

Introducción a R y RStudio

Empezando con R: algunos conceptos básicos

Índice

Ideas clave	3
2.1. Introducción y objetivos	3
2.2. Entorno de trabajo de RStudio	3
2.3. La consola de R	7
2.4. Variables	8
2.5. Objetos	11
2.6. Directorio de trabajo	15
2.7. Scripts	17
2.8. Creación de Proyectos	20
2.9. Manejo de la biblioteca: paquetes adicionales	22
2.10. Referencias bibliográficas	27

2.1. Introducción y objetivos

Para trabajar con R es necesario conocer un poco del vocabulario usado en este lenguaje de programación. En este tema trataremos conceptos básicos que usaremos a lo largo de todo el apunte, lo que nos permitirán familiarizarnos con el lenguaje en sí mismo como con el entorno de trabajo que proporciona RStudio.

Al finalizar este tema, habrás alcanzado los siguientes objetivos:

- ▶ Familiarizarse con el entorno de RStudio.
- ▶ Describir el propósito y el uso de cada panel de RStudio.
- ▶ Establecer el directorio de trabajo para una sesión de RStudio.
- ▶ Definir variables y asignarle valores.
- ▶ Conocer funcionalidades básicas de la consola de R.
- ▶ Crear, guardar y abrir scripts de código R.
- ▶ Comprender la utilidad de trabajar con proyectos autocontenidos.
- ▶ Gestionar paquetes.

2.2. Entorno de trabajo de RStudio

A pesar de que desde la consola de R y un editor de texto cualquiera es posible realizar cualquier tarea: escribir *scripts* y programas, crear nuevos paquetes, generar gráficos y documentación, etc., la mayor parte de esas tareas pueden simplificarse de manera considerable contando con un entorno de trabajo (IDE) adecuado.

Ya mencionamos que, para contar con más herramientas de apoyo en el uso de R, a largo de todo el apunte emplearemos el software RStudio. Por lo que todos los ejemplos e imágenes estarán referidas a esta interfaz y no al uso directo de R.

En el siguiente video, “Introducción a RStudio”, se introduce la interfaz RStudio, sus paneles y funcionalidades básicas:



Cuando abras RStudio por primera vez aparecerá una ventana similar a la que se muestra en la Figura 1, su apariencia exacta dependerá del sistema operativo y versión de RStudio que estés utilizando.

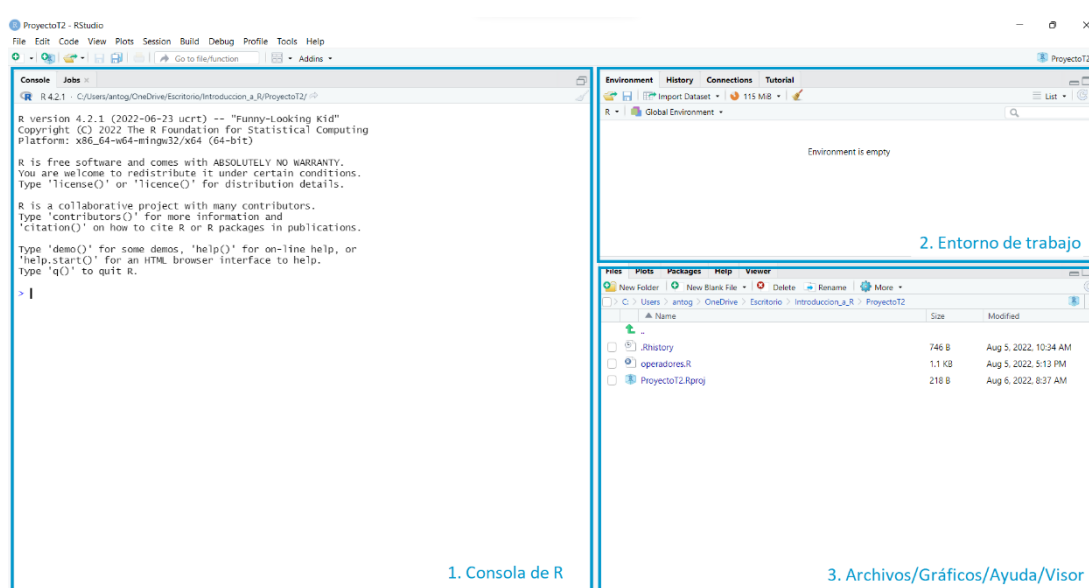


Figura 1: Las cuatro ventanas de RStudio.

Como podemos ver, RStudio está, a su vez, dividido en 3 paneles o ventanas.

- ▶ Ventana 1: es la consola de R. Corresponde a lo que sería el software R en su versión básica. Allí el software ejecuta las operaciones realizadas desde el editor de sintaxis. Por defecto, la consola se encuentra en el panel inferior-izquierdo.

¿Puedes observar la pestaña llamada *Console*? Inmediatamente debajo aparece un texto informativo (la versión de R que estamos usando, el directorio de trabajo actual, créditos, etc.) y al final una línea en blanco encabezada por el símbolo “>”. Este símbolo es la *marca de inicio* de una nueva línea y nos indica que R espera que le demos instrucciones. Para ejecutarlas y obtener el resultado pulsamos *Enter*.

- ▶ Ventana 2: el ambiente de trabajo: es un panel multifunción. Tres de sus pestañas son:
 - *Environment* del programa: aquí se irán registrando los objetos que vayamos creando en la sesión de trabajo. Incluye las opciones de abrir, guardar e importar el conjunto de datos con los que se podrían trabajar; buscar y limpiar objetos del área de trabajo, es decir, eliminar las variables definidas en la sesión actual; muestra la información de los objetos como lista o cuadrícula (más detallada).
 - *History* que nos muestra un histórico del conjunto de órdenes que hemos usado. Aquí podemos seleccionar algunas y reejecutarlas presionando el botón *To Console* o enviarlas al *script* clicando en *To Source*. También encontramos el símbolo de escoba para eliminar las órdenes y el buscador para encontrar aquella que se precise.
 - *Tutorial* pues eso, un poco de ayuda usando el paquete `{learnr}` que recomiendo fuertemente si disponéis de un poco de tiempo y ganas, para dar los primeros pasos.
- ▶ Ventana 3: también es un panel multifunción. Las distintas pestañas que posee son:
 - la pestaña *Files* permite ver el directorio de trabajo en su disco duro. Una característica interesante del panel *Files* es que podemos usarlo para configurar el directorio de trabajo: una vez que estamos a la carpeta en la que deseamos leer y guardar archivosⁱ, haciendo clic en *More* y luego en *Set As Working Directory*. Hablaremos sobre los directorios de trabajo con más detalle más adelante;
 - la pestaña *Plots* permite visualizar los gráficos que se van generando y ofrece la posibilidad de eliminarlo y/o salvarlo (*Export*);
 - la pestaña *Packages* permite ver los paquetes descargados y guardados en el disco duro, así como gestionar su instalación o actualización. Además, el hacer clic en el nombre del paquete, redirecciona a la solapa *Help*, que veremos a continuación.

- la ventana *Help* permite acceder al CRAN - *Comprehensive R Archive Network* (siempre que se cuente con conexión a Internet), página oficial del software que ofrece diferentes recursos para el programa: manuales para el usuario, cursos online, información general, descarga de paquetes, información de los paquetes instalados, etc. Esta última pestaña es bastante útil: empleando el motor de búsqueda se accede de manera rápida a manuales de uso de los diferentes paquetes (y sus funciones) instalados en el ordenador (esto no requiere conexión a Internet);
- la ventana *Viewer* muestra los resultados al construir reportes mediante funcionalidades tipo **R Markdown**.

Aparte de estas tres ventanas, RStudio dispone de una cuarta ventana que es el editor de sintaxis: es el lugar donde editamos la sintaxis para posteriormente ejecutarla. Es decir, donde vamos a escribir el código de R que nos interesa en scripts. Por defecto, al abrir el editor, se posicionará en el panel superior-izquierdo. En el apartado 2.6. Scripts, veremos en detalle lo relacionado al editor de sintaxis.

Por otro lado, RStudio contiene una **barra de herramientas general**, que incluye las opciones de crear un nuevo archivo, abrir uno ya creado, guardar el fichero actual o todos los ficheros abiertos, imprimir, el cuadro de texto *Go to file/function*, el icono de paneles que permite configurar a estos, y el botón *Addins*. Además, en el extremo derecho se encuentra la opción de abrir un nuevo proyecto, cerrarlo o seleccionar aquel sobre el que se vaya a trabajar.

Otra barra de herramientas fundamental es la **barra de menús** situada en la parte superior de la interfaz. Los diferentes menús que la componen (*File, Edit, Code, View*, etc.) ofrecen una amplia variedad de herramientas para realizar con RStudio.

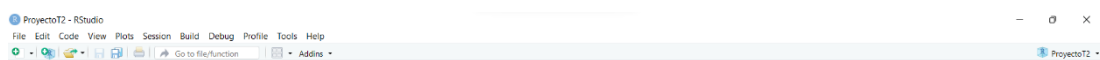


Figura 2: La barra de herramientas general y barra de menús de RStudio.

2.3. La consola de R

La consola de R corresponde al entorno computacional de este lenguaje. Es ahí donde es interpretado nuestro código.

Como lo indicamos anteriormente, en RStudio encontramos la consola de R en uno de los paneles de la interfaz, y, en caso de que trabajemos directamente desde R, será lo primero que encuentres al ejecutar R.

La consola es la herramienta básica para operar con R de manera interactiva. Esto significa que podemos escribir código directamente allí y R nos devolverá el resultado, sin necesidad de compilarlo.

Veamos las principales funcionalidades de la consola.

Para ejecutar cualquier comando R, simplemente tenemos que introducirlo y pulsar *Enter*. Por ejemplo, si a derecha de la *marca de inicio* escribimos $2+3*5$ y a continuación presionamos *Enter*, R mostrará en la línea siguiente el resultado, 17, y a continuación una nueva línea en blanco con la *marca de inicio* $>$, donde podremos seguir escribiendo instrucciones.

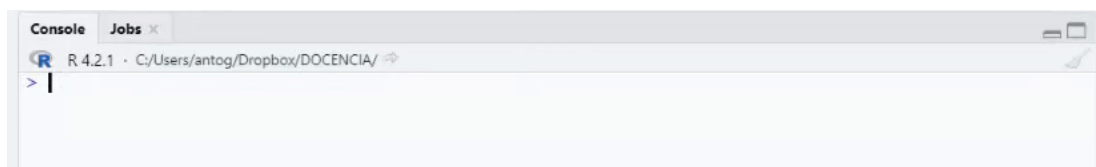
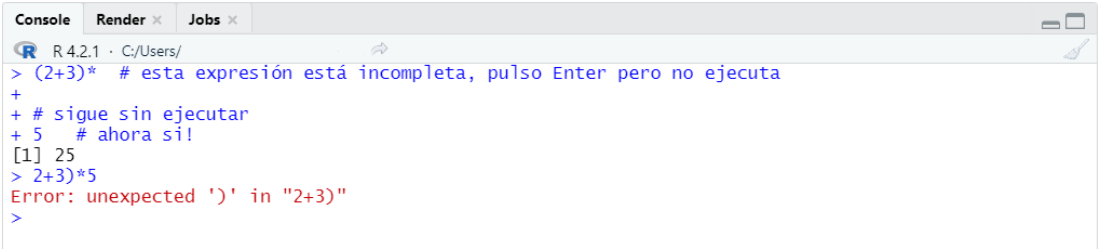


Figura 3: La consola de R.

Una instrucción puede extenderse en varias líneas. Si la expresión que entramos está incompleta, R no la evaluará y esperará a que la completemos en la línea siguiente. La marca de inicio mostrado en la consola cambiará de $>$ a $+$ siempre que se detecte

que aún faltan instrucciones para completar el comando. Además, nos avisará mediante un mensaje de error en caso de que cometamos algún error de sintaxis.



```
R 4.2.1 · C:/Users/
> (2+3)* # esta expresión está incompleta, pulso Enter pero no ejecuta
+
+ # sigue sin ejecutar
+ 5 # ahora sí!
[1] 25
> 2+3)*5
Error: unexpected ')' in "2+3)"
>
```

Figura 4: La consola de R: expresiones incompletas y mensajes de error.

Puede que hayas notado que R muestra cada línea de resultado anteponiendo [1]. Esto indica que ese valor es el primer resultado. En el ejemplo que se muestra en la Figura 3, 25 es el primer resultado (y en ese caso es el único). Ya volveremos sobre esto cuando veamos vectores, como objetos de R.

Ejecutar, llamar, correr y devolver

Cuando hablamos de *ejecutar*, *llamar* o *correr* nos referimos a pedirle a R realice algo, es decir, le estamos dando una instrucción o una entrada.

Cuando decimos que R nos *devuelve* algo, es que ha realizado algo que le hemos pedido, es decir, nos está dando una salida.

Por ejemplo, si escribimos en la consola

```
>2+3
```

y damos *Enter*, estamos pidiendo a R que realice la operación 2+3. En tal caso, R nos devolverá

```
[1] 5
```

2.4. Variables

Una variable refiere al nombre simbólico o *identificador* que se asocia a un dato que se almacena en la memoria principal del ordenador. A diferencia de una constante,

cuyo valor no puede cambiar, el dato que almacena la variable puede, como el término indica, variar durante la ejecución del programa.

Por ejemplo, para calcular el área de un círculo utilizamos la fórmula $a = \pi r^2$, donde a y r son variables, mientras que π y 2 son constantes.

Para crear una variable en R hay que asignarle un valor mediante una operación llamada *asignación*. La asignación consta de tres elementos:

1. el identificador o nombre de la variable
2. el operador de asignación
3. una expresión que produce el valor a asignar

Por ejemplo:

```
r <- 3
```

Aquí la variable tiene el nombre r , el operador de asignación está compuesto por los caracteres $<$ y $-$ escritos juntos ($<-$). Por último, la expresión es el valor 3. A partir de ahora, la variable r tiene asociado el valor 3 y puede utilizarse, a la vez, en otras expresiones y/o asignaciones. Por ejemplo,

```
a <- pi*r^2
a
## [1] 28.27433
```

Al ejecutar el código, se sustituye la variable r por su valor, 3, asignado a la variable a el valor 28.27433.

Hablaremos sobre asignaciones más adelante, en el apartado de operadores.

Nombre de variables

Al crear una variable debemos procurar que el nombre sea lo más claro y descriptivo posible, de forma que no conduzca a confusión y sea más fácil de comprender tanto por nosotros mismos como por otras personas.

Hay que tener en cuenta que existen algunas reglas a la hora de nombrar variables. Podemos usar letras, números, puntos y guiones bajos en el nombre de la variable, pero **los guiones bajos y los números no pueden ser el primer carácter** del nombre de la variable y en caso de que el nombre empiece con un punto, a este no puede seguirle un número.

También hay **palabras reservadas** que no se pueden usar como identificadores, como TRUE, FALSE, NULL, Inf, entre otras. La lista completa de palabras reservadas en R la puedes encontrar escribiendo `help(Reserved)` o `?Reserved` en la consola de comandos.

Si por alguna razón necesitas nombrar una variable con una palabra reservada o comenzar con un guion bajo o número, necesitarás usar acentos graves (*backticks*):

```
# Ejemplos de nombres válidos para variables en R
suma_123 <- 1+2+3
Suma <- 1+2+3
suma <- 1+2+3
SumaTres <- 1+2+3
Sumatres <- 1+2+3
.suma_123 <- 1+2+3
true <- 1+2+3 # notar que la palabra reservada es TRUE, todas mayúsculas

# Ejemplos de nombres que no son válidos para variables en R
_suma <- 1+2+3
Error: unexpected symbol in "_suma"
suma$123 <- 1+2+3
Error: unexpected numeric constant in "suma$123"
123suma <- 1+2+3
Error: unexpected symbol in "123suma"
.123suma <- 1+2+3
Error: unexpected symbol in ".123suma"
TRUE <- 1+2+3
```

```
Error in TRUE <- 1 + 2 + 3 :  
  invalid (do_set) left-hand side to assignment  
`TRUE` <- 1+2+3 # con los acentos graves funciona bien
```

2.5. Objetos

En R, todo es un objeto: variables, datos, funciones, resultados, etc., se guardan en la memoria RAM en forma de objetos con un nombre específico sin usar archivos temporales.

La explicación de esto es un tanto compleja y se sale del alcance de este apunte. Se relaciona con el paradigma de *programación orientada a objetos* y ese es todo un tema en sí mismo.

Simplemente tengamos presente que, al hablar de un objeto, estamos hablando de cualquier cosa que *existe* en una sesión de R y que tiene un nombre, como lo explica claramente Paradis en su texto:

«*Orientado a Objetos* significa que las variables, datos, funciones, resultados, etc., se guardan en la memoria activa del computador en forma de objetos con un nombre específico. El usuario puede modificar o manipular estos objetos con operadores (aritméticos, lógicos, y comparativos) y funciones (que a su vez son objetos)» (Paradis, E., 2003, p. 5).

Objetos atómicos

Como dijimos, todas las cosas que manipula R se llaman objetos. En general, éstos se construyen a partir de objetos más simples. De esta manera, se llega a los objetos más simples que son de cinco clases a las que se denomina atómicas y que son las siguientes:

- **character**: Los caracteres individuales y cadenas de caracteres tienen esta clase. Se delimitan mediante comillas simples o dobles.

- ▶ `numeric`: Todos los tipos numéricos, tanto enteros como en coma flotante y los expresados en notación exponencial, son de esta clase. También pertenecen a ellas las constantes `Inf` y `NaN`. La primera representa un valor infinito y la segunda un valor que no es numérico (*Not A Number*).

Aunque todos los tipos numéricos comparten una misma clase, que es `numeric`, el tipo de dato (que determina la estructura interna del almacenamiento del valor) es `double` (que significa de doble precisión). Distinguiremos entre clase y tipo más adelante.

- ▶ `integer`: En R por defecto todos los tipos numéricos se tratan como `double`. El tipo `integer` se genera explícitamente mediante la función `as.integer()`.
- ▶ `complex`: Cualquier valor que cuente con una parte imaginaria, denotada por el sufijo `i`, será tratado como un número complejo.
- ▶ `logical`: Esta es la clase de los valores booleanos, representados en R por las palabras reservadas `TRUE` y `FALSE`.

En el lenguaje, sin embargo, cada una de estas clases de datos no se encuentran ni se manejan de manera aislada, sino formando parte de alguna estructura más compleja. Estudiaremos las diferentes estructuras de datos que podemos manejar en R en el próximo tema.

Clase y tipo de un dato

Para los tipos de datos simples, en general la clase y el tipo coinciden salvo en el caso de datos numéricos no enteros, cuya clase es `numeric` siendo su tipo `double`.

Para obtener la clase y tipo de cualquier dato, ya sea constante o una variable, podemos utilizar las dos siguientes funciones:

- ▶ `class(objeto)`: Devuelve un vector con los nombres de las clases a las que pertenece el objeto.
- ▶ `typeof(objeto)`: Devuelve una cadena indicando el tipo de dato interno usado por el objeto.

Veamos algunos ejemplos.

```
class(3)
## [1] "numeric"
class(as.integer(3)) # para obtener un objeto de clase integer, debemos
indicarlo expresamente
## [1] "integer"
typeof(3)
## [1] "double"
typeof(as.integer(3))
## [1] "integer"
class(2+3i)
## [1] "complex"
typeof(2+3i)
## [1] "complex"
class('hola')
## [1] "character"
class("hola")
## [1] "character"
class(hola) # las comillas son obligatorias
Error: object 'hola' not found
typeof('hola') == typeof("hola") # el uso de comillas dobles o simples es
indistinguible
## [1] TRUE
V1 <- 2.6
V2 <- TRUE
(V3 <- 1/0)
## [1] Inf
(V4 <- 0/0)
## [1] NaN
class(V1)
## [1] "numeric"
class(V2)
## [1] "logical"
class(V3)
## [1] "numeric"
class(V4)
## [1] "numeric"
```

Coerción de tipos de datos en R

Los tipos de datos en R se pueden coercionar, es decir, forzar, para ser transformados de un tipo a otro, con las funciones que comienzan con `as.*()`, resumidas en la siguiente tabla:

Tipo	coerción
logical	as.logical()
integer	as.integer()
numeric	as.numeric()
complex	as.complex()
character	as.character()
factor	as.factor()

Tabla 1: Coerción de los tipos de datos más comunes

Podemos ver una lista completa de todas las funciones `as.*()` (¡116 en total!) en el paquete base mediante la instrucción `ls(pattern = "^as", baseenv())`.

La coerción de tipos se realiza de los tipos de datos más restrictivos a los más flexibles, esto es, en el siguiente orden: `logical -> integer -> numeric -> character`. Las coerciones no pueden ocurrir en orden inverso. Podemos coercionar un dato de tipo entero a uno numérico, pero no uno de cadena de texto a numérico.

Mención especial merecen los **factores**. Si bien un factor no se trata estrictamente un tipo de dato, los factores son un caso particular para la coerción. Los factores en R se utilizan para representar *datos categóricos*. Se puede pensar en ellos como vectores en los que cada elemento tiene una etiqueta asociada. Por ello pueden ser coercionados a tipo numérico y cadena de texto; y, a su vez, los datos numéricos y cadena de texto pueden ser coercionados a factor. Sin embargo, al coercionar un factor a tipo numérico, perdemos sus niveles. Volveremos sobre los factores más adelante.

2.6. Directorio de trabajo

El directorio o carpeta de trabajo es el lugar *en nuestra computadora* en el que se encuentran los archivos con los que estamos trabajando en R. Este es el lugar donde R buscará archivos para importarlos y al que serán exportados, a menos que indiquemos otra cosa.

Para saber cuál es tu directorio de trabajo actual, podemos ejecutar la función `getwd()` en la consola, que imprimirá la ruta del directorio de trabajo actual. Por tanto, la salida es la carpeta donde se guardarán todos tus archivos.

```
# Imprime la ruta del directorio de trabajo actual  
getwd()
```

¿Cómo podemos cambiar el directorio de trabajo?

Para evitar el engorroso procedimiento de tener que indicar todo el tiempo las rutas de acceso a los archivos (por ejemplo: `C:/Users/unir/Introduccion_a_R/datos/Poids_naissance.csv`) es posible establecer una carpeta de trabajo: esto es, una carpeta predeterminada donde el programa buscará los archivos a ejecutar y guardará los archivos que le indiquemos.

Existen varias alternativas para cambiar y definir el directorio de trabajo.

1. Desde la barra de menú: seleccionamos **Sesion > Set Working Directory > Choose directory...** y en la ventana que se abre elegir la carpeta que se desea.

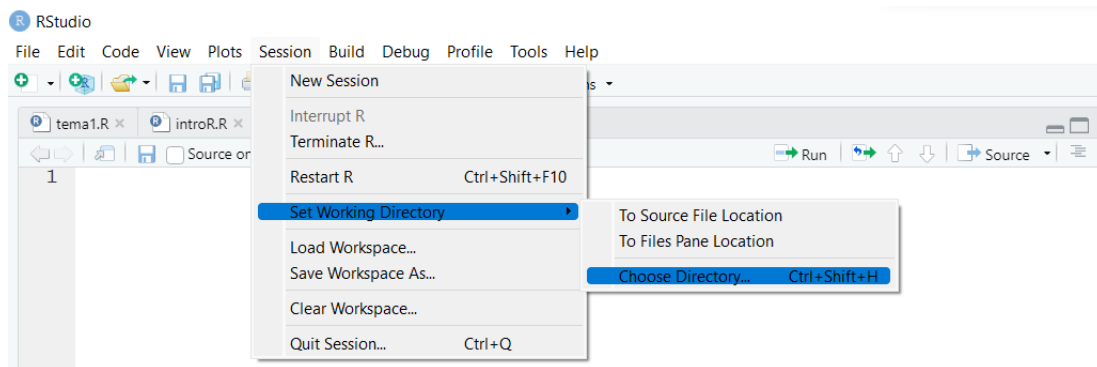


Figura 5: Cómo establecer el directorio de trabajo desde la barra de menú.

2. Desde la pestaña **Files**, del panel inferior derecho, y navegar por el árbol de directorios hasta llegar a la carpeta deseada o bien, clicando en los tres puntos que aparecen a la derecha, en el menú de opciones de la pestaña, elegir la carpeta deseada en la ventana que se abra y luego hacer clic en **More > Set As Working Directory**.

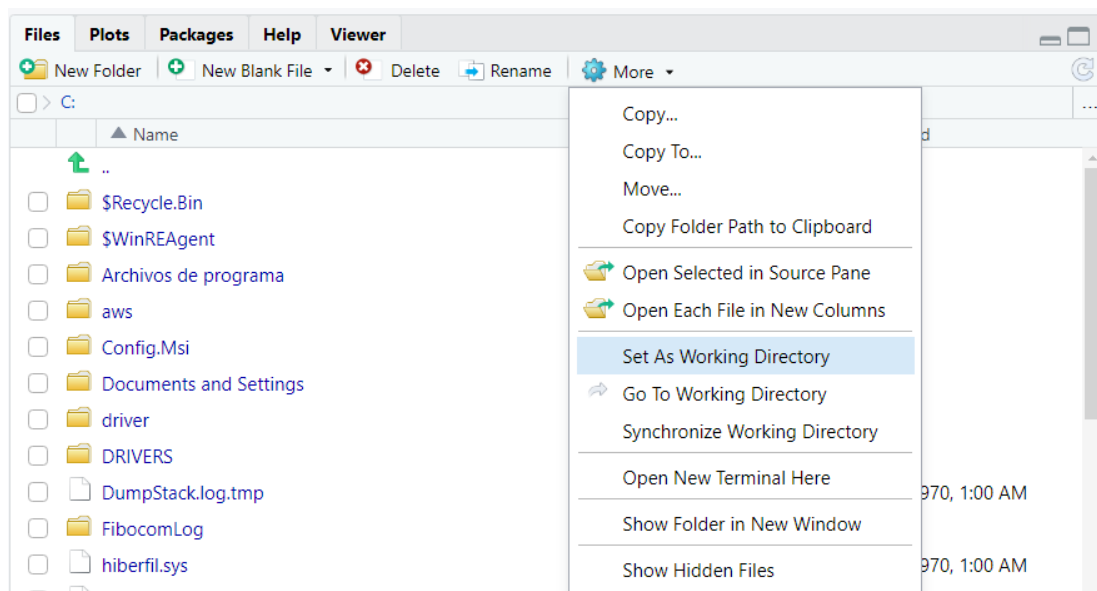


Figura 6: Cómo establecer el directorio de trabajo desde la pestaña Files.

3. Utilizar la función `setwd()` indicándole a R la ruta donde queremos trabajar.



```
R 4.2.1 · C:/Users/antog/OneDrive/Escritorio/Introduccion_a_R/ ↗
> setwd("C:/Users/antog/OneDrive/Escritorio/Introduccion_a_R")
>
```

Figura 7: Cómo establecer el directorio de trabajo utilizando la función `setwd()`.

Es importante destacar que esto lo tendrás que realizar cada vez que inicies RStudio. Para obtener un listado de los archivos que contiene la ruta establecida como directorio de trabajo, puedes utilizar la función `dir()`.

2.7. Scripts

Accede al video “*Los scripts de R*” para obtener una breve introducción al trabajo con scripts en R:



Accede al vídeo

Hasta ahora hemos utilizado la consola para ejecutar código. Aunque es un buen punto de partida, rápidamente se vuelve incómodo a medida que necesitamos trabajar con estructuras más complejas. Por otra parte, también mencionamos que RStudio cuenta con un editor de sintaxis que podemos utilizar a modo de cuaderno de trabajo: los *scripts*.

Los scripts son documentos de texto con la extensión de archivo `.R`, por ejemplo `Script_1.R`. Estos archivos son iguales a cualquier documento de texto, pero R los puede leer y ejecutar el código que contienen.

La edición de scripts R en RStudio cuenta con múltiples asistencias a la escritura de código: coloreado de código, autocompletado, ayuda sobre argumentos de funciones y miembros de objetos, etc.

Aunque R permite el uso interactivo, es recomendable que utilices scripts para guardar tu código en un archivo .R, de esta manera puedes volver a usarlo más tarde e, incluso, puedes compartirlo con otras personas.

Para crear un nuevo script en la barra de herramientas clicamos en **File > New File > R Script** (ver Figura 8) o bien podemos utilizar el atajo del teclado `Cmd/Ctrl+Shift+N`.

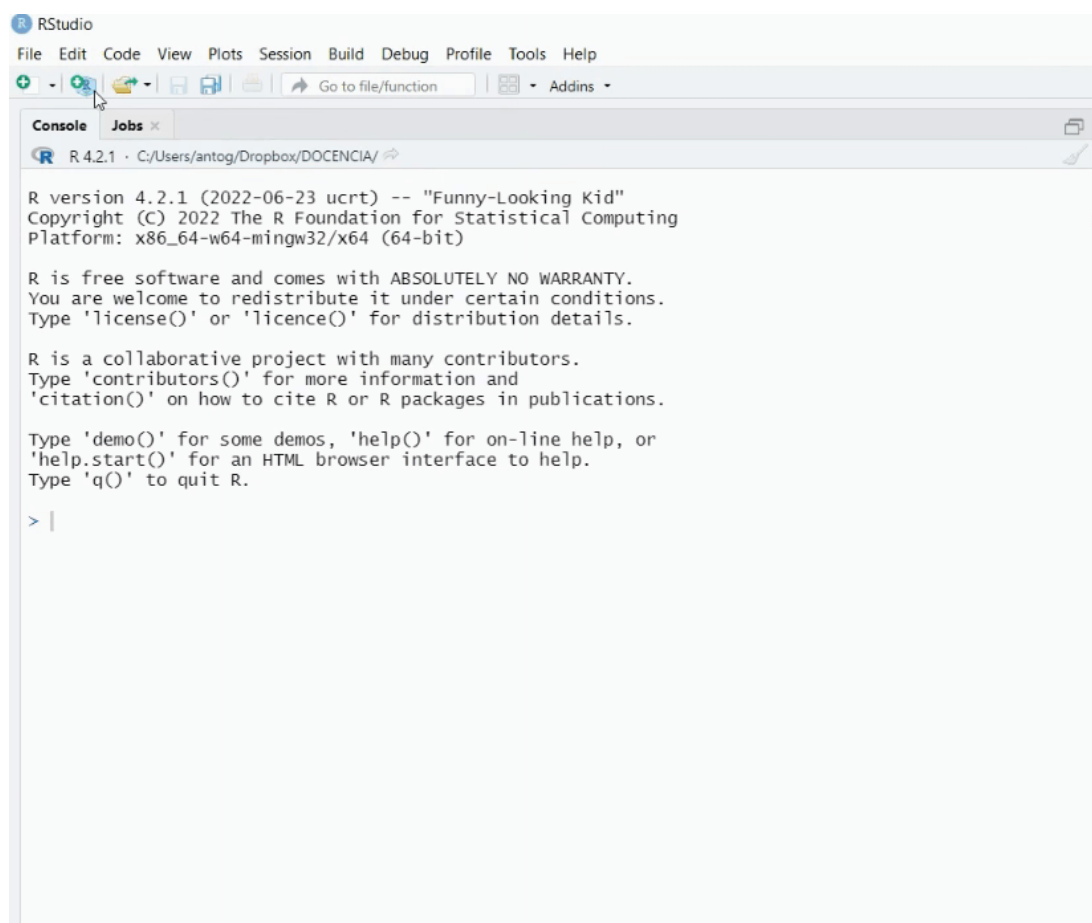


Figura 8: Crear un nuevo script.

Ahora podemos escribir las instrucciones líneas por línea. Las instrucciones las podemos ejecutar una a una o las podemos seleccionar y ejecutar en bloque. Para ejecutar las instrucciones tenemos varias alternativas:

- ▶ Hacemos clic en el botón *Run* (botón situado en la parte derecha de las opciones del panel de *script*).
- ▶ Pulsamos `Cmd+Return` (en Mac) o `Ctrl + Enter` (en Windows o Linux).

En el script podemos escribir como si fuera un cuaderno. El símbolo '#' sirve para indicarle a R que omita todo lo que hay a su derecha en la misma línea, lo que permite añadir comentarios en el script.

Probemos a escribir o copiar las instrucciones de arriba en un script:

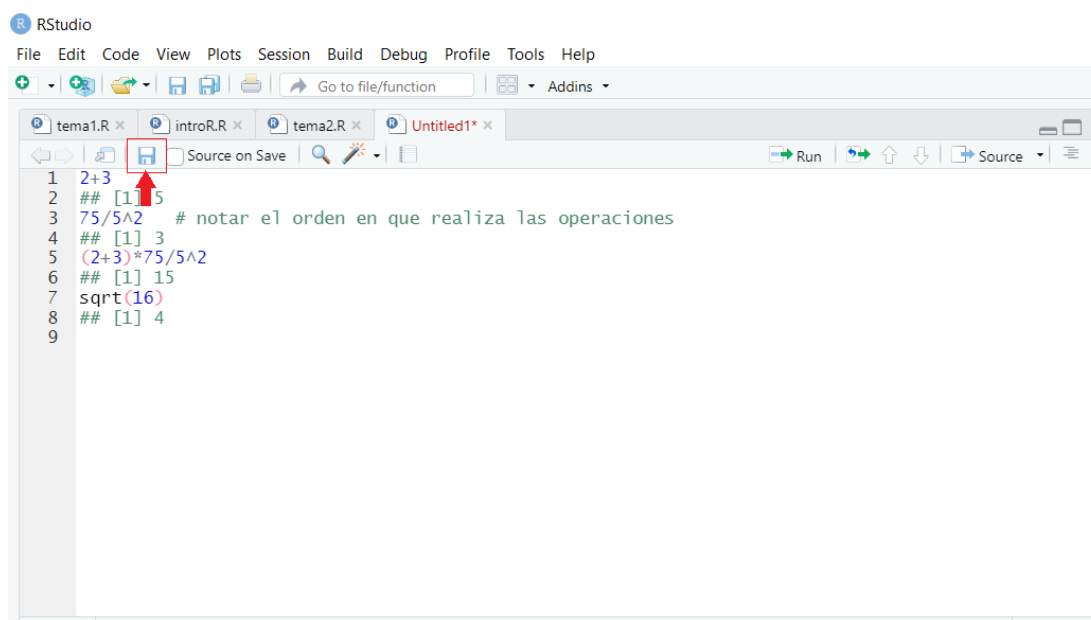


Figura 9: Ejemplo de script.

Para guardar un script:

- ▶ **File > Save as** y seleccionar la ruta donde se quiere guardar el fichero.
- ▶ Hacer clic en el ícono de un disquete que aparece en la parte izquierda de la cinta de opciones del script (ver Figura 9).

Si queremos abrir un script que hayamos guardado, tenemos varias opciones:

- ▶ **File > Open File**, esto abrirá una ventana en la que deberemos buscar el archivo .R que queremos abrir. Es una buena práctica hacer que la carpeta donde se encuentran los archivos de script sea el directorio de trabajo de R.
- ▶ Clicando sobre el script en la pestaña **Files**

2.8. Creación de Proyectos

Los proyectos de RStudio son muy útiles para organizar nuestros scripts en carpetas. De esta forma, al abrir un proyecto éste contendrá todos los archivos correspondientes al mismo. Además, al abrir el proyecto, la carpeta del proyecto se establecerá como directorio de trabajo, de modo que todo lo que guardes se guardará en la carpeta del proyecto.

En el siguiente video, “*Creación de proyectos*”, encontrarás el procedimiento a seguir para crear un nuevo proyecto:



Accede al vídeo

Vamos a ver cómo crear un nuevo proyecto. Para ello, haz clic en **File > New Project**

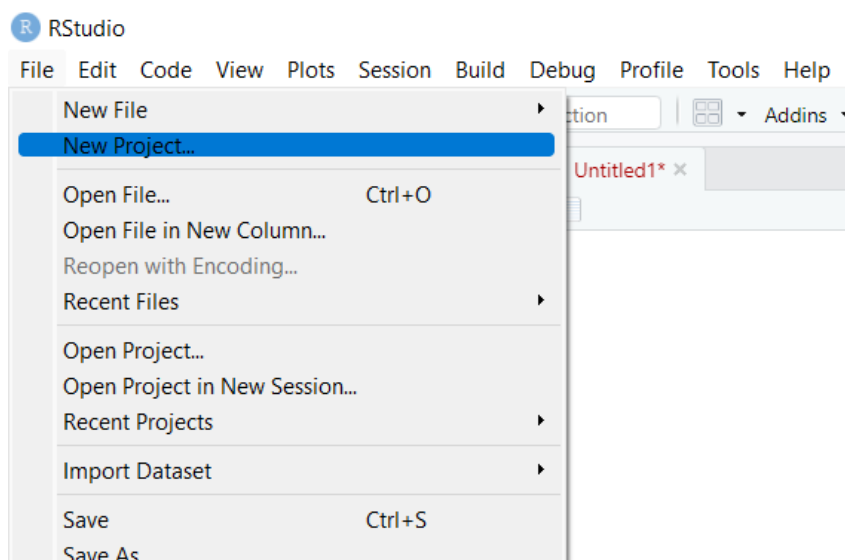


Figura 10: Creación de un nuevo proyecto.

y se abrirá la siguiente ventana:

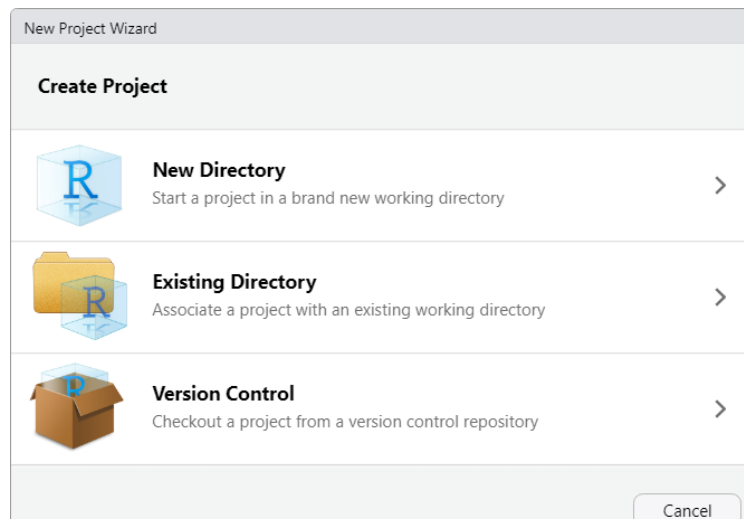


Figura 11: Creación de un nuevo proyecto.

Aquí, podremos elegir **New Directory** (crea un nuevo proyecto desde un nuevo directorio) o **Existing Directory** (crea el proyecto en un directorio existente).

Si eliges la opción **New Directory**, debes hacer clic en **New Project** y luego introducir el nombre del directorio (carpeta que se va a crear y que al mismo tiempo será el nombre del proyecto de R), por ejemplo: “ProyectoT2”, y establecer la ruta para el proyecto. Para terminar, hacemos clic en el botón **Create Project**. Al seguir este proceso se habrá creado una carpeta en el directorio seleccionado y un fichero nombre_directorio.Rproj.

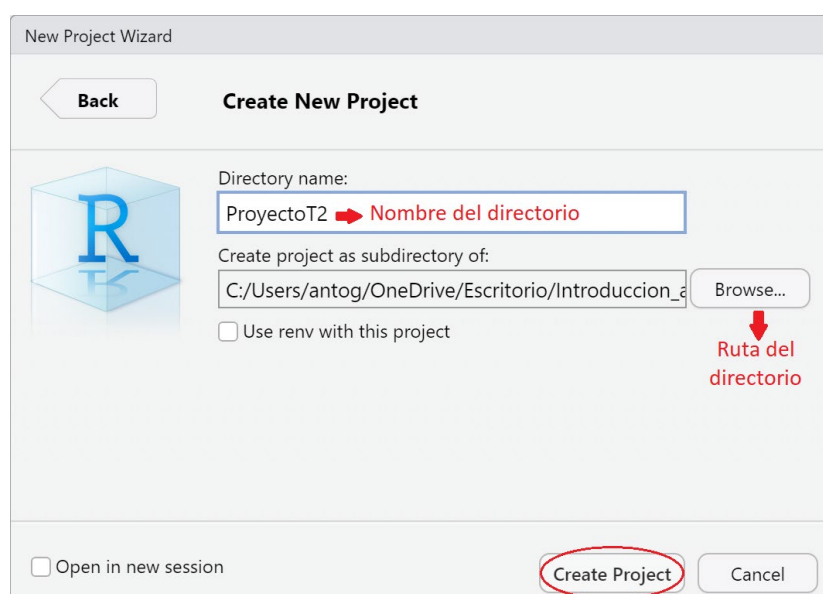


Figura 12: Creación de un nuevo proyecto en nuevo directorio.

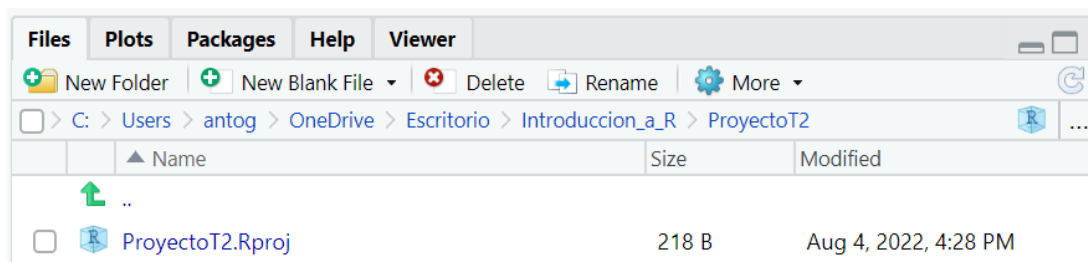


Figura 13: Archivo .Rproj que se crea luego de seguir todo el proceso.

Para crear un proyecto en una carpeta que ya existe, haciendo clic en **Browse**, seleccionamos la carpeta que queremos que sea el directorio donde se aloje el nuevo proyecto. Una vez elegida la carpeta, clicamos en **Create Project**.

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

¿Por qué es recomendable trabajar con proyectos?

Cualquier fichero que creamos (script de R, documento de Rmarkdown, history, etc.) y guardemos se guardará en la carpeta del proyecto. Esto es así porque, al abrir el proyecto, automáticamente queda establecida la carpeta del proyecto como directorio de trabajo de la sesión RStudio y, por lo tanto, como lugar donde R buscará y guardará los archivos que le indiquemos. Dicho de otra manera, cuando inicies R en este directorio de proyecto, o abras este proyecto con RStudio, todo el trabajo estará completamente autocontenido en este directorio.

2.9. Manejo de la biblioteca: paquetes adicionales

La versión básica del software R trae incorporada unas cuantas herramientas, siempre disponibles desde que abrimos el software, que ofrecen un gran abanico de funcionalidades con las que podemos, por ejemplo, cargar datos de fuentes externas, llevar a cabo análisis estadísticos y también obtener representaciones gráficas. Sin

embargo, en numerosas ocasiones, necesitaremos usar otras funcionalidades por lo que será necesario recurrir a paquetes externos, incorporando al entorno de trabajo las funciones y objetos definidos en ellos. Cada paquete es una colección de funciones, datos y documentación diseñada para atender una tarea específica. Veremos a continuación cómo descargar nuevos paquetes y cómo cargarlos al entorno de trabajo para poder utilizarlos.

En el video “La biblioteca de R”, se introduce como instalar, cargar y actualizar paquetes a en R:



Accede al vídeo

2.9.1. Descarga e instalación de paquetes

Cualquier paquete disponible en el repositorio oficial (CRAN) puede ser instalado fácilmente desde el propio RStudio. Esto puede hacerse de varias formas:

1. A través de la barra de menú de RStudio: **Tools > Install packages ...** En la ventana que se abre, escribimos el nombre del paquete que queremos y luego clicamos en *Install*. Conviene dejar tildada la opción *Install dependencies* para que se instalen también los paquetes necesarios para su correcto funcionamiento.

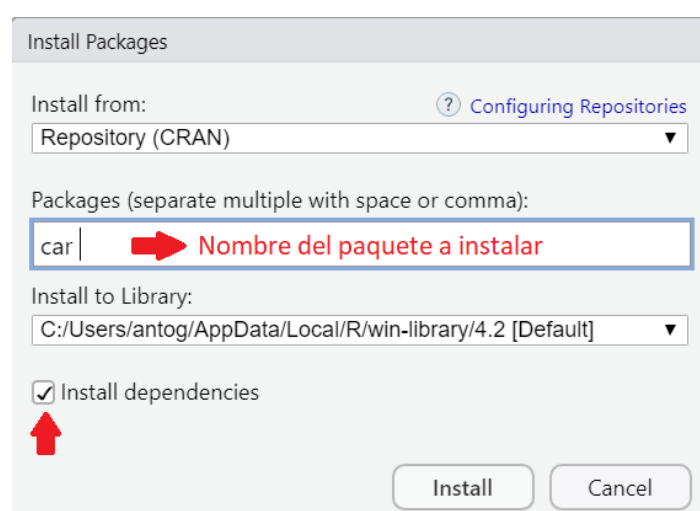
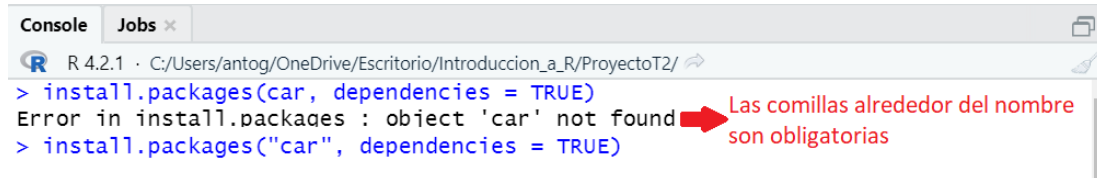


Figura 14: Ventana para instalar paquetes.

2. Desde el panel inferior derecho del escritorio de RStudio, en la pestaña **Packages**, haciendo clic en **Install**, nos abre la misma ventana de la Figura 14 (y por tanto el procedimiento es el mismo)
3. En la consola de R, usando la función `install.packages()`, dando como argumento el nombre del paquete que deseamos instalar, entre comillas.



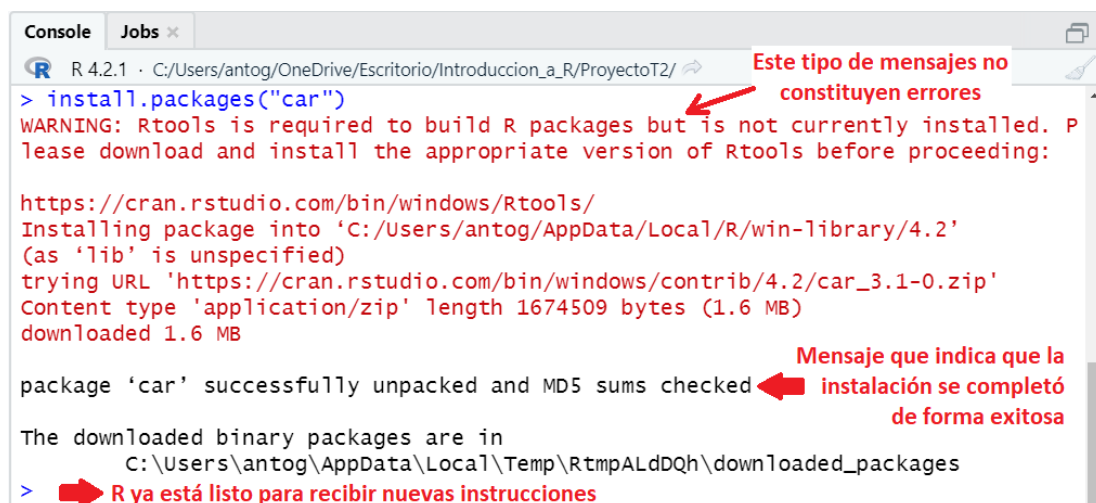
```
Console Jobs x
R 4.2.1 · C:/Users/antog/OneDrive/Escritorio/Introduccion_a_R/ProyectoT2/
> install.packages(car, dependencies = TRUE)
Error in install.packages : object 'car' not found
> install.packages("car", dependencies = TRUE)
```

Las comillas alrededor del nombre son obligatorias

Figura 15: Instalar paquetes desde la consola.

El argumento `dependencies = TRUE` obliga a R a instalar no sólo el paquete requerido, sino todos aquellos de los que dependa para funcionar correctamente.

Cualquiera sea la opción que hayamos elegido para instalar el paquete, una vez ejecutado tal comando saldrán distintos mensajes en la consola de R, generalmente con una apariencia como la que se muestra en la Figura 16. Vale destacar que muchas veces salen mensajes en rojo e incluso mensajes de alerta (*warnings*); por lo general se trata de mensajes que el programa muestra al usuario, pero que no implican que algo haya salido mal en la ejecución del comando. Como se aprecia en la Figura 16, R nos indica cuándo el paquete se descargó e instaló correctamente en la computadora, a la vez que la consola queda lista para seguir ejecutando análisis.



```
Console Jobs x
R 4.2.1 · C:/Users/antog/OneDrive/Escritorio/Introduccion_a_R/ProyectoT2/
> install.packages("car")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate version of Rtools before proceeding:
https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/antog/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.2/car_3.1-0.zip'
Content type 'application/zip' length 1674509 bytes (1.6 MB)
downloaded 1.6 MB

package 'car' successfully unpacked and MD5 sums checked
The downloaded binary packages are in
C:\Users\antog\AppData\Local\Temp\RtmpALdDQh\downloaded_packages
>
```

Este tipo de mensajes no constituyen errores

Mensaje que indica que la instalación se completó de forma exitosa

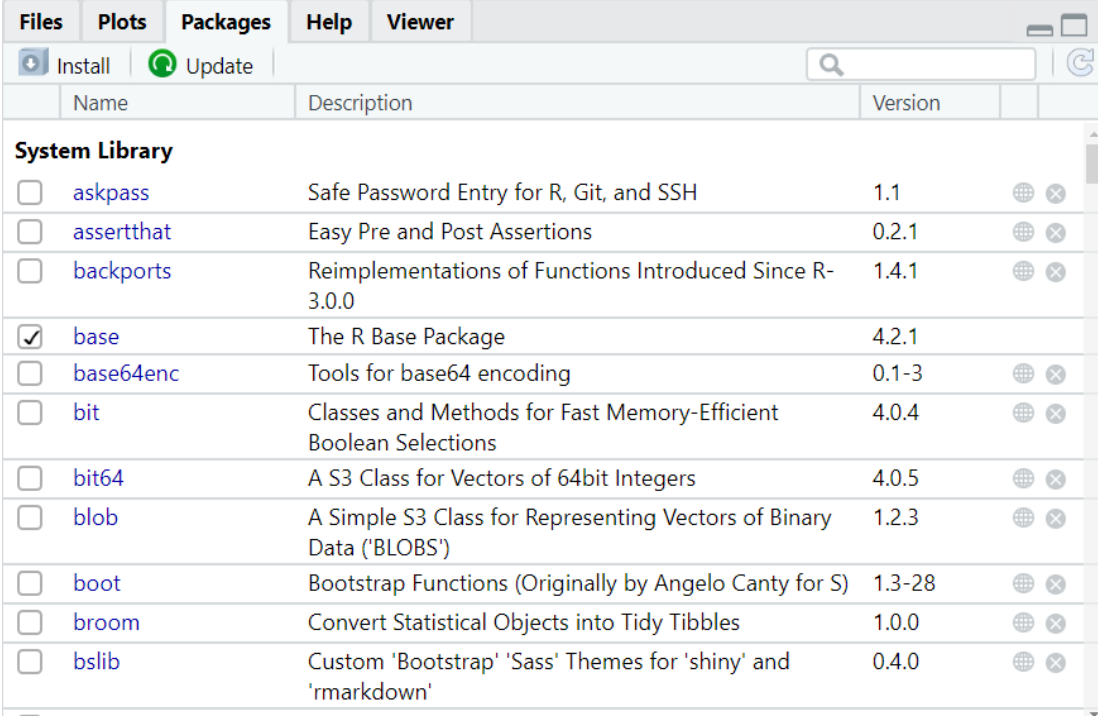
R ya está listo para recibir nuevas instrucciones

Figura 16: Consola de R indicando la instalación exitosa del paquete.

2.9.2. Cargar paquetes

Los paquetes deben **instalarse** en R **una única vez**, pero es necesario **cargarlos** en **cada sesión** (cada vez que abrimos R) para hacerlos disponible para su uso.

En la pestaña **Packages**, del panel inferior derecho, podemos consultar la lista de paquetes que tenemos instalados. Aquellos paquetes que aparecen marcados en esta lista son los que están cargados en la sesión de trabajo actual (ver Figura 17).



The screenshot shows the 'Packages' tab in RStudio. At the top, there are buttons for 'Install' and 'Update', and a search bar. Below this is a table with columns for 'Name', 'Description', and 'Version'. The table is divided into two sections: 'System Library' and a list of other packages. In the 'System Library' section, the 'base' package is checked, indicating it is loaded. Other packages like 'askpass', 'assertthat', 'backports', 'base64enc', 'bit', 'bit64', 'blob', 'boot', 'broom', and 'bslib' are listed with their descriptions and versions, but they are not checked.

Name	Description	Version
System Library		
<input type="checkbox"/> askpass	Safe Password Entry for R, Git, and SSH	1.1
<input type="checkbox"/> assertthat	Easy Pre and Post Assertions	0.2.1
<input type="checkbox"/> backports	Reimplementations of Functions Introduced Since R-3.0.0	1.4.1
<input checked="" type="checkbox"/> base	The R Base Package	4.2.1
<input type="checkbox"/> base64enc	Tools for base64 encoding	0.1-3
<input type="checkbox"/> bit	Classes and Methods for Fast Memory-Efficient Boolean Selections	4.0.4
<input type="checkbox"/> bit64	A S3 Class for Vectors of 64bit Integers	4.0.5
<input type="checkbox"/> blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.3
<input type="checkbox"/> boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-28
<input type="checkbox"/> broom	Convert Statistical Objects into Tidy Tibbles	1.0.0
<input type="checkbox"/> bslib	Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'	0.4.0

Figura 17: Lista de paquetes instalados en la computadora.

Así, si queremos cargar un paquete ya instalado basta con tildar el paquete requerido en esta lista. Otra forma es, desde la consola, utilizando la función `library()`:

```
library(car)      # las comillas no son necesarias
```

Una vez cargado puedes usar la función `help(package=)` con el nombre del paquete para ver su documentación. También encontrarás ejemplos útiles para comprender cómo funciona el paquete.

```
help(package=car)
```

Diferencia entre un Paquete y una Librería

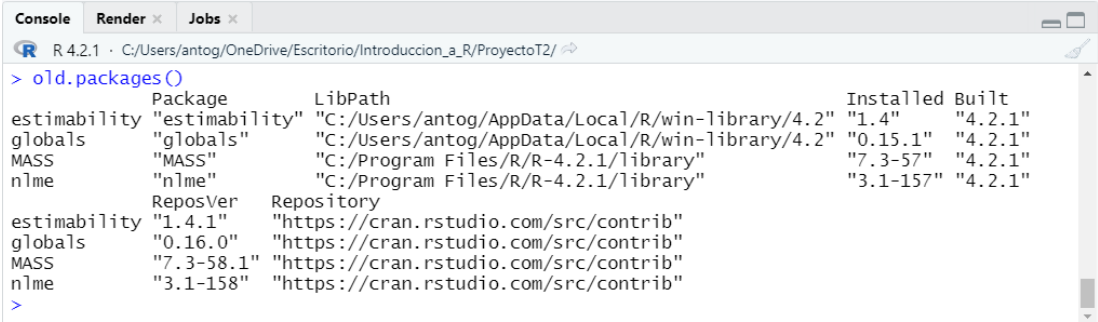
En muchas ocasiones existe la confusión entre lo que es un paquete y una librería, y podemos encontrarnos con gente llamando "librerías" a los paquetes.

Una cosa es la función `library()` usada para cargar un paquete, y que se refiere al lugar en dónde el paquete es localizado, habitualmente una carpeta en nuestro ordenador, y otra un paquete que es una colección de funciones y datos empaquetados de forma conveniente.

Dicho de otra manera, y en palabras de [Hadley Wickham](#), director científico de RStudio, y creador de la gran mayoría de paquetes del ecosistema *tidyverse*: “a package is a like a book, a library is like a library; you use `library()` to check a package out of the library” [Un paquete es como un libro, una librería es como una biblioteca; usas `library()` para sacar un paquete de la biblioteca] (Wickham, 2014)

2.9.3 Actualizar paquetes instalados

La función `old.packages()` compara los paquetes que tenemos instalados en nuestro ordenador con los que se encuentran en CRAN, y nos proporciona una lista de aquellos que cuentan con una versión más moderna.



	Package	LibPath	Installed	Built
estimability	"estimability"	"C:/Users/antog/AppData/Local/R/win-library/4.2"	"1.4"	"4.2.1"
globals	"globals"	"C:/Users/antog/AppData/Local/R/win-library/4.2"	"0.15.1"	"4.2.1"
MASS	"MASS"	"C:/Program Files/R/R-4.2.1/library"	"7.3-57"	"4.2.1"
nlme	"nlme"	"C:/Program Files/R/R-4.2.1/library"	"3.1-157"	"4.2.1"
	ReposVer	Repository		
estimability	"1.4.1"	"https://cran.rstudio.com/src/contrib"		
globals	"0.16.0"	"https://cran.rstudio.com/src/contrib"		
MASS	"7.3-58.1"	"https://cran.rstudio.com/src/contrib"		
nlme	"3.1-158"	"https://cran.rstudio.com/src/contrib"		

Figura 18: Lista de paquetes que pueden ser actualizados.

Para actualizarlos basta con ejecutar `update.packages()` R nos irá preguntando *paquete por paquete* si deseamos proceder o no a su actualización.

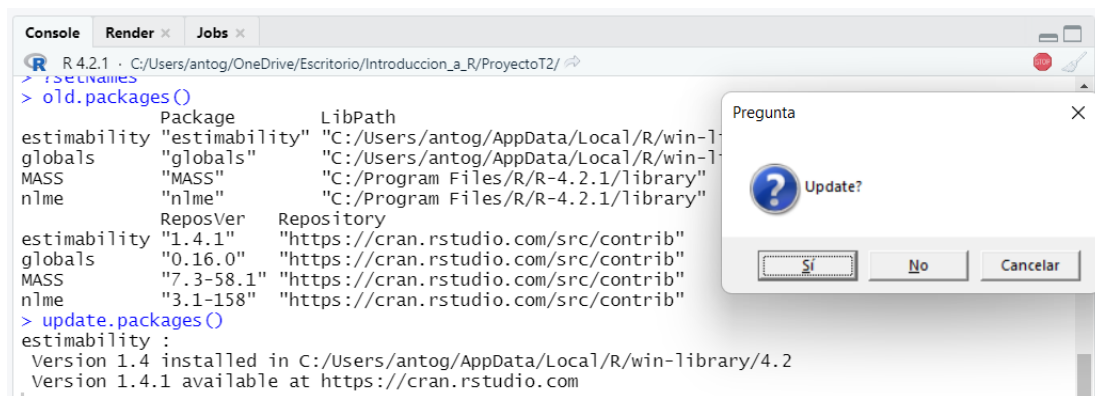


Figura 19: Cuadro de diálogo para confirmar o no la actualización de un paquete.

En RStudio, la actualización de los paquetes puede llevarse a cabo simplemente clicando en el icono *Update* en la pestaña *Packages*.

2.10. Referencias bibliográficas

Wickham, H. [@hadleywickham] (8 de diciembre de 2014). *a package is a like a book, a library is like a library; you use library() to check a package out of the library.* [Tweet]. Twitter.

<https://twitter.com/hadleywickham/status/541948905009586176?lang=en>

ⁱ Para establecer una carpeta de trabajo, puedes clicar los tres puntos que aparecen en la ventana *Files*, y en la ventana emergente que se abre, elegir la carpeta que desees.