/>

Así, por ejemplo, en el módulo de Introducción al Machine Learning, usaremos algunos de los paquetes incluidos en dicha librería, y que están accesibles en el siguiente enlace: <https://cran.r-project.org/view=MachineLearning>

Para instalar un package utilizando RStudio tenemos que hacer lo siguiente:

Ir a la barra de menú **Tools** → **Install Packages** o bien seleccionar **Install**  en la pestaña **Packages**. Con cualquiera de estas acciones se accede a la Figura 2.5-7. Escribimos en la ventana que se despliega el nombre del paquete o paquetes a instalar, en este caso será el paquete `e1071`.

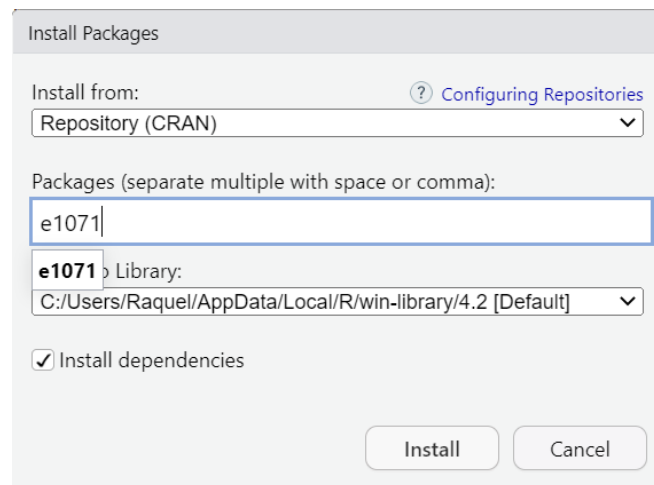


Figura 2.5-7: Instalando package.

Observamos que, para ayudar en su selección, aparece una lista de potenciales paquetes, escogemos `e1071` y hacemos clic en **Install**. La ventana de la consola irá indicando el proceso de instalación (ver Figura 2.5-8).

```
> install.packages("e1071")
Installing package into 'C:/Users/Raquel/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/e1071_1.7-12.zip'
Content type 'application/zip' length 663514 bytes (647 KB)
downloaded 647 KB
```

```
package 'e1071' successfully unpacked and MD5 sums checked
```

```
The downloaded binary packages are in
C:\Users\Raquel\AppData\Local\Temp\RtmpI1N5Zr\downloaded_packages
```

Figura 2.5-8: Proceso de instalación del package `e1071` en la consola.

Para usar el paquete recién instalado en la sesión actual debemos primero cargarlo, bien con la instrucción:

```
> library(e1071)
```

O bien seleccionándolo desde el panel de **Packages** (ver Figura 2.5-9)

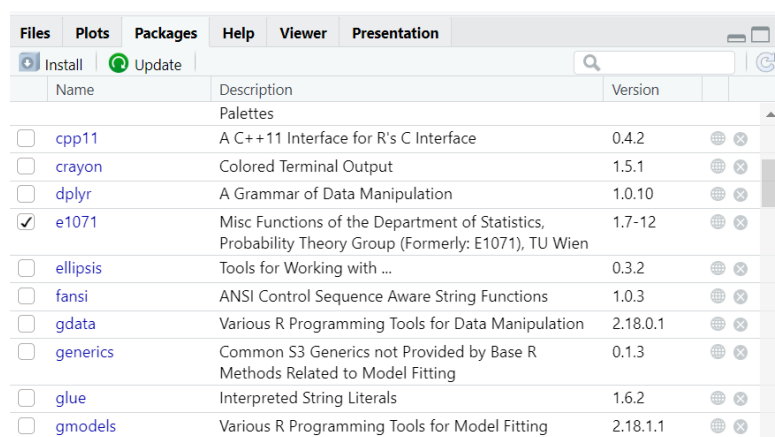



Figura 2.5-9: Carga del package `e1071`

Para actualizar los paquetes, debe ir al menú principal y en **Tools** → **Check for Package Updates**, aparece una ventana mostrando los paquetes que se pueden actualizar. También se puede actualizar desde el botón  **Update** del panel **Packages**.

\*\*\*\*\*

## EJERCICIO 8.

Instalar el package `ggplot2` siguiendo los pasos explicados en la sección.

\*\*\*\*\*

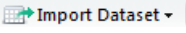
## 2.6. E/S. Lectura de datos desde distintos tipos de fuentes

Los datos en R se pueden cargar en la estructura de datos bien manualmente o bien pueden ser importados desde una fuente externa. R trae con su instalación múltiples conjuntos de datos. Para ver un listado de los mismos basta teclear `data()`, y para cargar uno concreto debemos utilizar el comando `data(name)`.

La introducción manual de datos es una tarea tediosa, especialmente cuando se tiene un gran volumen de información. En estos casos, lo más normal es disponer de un fichero de datos externo que el programa se encarga de leer. R es capaz de leer ficheros con multitud de formatos: Excel (con extensión `.xls` o `.xlsx`), SPSS (con extensión `.sav`), `.csv`, `.txt`

### 2.6.1. Importando datos de ficheros de texto: `.csv` y `.txt`

Los formatos de archivo de texto que se utilizan habitualmente son de dos tipos `.txt` o `.csv`. En los archivos de texto delimitados (`.txt`), el carácter de tabulación (el código de carácter ASCII 009) separa normalmente cada campo de texto. En los archivos de texto de valores separados por comas (`.csv`), es el carácter de coma (,) el que suele separar cada campo.

Independientemente del tipo de archivo que queramos importar, debemos ir a la pestaña **Environment** y en **Import DataSet** (  ) se selecciona el tipo de fichero desde el que queremos realizar la importación. Podemos acceder a estas mismas opciones de importación desde el menú **File-> Import Data Set**

#### 2.6.1.1. IMPORTANDO `.csv`

Al importar un fichero `.csv`, tanto desde un directorio local como desde una URL, RStudio nos permite diferentes acciones como veremos a continuación.

Supongamos que queremos importar el fichero denominado **Datos personas.csv** que se encuentra entre los ficheros que te puedes descargar del curso virtual. En primer lugar, deberemos seleccionar **File->Import DataSet** (también accesible desde la pestaña **Environment**) y a continuación **From Text (readr)**... En ese momento es probable que tengamos que instalar algunos paquetes (`readr`, `Rcpp`), si no lo hemos hecho previamente. Terminada la instalación de los paquetes que nos indica RStudio seleccionamos el fichero de la ubicación donde lo tengamos almacenado en nuestro PC y clicamos en **Abrir**. Entonces RStudio muestra la pantalla que se muestra en la Figura 2.6-1.

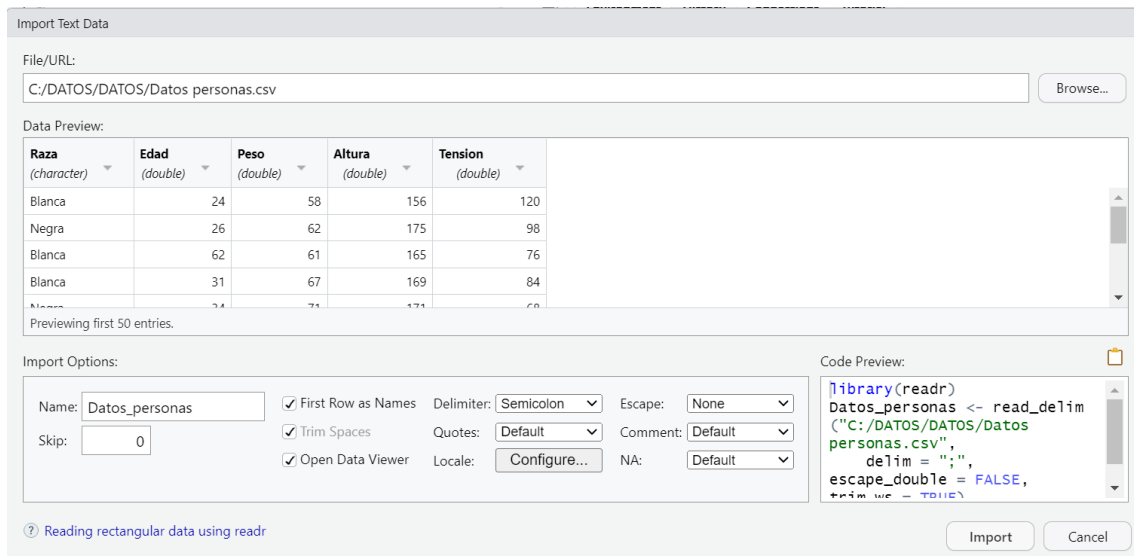


Figura 2.6-1: Proceso de importación de datos .csv

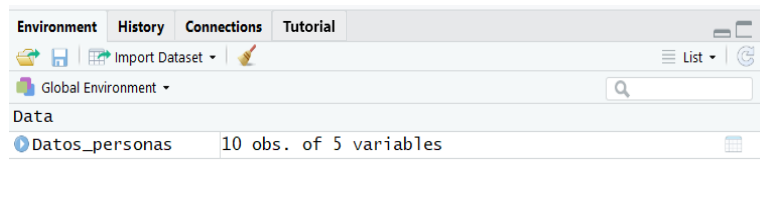
El campo **Delimiter** (delimitador de columna) dispone de diferentes opciones para poder seleccionar según cuál sea el tipo de delimitador de columna que tenemos en el fichero de entrada (Tabulador, espacio en blanco, coma o punto y coma). En este caso debemos cambiar el **Delimiter** de Comma a Semicolon, dado que el carácter separador es punto y coma, y a continuación, dar a **Import**. Y veremos entonces los datos ya importados (ver Figura 2.6-2).

Como se aprecia en la Figura 2.6-1 se pueden realizar otras opciones previas a la importación como por ejemplo renombrar el data frame en el que se almacenarán los datos del fichero. Esto se hace en el campo **Name**. En nuestro caso dejamos el nombre por defecto, Datos\_personas. O indicar el salto de un determinado número de líneas (en **Skip**).

	Raza	Edad	Peso	Altura	Tension
1	Blanca	24	58	156	120
2	Negra	26	62	175	98
3	Blanca	62	61	165	76
4	Blanca	31	67	169	84
5	Negra	34	71	171	68
6	Negra	65	69	150	80
7	Negra	76	68	190	95
8	Blanca	32	73	156	86
9	Blanca	12	43	130	76
10	Blanca	56	82	178	89

Figura 2.6-2: Data frame importado

Una vez seleccionadas las opciones de importación seleccionado en **Import** aparece cargado en el **Environment** el data frame (ver Figura 2.6-3):

Figura 2.6-3: Data frame en **Environment** una vez importado el fichero.

Esto mismo se podría haber realizado desde la consola utilizando las líneas:

```
Datos.personas <- read.csv("D:/R/Datos personas.csv", sep=";")
View(Datos.personas)
```

Y se visualizan de nuevo los datos importados (ver Figura 2.6-4):

	Raza	Edad	Peso	Altura	Tension
1	Blanca	24	58	156	120
2	Negra	26	62	175	98
3	Blanca	62	61	165	76
4	Blanca	31	67	169	84
5	Negra	34	71	171	68
6	Negra	65	69	150	80
7	Negra	76	68	190	95
8	Blanca	32	73	156	86
9	Blanca	12	43	130	76
10	Blanca	56	82	178	89

Figura 2.6-4: Contenido del Data frame.

Para poder acceder a los datos del data frame tenemos que hacer un attach:

```
> attach(Datos.personas)
> summary(Edad)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 12.00  27.25   33.00   41.80   60.50   76.00
```

\*\*\*\*\*

### EJERCICIO 9.

Importar desde la siguiente URL un fichero .csv:

<http://winterolympicsmedals.com/medals.csv>

y obtener un summary de la variable Gender.

\*\*\*\*\*



### 2.6.1.2. IMPORTANDO .txt

La importación de un fichero .txt es muy parecida a la de un fichero .csv, se puede realizar desde **Import Dataset From Text (base)**. Si importamos desde un fichero local denominado **Datos personas.txt** desde esta opción llegamos a la pantalla de la Figura 2.6-5.

The dialog box shows the following settings:

- Name: Datos.personas
- Encoding: Automatic
- Heading: Yes (selected)
- Row names: Automatic
- Separator: Tab
- Decimal: Period
- Quote: Double (")
- Comment: None
- na.strings: NA
- ☐ Strings as factors

The 'Input File' section displays the following data:

Raza	Edad	Peso	Altura	Tension
Blanca	24	58	156	120
Negra	26	62	175	98
Blanca	62	61	165	76
Blanca	31	67	169	84
Negra	34	71	171	68
Negra	65	69	150	80
Negra	76	68	190	95
Blanca	32	73	156	86
Blanca	12	43	130	76
Blanca	56	82	178	89

The 'Data Frame' section shows the same data as a table:

Raza	Edad	Peso	Altura	Tension
Blanca	24	58	156	120
Negra	26	62	175	98
Blanca	62	61	165	76
Blanca	31	67	169	84
Negra	34	71	171	68
Negra	65	69	150	80
Negra	76	68	190	95
Blanca	32	73	156	86
Blanca	12	43	130	76
Blanca	56	82	178	89

Figura 2.6-5: Proceso de importación de datos .txt

En este caso los delimitadores del Input File son tabuladores, por lo que habrá que seleccionar **Tab** como delimitador. Seleccionando **Import** completamos la importación de datos.

### 2.6.2. Importando datos de ficheros Excel

Una manera sencilla y muy habitual de importar ficheros Excel (.xls, .xlsx) es abrir el fichero Excel y guardarlo como Texto (delimitado por tabulaciones) (\*.txt) o bien como .csv y aplicar directamente los pasos explicados en las secciones anteriores.

Otra forma es directamente desde **Import Dataset From Excel...** En este caso, la primera vez que se accede a esta opción se debe permitir la instalación de la librería readxl. Instalada la librería se accede a la pantalla de importación que se muestra en la Figura 2.6-6. Podemos probar con el fichero **Datos personas.xlsx**.

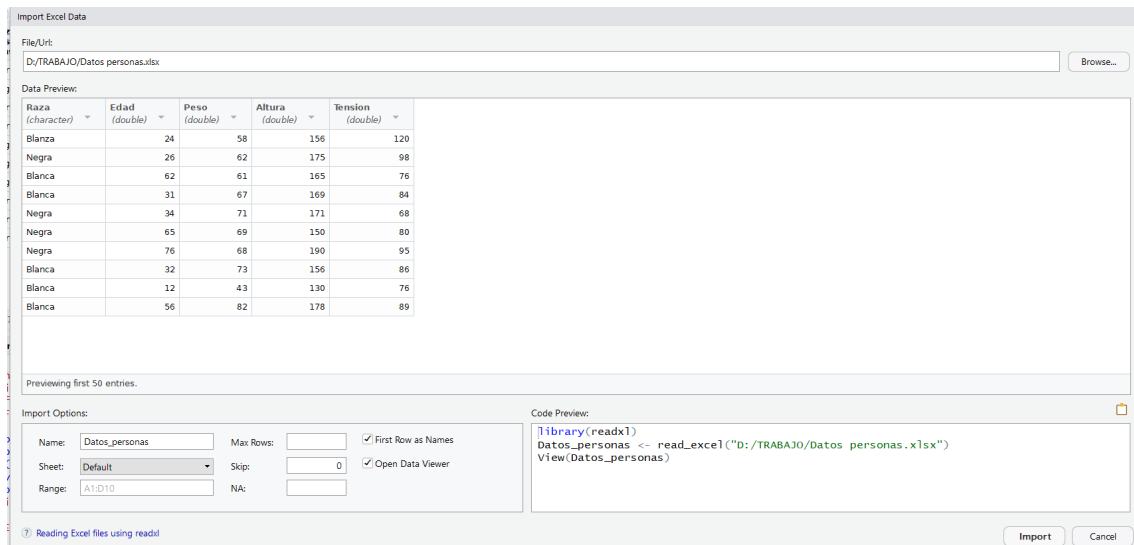


Figura 2.6-6: Proceso de importación de datos .xlsx

También se pueden utilizar directamente determinados packages disponibles para este fin. Por ejemplo, se puede instalar el package `xlsx` y utilizar la función `read.xlsx` de la siguiente manera:

```
> ficheroexcel<-"C:/Datos personas.xls"
> midataframe<-read.xlsx(ficheroexcel,1)
```

```
> midataframe
```

	Raza	Edad	Peso	Altura	Tension
1	Blanca	24	58	156	120
2	Negra	26	62	175	98
3	Blanca	62	61	165	76
4	Blanca	31	67	169	84
5	Negra	34	71	171	68
6	Negra	65	69	150	80
7	Negra	76	68	190	95
8	Blanca	32	73	156	86
9	Blanca	12	43	130	76
10	Blanca	56	82	178	89

### 2.6.3. Importando datos de ficheros con otros formatos

R permite importar otros tipos de datos: SAS (**Import Dataset /From SAAS...**), SPSS (**Import Dataset /From SPSS...**), Stata (**Import Dataset /From SPSS...**). Para otros tipos de datos como por ejemplo Systat, Weka algunos de los paquetes más utilizados para importar son: *RODBC* o el paquete *Hmisc* (que a su vez requiere el paquete *foreign*). Aunque su manejo no es objeto de este módulo.

### 2.6.4. Exportando distintos tipos de datos

R también permite exportar datos en distintos formatos. Para ello utilizamos la función `write.table`. Veamos cómo se realizaría la exportación en algunos de estos formatos.

#### 2.6.4.1. EXPORTANDO A .CSV

Vamos a cargar los datos mtcars disponibles en R y después los vamos a exportar. Para esto utilizamos los comandos siguientes:

```
> data(mtcars)
> write.csv(mtcars, 'mtcars.csv')
```

En este caso en el directorio que tengamos como espacio de trabajo se habrá almacenado el fichero mtcars.csv

#### 2.6.4.2. EXPORTANDO A .txt

Si queremos exportar los datos a un fichero de texto separado por tabulaciones debemos utilizar el comando:

```
> write.table(mtcars, 'mtcars.txt', sep='\t')
```

R usa por defecto el '.' como separador de decimales, en España se usa ',' como separador de decimales y '.' como separador de miles. Para corregir esto en la exportación tenemos que realizar lo siguiente:

```
> write.table(mtcars, 'mtcars.txt', sep='\t', dec=',')
```

#### 2.6.4.3. EXPORTANDO A Excel

En este caso tenemos que cargar un paquete adicional (WriteXLS). Podemos ver detalles de cómo hacerlo escribiendo en la consola lo siguiente:

```
> install.packages('WriteXLS')
> library(WriteXLS)
> ?WriteXLS # para ver la ayuda.
```

#### 2.6.4.4. EXPORTANDO A OTROS FORMATOS

Con el paquete foreign podemos exportar a formato SPSS de esta manera:

```
> write.foreign(mtcars, 'File_spss.txt', 'Code_spss.sps', package='SPSS')
```

Esto muestra qué ha generado R para que sea leído por SPSS

```
> file.show('File_spss.txt') file.show('Code_spss.sps')
```

Con el paquete foreign podemos también exportar a STATA mediante:

```
> write.dta(mtcars, 'mtcars.dta')
```