Herramientas del BIG DATA y analytics

Empezando a usaR. Una guía paso a paso

Autores
Julio César Alonso
Maria Paula Ocampo

Editorial Universidad Icesi
UNIVERSIDAD ICESI

Empezando a usaR: Una guía paso a paso

Julio César Alonso[1] Maria Paula Ocampo[2]

1 de agosto de 2022

[1]CIENFI - Universidad Icesi, jcalonso@icesi.edu.co
[2]CIENFI - Universidad Icesi, mpocampo@icesi.edu.co

Julio César Alonso C. y Maria Paula Ocampo A.

Colección «Herramientas del Big Data y Analytics», vol. 1

Cali. Universidad Icesi, 2022.

110 páginas.

Incluye referencias bibliográficas.

ISBN: 978-628-7538-75-7 (eBook).

DOI: https://doi.org/10.18046/EUI/bda.h.1

Palabras Clave: 1. R | 2. Analítica | 3. Introducción | 5. Lenguaje estadístico | 5. Big Data Analytics

Clasificación Dewey: 545 ddc 21

CIENFI - Centro de Investigación en Economía y Finanzas

www.icesi.edu.co/centros-academicos/cienfi

Rector: Esteban Piedrahita Uribe

Secretaria General: María Cristina Navia Klemperer

Director Académico: José Hernando Bahamón

Coordinador editorial: Adolfo A. Abadía

Corrección de estilo: Claudia L. González G.

Diseño de portada: Sandra Moreno

Fotos tomadas por: Julio César Alonso

Editorial Universidad Icesi

Calle 18 No. 122-135 (Pance), Cali – Colombia

Teléfono: +57 (2) 555 2334 | E-mail: editorial@icesi.edu.co

http://www.icesi.edu.co/editorial

Publicado en Colombia – *Published in Colombia*

La publicación de este libro se aprobó luego de superar un proceso de evaluación doble ciego.

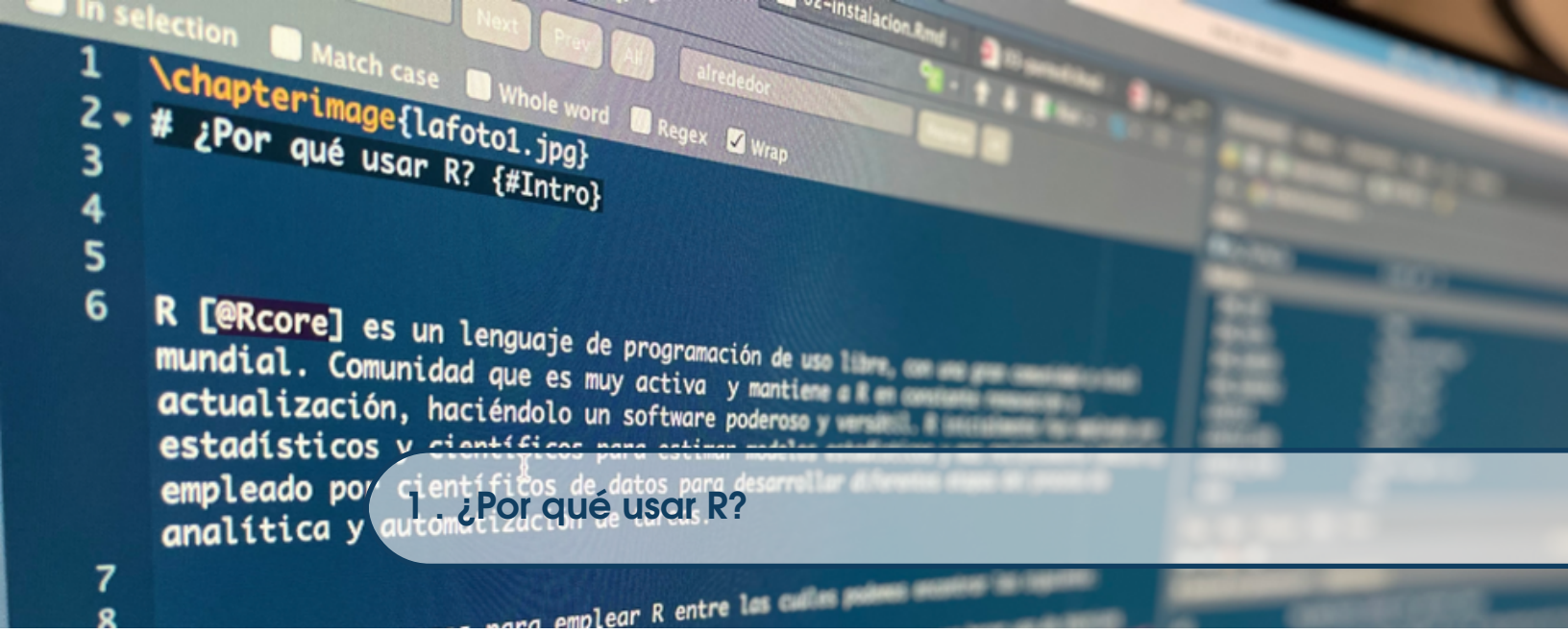
Índice general

Prefacio

Si estás leyendo este libro, es probable que ya uses o estés decidiendo emplear el software R para tus análisis cuantitativos. Ya sea que vayas a emplear herramientas cuantitativas o cualitativas, para hacer análisis de carácter científico o resolver preguntas de negocio, R te permitirá cumplir con tu objetivo.

Esta obra tiene como propósito presentar una primera aproximación a R, que va desde su instalación, hasta presentar *tips* de buenas prácticas para escribir código. Si eres nuevo en el universo de R, este libro será un buen punto de arranque. En el caso de que hayas tomado un curso de R previamente y eres un usuario independiente de R, tal vez este libro no te aporte nuevos conocimientos, pero será una herramienta de consulta de algunos conceptos básicos.

Esta obra recoge nuestra experiencia trabajando con R en el CIENFI (Centro de Investigación en Economía y Finanzas) de la Universidad Icesi para resolver problemas con datos. En el CIENFI, empleamos R para la transformación de datos en conclusiones que faciliten la toma de decisiones en organizaciones privadas y públicas. Así mismo, la Universidad Icesi tiene el único *mirror* del CRAN (The Comprehensive R Archive Network) en Colombia desde 2004. Toda esta experiencia la plasmaremos en en esta obra para así asegurarnos de que nuevas generaciones de profesionales continúen fortaleciendo a la comunidad de R alrededor del mundo. Esperamos que encuentres esta obra útil y la compartas con otros futuros usuarios interesados.



R (R Core Team, 2018) es un lenguaje de programación de uso libre con una gran comunidad a nivel mundial. La cual al ser activa mantiene a R en constante renovación y actualización, esto lo hace un software poderoso y versátil. R inicialmente fue empleado por estadísticos y científicos para estimar modelos estadísticos, recientemente es también empleado por científicos de datos para desarrollar diferentes etapas del proceso de analítica y automatización de tareas.

Hay múltiples razones para emplear R, entre las cuáles podemos encontrar las siguientes:

- **R es gratuito**. No tenemos que pagar licencias costosas para usar técnicas cuantitativas de punta desarrolladas por científicos alrededor del mundo. R es un software de procesamiento y análisis de datos particularmente popular entre la comunidad científica y estadística, el cual es distribuido gratuitamente bajo licencia pública general (GNU).
- **R es multiplataforma**. R está disponible para usuarios de tres sistemas operativos: Unix-Like, Windows y Mac-OS. Los códigos y análisis que se realizan bajo un sistema operativo pueden ser empleados por otros usuarios que manejen otro sistema operativo.
- **R tiene métodos estadísticos y de inteligencia artificial de última tecnología, además crece todos los días**. R tiene más de 2000 paquetes[1] (conjuntos de códigos que se pueden adicionar a R) diseñados para efectuar operaciones especiales. Encontraremos numerosos paquetes para realizar numerosas tareas y cálculos. Es poco común encontrar un método estadístico o de inteligencia artificial, para el cuál no exista ya un paquete. Los paquetes también

[1] En el enlace https://cran.r-project.org/web/views/ se puede encontrar un listado no comprensivo de paquetes organizados por temas.

son gratuitos.

- **R tiene documentación en línea de buena calidad y completa** construida por una comunidad científica excepcional. Esto hace que R se encuentre respaldado por la comunidad científica.
- **R crea gráficos de alta calidad**. R crea visualizaciones sensacionales incomparables a los de otros paquetes gratuitos o pagos[2]. Adicionalmente, es posible crear visualizaciones interactivas[3].
- **R nos hace pensar en cómo resolver el problema.** Al ser un lenguaje de programación, es necesario pensar en los pasos necesarios para resolver el problema. La interfaz de línea de comando es ideal para aprender haciendo.
- **R permite hacer investigación reproducible** (*reproducible research*). Es decir, R permite documentar los resultados obtenidos paso a paso, mostrando el flujo completo de procesamiento de los datos por medios de *scripts* e informes que cualquier investigador puede constatar. La reproducibilidad en la investigación es importante debido a que garantiza transparencia y confianza en los hallazgos. Además, la reproducibilidad permite entender el procedimiento realizado paso a paso.
- **Con R puede hacerse "lo mismo" de diversas formas** Al ser un lenguaje de programación, cada persona puede expresarse de manera diferente para resolver una misma pregunta y llegar a una misma respuesta.
- **R no es solamente utilizado en la academia**, también es empleado en diferentes áreas de la industria. Las organizaciones emplean cada vez más a R para hacer análisis de datos y responder sus preguntas de negocio (*business analytics*). Esto hace que los profesionales que son usuarios de R tengan ventajas al conseguir un trabajo en la industria, frente a aquellos que no lo emplean.
- **R está listo para trabajar en el mundo del *Big Data* y *business analytics*.** Por eso, cada vez más científicos de datos emplean este software para sus análisis como parte de un arsenal de lenguajes disponibles para hacer ciencia de datos.
- **R se integra a otros softwares**. Cada vez son más los softwares comerciales que crean "puertas" para realizar análisis en R y poderlos integrar al flujo de trabajo diario. También es posible integrar el uso de R con otros lenguajes como Python.

- **R permite el cálculo distribuido**. R puede emplear los diferentes núcleos que tienen los computadores en la actualidad, de esta manera permite realizar cálculos distribuidos en diferentes núcleos de procesamiento de los computadores. Lo anterior agiliza los cálcu-

[2] Algunos ejemplos de visualizaciones creadas en R se pueden encontrar en la Galería de Gráficos de R

[3] Algunos ejemplos de visualizaciones creadas en R se pueden encontrar en la página web del Cienfi. Por ejemplo, en este link encontraras algunas visualizaciones interactivas.

los, reduciendo los tiempos de procesamiento.

En últimas, R es una comunidad de colaboración alrededor del mundo que está disponible de manera gratuita para quien desee emplearla. En la actualidad, este lenguaje de programación estadístico permite realizar desde cálculos muy sencillos, hasta programar diferentes rutinas que permiten realizar operaciones más complejas, a partir de líneas de código.

El proyecto R inicia en 1992 en la Universidad de Auckland en Nueva Zelanda. Los profesores Robert Gentleman y Ross Ihaka crearon este software como una evolución del software estadístico comercial S-PLUS. El nombre de R se originó por las iniciales de sus primeros autores, pero también como un símbolo de evolución sobre el software comercial S-PLUS. En 1995 apareció la primera versión de R distribuida como software de código abierto bajo el tipo de licencia GPL2. En 1997 se conforma el *Core Team* que desarrolla constantemente a R.

La primera versión estable de R fue publicada el 29 de febrero de 2000. En ese momento la comunidad de R creció inicialmente entre los estadísticos. Posteriormente, la comunidad científica de diferentes áreas del conocimiento comenzó a adoptar R, haciendo de este un lenguaje estadístico alternativo a los softwares comerciales disponibles. Hoy, R ha superado el mundo de la estadística y se ha convertido en un lenguaje para procesar y visualizar datos, incluyendoal mundo del *Big Data*. En 2020 se lanzó la versión 4.0 de R tras 20 años de crecimiento exponencial de los usuarios de este lenguaje de programación.

Toda la información relacionada con R puede ser encontrada en "Proyecto R" (R project en inglés). Los archivos del paquete central y los demás paquetes se pueden encontrar en la CRAN (*Comprehensive R Archive Network*), a la cual puede accederse a través de una serie de *mirrors* distribuidos en todo el mundo. Por ejemplo, la Universidad Icesi mantiene un *mirror* de R, al cual se accede a través de la página http://www.icesi.edu.co/CRAN/. Existen diferentes *mirrors* en los cinco continentes.

Debido a que la interfaz de R puede ser poco amigable con un usuario sin experiencia o familiarizado con códigos y *scripts* de programación, se han desarrollado diferentes interfaces para hacer más agradable su visualización. Ejemplos de ellas van desde aproximaciones que permiten el uso de R con el ratón usando clic como RCommander (Fox, 2017)[4], hasta RStudio que provee una interfaz amigable para controlar a R empleando comandos como se haría directamente en R. RStudio fue diseñado para facilitar el trabajo en R sin importar la tarea desem-

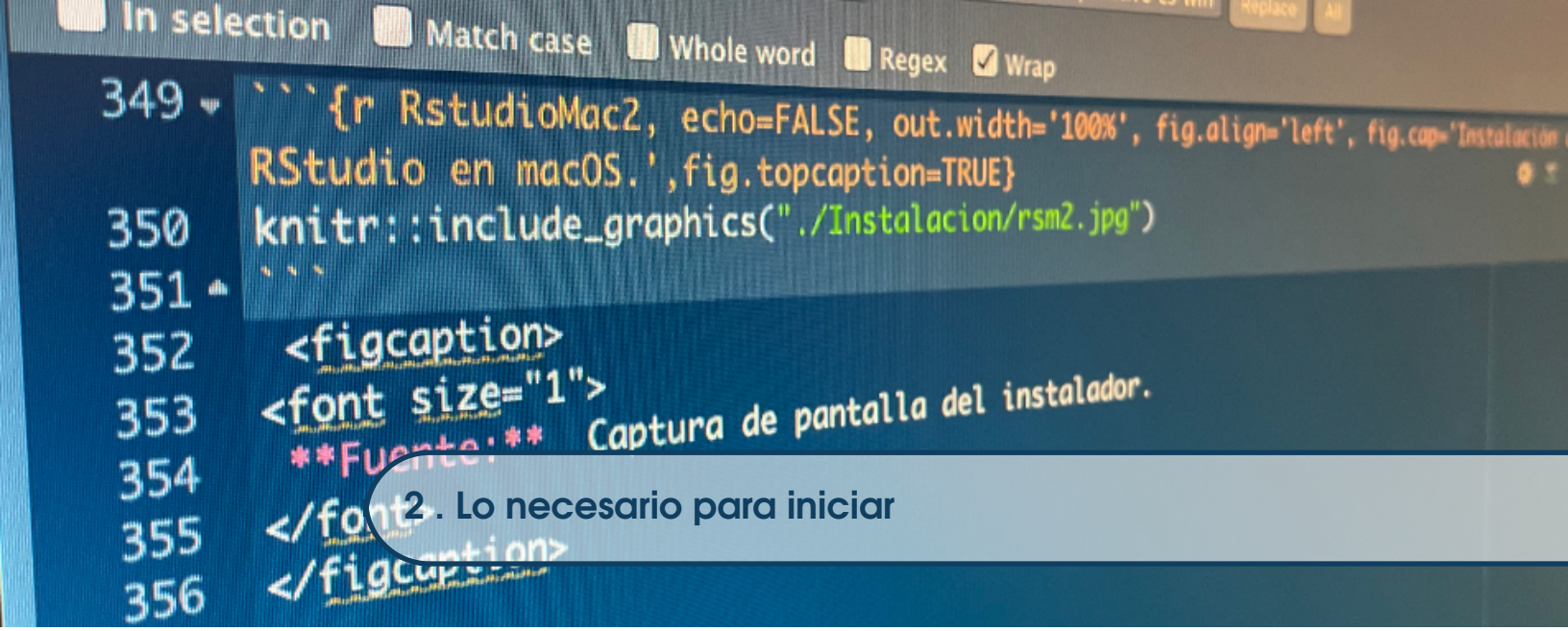
[4]Para una introducción a Rcommander puede consultar a Alonso Cifuentes y Jaramillo (2011) .

peñada[5]. Otra opción es Tinn-R que sólo está disponible para Windows[6].

En el Capítulo 2 veremos el paso a paso de la instalación de R y RStudio para que podamos empezar a correr los primeros códigos. Posteriormente, en el Capítulo 3 discutiremos las partes que componen la interfaz de R y RStudio. En el Capítulo 4 presentaremos una introducción a los objetos de R y cómo hacer operaciones básicas. En el Capítulo 5 se presentan los objetos más sencillos y algunos compuestos. En el Capítulo 6 se presenta el directorio de trabajo de R. En el Capítulo 7 se ofrece una introducción a los paquetes en R. Finalmente, en el Capítulo 8 se muestran diferentes maneras de cargar una base de datos en R.

[5]RStudio puede ser descargado desde http://www.rstudio.com/ide/download/ para cualquier sistema operativo

[6]Se puede descargar desde http://sourceforge.net/projects/tinn-r/.



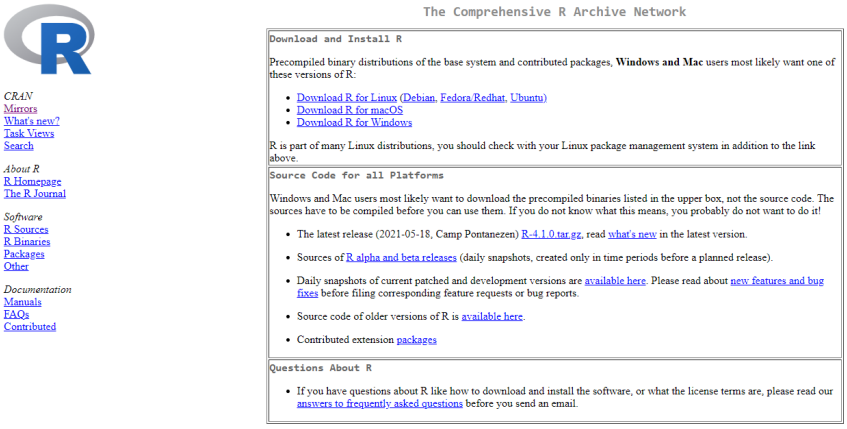
2. Lo necesario para iniciar

Para empezar a trabajar con R primero debemos instalarlo. Con esto será suficiente para cargar los datos, estimar modelos estadísticos, hacer visualizaciones e incluso entrenar modelos de inteligencia artificial. No obstante, es recomendado instalar una interfaz que permita trabajar de manera más cómoda con los diferentes elementos de R. Es decir, ponerle una suerte de chasis bonito al potente motor de análisis que es R, y mejorar nuestra experiencia de uso del programa y así tenerlo todo en un mismo *Core layout*. Aunque existen varias interfaces, RStudio es tal vez el chasis multiforma más empleado. En este capítulo veremos paso a paso la instalación de R y RStudio para Windows y macOS. Luego, haremos un recorrido por cada elemento del software R y su integración en la interfaz de RStudio. Si ya tienes instalado R y RStudio en tu computador, puedes omitir este capítulo y continuar con el Capítulo 3.

2.1 Instalación de R

El primer paso para instalar R es descargarlo desde un repositorio. Alrededor del mundo existen diferentes repositorios que permiten descargar R del sitio más cercano a nuestra ubicación. Cualquier buscador te llevará rápidamente a un repositorio cercano si buscas el término **CRAN** (por la sigla en inglés Comprehensive R Archive Network). O puedes acceder al repositorio empleando el siguiente link: https://cran.r-project.org/. Verás una página principal como la presentada en la Figura 2.1. El siguiente paso es descargar el instalador para tu respectivo sistema operativo y ejecutarlo.

Figura 2.1. Página principal de CRAN



Fuente: Esta captura de pantalla fue tomada de CRAN (https://cran.r-project.org/).

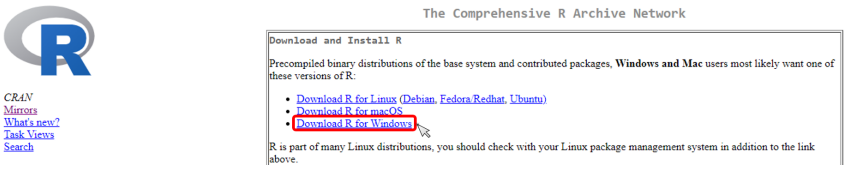
En la sección 2.1.1 veremos paso a paso cómo instalar R en Windows. Para los usuarios de macOS, la misma información se presentará en la sección 2.1.2.

2.1.1 Instalación de R para Windows

(Si eres usuario de macOS, continúa en la sección 2.1.2.)

Para descargar el instalador de R, elegiremos la versión correspondiente a Windows haciendo clic en **Download R for Windows**, como se indica en la Figura 2.2.

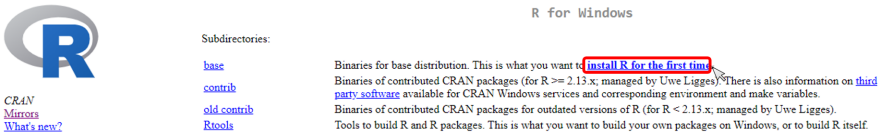
Figura 2.2. Paso 1 para descargar instalador de R en Windows



Fuente: Esta captura de pantalla fue tomada de CRAN (https://cran.r-project.org/bin/windows/).

Luego damos clic en **install R for the first time** (Ver Figura 2.3).

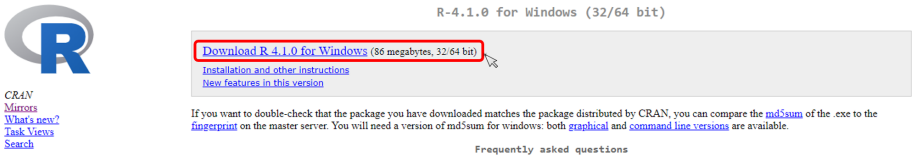
Figura 2.3. Paso 2 para descargar instalador de R en Windows



Fuente: Esta captura de pantalla fue tomada de CRAN (https://cran.r-project.org/bin/windows/).

En la siguiente ventana emergente damos clic en **Download R 4.1.2 for Windows** (Ver Figura 2.4). Esto iniciará la descarga del archivo con el instalador. Ejecuta el archivo descargado. Vale la pena mencionar que se te mostrará la versión más reciente de R (es probable que ahora yesté una versión más actualizada disponible).

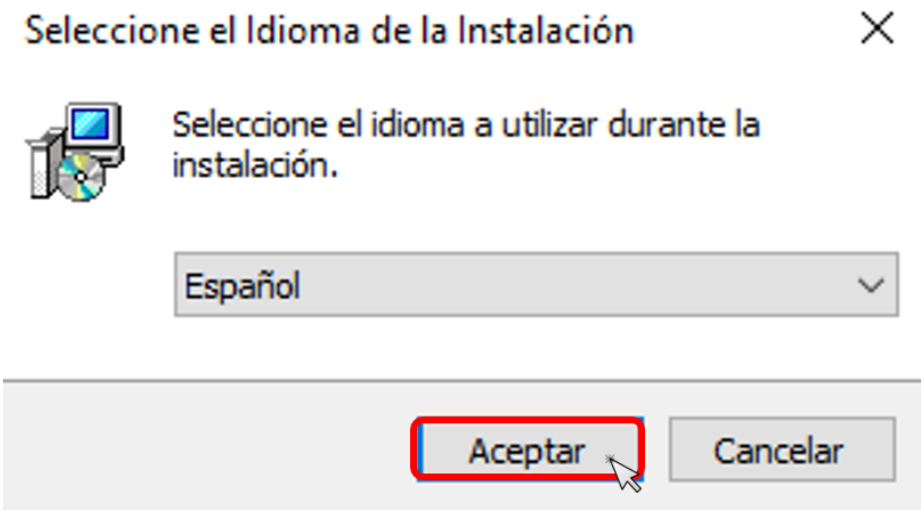
Figura 2.4. Paso 3 para descargar instalador de R en Windows



Fuente: Esta captura de pantalla fue tomada de CRAN (https://cran.r-project.org/bin/windows/base/).

Tras ejecutar el archivo descargado, lo primero que aparecerá será una ventana que permitirá elegir el idioma a utilizar durante la instalación. Para este manual utilizaremos "Español" (Ver Figura 2.5).

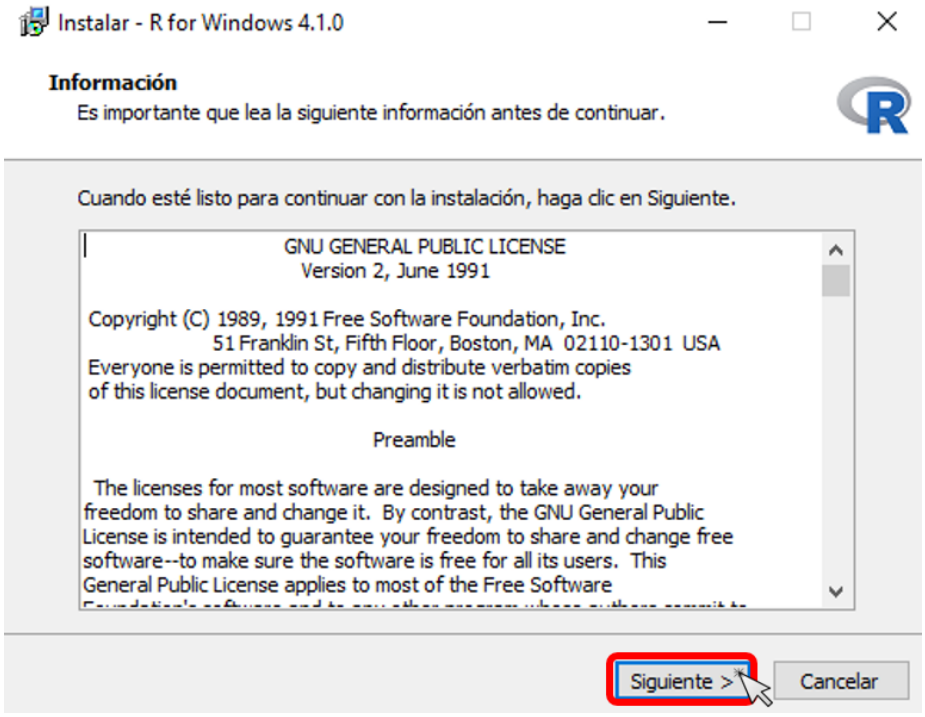
Figura 2.5. Paso 1 instalación de R en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

Después de elegir el idioma, aparecerá la licencia de uso de R. Ésta en términos generales nos indica que el programa es de uso libre y gratuito. Haz clic en **Siguiente** después de leer los términos de la licencia (Ver Figura 2.6).

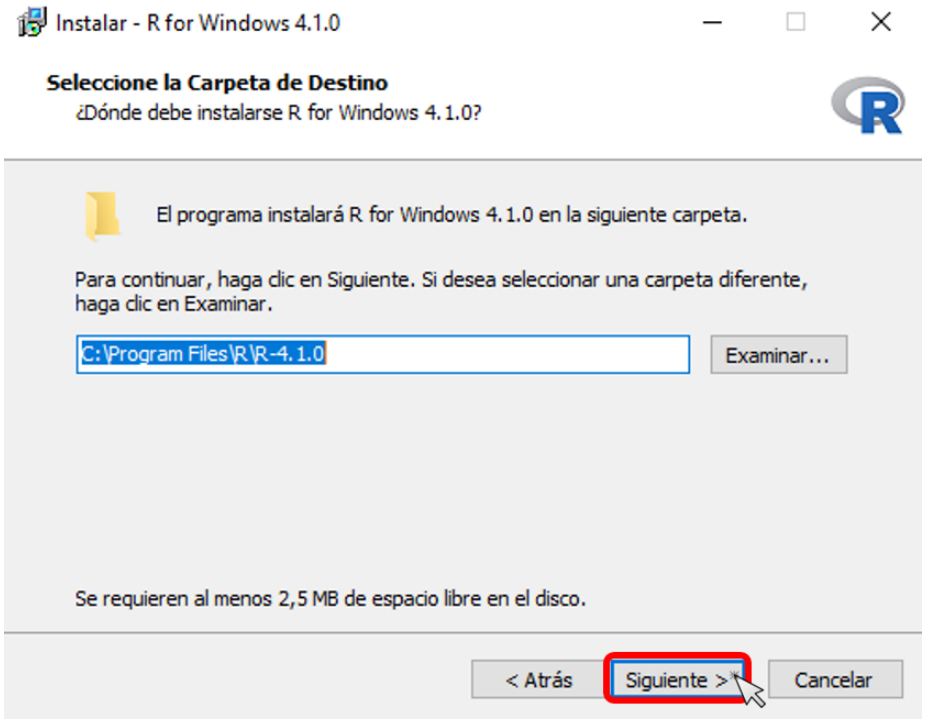
Figura 2.6. Paso 2 instalación de R en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

La ventana siguiente muestra la dirección o carpeta de destino en donde se instalará R. Recomendamos que uses la dirección predeterminada y sólo da clic a **Siguiente**(Ver Figura 2.7).

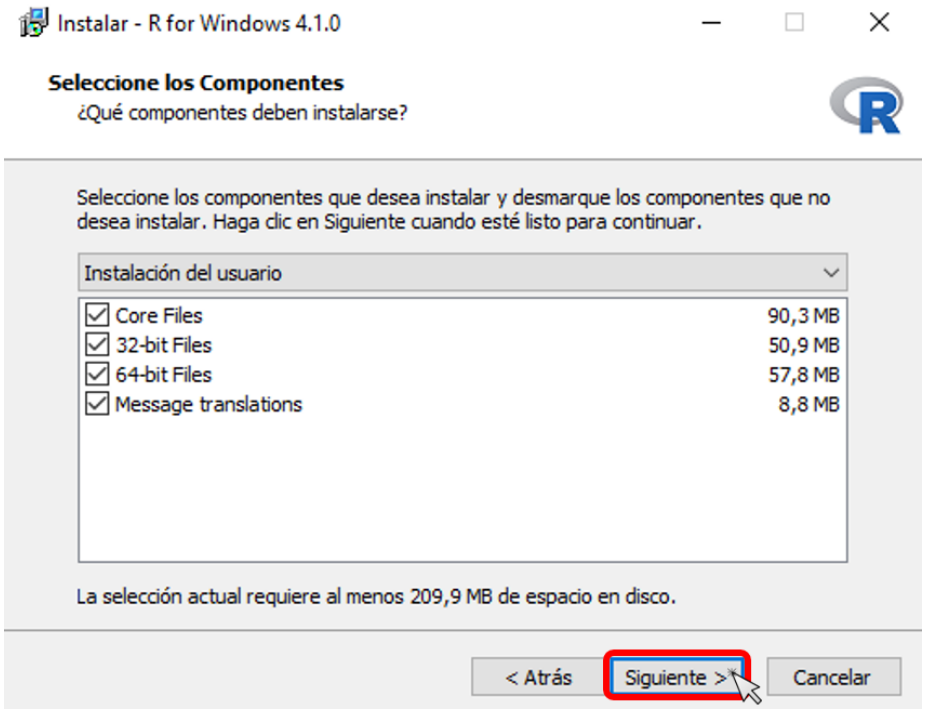
Figura 2.7. Paso 3 instalación de R en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

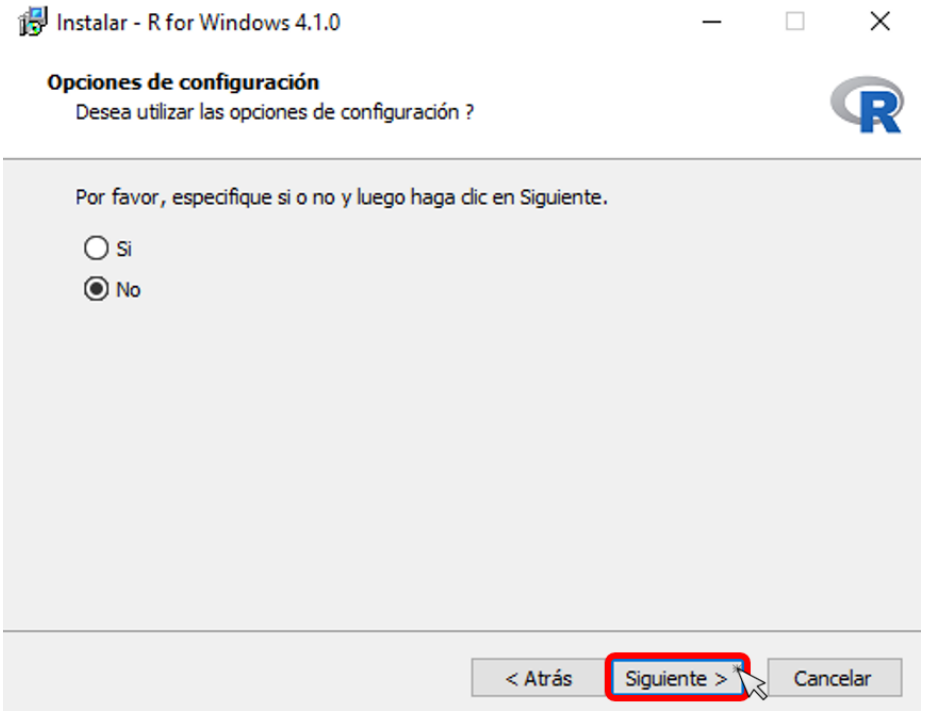
Para continuar, haga clic en **Siguiente** en las ventanas que corresponden a componentes (Figura 2.8), opciones de configuración (Figura 2.9) y creación de carpeta en el menú inicio (Figura 2.10). En ninguno de estos casos es necesario cambiar las opciones predeterminadas.

Figura 2.8. Paso 4 instalación de R en Windows



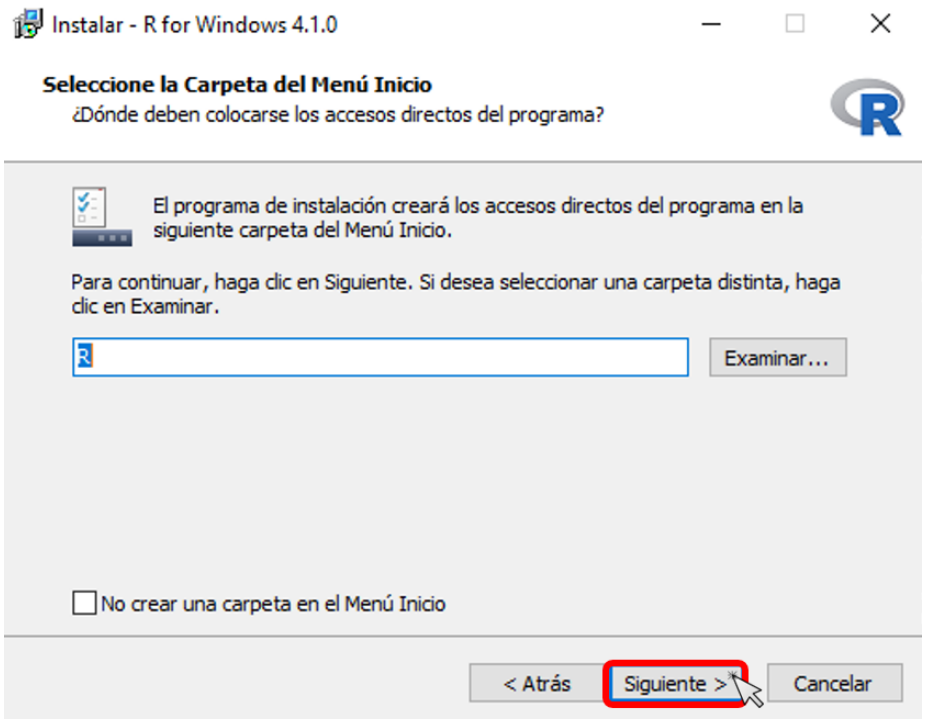
Fuente: Esta captura de pantalla fue tomada del instalador de R.

Figura 2.9. Paso 5 instalación de R en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

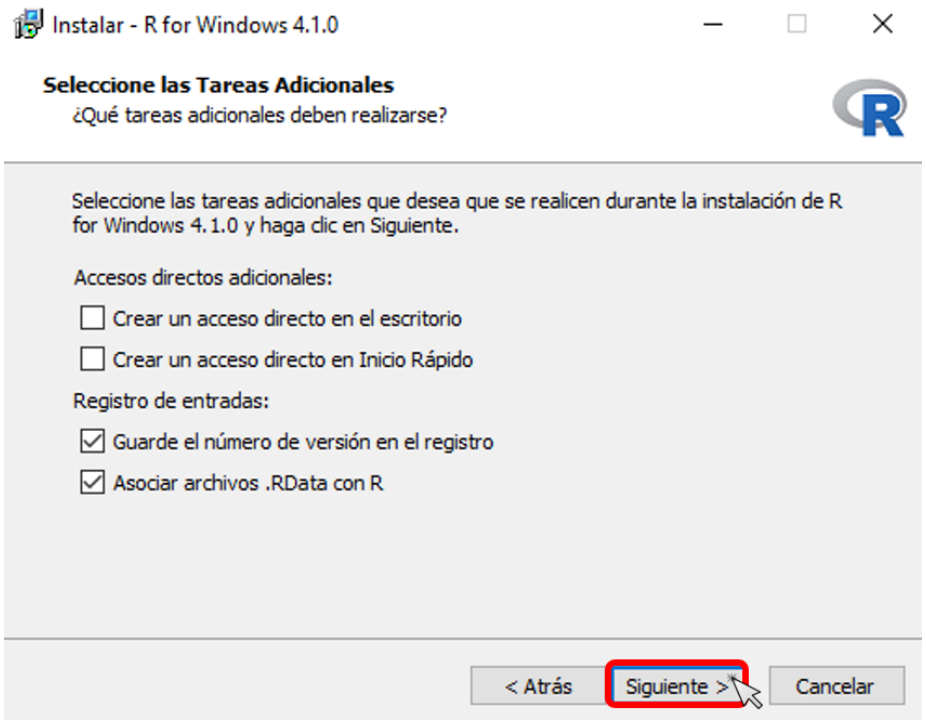
Figura 2.10. Paso 6 instalación de R en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

En la ventana donde se seleccionan las tareas adicionales (Ver Figura 2.11), recomendamos no elegir la opción "de"crear un acceso directo en el escritorio", dado que más adelante instalaremos una interfaz gráfica que nos permitirá manipular R de manera más sencilla y no será necesario el acceso directo.

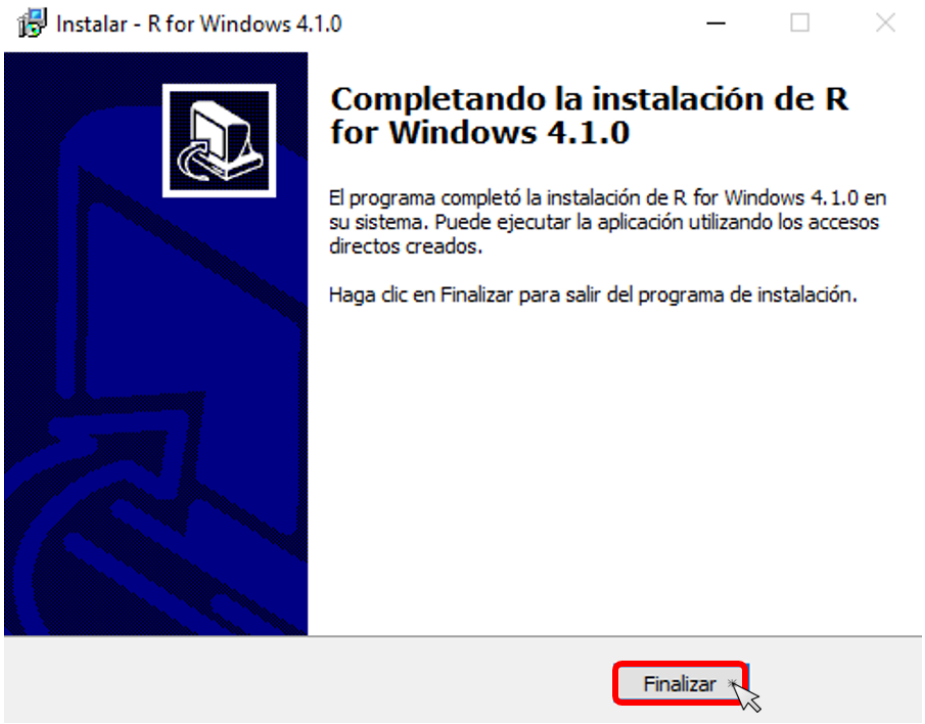
Figura 2.11. Paso 7 instalación de R en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

Una vez termines todos los pasos descritos anteriormente, da clic en la opción **Finalizar** (Ver Figura 2.12) y ya estás listo para continuar con la instalación de la interfaz gráfica RStudio (pasa a la sección 2.2.1).

Figura 2.12. Paso 8 instalación de R en Windows



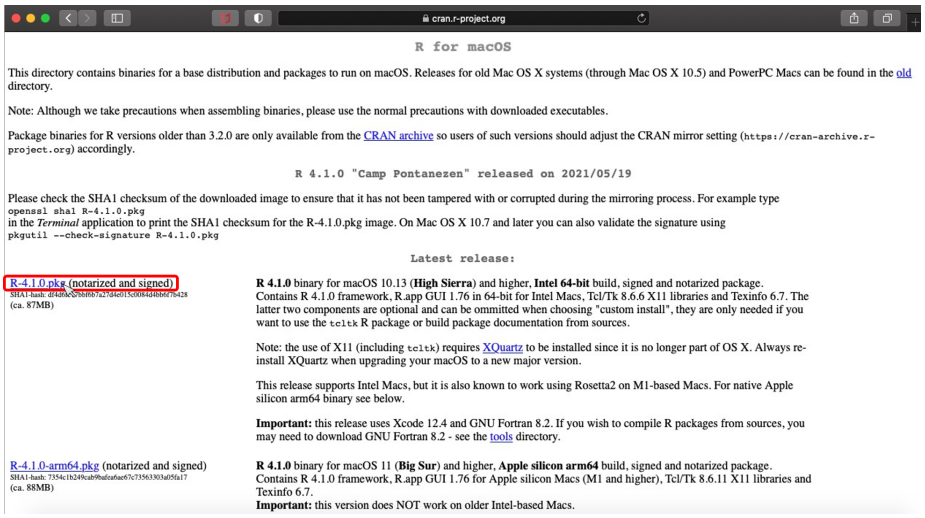
Fuente: Esta captura de pantalla fue tomada del instalador de R.

2.1.2 Instalación de R para macOS

(Si tu sistema operativo es Windows, puedes omitir esta sección. Continúa con la sección 2.2.1.)

Para instalar la versión de macOS, haz clic en **Download R for macOS** desde el link de descarga de R (ver Figura 2.1). Luego observarás la ventana que se presenta en la Figura 2.13).

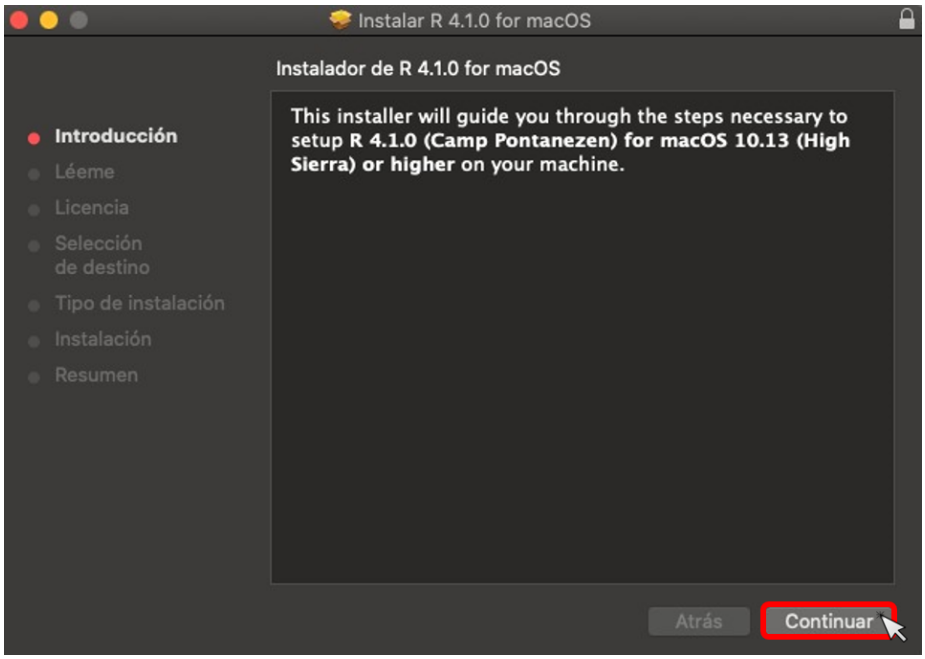
Figura 2.13. Paso 1 Descarga de R en macOS



Fuente: Esta captura de pantalla fue tomada de CRAN (https://cran.r-project.org/bin/macOSx/).

Una vez eliges la última versión disponible (Ver Figura 2.13) y ejecutas el archivo descargado, iniciará la instalación (ver 2.14). En esta primera ventana haz clic en **Continuar**.

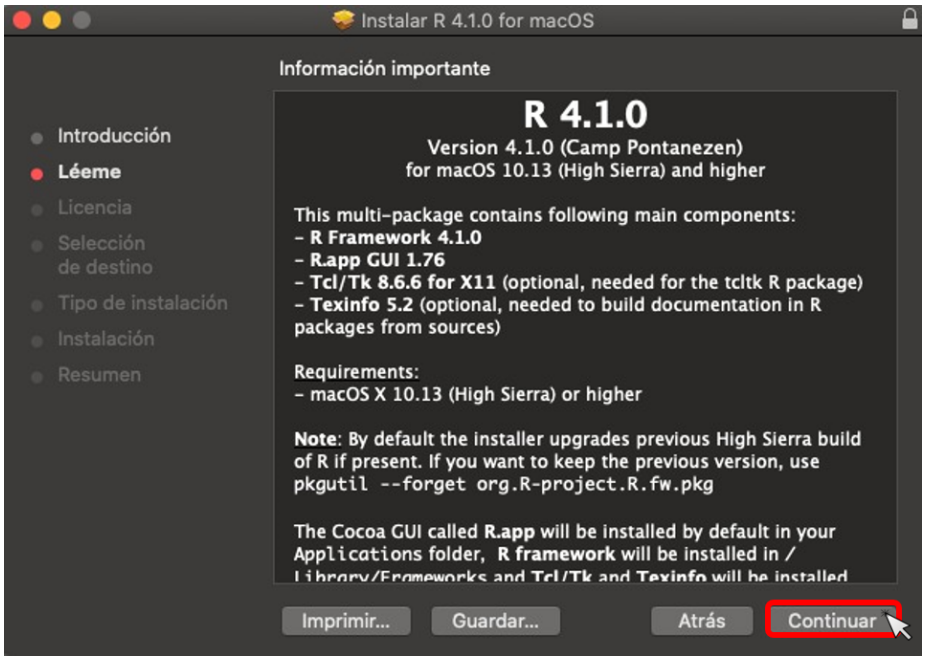
Figura 2.14. Paso 1 instalación de R en macOS



Fuente: Esta captura de pantalla fue tomada del instalador de R.

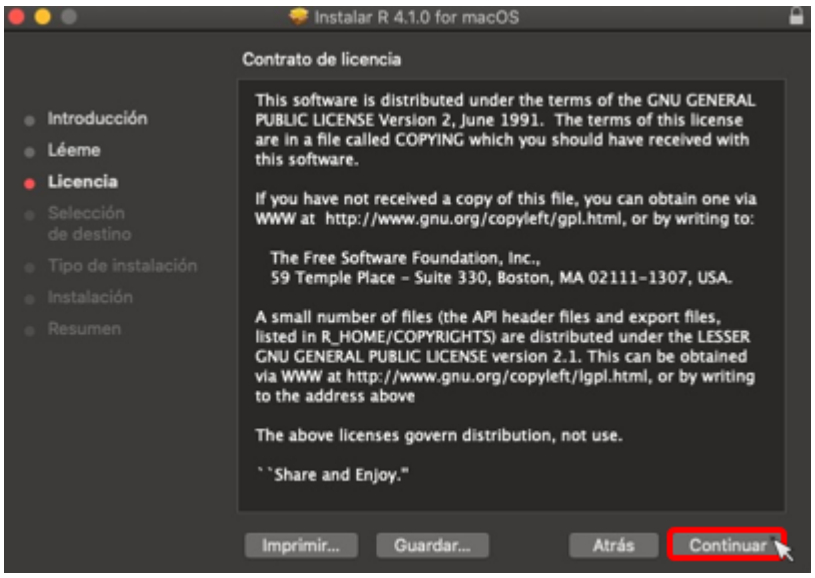
Luego, aparecerá primero información importante sobre el programa (Ver Figura 2.15) y la licencia de uso de R (Ver Figura 2.16). Esta última ventana, en términos generales, nos indica que el programa es de uso libre y gratuito. En ambas ventanas debemos seleccionar **Continuar** (después de leer la información de la licencia).

Figura 2.15. Paso 2 instalación de R en macOS



Fuente: Esta captura de pantalla fue tomada del instalador de R.

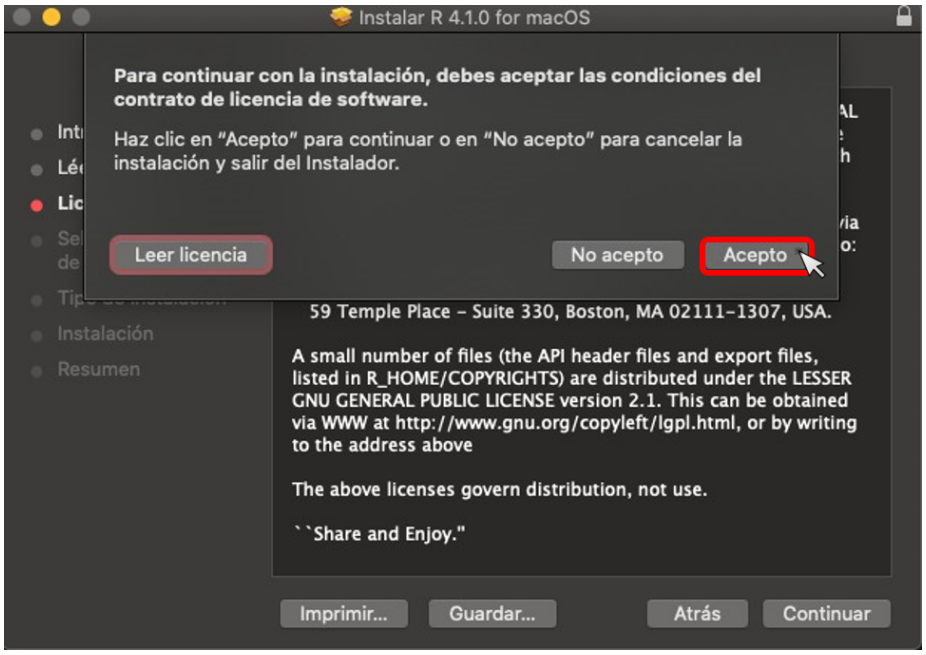
Figura 2.16. Paso 3 instalación de R en macOS



Fuente: Esta captura de pantalla fue tomada del instalador de R

Debemos aceptar las condiciones de la licencia para continuar (Ver Figura 2.17).

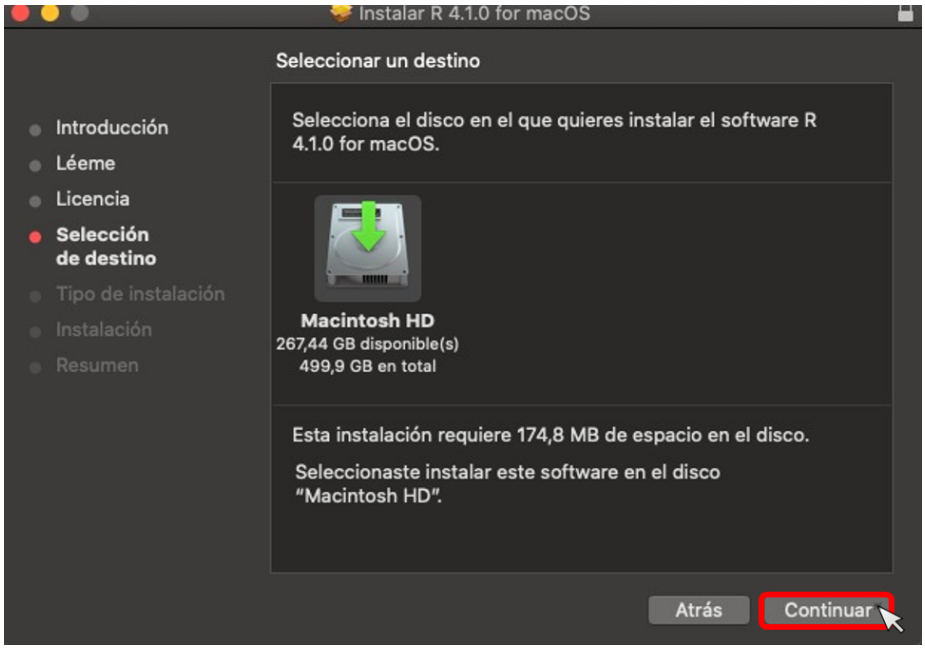
Figura 2.17. Paso 4 instalación de R en macOS



Fuente: Esta captura de pantalla fue tomada del instalador de R.

La siguiente ventana nos muestra la dirección o carpeta de destino en donde se instalará R, recomendamos que uses la predeterminada. Haz clic en **Continuar** (Ver Figura 2.18).

Figura 2.18. Paso 5 instalación de R en macOS



Fuente: Esta captura de pantalla fue tomada del instalador de R.

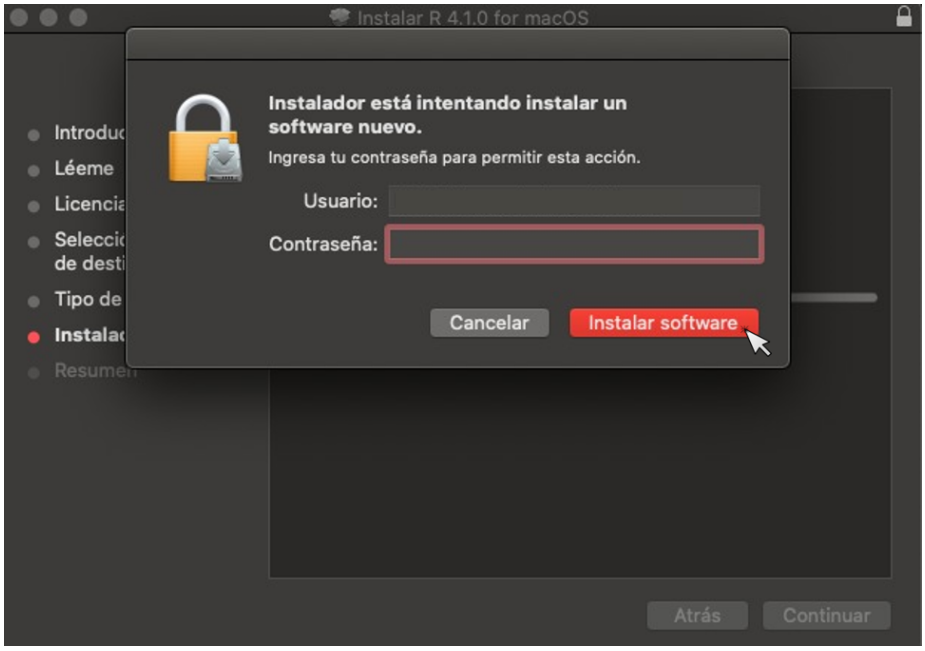
La siguiente ventana corresponde al tipo de instalación, da clic en **Instalar** (Ver Figura 2.19) y a continuación aparecerá una ventana para validar el usuario y permitir que inicie la instalación (Ver Figura 2.20).

Figura 2.19. Paso 6 instalación de R en macOS

Fuente: Esta captura de pantalla fue tomada del instalador de R.

Por último, da clic en **Finalizar** y ya estamos listos para continuar con la instalación de la interfaz gráfica RStudio (pasa a la sección 2.2.2).

Figura 2.20. Paso 7 instalación de R en macOS

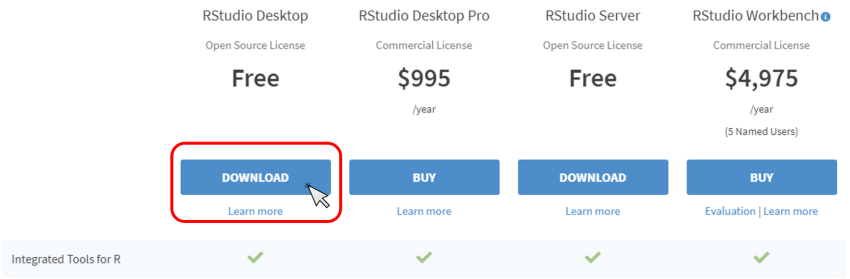


Fuente: Esta captura de pantalla fue tomada del instalador de R.

2.2 Instalación de RStudio

Para descargar RStudio, primero debemos visitar el siguiente enlace: https://rstudio.com/products/rstudio/download. Allí encontraremos la versión gratuita de RStudio Desktop. Verás una página como la presentada en la Figura 2.21.

Figura 2.21. Página de descarga de RStudio



Fuente: Captura de pantalla de https://rstudio.com/products/rstudio/download

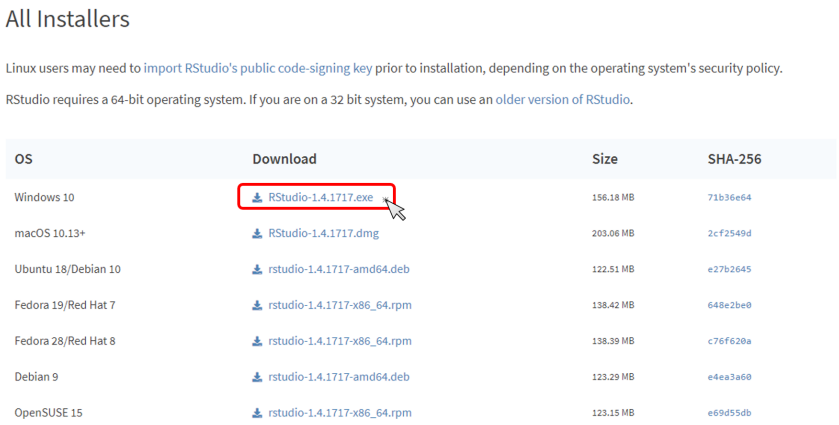
Al hacer clic en **DOWNLOAD** (Ver Figura 2.21), se abrirá la lista de instaladores para los diferentes sistemas operativos. En las siguientes secciones cubriremos la instalación en Windows (sección 2.2.1) y macOS (sección 2.2.2).

2.2.1 Instalación de RStudio para Windows

(Esta sección la puedes omitir si eres usuario de macOS. Continúa en la sección 2.2.2.)

Para iniciar, descargue el instalador correspondiente a Windows 10 como se indica en la Figura 2.22 y ejecute el archivo descargado.

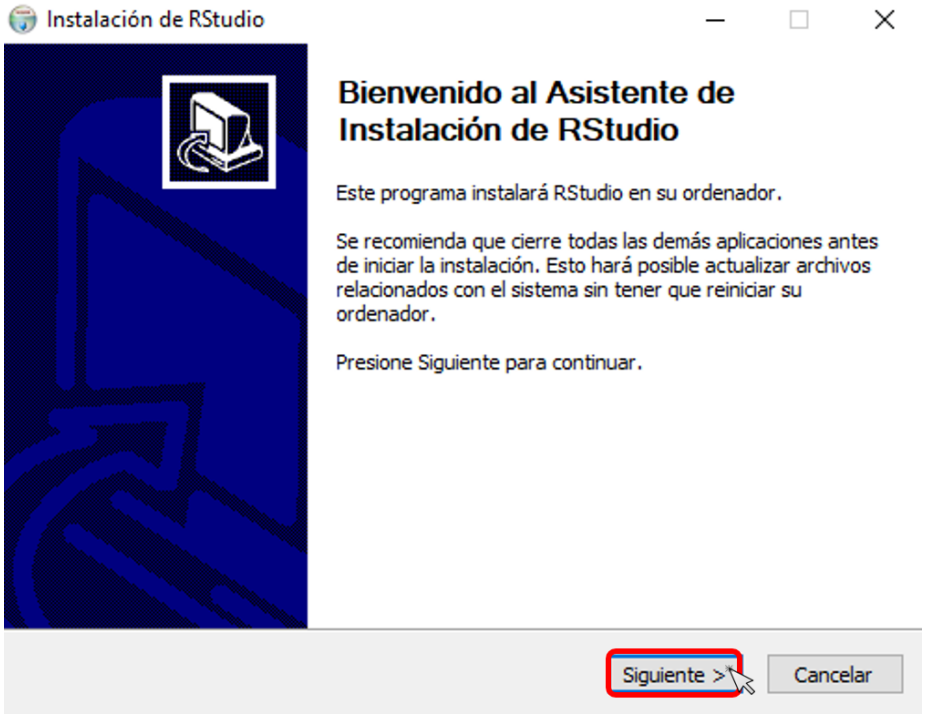
Figura 2.22. Página para descargar Rstudio para Windows



Fuente: Captura de pantalla de https://www.rstudio.com/products/rstudio/download/#download

En la ventana emergente (Ver Figura 2.23), haz clic en **Siguiente** para iniciar la instalación.

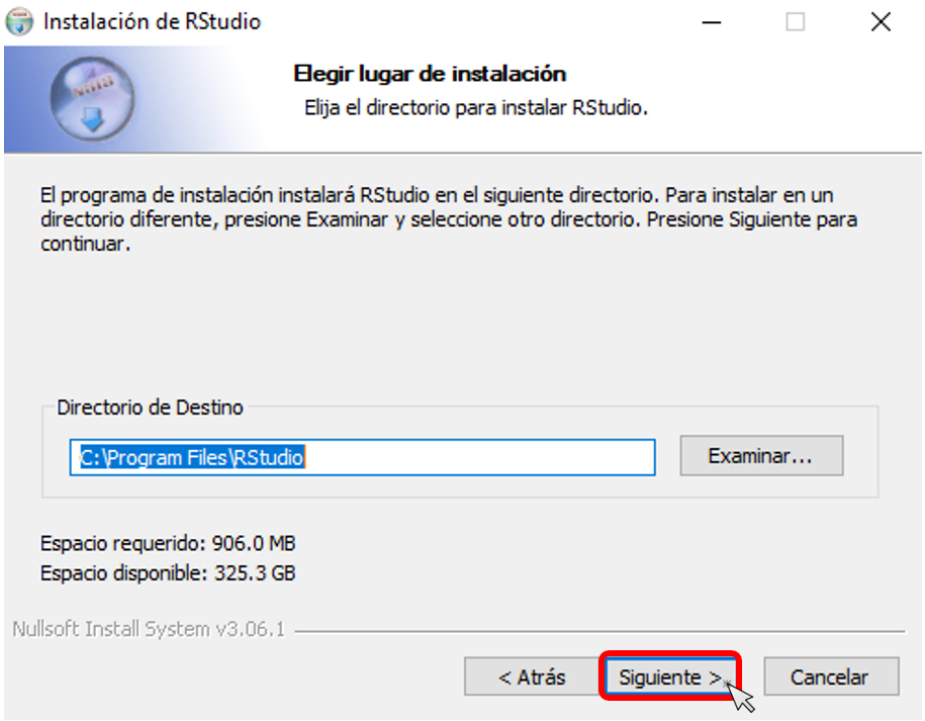
Figura 2.23. Paso 1 instalación de RStudio en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

En la siguiente ventana (ver Figura 2.24) se muestra la dirección o carpeta de destino en donde se instalará RStudio, recomendamos que uses la dirección predeterminada y sólo haz clic en **Siguiente**.

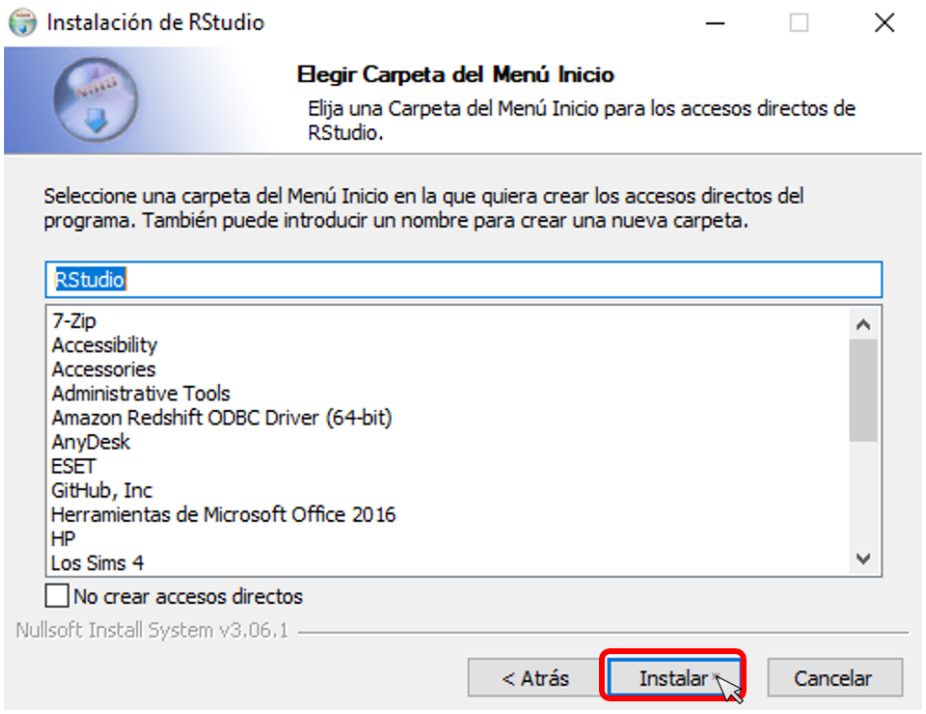
Figura 2.24. Paso 2 instalación de RStudio en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

En la ventana siguiente (ver Figura 2.25), haz clic en el botón **Instalar**, en instantes el programa comenzará la instalación.

Figura 2.25. Paso 3 instalación de RStudio en Windows



Fuente: Esta captura de pantalla fue tomada del instalador de R.

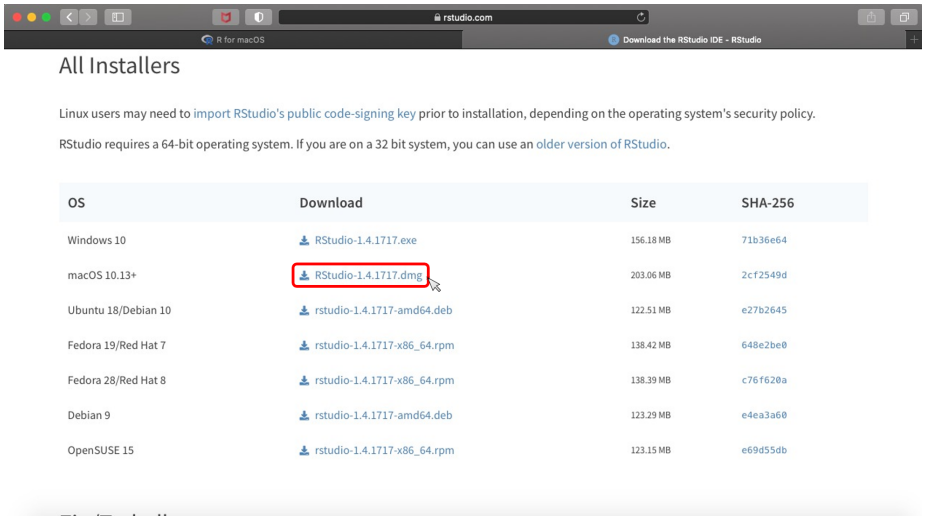
Finalmente, aparecerá una última ventana que indica la opción **Finalizar**. Al dar clic allí habremos terminado y estamos listos para dar nuestros primeros pasos en R. En el siguiente capítulo abordaremos las partes de R y RStudio.

2.2.2 Instalación de RStudio para macOS

(Si tu sistema operativo es Windows puedes omitir esta sección continúa con el Capítulo 3).

Para iniciar, descargue el instalador correspondiente a macOS como se indica en la Figura 2.26 y ejecute el archivo.

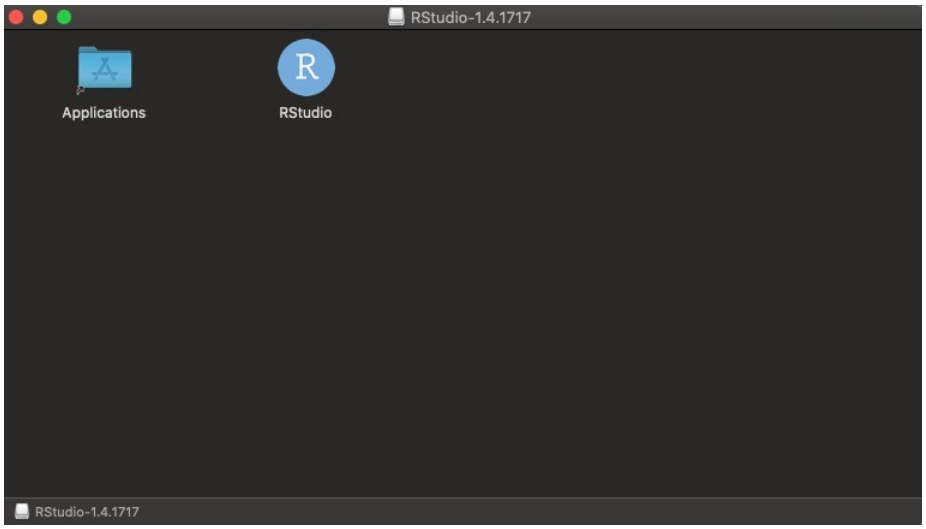
Figura 2.26. Página para descargar Rstudio para macOS



Fuente: Captura de pantalla de https://www.rstudio.com/products/rstudio/download/#download

Al ejecutar el archivo descargado, se generará una carpeta con la aplicación de RStudio ya instalada. El último paso necesario para terminar las instalaciones es arrastrar el ícono de RStudio a la carpeta de aplicaciones y soltarlo en dicha carpeta (Ver Figura 2.27).

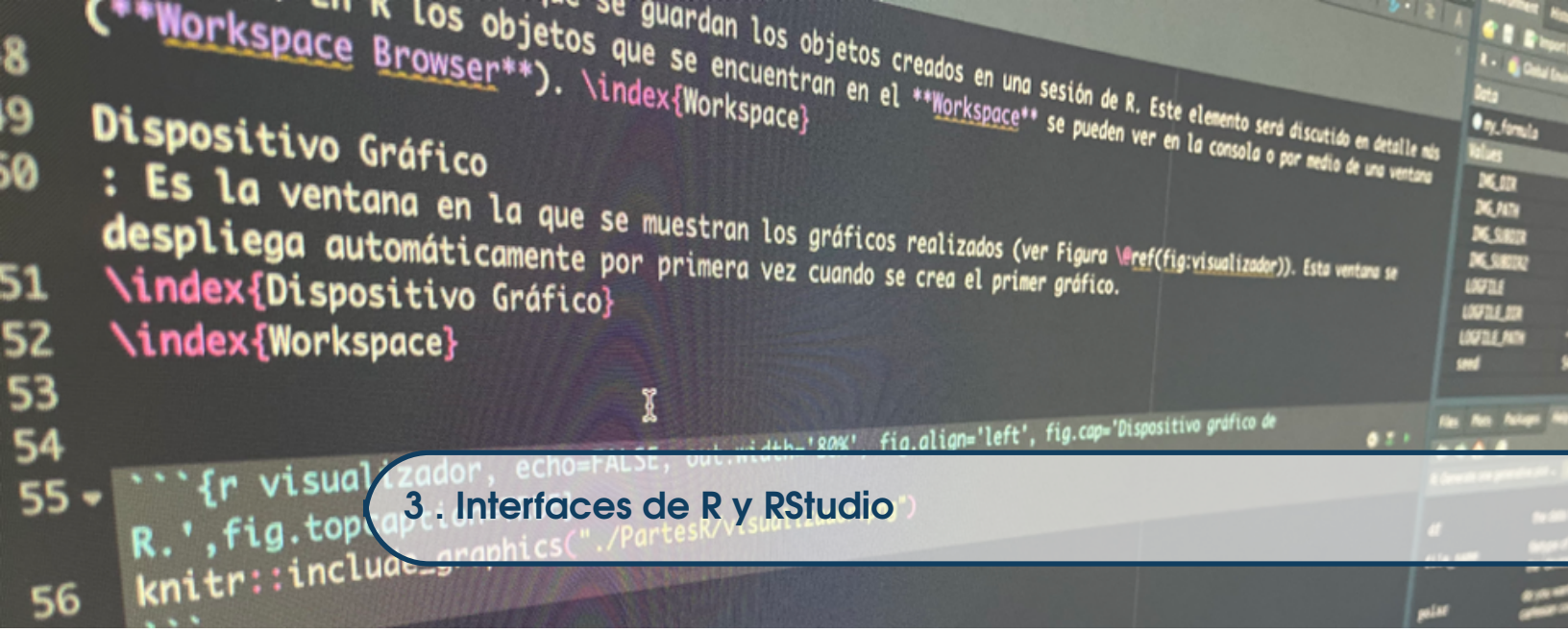
Figura 2.27. Instalación de RStudio en macOS



Fuente: Esta captura de pantalla fue tomada del instalador de R.

2.3 Comentarios finales

Con esto habremos terminado las instalaciones y estamos listos para dar nuestros primeros pasos en R. En el siguiente Capítulo discutiremos las partes de R y RStudio. En el Capítulo 4 construiremos nuestras primeras líneas de código, la creación de objetos y el uso de funciones. En el Capítulo 5 estudiaremos los diferentes tipos de objetos y en el Capitulo 7 veremos cómo instalar y cargar paquetes en R.



3. Interfaces de R y RStudio

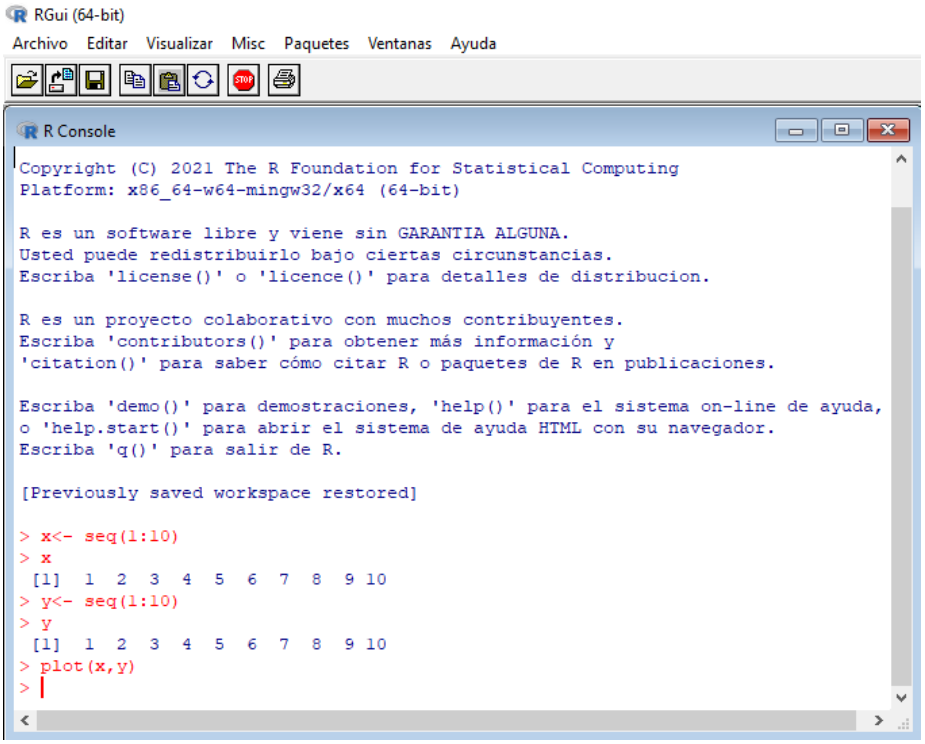
En el capítulo anterior explicamos cómo instalar R y la interfaz RStudio. En el transcurso de este libro hemos afirmado que RStudio es una suerte de chasis que se le pone al motor potente que es R. Emplear RStudio mejora nuestra experiencia de uso del programa para tenerlo todo en un mismo *Core layout*. En este capítulo discutiremos las partes que componen la interfaz nativa de R y el cómo RStudio facilita nuestro trabajo en R.

3.1 Conociendo R

R es un ambiente de programación formado por un conjunto de elementos que nos permiten cargar, transformar, analizar y visualizar datos. En R contamos con los siguientes elementos que corresponden a ventanas flotantes en nuestra pantalla:

Consola. Es el espacio en el cual se ejecutan las órdenes que damos a R a través de la línea de código. También es el espacio en donde se muestran los resultados inmediatos de los cálculos (ver Figura 3.1). En la consola se ejecutan comandos que se pueden escribir directamente en ella o se pueden enviar desde el **script** a la consola. Esta es la única ventana que es visible por defecto cuando abrimos R. La consola presenta el signo > en la linea donde se puede escribir los comandos y un cursor parpadiante (signo I) que muestra que R está listo para recibir comandos. Los comandos son digitados a esa línea y enviados al motor de R al presionar la tecla `enter` o `return`.

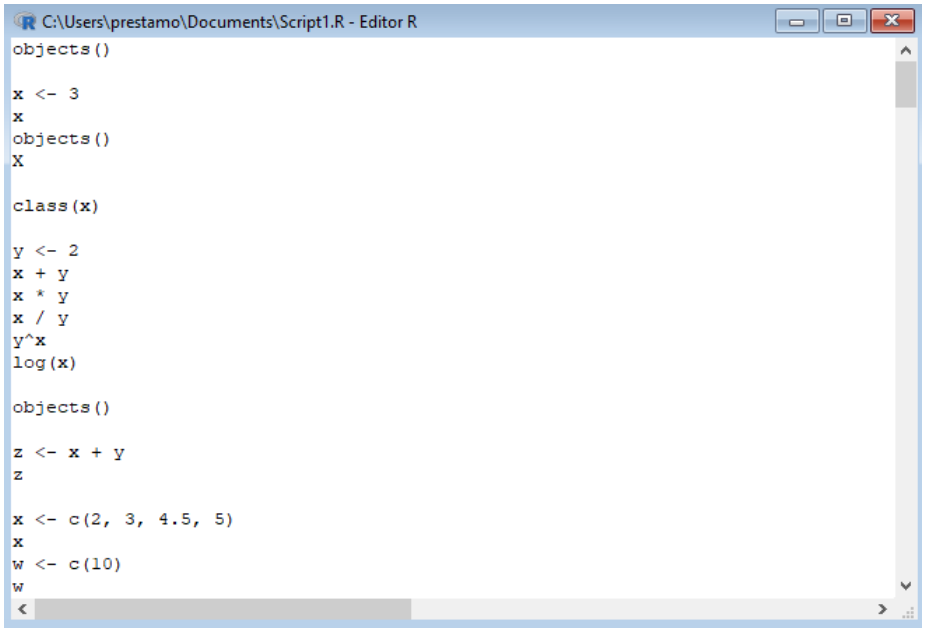
Figura 3.1. Consola de R



Fuente: Esta captura de pantalla fue tomada de R.

Script. Es un archivo en el cual podemos guardar las líneas de código (lista de comandos) que elaboramos. Podemos pensar en el *script* como el archivo donde almacenamos la lista de instrucciones que llevamos a cabo para desarrollar un proyecto. (ver Figura 3.2).

Figura 3.2. Ventana de un script en R

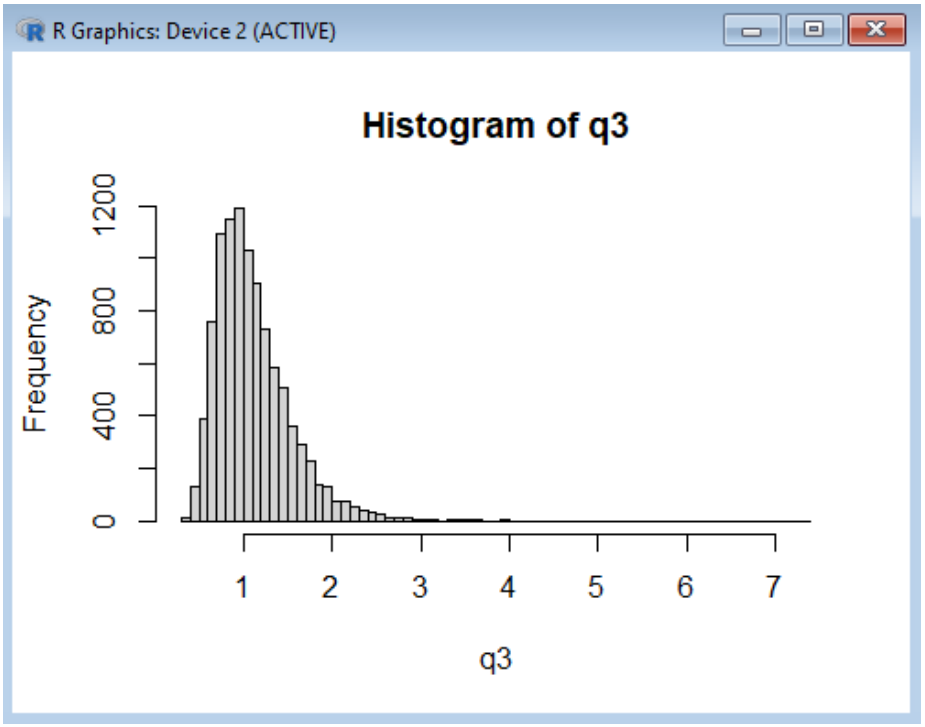


Fuente: Esta captura de pantalla fue tomada de R.

Workspace. Es un espacio en el cual se guardan los objetos creados en una sesión de R. Este elemento será discutido en detalle más adelante. En R, los objetos que se encuentran en el *workspace* se pueden ver en la consola por medio de un comando o por medio de una ventana (*Workspace Browser* que se encuentra en el menú *Workspace*).

Dispositivo Gráfico. Es la ventana en la cual se muestran los gráficos realizados (ver Figura 3.3). Esta ventana se despliega automáticamente por primera vez cuando se crea el primer gráfico.

Figura 3.3. Dispositivo gráfico de R.

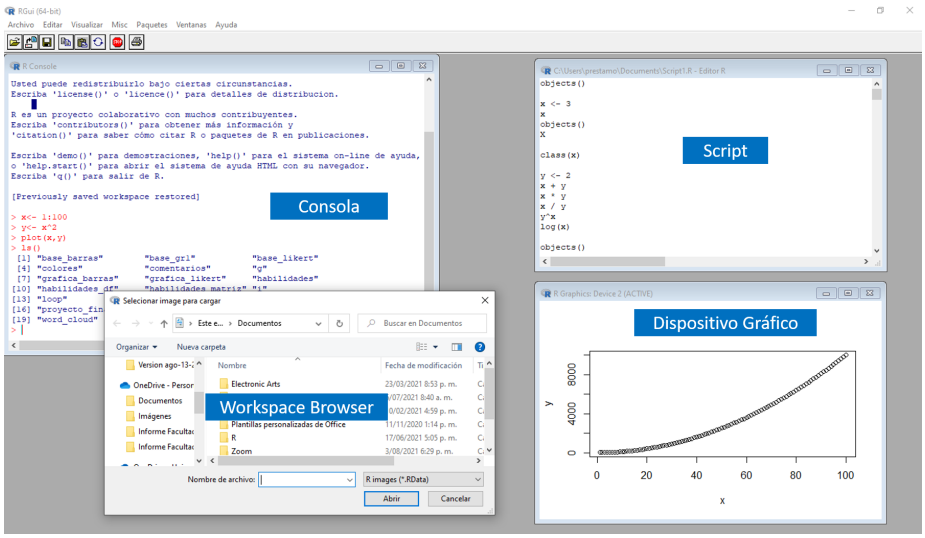


Fuente: Esta captura de pantalla fue tomada de R.

Working directory. Es una carpeta donde se guardan el *workspace*, datos originales y otros elementos. Es la carpeta en donde se almacenará o desde la cual se leerán archivos por defecto. El *working directory* se puede cambiar como se discutirá más adelante (Ver Capítulo 6).

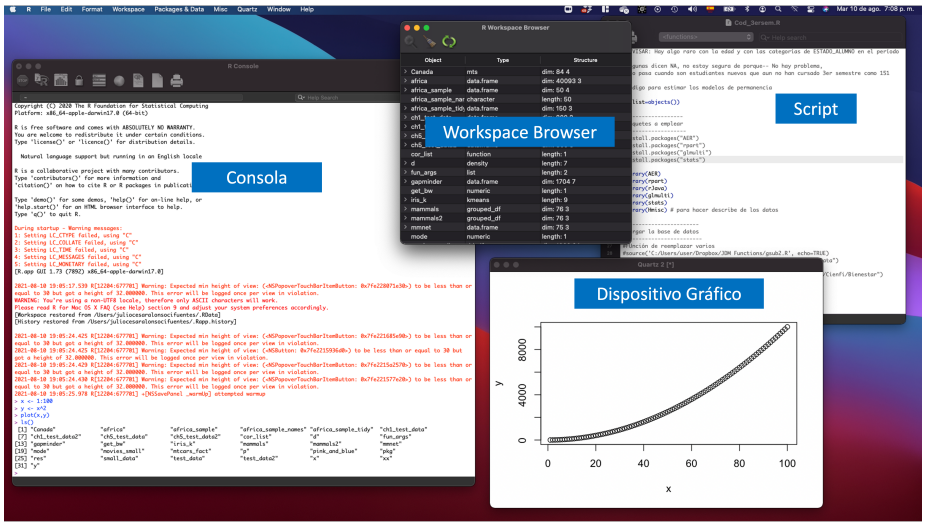
Como se puede observar en las Figuras 3.1, 3.2 y 3.3 las ventanas que contienen estos elementos no son muy agradables visualmente y pueden ser algo desordenadas. En las Figuras 3.4 y 3.5 se presenta una visualización de cómo se ve la interfaz de R en los sistemas operativos Windows y macOS, respectivamente. En la siguiente sección veremos como RStudio puede ayudar e mejorar la apariencia de R y facilitar nuestro trabajo.

Figura 3.4. Visualización completa de la interfaz de R en Windows



Fuente: Esta captura de pantalla fue tomada de R.

Figura 3.5. Visualización completa de la interfaz de R en macOS



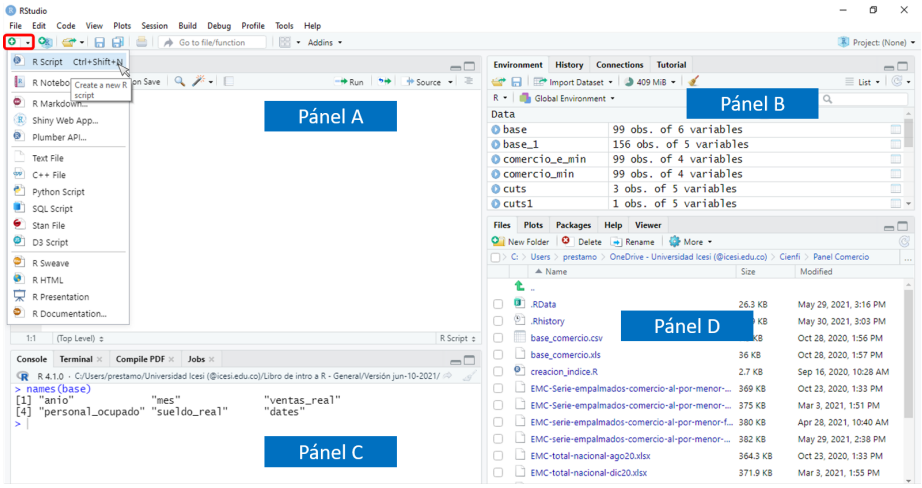
Fuente: Esta captura de pantalla fue tomada de R.

3.2 La interfaz de RStudio

RStudio es una interfaz gráfica que permite visualizar los elementos básicos de R descritos en la sección anterior. Verás que los diferentes cuadrantes representan o integran cada elemento de R.

La Figura 3.6 muestra lo primero que veremos al ejecutar RStudio. El **Pánel A** se encuentra en la parte superior derecha, es el espacio para escribir un nuevo *script* o cargar uno guardado previamente en nuestra computadora. En este cuadrante podemos seleccionar una o varias líneas de código para ejecutarlas presionando las teclas `Ctrl + Return` en Windows y `Command + Return` en macOS. Es decir, esta combinación de teclas envían directamente el código que está en el *scripte* a la consola. Esto te ahorrará el tiempo de copiar el código del *script* y pegarlo en la consola para su ejecución.

Figura 3.6. Cuadrantes de RStudio



Fuente: Esta captura de pantalla fue tomada de RStudio.

Para empezar un nuevo *script* das clic al logo señalado en la Figura 3.6 con un rectángulo rojo en la parte superior izquierda de la ventana de RStudio. Otra manera de crear un script es emplear los menús. En este caso haremos clic en `File | New File | R Script`. También pueden emplearse atajos del teclado para crear un *script*. En Windows un nuevo *script* se crea con `Shift + Ctrl + n`. En macOS puedes usar `Shift + Command + n`.

Uno de los beneficios de usar RStudio para escribir nuestro código, es que los comandos de R serán automáticamente formateados con diferentes colores, lo cuál permitirá identificar fácilmente cada una de las partes de tu código. Además, RStudio te ayudará a llenar automáticamente algunas funciones y te recordará los argumentos de las funciones cuyos paquetes hayan sido cargados.

En el **Pánel B** está la consola, el signo mayor (>) significa que podemos escribir un comando, ejecutarlo con la tecla "Enter" y ver inmediatamente el resultado.

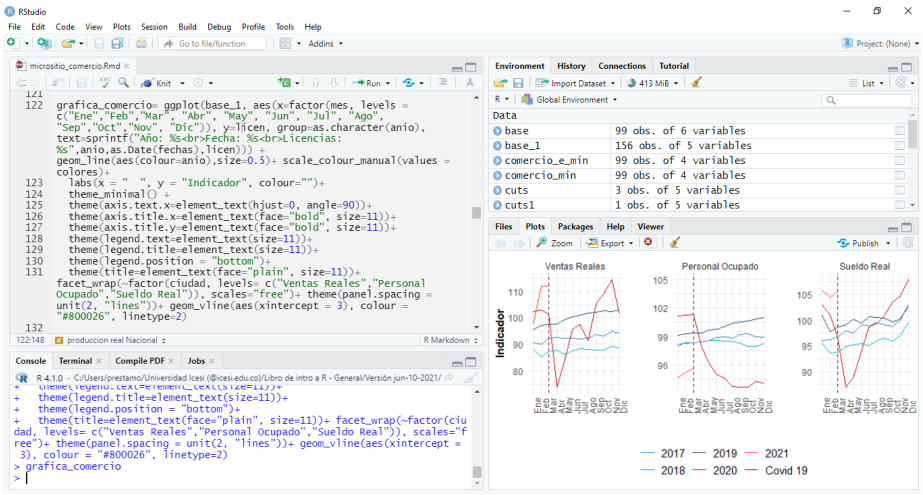
En el **Pánel C** se encuentra el *workspace* (pestaña `Enviroment`). Ahí se muestran los objetos que están en el *workspace* actual. En ese mismo panel encontramos la pestaña de *History* que contiene la historia de todas las líneas código que han sido ejecutadas previamente en la consola.

En el **Pánel D** te toparás con la pestañas `Files` que muestra los archivos que están en el *Working directory* actual. En el mismo pánel está la

pestaña `Help` en la que puedese buscar la documentación de las funciones y su sintaxis. La pestaña `Packages` muestra los paquetes instalados en tu R, y con un chulo aquellos que ya han sido cargados. Finalmente, están las pestañas de `Viewer` y `Plots` que contienen los dispositivos gráficos.

En la Figura 3.7 se muestra una sesión de RStudio más detallada para que puedas visualizar los diferentes elementos.

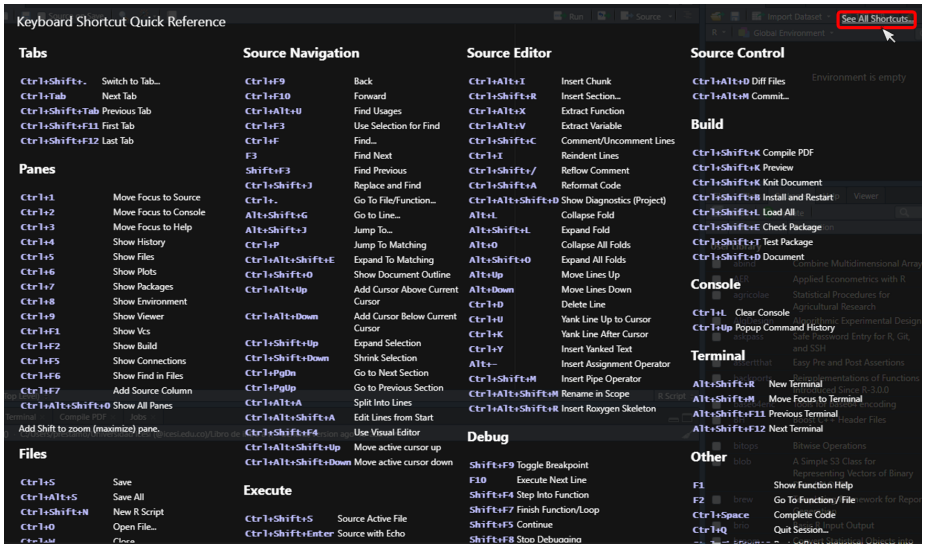
Figura 3.7. Sesión de RStudio



Fuente: Esta captura de pantalla fue tomada de RStudio.

Antes de dar nuestros primeros pasos con código, veamos algunos trucos y personalización de RStudio.Para comenzar, los páneles de la ventana de RStudio mantienen accesible la información importante sobre nuestro proyecto. Saber cómo movernos entre ellos sin tocar el ratón, puede ahorrarnos tiempo y mejorar nuestro flujo de trabajo. Esta herramienta se conoce como *shorcuts* y se encuentra descrita en la barra superior de RStudio en la pestaña `Tools`. Allí daras clic en `Keyboard shortcuts Help` y verás una lista como se muestra en la Figura 3.8. Al hacer clic donde está señalado, serás dirigido a un *link* en donde encontrarás todos los trucos disponibles para cada sistema operativo. Así, si quisiéramos pasar de cualquier pánel de RStudio a la consola, bastaría con presionar la tecla `Ctrl + 2`, tanto en Mac cómo en Windows y Linux. De manera similar para movernos al panel del *script* podemos emplear la tecla `Ctrl + 1`, en todas las plataformas.

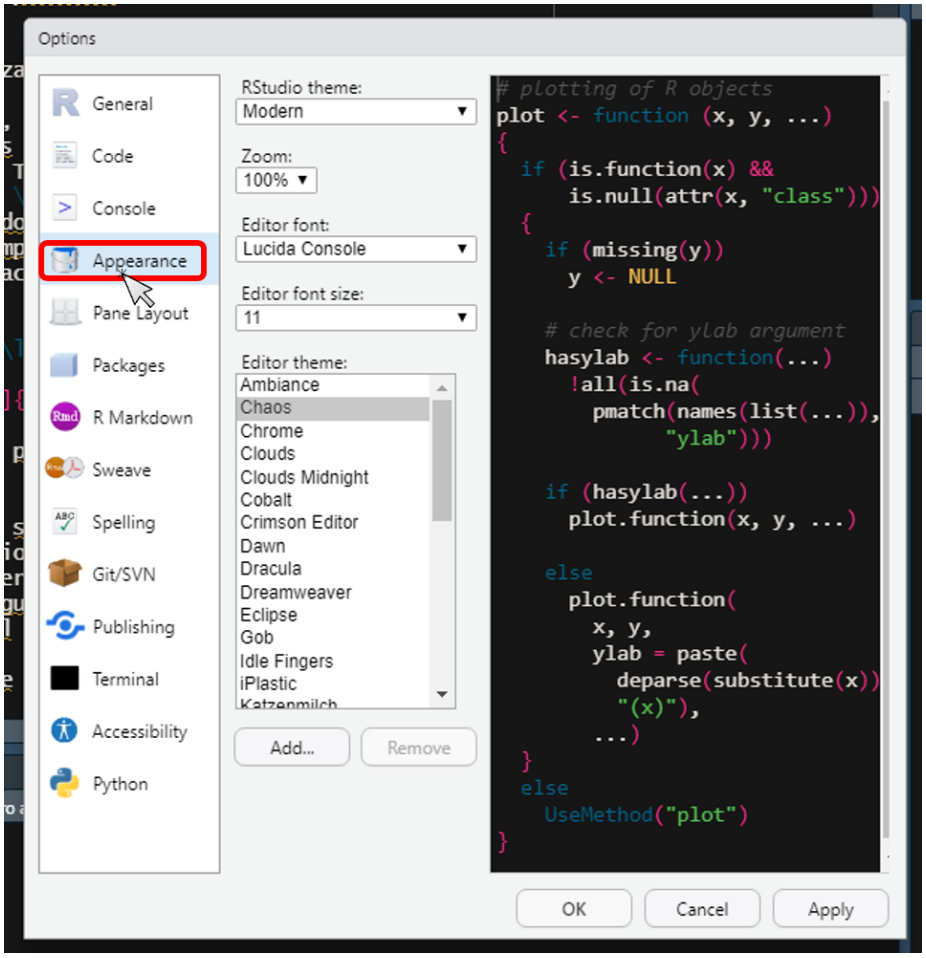
Figura 3.8. Guía de Shortcuts en RStudio



Fuente: Esta captura de pantalla fue tomada de RStudio.

Otra herramienta que puede ayudar a la salud de nuestros ojos y hacer más agradable nuestro trabajo, es personalizar la apariencia de RStudio. Si pasas mucho tiempo usando R es recomendable que cuides de tus ojos cambiando los colores de RStudio. Esta opción se encuentra también en el menú *Tools* `Tools | Global Options`, al elegirla aparecerá una nueva ventana, como se muestra en la Figura 3.9. Vamos a seleccionar el menú de *Appereance* y en la pestaña `Editor Theme` podemos elegir el tema que queramos. Recomendamos las opciones de fondo oscuro y letras de colores fuertes así serán distinguibles fácilemnte para una mejor experiencia de uso y cuidado de nuestros ojos.

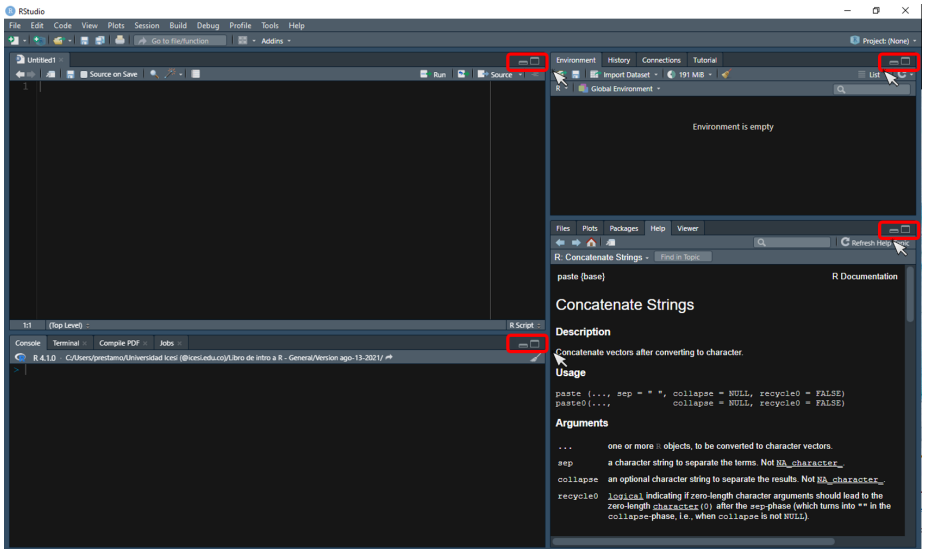
Figura 3.9. Menú para cambiar la apariencia de RStudio



Fuente: Esta captura de pantalla fue tomada de RStudio.

También es posible maximizar o minimizar los páneles en RStudio dando clic a los iconos en la barra superior de cada pánel como se muestra en la Figura 3.10, o bien puedes personalizar los tamaños de cada pánel arrastrando el cursor en los bordes de cada uno.

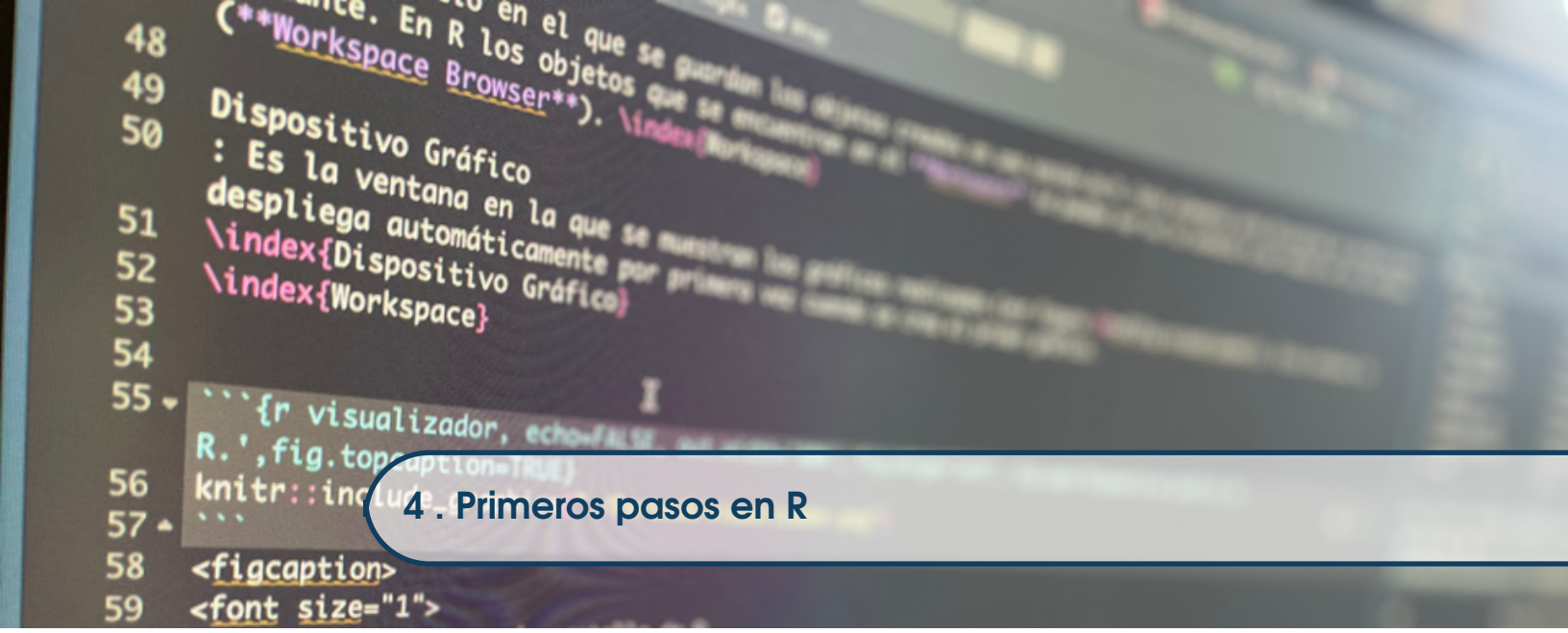
Figura 3.10. Ícono para minimizar o maximizar los páneles en RStudio



Fuente: Esta captura de pantalla fue tomada de RStudio.

3.3 Comentarios finales

Ya instalamos R y RStudio (Capítulo 2) y en este capítulo estudiamos las diferentes partes de R y RStudio. Ya podemos usar R para hacer cálculos. En el siguiente capítulo (Capítulo 4) haremos nuestras primeras líneas de comando, la creación de objetos y el uso de funciones. En el Capítulo 5 abordaremos los diferentes tipos de objetos. En el Capitulo 7 discutiremos cómo instalar y cargar paquetes en R, lo que permitirá ampliar las funciones disponibles para cargar, transformar y analizar datos. En el Capítulo 8 plantearemos cómo cargar datos de diferentes fuentes.



4 . Primeros pasos en R

Ya tienes instalado Rstudio y R en to computador (R Core Team, 2018), te preguntarás ¿y ahora qué? Un *script* vacío y una consola con un cursor parpadeante pueden ser intimidantes a primera vista. ¡No te preocupes!, ya estamos listos para empezar con nuestros primeros comandos.

4.1 Primeras línea de código

Antes de iniciar, es importante recordar que si aún no tienes un *script* abierto, puedes crear uno empleando la barra de menú `File | New File | R Script`. O con los atajos del teclado: `Shift + Ctrl + n` (en Windows) o `Shift + Command + n` (en macOS). Es necesario crear un *script* nuevo, lo usaremos para guardar los comandos que estudiaremos a continuación.

Generalmente, lo más conveniente es escribir y guardar el *script* y, a partir de éste, ejecutar los comandos necesarios. Si escribiesemos los comandos directamente en la consola, los perderíamos y no podríamos recuperarlos para repetir el ejercicio o reciclar código en un proyecto futuro.

Antes de empezar a ver algunos comandos, es importante mencionar que los comandos se dividen en dos grandes tipos:

- Asignaciones
- Expresiones

Las **asignaciones** piden a R que guarde el resultado de ejecutar una orden en un elemento al que llamaremos objeto. Veamos un ejemplo

muy sencillo. En tu *script* escribe la siguiente línea y ejecútala en la consola[1]:

```
#Asignación de un valor a un objeto
x <- 1
```

En este caso asignamos al objeto `x` el valor de uno. Noten que inmediatamente en su *workspace* se registra el objeto `x` que se ha creado (ver panel superior izquierdo en la pestaña `Enviroment`).

El signo <- representa en R el comando asignar lo que está a la derecha del signo al objeto a la izquierda del signo[2]. Vale la pena mencionar que R es sensible al uso de mayúsculas. Así que, el objeto `x` es diferente al `X`.

También puedes notar que realmente escribimos dos líneas de código, la primera inicia con el signo #. Este signo le informa a R que todo lo que se encuentra a la derecha de este signo no será evaluado. Es decir, el signo # se puede emplear para hacer comentarios del código que se escribe. Comentar el código es una buena práctica, pues nos ayuda en un futuro a recordar lo que está haciendo cada parte del código. Y más importante aún, si otra persona mira el código podrá entenderlo más fácilmente. Recuerda que R es un lenguaje de programación y cada persona se expresa de manera diferente en cada lenguaje.

Recomendación de Estilo

Cuando se asigne un valor o resultado a un objeto emplea el signo <- y no =.

```
# Buena práctica
x <- 1

# Mala práctica
x = 1
```

Antes mencionamos que existen dos tipos de comandos: las asignaciones y las expresiones. Las **expresiones** son una orden que se da a R de ejecutar una o muchas tareas empleando objetos y parámetros (de

[1] Recuerda, si tienes el cursor en una línea (renglón) de tu *script* lo puedes mandar a ser ejecutado empleando `Ctrl + Return` en Windows o `Command + Return` en macOS. Si seleccionas con el cursor (sombreas) una parte del script y empleas la combinación de teclas descritas anterioremente, enviarás la selección completa.

[2] Se puede obtener el mismo resultado empleando el signo =, pero no es una práctica buena en la comunidad de R, pues este signo no implica en término lógicos lo mismo que asignar algo a un objecto.

ser necesarios). Estos comandos son conocidos en la comunidad de R como **funciones**. Y los objetos y parámetros que emplea el comando para hacer una operación se conocen como argumentos. En general, las funciones en R tienen la forma:

función(argumentos)

Por ejemplo, la función **print()** emplea como argumento principal un `objeto` y realiza la tarea de imprimir dicho objeto en la consola. Veamos un ejemplo:

```
print(x)
```

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

Otra función básica[3] es **help()**, que sirve para visualizar la ayuda asociada a otra función y tiene como uno de sus argumentos el nombre de la función sobre el cual se quiere pedir ayuda. Por ejemplo,

```
#Pedir ayuda sobre la función print()
help(print)
```

Tras ejecutar esta línea de código, en el panel inferior izquierdo se nos activa la pestaña *Help* y se muestra la documentación de la función **print()**. Nota que en esa ventana puedes buscar la documentación de cualquier función digitando la palabra clave en el espacio que tiene una lupa. Esta es una de las ventajas de Rstudio frente a la interfaz de R que no permite esta búsqueda de ayuda fácilmente.

4.2 Operaciones básicas

Veamos las operaciones básicas que podemos hacer con objetos. Creemos un segundo objeto `y`:

```
y <- 2
```

4.2.1 Operaciones aritméticas básicas

Con los objetos podemos hacer operaciones aritméticas sencillas, como sumar o restar. En el Cuadro 4.1 se presentan los operadores aritméticos más sencillos que tiene R.

[3] Esta función es una de las pocas excepciones en la cual el argumento no es un objeto sino otra función.

```
# suma
x + y
```

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

```
# resta
x - y
```

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

```
# multiplicación
x * y
```

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

```
#división
x / y
```

```
## [1] 0.5
```

```
# x elevada a la y
y^x
```

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

Tabla 4.1. Operadores aritméticos en R

Operador	Descripción
$+$	Suma
$-$	Sustracción
$*$	Multiplicación
$/$	División
$\wedge$	Exponente
$\%\%$	Módulo (Residuo de la división)
$\%/\%$	Parte entera de la división

Recomendación de Estilo

Deja un espacio antes y después de operadores binarios como `+`, `-`, `*` y `/`. También deja espacios antes de abrir y cerrar paréntesis, a menos que estés evaluando una función.

```
# Buena práctica
w <- z * (x + y) + 2

# Mala práctica
w<-z*(x+y)+2
```

También podemos asignar el valor de una operación a un objeto, por ejemplo

```
z <- x^2 + y^4
z
```

```
## [1] 17
```

En este caso ya tenemos tres objetos en nuestro *workspace* (ver panel superior izquierdo en la pestaña *Enviroment*). Si se deseas monitorear en la consola todos los objetos que tenemos en el *workspace* podemos emplear la función **objects()** sin necesidad de argumentos[4]

```
objects()
```

```
## [1] "regular_plot" "source"       "x"            "y"
## [5] "z"
```

También existen funciones que permiten realizar operaciones relativamente sencillas, como por ejemplo, encontrar la raíz cuadrada (**sqrt()**), el logaritmo natural (**log()**), o el logaritmo base 10 (**log10()**).

```
# raíz cuadrada
sqrt(z)
```

```
## [1] 4.123106
```

[4]Otra función que hace exactamente la misma tarea es **ls()**. ¡Inténtalo!

```
# logaritmo natural
log(10)
```

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

```
#logaritmo base 10
log10(10)
```

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

Bueno, ya te puedes hacer una idea de cómo funciona R para hacer estas operaciones. Recuerda que este lenguaje de programación tiene muy buena documentación y puedes encontrar rápidamente la función que necesitas para hacer una operación aritmética. Por ejemplo, en Google puedes buscar la operación o tarea que desees seguido del término "en R"[5] . ¡Inténtalo!

4.2.2 Operaciones de relación y lógicos

R también permite comparar dos objetos y determinar si son iguales, diferentes, etc. En el Cuadro 4.2 se presentan los operadores disponibles en R para hacer comparaciones entre objetos (que sean comparables).

Tabla 4.2. Operadores de relación en R

Operador	Descripción
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	No es igual a

Por ejemplo, si queremos saber si los objetos `x` y `y` son iguales empleamos la siguiente linea de código:

```
# ¿es x igual a y?
x == y
```

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

[5]Encontrarás muchos mas resultados si tu búsqueda es en inglés.

Nota que el resultado será una respuesta de verdadero (*TRUE*) o falso (*FALSE*). En este caso al nos ser iguales la respuesta es *FALSE*. Naturalmente, el resultado de estas operaciones de comparación, también se pueden guardar en objetos.

```
# asignar la respuesta de ¿es x igual a y
# al objeto w
 w <- x == y
w
```

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

También podemos hacer operaciones lógicas en R empleando los operadores que se presentan en el Cuadro 4.3.

Tabla 4.3. Operadores lógicos en R

Operador	Descripción
!	NO lógico
&	Y lógico
|	ó lógico

Por ejemplo, supongamos que queremos chequear si el objeto `y` es más grande que `y`, y al mismo tiempo si la raíz cuadrada del objeto `z` es mayor que 3. Esto lo podemos hacer con la siguiente línea de código:

```
y > x & sqrt(z) > 3
```

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

Recomendación de Estilo

Deja un espacio antes y después de operadores de comparación y lógicos.

```
# Buena práctica
 z != y & log(z) > 3

# Mala práctica
 z!=y&log(z) > 3
```

4.3 Diferentes formas de crear objetos

Hasta ahora hemos discutido cómo crear objetos que contienen un número o que son resultado de evaluar una función. Los objetos con solo un número no son muy interesantes, pero antes de pasar a crear objetos más complejos (ver Capítulo 5) o con bases de datos (ver Capítulo 8), veamos cómo crear un objeto con varios números.

Supongamos que queremos guardar en un objeto las compras en millones de pesos durante el primer año de operación de la compañía de 4 clientes. Eso puede hacerse directamente de la siguiente manera:

```
# asignamos al objeto ventas_1 los números 2 , 3 , 4.5 y 5
# (en ese orden)
ventas_1 <- c(2, 3, 4.5, 5)
ventas_1
```

```
## [1] 2.0 3.0 4.5 5.0
```

Nota que en este caso hemos creado un objeto que tiene un nombre que incluye números, letras (alfanumérico) y un carácter especial (_). De hecho, en los nombres de los objetos se pueden emplear números, letras y puntos y guiones bajos (_).

Para crear nuestro objeto empleamos la función **c()**, ella toma su nombre de la tarea que hace: combinar los elementos que se coloquen como argumentos. Además, que cuando las funciones emplean más de un argumento, estos se separan con comas.

Recomendación de Estilo

Después de una coma siempre deja un espacio.

```
# Buena práctica
x1 <- c(2, 3, 5)

# Mala práctica
x1 <- c(2,3,5)
```

Podemos crear objetos empleando secuencias de números. El operador : permite crear una secuencia que va desde el número a la izquierda del operador hasta el número a la derecha, con incrementos de a uno.

```
# secuencia de 1 a 10 con incrementos de a uno.
1:10
```

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

Combinando las dos últimas partes, creemos un objeto que contenga todos los números del 1 al 10, sin tener que digitarlos.

```
# se asigna al objeto x1 la secuencia de
# 1 a 10 con incrementos de a uno.
x1 <- c(1:10)
x1
```

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

También podemos crear una secuencia descendente, incoando la secuencia con el número mayor y terminándola con el menor, por ejemplo

```
# se asigna al objeto x2 la secuencia de
# 10 a 1 con reducciones de a una unidad.
x2 <- c(10:1)
x2
```

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

Otra forma de hacer los mismo, pero con más versatilidad es emplear la función **seq()** para crear secuencias. Esta función típicamente incluye los siguientes argumentos:

seq(from, to, by)

Donde:

- **from**: es el valor de inicio de la secuencia.
- **to**: es el valor final de la secuencia.
- **by**: es el número en el que se incrementará la secuencia.

Por ejemplo, si queremos una secuencia de números que vaya desde -2 hasta 10 en incrementos de dos unidades, podemos emplear el siguiente código:

```
seq(from =-2, to = 10, by=2)
```

```
## [1] -2  0  2  4  6  8 10
```

```
seq(-2, 10, 2)
```

```
## [1] -2  0  2  4  6  8 10
```

Observa que podemos obtener el mismo resultado sin tener que escribir el nombre de los argumentos. Sino se pone el nombre del argumento, R tomará los argumentos en el orden predeterminado por la función.

Otra función útil para crear objetos es **rep()**. Esta función crea un objeto repitiendo otro. Esta función incluye los siguientes argumentos:

rep(x, times, each)

Donde:

- **x**: es el objeto a replicar.
- **times**: es el número de veces a repetir el objeto `x`. Si se emplea este argumento no se utiliza **each**.
- **each**: número de veces a repetir cada elemento del objeto `x`. Si se emplea este argumento no se utiliza **times**.

Repitamos el objeto `ventas_1` tres veces y cada uno de los elementos de ese objeto tres veces.

```
# se repite el objeto ventas_1 3 veces
v2 <- rep(ventas_1, times=3)
v2
```

```
##  [1] 2.0 3.0 4.5 5.0 2.0 3.0 4.5 5.0 2.0 3.0 4.5 5.0
```

```
# se repite cada elemento del objeto ventas_1 3 veces
v3 <- rep(ventas_1, each=3)
v3
```

```
##  [1] 2.0 2.0 2.0 3.0 3.0 3.0 4.5 4.5 4.5 5.0 5.0 5.0
```

También podemos crear objetos que contengan texto, estos pueden ser útiles para generar las etiquetas de bases de datos, cuadros o gráficos. Por ejemplo, supongamos que los cuatro clientes para los cuales registramos sus ventas del primer año en el objeto `ventas_1` son Juliana, Matías, Mariana y Luís, respectivamente. Guardemos esos nombres en el objeto `nombres`.

```
# creamos el objeto de nombres
nombres <- c("Juliana", "Matías", "Mariana", "Luís")
nombres
```

```
## [1] "Juliana" "Matías"  "Mariana" "Luís"
```

Es más, en algunas ocasiones desearemos crear nombres de manera automática y no digitarlos. Esto se hace combinando lo que hemos visto hasta ahora y la función **paste()**, esta "pega" los elementos de objetos empleando un separador determinado (argumento **sep**).

Creemos, por ejemplo, un objeto que tenga los nombres x_1 a x_10 y y_1 a y_10, pero alternando las equis (x) y las yes (y).

```
# creamos el objeto de nombres2 con nombres para la variable

nombres2 <- paste(c("x","y"), rep(1:10, each=2), sep="_")
nombres2
```

```
##  [1] "x_1"  "y_1"  "x_2"  "y_2"  "x_3"  "y_3"  "x_4"  "y_4"
##  [9] "x_5"  "y_5"  "x_6"  "y_6"  "x_7"  "y_7"  "x_8"  "y_8"
## [17] "x_9"  "y_9"  "x_10" "y_10"
```

4.4 Comentarios finales

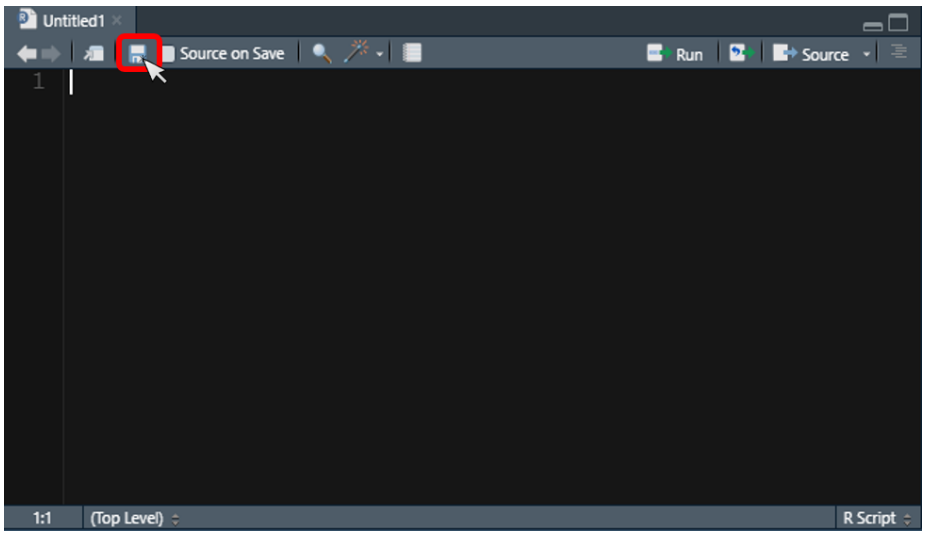
En este capítulo empleamos comandos para crear y hacer operaciones con objetos. En el siguiente veremos más aspectos importantes de los objetos como los diferentes tipos de objetos que existen y cómo transformar los tipos de objetos. Posteriormente en el Capítulo 8, discutiremos cómo construir objetos que contengan bases de datos que leeremos de archivos.

Por hora, guarda el *script* que has construido durante este capítulo. Lo necesitarás para el siguiente. Si deseas guarda el *workspace*, esto te permitirá cargar en cualquier momento todos los objetos que has creado. En algunas ocasiones, para ahorrar espacio de almacenamiento, los usuarios prefieren guardar los *scripts* y correrlos nuevamente cuando se requiera retomar el trabajo. Pero en otras ocasiones, es mejor guardar tanto el *script* como el *workspace* para ahorrar tiempo, pues la ejecución del *script* puede tomar mucho tiempo.

Para grabar el *script* puedes optar por cualquiera de las siguientes opciones que generarán un archivo con la extensión `.R`:

- Hacer clic en el ícono de un disquete (ver Figura 4.1) en la parte superior del panel donde está el *script*.
- Emplear los atajos de teclas. Teniendo el cursor activo en el *script* puedes presionar al mismo tiempo `Ctrl` + `s` en Windows o `Command` + `s`) en macOS.

Figura 4.1. Guardar un Script



Fuente: Esta captura de pantalla fue tomada de R.

- Emplear el menú de "File". Teniendo el cursor activo en el *script* puedes hacer clic en `File | Save`.

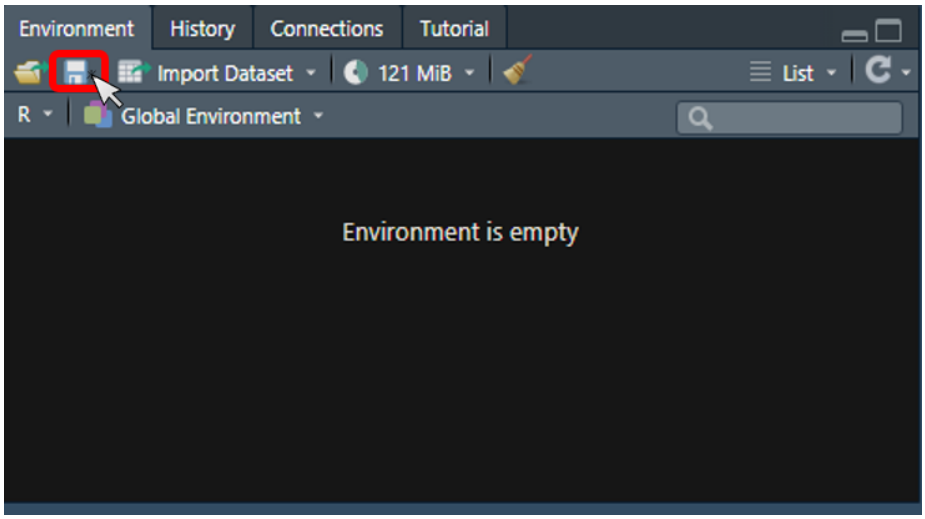
Grabar el *workspace* es muy similar. Puedes optar por cualquiera de las siguientes opciones que generarán un archivo con la extensión `.RData`:

- Hacer clic en el ícono de un disquete (ver Figura 4.2) en la parte superior de la pestaña *Enviroment* del panel en la parte superior derecha.
- Emplear el menú de "Session". Haz clic en `Session | Save Workspace As...`.
- Emplear código. La función **save.image()** que guardará todos los objetos en el *worspace* en el archivo con el nombre que deseemos (argumento **file**). El archivo se creará por defecto en el *working directory*, por ejemplo

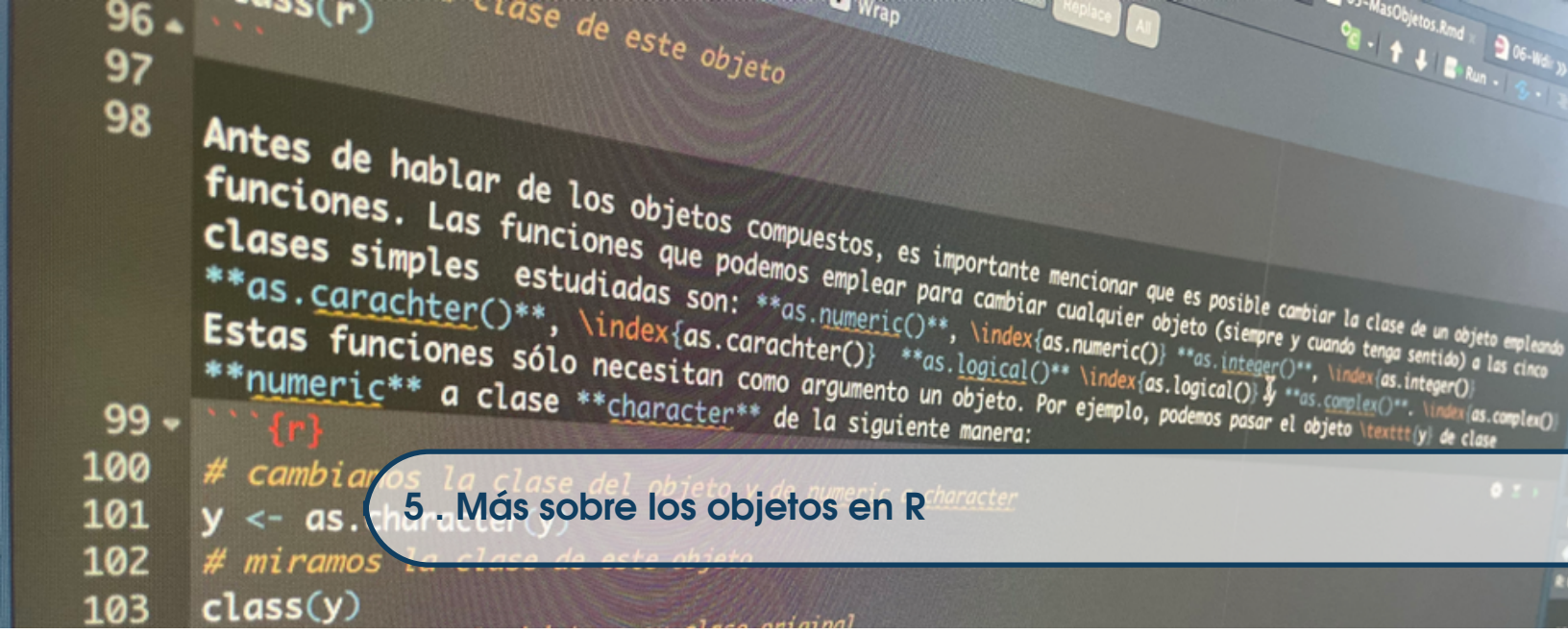
```
save.image(file = "res_cap4.RData")
```

En todo caso, cuando cierres la aplicación Rstudio, verás un menú que te pregunta si quieres guardar los *scripts* que no estén guardados y el *workspace*.

Figura 4.2. Guardar el Workspace



Fuente: Esta captura de pantalla fue tomada de R.



R es un lenguaje de programación orientado a la creación de objetos[1]. Los objetos son cualquier cosa a la que pueda asignársele una variable, además permiten que el usuario almacene la información que necesita para llevar a cabo su proyecto.

En el capítulo anterior creamos diferentes objetos: unos que contenían un solo número; otros que contenían resultados lógicos (TRUE o FALSE); otros que estaban compuestos por varios números, y otros que eran colecciones de texto.

En este capítulo estudiaremos algunos tipos de objetos importantes para continuar con nuestra introducción a R.

Los tipos de objetos en R se conocen en la comunidad como clases. Existen muchas clases de objetos, pero las clases más sencillas son 5:

- **numeric**: números reales o decimales
- **integer**: números enteros
- **complex**: números complejos
- **character**: carácteres
- **logical**: resultados lógicos (TRUE o FALSE)

Como lo vimos en el Capítulo 4, los objetos pueden combinarse para crear otros nuevos. Es más, hay objetos que tienen otros objetos que los conforman y cada uno puede ser de diferente clase. Así también tendremos otras clases para objetos compuestos, por ejemplo,

- **list**: listas
- **matrix**: matriz
- **array**: colección de objetos

[1] Los lenguajes de programación orientados a objetos se conocen por la OOP (*Object-Oriented Programming*.

- **data.frame**: base de datos

En este capítulo discutiremos estas clases de objetos.

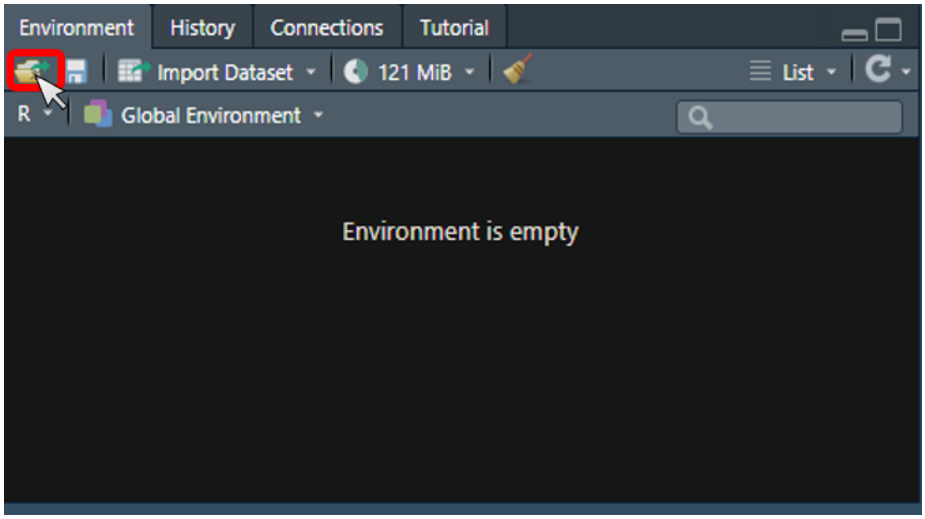
5.1 Clases sencillas de objetos

Los objetos más sencillos pueden ser los de la clase *numeric* que se emplea para números reales o decimales. Estos son tratados como variables continuas. La clase *integer* corresponde a un número entero (sin decimales). En este caso R trata a estos objetos como variables discretas. La clase *complex* corresponde a aquellos que contienen un número complejo. La clase *logical* se refiere a objetos que toman valores FALSE (falso) o TRUE (verdadero). Y la clase *character* se emplea para almacenar texto, conocido como cadenas de carácteres en R. La forma más sencilla de almacenar datos (de texto) bajo el formato de carácteres es utilizando comillas alrededor del fragmento de texto.

En el capítulo anterior (Capítulo 4) creamos 11 objetos. A continuación, carga el *workspace* o corre de nuevo el *script* del capítulo anterior. Un *workspace* previamente guardado (archivo con la extensión `.RData`) puede cargarse de las siguiente forma:

- Hacer clic en el icono de un folder con flecha saliente (ver Figura 5.1) en la parte superior de la pestaña `Enviroment` del panel en la parte superior derecha.

Figura 5.1. Cargar un Workspace



Fuente: Esta captura de pantalla fue tomada de R.

- Emplear el menú de "Session". Puedes hacer clic en `Session | Load Workspace...`.
- Emplear código. Puedes emplear la función **load()**, cuyo único argumento es el nombre del archivo (entre comillas) con el que grabaste el archivo `.RData` y su ruta si se requiere. Recuerda que R buscará cualquier archivo en el *working directory*, a menos que se especifique lo contrario; por ejemplo

```
load(file = "res_cap4.RData")
```

Ahora que ya tienes el mismo *workspace* que empleamos en el Capítulo 4, comprobemos los objetos que tenemos empleando la función **objets()** que vimos en el capítulo anterior.

```
objects()
```

```
##  [1] "nombres"      "nombres2"     "regular_plot" "source"
##  [5] "v2"           "v3"           "ventas_1"     "w"
##  [9] "x"            "x1"           "x2"           "y"
## [13] "z"
```

Para conocer la clase de cada objeto podemos emplear la función **class()** , por ejemplo

```
class(nombres)
```

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

Puedes emplear esta función para comprobar la clase de todos los 11 objetos como se reporta en el Cuadro @5.1

Tabla 5.1. Clase de los objetos creados en el capítulo anterior

Objeto	Clase	Objeto	Clase
nombres	character	x	numeric
nombres2	`rclass(nombres2)`	x1	`rclass(x1)`
v2	`rclass(v2)`	x2	`rclass(x2)`
v3	`rclass(v3)`	y	`rclass(y)`
ventas_1	`rclass(ventas_1)`	z	`rclass(z)`
w	`rclass(w)`		

Ya creamos objetos de todas las clases, menos de la clase **complex**. No es común emplear este tipo de objetos, pero si alguna vez necesitaras crear uno con un número complejo, es muy sencillo; por ejemplo:

```
# asingamos al objeto r el número complejo 1+2i
r <- 1 + 2i
r
```

```
## [1] 1+2i
```

```
# miramos la clase de este objeto
class(r)
```

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

Antes de hablar de los objetos compuestos, es importante mencionar que es posible cambiar la clase de un objeto empleando funciones. Las funciones que podemos emplear para cambiar cualquier objeto (siempre y cuando tenga sentido) a las cinco clases simples estudiadas son: **as.numeric()**, **as.integer()**, **as.carachter()**, **as.logical()** y **as.complex()**. Estas funciones sólo necesitan como argumento un objeto. Por ejemplo, podemos pasar el objeto `y` de clase **numeric** a clase **character** de la siguiente manera:

```
# cambiamos la clase del objeto y de numeric a character
y <- as.character(y)
# miramos la clase de este objeto
class(y)
```

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

```
# regresemos el objeto a su clase original
y <- as.numeric(y)
class(y)
```

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

Existe otra clase de objeto simple que es útil cuando trabajamos con información cualitativa: la clase **factor** . Esta clase se emplea para registrar datos cualitativos, como el sexo, los colores y si se está defectuoso o no. Esta se diferencia de la clase **character** en que puede tomar un número limitado de opciones (niveles o *levels* en inglés).

Continuemos con el ejemplo con el que empezamos en el capítulo anterior (Capítulo 4); en este registramos las compras en millones de pesos durante el primer año de operación de la compañía de 4 clientes en el objeto `ventas_1` y cuyos nombres guardamos en el objeto `nombres`. Ahora creemos el objeto `sexo` en el que registraremos el sexo (femenino o masculino) de cada uno de los clientes.

```
# recordando los nombres
nombres
```

```
## [1] "Juliana" "Matías"  "Mariana" "Luís"
```

```
# creamos el objeto sexo
sexo <- factor(c("Femenino", "Masculino", "Femenino",
                 "Masculino"))
# constatamos la clase
class(sexo)
```

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

También podemos constatar cuales son los niveles de un objeto factor empleando la función **levels()**.

```
# constatamos los niveles del objeto de clase factor
levels(sexo)
```

```
## [1] "Femenino"  "Masculino"
```

Observa que los objetos sencillos que hemos tratado hasta hora pueden ser de un solo dato (como el objeto `x1`) o pueden tener varios (como el objeto `ventas_1`). Los objetos como `ventas_1` y `v2` son conjuntos unidimensionales de datos cuyos elementos son numéricos, y por tanto, el objeto como tal es de clase **numeric** numérico. A este tipo de objetos se les conoce como vectores, estos son objetos que contienen conjuntos unidimensionales de datos (solo una fila), los cuales pueden ser de clase **integer**, **complex**, **character**, **logical** o **fator**. En todo caso, todos los elementos dentro del vector deben ser de la misma clase.

5.2 Clases de objetos compuestos

Hay objetos que son composiciones de otros y generan nuevas clases de objetos como la clase **list**. El objecto de esta clase es una estructura de datos que contiene elementos de diferentes tipos como **integer**, **complex**, **character**, **logical**, o **fator**. En R la lista puede ser creada usando la función **list()**; por ejemplo, podemos crear el objeto `l1` que tenga los objetos `w`, `x` y `nombres2`:

```
# creamos el objeto l1 de clase lista
l1 <- list(w, x, nombres2)
l1
```

```
## [[1]]
## [1] FALSE
##
## [[2]]
## [1] 1
##
## [[3]]
##  [1] "x_1"  "y_1"  "x_2"  "y_2"  "x_3"  "y_3"  "x_4"  "y_4"
##  [9] "x_5"  "y_5"  "x_6"  "y_6"  "x_7"  "y_7"  "x_8"  "y_8"
## [17] "x_9"  "y_9"  "x_10" "y_10"
```

```
# miramos la clase de este objeto
class(l1)
```

```
## [1] "list"
```

Este tipo de objetos tendrán diferentes compartimientos (*slots*), uno para cada objeto que conforma la lista. A los *slots* se puede acceder empleando los corchetes cuadrados (`[]`) para determinar el número del *slot* al que se quiere acceder; por ejemplo, en nuestro objeto `l1` tenemos tres *slots*. Accedamos al tercer *slot*:

```
# consultando el tercer slot del objeto l1
l1[3]
```

```
## [[1]]
##  [1] "x_1"  "y_1"  "x_2"  "y_2"  "x_3"  "y_3"  "x_4"  "y_4"
##  [9] "x_5"  "y_5"  "x_6"  "y_6"  "x_7"  "y_7"  "x_8"  "y_8"
## [17] "x_9"  "y_9"  "x_10" "y_10"
```

También podemos ponerle nombres a los *slots*; por ejemplo,

```
# poniéndole el nombre a los slots del objeto l1
names(l1) <- c("objeto_w", "objeto_x", "objeto_nombres")
l1
```

```
## $objeto_w
## [1] FALSE
##
## $objeto_x
## [1] 1
##
## $objeto_nombres
##  [1] "x_1"  "y_1"  "x_2"  "y_2"  "x_3"  "y_3"  "x_4"  "y_4"
##  [9] "x_5"  "y_5"  "x_6"  "y_6"  "x_7"  "y_7"  "x_8"  "y_8"
## [17] "x_9"  "y_9"  "x_10" "y_10"
```

Al ponerle nombre a los *slots*, podemos acceder a cada *slot* por su nombre empleando el signo `$` entre el nombre del objeto y el *slot*; por ejemplo,

```
# consultando los slots del objeto l1 por sus nombres
l1$objeto_w
l1$objeto_nombres
```

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

```
##  [1] "x_1"  "y_1"  "x_2"  "y_2"  "x_3"  "y_3"  "x_4"  "y_4"
##  [9] "x_5"  "y_5"  "x_6"  "y_6"  "x_7"  "y_7"  "x_8"  "y_8"
## [17] "x_9"  "y_9"  "x_10" "y_10"
```

Otra clase de objeto compuesto muy común es **matrix** (matriz). Una matriz en R es un conjunto de 2 dimensiones (filas y columnas) en los que cada elemento tiene la misma clase. Los objetos de clase **matrix** se pueden crear con la función **matrix()** , cuyos argumentos son:

matrix(data , nrow , ncol, byrow = FALSE)

Donde:

- **data**: son los datos que serán incluidos en la matriz.
- **nrow**: número de filas de la matriz.
- **ncol**: número de columnas de la matriz.
- **byrow**: le dice a R si los datos se llenarán por filas (`byrow = TRUE`) o por columnas (`byrow = FALSE`). Este último es el valor por defecto.

Por ejemplo, podemos crear una matriz de dos filas y tres columnas de la siguiente manera:

```
# creando una matriz de dos filas por tres columnas
m1 <- matrix( c(1, 2, 3, 3, 4, 5), nrow = 2, ncol = 3, byrow = FALSE)
m1
```

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

```
# miramos la clase de este objeto
class(m1)
```

```
## [1] "matrix" "array"
```

También podemos crear una matriz combinando vectores de las mismas dimensiones; por ejemplo, creemos una matriz cuyas columnas sea los objetos `v2` y `v3`. Esto lo podemos hacer con la función **cbind()**. Esta función combina los vectores (columnas) (de ahí la c de columna y **bind** de unir) que se especifiquen como argumento de la función[2]; por ejemplo,

```
v2
```

```
##  [1] 2.0 3.0 4.5 5.0 2.0 3.0 4.5 5.0 2.0 3.0 4.5 5.0
```

```
v3
```

```
##  [1] 2.0 2.0 2.0 3.0 3.0 3.0 4.5 4.5 4.5 5.0 5.0 5.0
```

[2]También existe una función que permite unir filas: **rbind()**.

```
# creando una matriz uniendo los dos vectores como columnas
m2 <- cbind(v2, v3)
m2
```

```
##        v2  v3
##  [1,] 2.0 2.0
##  [2,] 3.0 2.0
##  [3,] 4.5 2.0
##  [4,] 5.0 3.0
##  [5,] 2.0 3.0
##  [6,] 3.0 3.0
##  [7,] 4.5 4.5
##  [8,] 5.0 4.5
##  [9,] 2.0 4.5
## [10,] 3.0 5.0
## [11,] 4.5 5.0
## [12,] 5.0 5.0
```

```
# miramos la clase de este objeto
class(m2)
```

```
## [1] "matrix" "array"
```

Con las matrices podemos hacer numerosas operaciones similares a las discutidas en la sección 4.2. En el Cuadro 5.2 se presenta un resumen de las principales operaciones que se pueden realizar con los objetos de clase **matrix**.

Tabla 5.2. Operaciones con matrices en R

Operación	Ejemplo	Función o Código en R
Obtener el elemento i, j de una matriz	$\mathbf{A}_{1,2}$	**A[1, 2]**
Transponer una matriz	$\mathbf{A}^T$	**t(A)**
Determinante de una matriz cuadrada	$\det(\mathbf{A}_{n\times n})$	**det(A)**
Inversa de una matriz cuadrada	$\mathbf{A}^{-1}_{n\times n}$	**solve(A)**
Elegir una columna	Columna 3 de la matriz $\mathbf{A}$	**A[, 3]**

Operación	Ejemplo	Función o Código en R
Columna tiene nombre X		**A$X**
Elegir una fila	Fila 5 de la matriz $\mathbf{A}$	**A[5,]**
	Datos	
Suma de matrices conformables	$\mathbf{A}_{n\times m} + \mathbf{B}_{n\times m}$	**A + B**
Resta de matrices conformables	$\mathbf{A}_{n\times m} - \mathbf{B}_{n\times m}$	**A - B**
Multiplicación de matrices conformables	$\mathbf{A}_{n\times m} \times \mathbf{B}_{m\times k}$	**A %*% B**

Otra clase de objetos compuestos es **array** (arreglo en español). Un objeto de esta clase es similar a las matrices pero pueden tener más de dos dimensiones. Esto nos permite construir cubos de datos u objetos de datos con más de dos dimensiones. Por ejemplo, un objeto de clase **array** de dimensión (2, 3, 4) contiene 4 matrices cada una con 2 filas y 3 columnas. Un objeto de esta clase se crea con la función **array()** , la cuál tiene dos argumentos, los datos (**data**) y las dimensiones (**dim**).

Por ejemplo, creemos un objeto que contenga dos matrices, cada una con 3 filas y 3 columnas:

```
# creando un array de dos matrices cada una con
# 3 filas y 3 columnas
a1 <- array(c(5, 9, 3, 10, 11, 12, 13, 14, 15),dim = c(3,3,2))
a1
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

```
# miramos la clase de este objeto
class(a1)
```

```
## [1] "array"
```

Finalmente, discutamos la clase de objetos compuestas más útil para hacer análisis de datos: la clase **data.frame**. Esta clase es similar a las matrices, pero se entiende que cada columna corresponde a una variable y cada fila a las observaciones. Y su gran diferencia con una matriz es que cada variable (columna) puede ser de diferente clase.

Continuemos con el ejemplo con el cual empezamos, en aquel en el que registramos las ventas en millones de pesos durante el primer año de operación de la compañía de 4 clientes. En el objeto `ventas_1` de clase **numeric** tenemos las ventas realizadas a cada cliente, en el objeto `nombres` de clase **character** tenemos los nombres de los clientes y en el objeto `sexo` de clase **factor** tenemos el sexo de cada cliente. Pongamos estas variables en una sola base de datos para su análisis posterior, empleando la función **data.frame()** .

```
# creando un data frame
d_ventas <- data.frame(ventas_1, nombres, sexo)
d_ventas
```

```
##   ventas_1 nombres      sexo
## 1      2.0 Juliana  Femenino
## 2      3.0  Matías Masculino
## 3      4.5 Mariana  Femenino
## 4      5.0    Luís Masculino
```

```
# miramos la clase de este objeto
class(d_ventas)
```

```
## [1] "data.frame"
```

Como lo realizamos anteriormente, podemos modificar el nombre de las variables (columnas) empleando la función **names()**; por ejemplo, cambiemos el nombre de la primera variable a solo **ventas**:

```
# cambiando el nombre d ela preimera variable
names(d_ventas)[1] <- "ventas"
d_ventas
```

```
##   ventas nombres      sexo
## 1    2.0 Juliana  Femenino
## 2    3.0  Matías Masculino
## 3    4.5 Mariana  Femenino
## 4    5.0    Luís Masculino
```

Ahora podemos ver la clase de cada una de las variables del objeto de clase **data.frame**. Esto lo podemos hacer rápidamente con la función **str()**:

```
# constatando las clases de cada variable en el data frame
str(d_ventas)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ ventas : num  2 3 4.5 5
##  $ nombres: chr  "Juliana" "Matías" "Mariana" "Luís"
##  $ sexo   : Factor w/ 2 levels "Femenino","Masculino": 1 2 1 2
```

En el caso de los objetos de clase **data.frame**, así como de los objetos de clase **lista**, **matrix** y **array**, se pueda acceder a una columna (variable) empleando el signo $ entre el nombre del objeto y el nombre de la variable. Veamos un ejemplo[3] .

```
# constatando la clase de cada variable en el data frame
class(d_ventas$sexo)
```

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

También podemos acceder a un dato específico, como lo hacemos en el caso de las matrices, empleando los corchetes. El siguiente ejemplo muestra el segundo dato de la variable que se encuentra en la tercera columna. Recuerda que el primer elemento representa la fila y el segundo después de la coma es la columna:

```
# extrayendo el segundo dato de la variable que se encuentra
# en la tercera columna
d_ventas[2,3]
```

```
## [1] Masculino
## Levels: Femenino Masculino
```

[3]Esto también se podría hacer con el siguiente código `d_ventas[ ,3]` o `d_ventas[ ,"sexo"]`.

O podemos ver una observación completa (fila). Extraigamos la segunda observación con todos sus datos para las variables:

```
# extrayendo la segunda observación para todas las variables
d_ventas[2,]
```

```
##   ventas nombres      sexo
## 2      3  Matías Masculino
```

Recomendación de Estilo

Emplea en los nombres de los objetos sólo letras minúsculas, números y _. No emplees tildes ni la letra "eñe". Los nombres de los objetos pueden ser sencillos, una sola letra o sustantivos, pero no uses verbos (estos se reservan para las funciones). Intenta que los nombres sean cortos y con significado. Si se requiere, utiliza los guiones bajos para separar las palabras dentro de un nombre. Siempre que sea posible, evita utilizar nombres de objetos ya usados por R y sus paquetes; por ejemplo, evita emplear el nombre `data` para los `data.frames`.

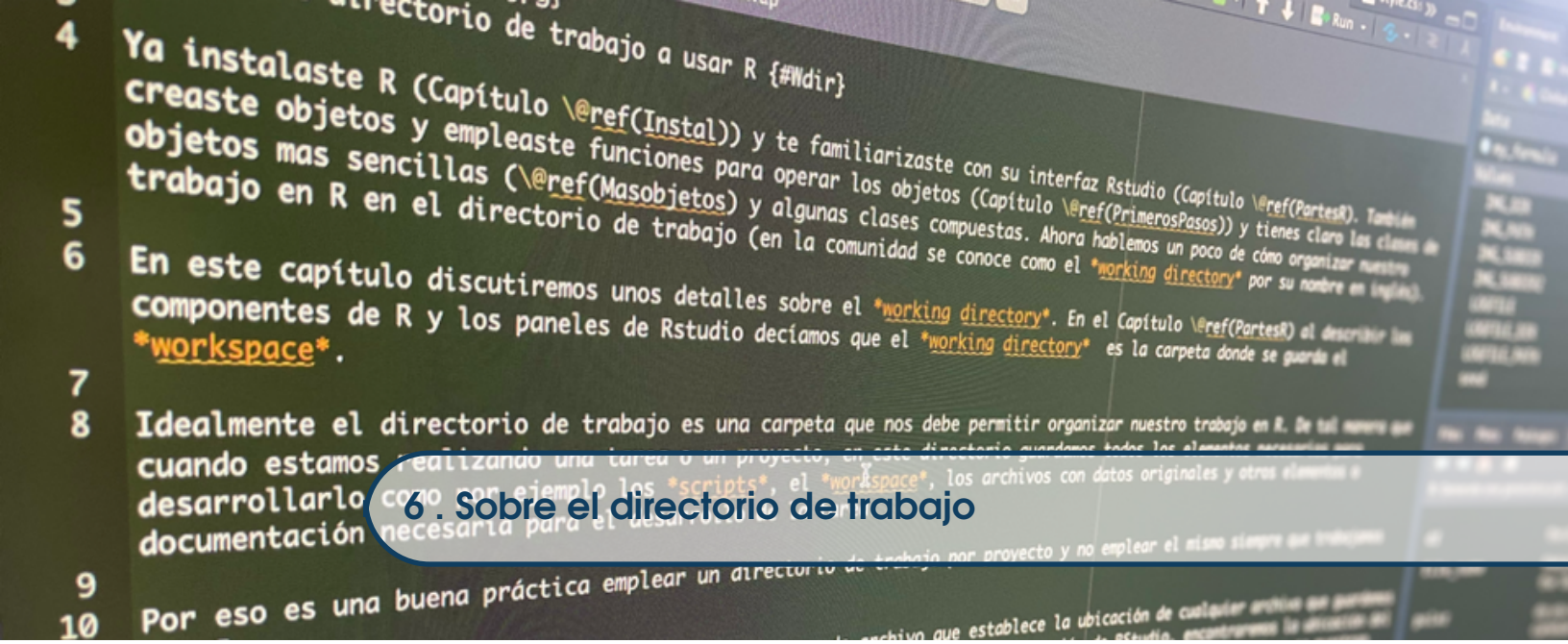
```
# Buena práctica
empleados_hora

# Mala práctica
empleadosHora
empleados_por_hora_para_el_primer_mes
```

5.3 Comentarios finales

En este capítulo discutimos diferentes clases de objetos que te permitirán trabajar en un futuro con otras clases. De hecho, existen innumerables clases de objetos, pero con lo que hemos visto aquí te podrás defender fácilmente en tu proceso de aprendizaje de este lenguaje.

Tal vez la clase de objeto más importante para trabajar con datos es el **data.frame**. Aquí vimos cómo construir de manera muy manual un **data.frame**, pero este no es el caso en la mayoría de las aplicaciones. Normalmente, las bases de datos se encuentran en archivos aparte. En el Capítulo 8 veremos cómo crear objetos de clase **data.frame** leyendo datos de archivos o leyendo datos de paquetes. Pero antes, en el Capítulo 7 discutiremos un poco más sobre las funciones que vienen en paquetes especiales. Y en el Capítulo @6 hablaremos del *working directory*.



6 . Sobre el directorio de trabajo

Ya instalaste R (Capítulo 2) y te familiarizaste con su interfaz Rstudio (Capítulo 3. También creaste objetos y empleaste funciones para operar los objetos (Capítulo 4) y tienes claro las clases de objetos mas sencillas (Capítulo 5) y algunas clases compuestas. Ahora hablemos de cómo organizar nuestro trabajo en R en el directorio de trabajo (en la comunidad se conoce como el *working directory* por su nombre en inglés).

En este capítulo discutiremos unos detalles sobre el *working directory*. En el Capítulo 3, al describir los componentes de R y los paneles de Rstudio, decíamos que el *working directory* es la carpeta en donde se guarda el *workspace*.

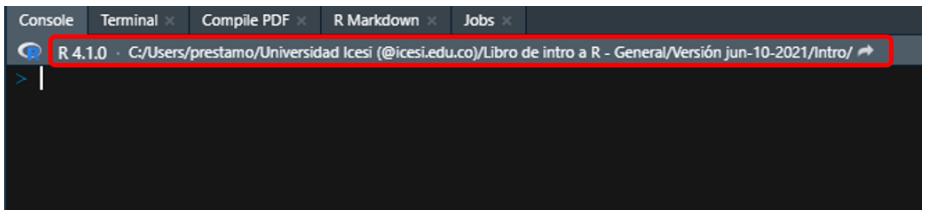
Idealmente el directorio de trabajo es una carpeta que nos debe permitir organizar nuestro trabajo en R. De manera que cuando estemos realizando una tarea o un proyecto en este directorio guardemos todos los elementos necesarios para desarrollarlo, como por ejemplo los *scripts*, el *workspace*, los archivos con datos originales y otros elementos o documentación necesaria para el desarrollo de la tarea.

Por eso una buena práctica es emplear un directorio de trabajo por proyecto y no el mismo siempre que trabajamos con R.

Técnicamente, el directorio de trabajo es una ruta de archivo que establece la ubicación de cualquier archivo que guardemos desde R y también de los datos que vayamos a importar a R. Al iniciar una sesión de RStudio, encontraremos la ubicación del *working directory* como se muestra en la Figura 6.1. En el pánel inferior derecho en la pestaña `Files` se muestran los archivos que están en el *working directory* actual.

Si abres Rstudio desde el ícono del programa, R empleará como directorio de trabajo predeterminado una ruta a una carpeta en mis documentos o una carpeta similar. Si quieres conocer cuál es el directorio

Figura 6.1. Directorio de trabajo preestablecido.



Fuente: Esta captura de pantalla fue tomada de RStudio

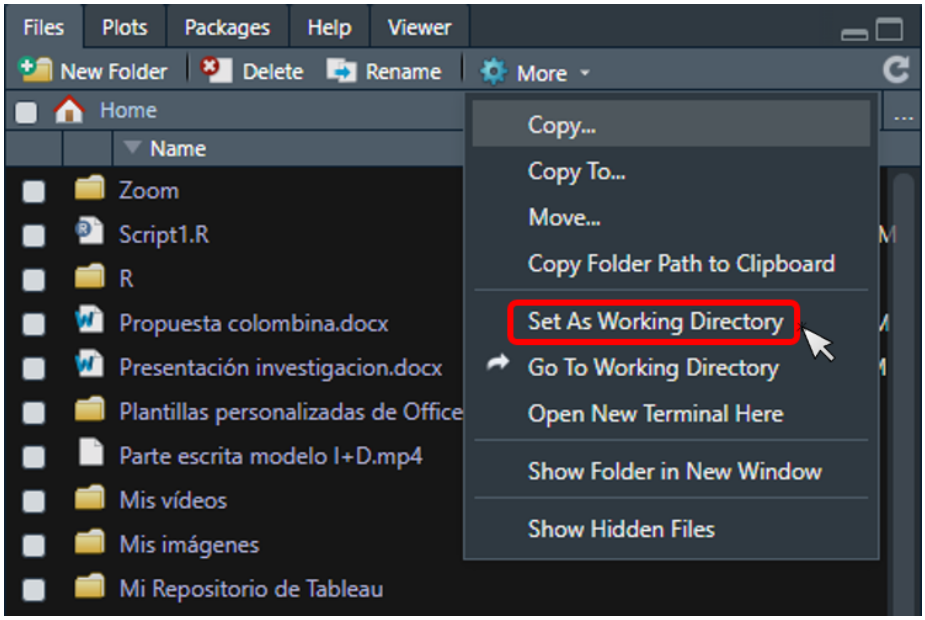
de trabajo actual puedes emplear la función **getwd()**; por ejemplo,

```
# recuperando el directorio de trabajo
getwd()
```

Esto hace necesario que tengamos que cambiar el directorio de trabajo cuando iniciamos sesión en R, si queremos organizar nuestro trabajo en una carpeta diferente cada vez que desarrollemos un proyecto diferente. Esto puede ejecutarse de diferentes maneras, una es empleando el menú de "Session". Puedes hacer clic en `Session | Set Working Directory | Choose Directory...`. Otra opción es emplear los atajos de teclas. Puedes presionar simultaneamente `Ctrl + shift + h` en Windows o `control + shift + h` en macOS.

Otra forma de hacerlo es desde la pestaña de `Files`. Navega hasta el folder que se quiere emplear como directorio de trabajo y una vez te encuentres en ese directorio puedes establecer ese directorio como el de trabajo haciendo clic en el ícono del piñón (*More*) como se muestra en la Figura 6.2.

Figura 6.2. Establecer el directorio de trabajo desde la pestaña 'Files'



Fuente: Esta captura de pantalla fue tomada de RStudio

Como viste, en la comunidad R es mas "elegante" emplear líneas de código para hacer una tarea. Esto porque permite automatizar las tareas, y correr un *script* que haga todas las tareas deseadas al tiempo. Por eso también existe una manera de cambiar el directorio de trabajo empleando una función. La función **setwd()** permite fijar el directorio de trabajo a cualquier ruta que se desee. La ruta debe escribirse entre comillas; por ejemplo, supongamos que empleas Windows y quieres cambiar tu directorio de trabajo a una carpeta que se llama *cap6* en la carpeta *libroIntroR* (estas carpetas deben ser creadas previamente) que está en mis documentos en la unidad C del computador. El código sería:

```
# cambiando el directorio de trabajo en windows
setwd("C:/Documentos/libroIntroR/cap6")
```

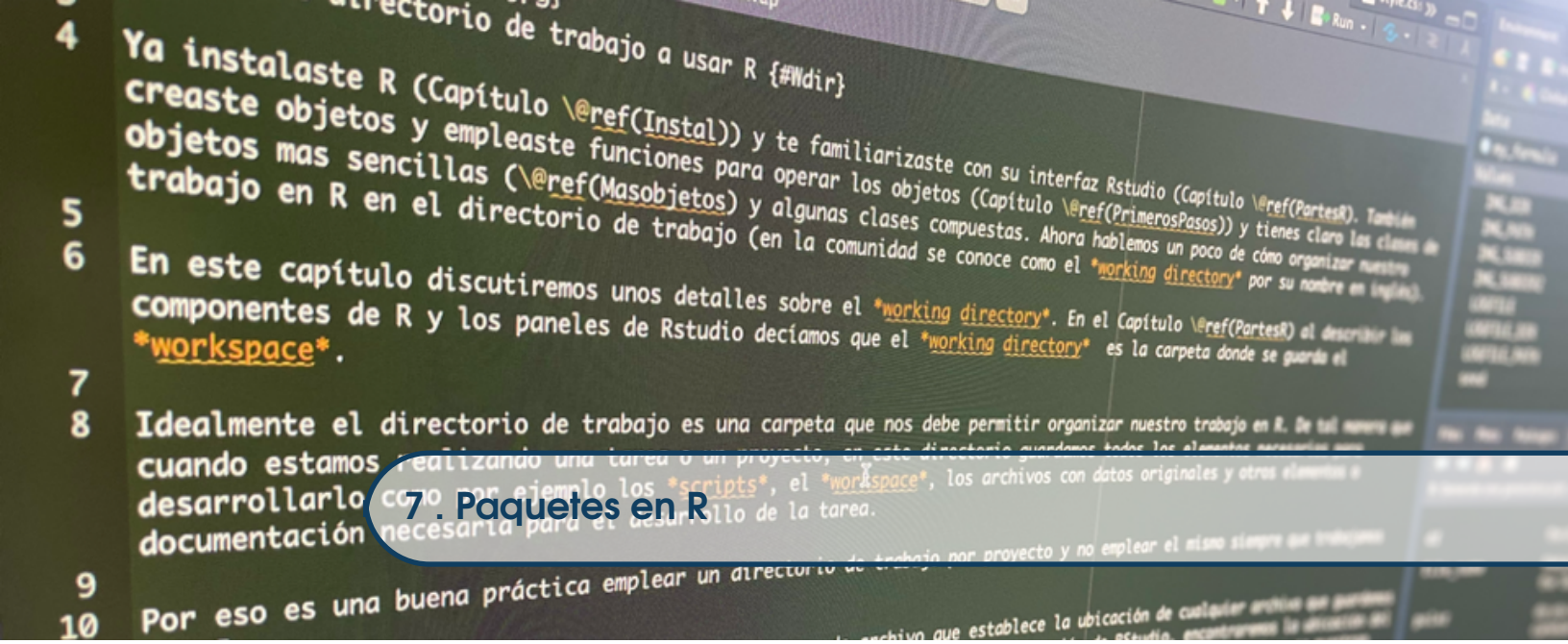
Ahora supongamos que estas usando macOS, el código sería[1]

```
# cambiando el directorio de trabajo en macOS
setwd("~/Documents/libroIntroR/cap6")
```

Una nota importante: en muchas ocasiones podemos estar trabajando en un servidor o compartiendo *scripts* entre diferentes computadores y hasta sistemas operativos. Por lo tanto, se debe tener mucho cuidado con fijar las rutas a los directorios de trabajo.

Un truco que te puede ayudar a no tener que fijar el directorio de trabajo desde el código, es abrir Rstudio no haciendo clic en el ícono del programa, sino por el contrario, hacer clic en el ícono del *script* que quieras emplear. **Cuando abres RStudio desde el *script*, el *working directory* queda automáticamente preestablecida al folder en el que está el archivo con el *script*** (archivo con extensión `.R`). ¡Este truco puede ahorrarte mucho tiempo!

[1] Estas carpetas deben ser creadas previamente.



7. Paquetes en R

En este capítulo regresaremos a la discusión de las funciones en R. Ya discutimos ampliamente los objetos (Capítulos 4 y 5). Ahora centremos la atención en las funciones en el lenguaje R. Recuerda que las funciones permiten desarrollar tareas con objetos.

En el Capítulo 4 discutimos que los comandos en R se dividen en dos grandes tipos: asignaciones y expresiones. Las **asignaciones** sirven para guardar datos en objetos. Las **expresiones**, más conocidas en la comunidad de R como **funciones**, son una o más ordenes que le damos a R para que actúe sobre objetos.

R tiene un *Core* (Núcleo o base) principal que incluye funciones que conforman la parte central de R. Todas las que hemos discutido hasta ahora hacen parte del *Core*. Pero precisamente, una de las fortalezas de R, es su capacidad de expandirse para realizar diferentes tareas. La comunidad de R ha construido **paquetes** que incluyen diversas funciones tanto como para hacer tareas específicas, así como generales.

En otras palabras, los *paquetes de R son un conjunto de funciones y bases de datos desarrollados por la comunidad*. Aumentan la potencia de R mejorando las funcionalidades básicas de R existentes o añadiendo otras nuevas.

O dicho de otra manera, en R la unidad fundamental de código compartible es el paquete. Un paquete agrupa funciones y datos que se acompañan con documentación y ejemplos (de ahí la necesidad de incluir datos en los paquetes).

El primero de agosto del 2022, estaban registrados 18,183 paquetes disponibles en la Comprehensive R Archive Network (CRAN). CRAN es el espacio virtual en donde descargaste el R (su *Core*) y donde está el repositorio de los paquetes "oficiales" de R. Recuerda que hay muchos

mirrors de CRAN al rededor del mundo que facilitan el acceso a este repositorio.

También podemos encontrar otros paquetes "no oficiales" de R en otros repositorios. Pero los que se encuentran en CRAN siguen el mismo formato y cuentan con toda la documentación necesaria para entender qué hace cada una de las funciones que lo componen, así como la descripción de las bases de datos que incluyen. Y mejor aún, si se desea se pueden modificar las funciones de los paquetes de ser necesario.

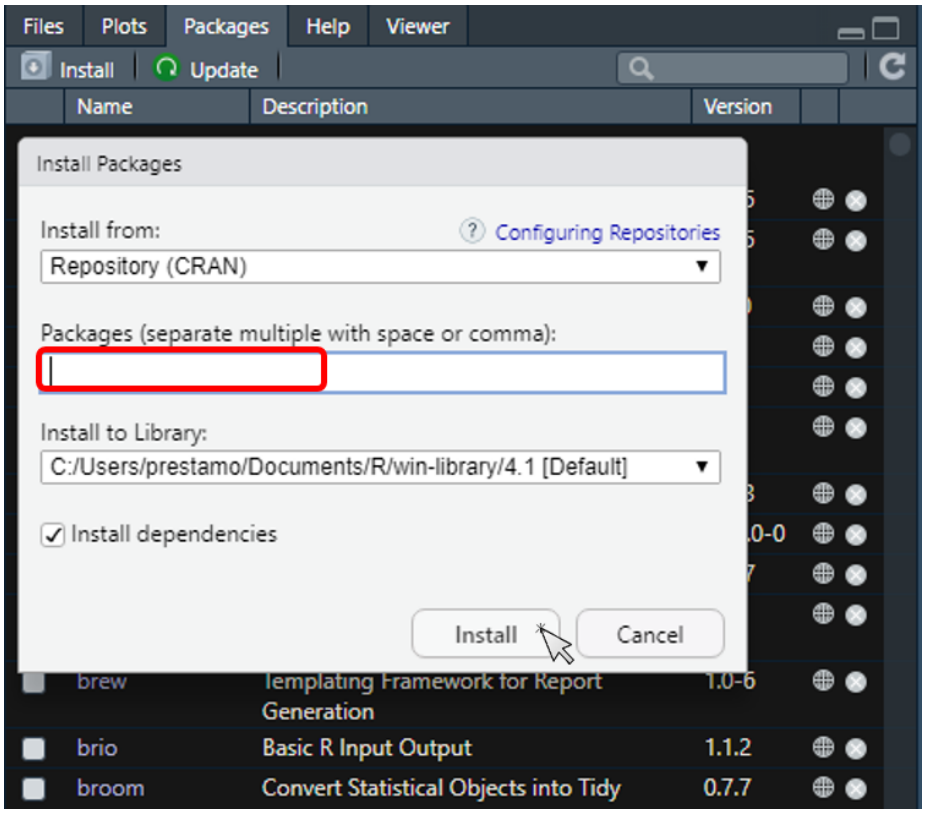
La enorme variedad de paquetes es una de las razones por las cuales R tiene tanto éxito. De hecho, es muy probable que alguien ya haya creado un paquete para realizar la tarea o resolver el problema en el que estás trabajando. Y si no existe, ¡tienes la oportunidad de crear uno y contribuir a esta comunidad!

Para instalar un paquete en R podemos usar la función **install.packages()** . El argumento de esta función es el nombre del paquete y se expresa entre comillas; es decir,

```
install.packages("nombre del paquete")
```

Adicionalmente, podemos emplear el menú de "Tools". Haz clic en `Tools | Install Packages...`. También se puede llegar a la misma parte a través de la pestaña `Packages` en el panel inferior derecho de RStudio, aquí podemos dar clic en `install` y aparecerá una nueva ventana en donde puedes escribir el nombre del paquete que deseas instalar (como se muestra en la Figura 7.1). Ahí también puedes seleccionar el *mirror* desde el cuál quieres descargar el paquete. Todos los *mirors* al rededor del mundo están sincronizados, así que no hace diferencia el *mirror* que se escoja encontrarás los mismos paquetes siempre. La gran diferencia es la velocidad de descarga. En principio, entre más cerca geográficamente estés de un *mirror*, mayor velocidad de descarga de los paquetes. La primera vez que instalemos un paquete, aparecerá un cuadro de diálogo en el que se nos pide elegir el CRAN *mirror* más cercano (Ver Figura 7.1).

Figura 7.1. Instalación un paquete empleando menús



Fuente: Esta captura de pantalla fue tomada de RStudio.

Las funciones y demás material de un paquete de R se almacenan en una *library*. Técnicamente, una *library* es un lugar (directorio) donde R guarda en tu computador, y por tanto, sabe que ahí encontrará el paquetes que se quiere usar.

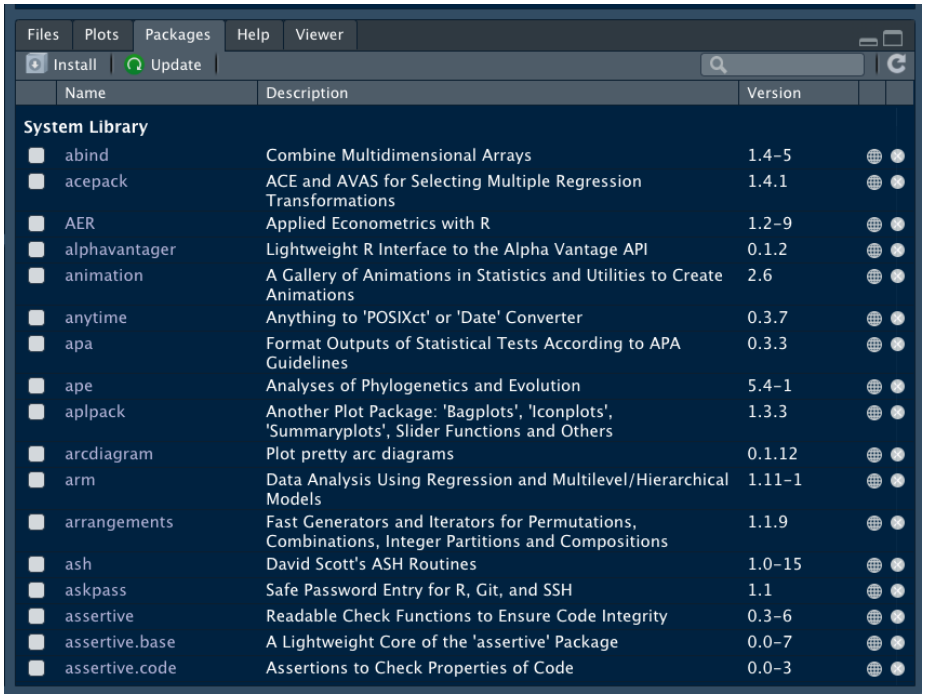
En otras palabras, un paquete es una colección de funciones y datos. La ubicación donde se almacenan los contenidos de un paquete se llama *library*. Por tanto, para decirle a R que use un paquete (para "cargarlo") se emplea la función **library()** . El argumento es el nombre del paquete que se quiere cargar (no se emplea comillas).

Es importante aclarar que para ahorrar memoria y mejorar el desempeño, cada vez que inicies una sesión en R no se cargan paquetes. Dependiendo de la tarea que realices cargarás los paquetes adecuados para dicha labor.

Una vez instalado un paquete la librería será creada y no será necesario instalar de nuevo el paquete. No obstante, para usar el paquete, debemos pedirle a R que los active en cada sesión. Así, para usar los paquetes es necesario emplear la función **library()** para cargarlos.

Otra opción, menos elegante, es emplear la pestaña `Packages` que se encuentra en el panel inferior derecho en RStudio. Puedes cargar los paquetes haciendo clic en los cuadros que se encuentran a la derecha de cada paquete (Ver 7.2). Aquí también encontrarás la descripción de cada uno de los paquetes.

Figura 7.2. Pestaña ‘Packages‘.

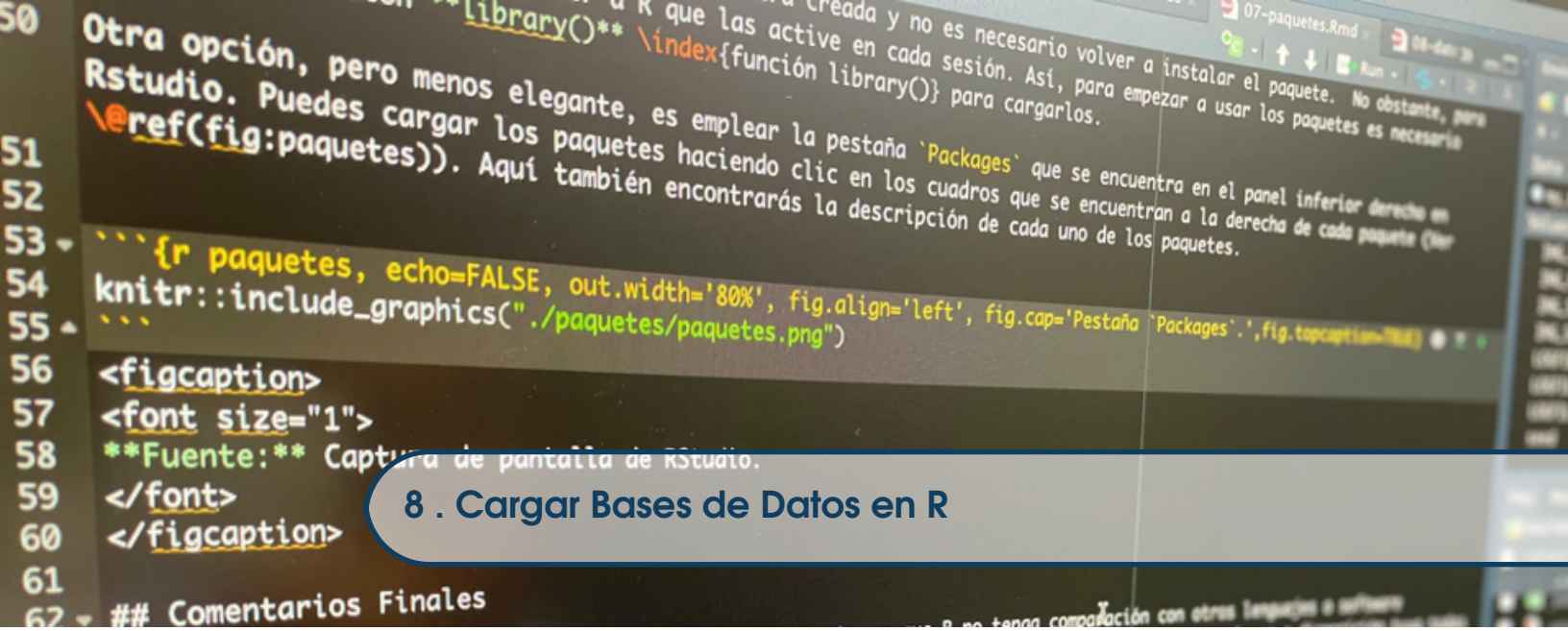


Fuente: Esta captura de pantalla fue tomada de RStudio.

7.1 Comentarios finales

Los paquetes agregan gran versatilidad a R. Hacen que R no tenga comparación con otros lenguajes o software disponibles. Como vimos en este capítulo es muy sencillo instalarlo y cargarlos. Así que ya tienes a tu disposición todos los paquetes disponibles en CRAN. Es virtualmente imposible hacer un recorrido en un libro sobre todos los paquetes disponibles en R y cómo usarlos. Puedes buscar los paquetes disponibles en CRAN por tema en el siguiente enlace: https://cran.r-project.org/web/views/. Ahí veras la variedad de temas para los que existen paquetes. Si quieres un listado en orden alfabético de estos paquetes lo puedes encontrar en este otro enlace: https://cran.r-project.org/web/packages/available_packages_by_name.html.

En todo caso, los paquetes te permitirán explorar y analizar los datos. Precisamente, la intención del Capítulo 8 es discutir cómo cargar bases de datos en R que después podrás analizar con los diversos paquetes disponibles en R.



8 . Cargar Bases de Datos en R

Ya tenemos una "caja de herramientas" que permite empezar a usar R con su interfaz RStudio. Ya dominas las clases más comunes de objetos en R y sabes cómo instalar paquetes. Antes de terminar este libro es importante explicar cómo crear objetos de clase **data.frame** a partir de archivos externos o extraer los datos incluidos en diferentes paquetes.

8.1 Importando Base de Datos de Archivos

Las bases de datos las podemos encontrar en diferentes formatos, pero tal vez los tres formatos más comunes son:

- Archivo de texto con extensión `.txt`,
- Archivo de texto de valores separados por comas (CSV por su sigla del inglés *comma-separated values*) con extensión `.csv`[1] y
- Archivo de Excel con extensión `.xls` o `.xlsx`.

A continuación discutiremos cómo cargar cada uno de estos tres formatos de datos.

8.1.1 Leyendo Archivos .txt

Los archivos de texto son conocidos como archivos "planos", pues no incluyen ninguna información de formato, solo datos "puros". Es decir, en un archivo de Excel podemos contar con formato para los bordes

[1] De hecho en algunas ocasiones los archivos con extensión `.csv` pueden estar delimitados por ";". En español es común que en los archivos csv los datos se separen por puntos y comas para evitar la confusión que puede generar el delimitador de decimales que es una coma.

o para los mismos números. En los archivos planos no hay formato, por lo tanto, estos archivos son más pequeños. Otra ventaja es que este formato de datos no tiene limitaciones de filas o columnas. Excel (la versión Microsoft 365) tiene una limitación de 1,048,576 filas y 16,384 columnas. Esto usualmente no es un problema, pero si podría ser una limitación en el mundo del *Big Data*.

Los archivos con extensión `.txt` pueden ser leídos empleando la función **read.table()** . Los principales argumentos de esta función son la ruta y el nombre del archivo (entre comillas), un argumento que le dice a R si los datos tiene un encabezado `header = TRUE` o no `header = FALSE` y el delimitador de los datos (argumento **sep**).

Para nuestro ejemplo, emplearemos el archivo *Employee.txt*[2], el cual tiene datos recopilados por el área de gestión humana de una de las principales empresas que venden automóviles en todo el mundo. El archivo cuenta con información del número de vendedores y las unidades fabricadas (en miles). Los datos son mensuales para los últimos 5 años.

El archivo lo podemos leer con el siguiente código:

```
# leyendo el archivo .txt
employee <- read.table("employee.txt", header = TRUE, sep = ",")
# verificando la clase del objeto
class(employee)
```

```
## [1] "data.frame"
```

Siempre que leamos datos de un archivo es importante estar seguros que los datos quedaron bien cargados; por ejemplo, veamos los primeros 5 datos del objeto `employee` con la función **head()**:

```
# inspeccionando los primeros 5 datos
head(employee, 5)
```

```
##   mes vendedores unidades
## 1   1        322     44.2
## 2   2        317     44.3
## 3   3        319     44.4
## 4   4        323     43.4
## 5   5        327     42.8
```

Todo parece bien. Veamos los últimos 3 datos con la función: **tail()**.

[2]El archivo se puede encontrar en la página web del libro: http://www.icesi.edu.co/editorial/empezando-usar

```
# inspeccionando los últimos 3 datos
tail(employee, 2)
```

```
##    mes vendedores unidades
## 59  59        392     49.2
## 60  60        396     48.1
```

Ahora verifiquemos la clase de las variables que componen este **data.frame** con la función **str()**:

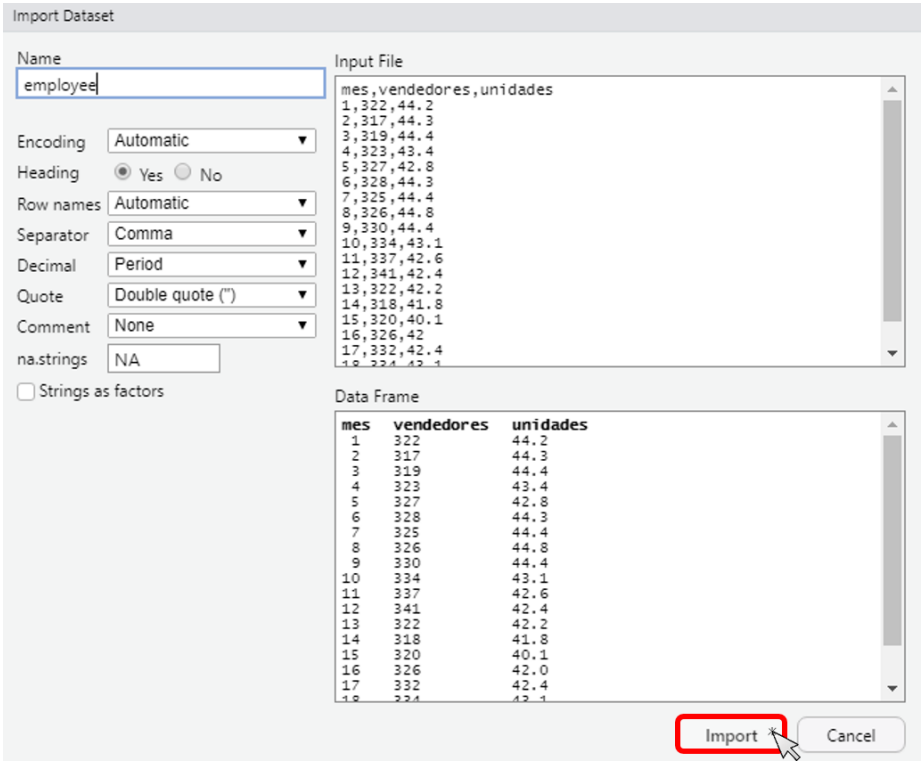
```
# inspeccionando la clase de cada variable
str(employee)
```

```
## 'data.frame':    60 obs. of  3 variables:
## $ mes : int 1 2 3 4 5 6 7 8 9 10 ...
## $ vendedores: int 322 317 319 323 327 328 325 326 330 334
##    ...
## $ unidades : num 44.2 44.3 44.4 43.4 42.8 44.3 44.4 44.8
##    44.4 43.1 ...
```

Todo parece estar en orden. Los datos han sido leídos correctamente.

RStudio también permite importar datos empleando el menú de "File". Haciendo clic en `File | Import Dataset | From Text (base)...`. Observarás primero una ventana en la que debes seleccionar el archivo a ser leído. Haz clic en el archivo y verás una ventana como la de la Figura 8.1. Esa ventana es el área de vista previa de datos. Es decir, cómo está R interpretando los datos. Si observas que está leyendo bien los datos haz clic en `import`, de lo contrario puedes usar las opciones a mano izquierda hasta encontrar las opciones que se ajusten a tu archivo plano de datos. Nota que el código que permitió hacer dicha importación de los datos aparecerá en la consola. Puedes copiar ese código en tu *script* para no hacer todo este procedimiento la próxima vez que quieras leer esos datos. Recuerda que en esta comunidad siempre será más elegante emplear código para hacer cualquier tarea.

Figura 8.1. Ventana emergente al cargar un archivo '.txt'



Fuente: Esta captura de pantalla fue tomada de RStudio

Puedes llegar a las mismas ventanas desde la pestaña `Environment` en el panel superior derecho donde hay un menú desplegable llamado `Import Dataset`. El menú desplegable ofrece diferentes opciones del tipo de datos que deseamos importar, ya sea un archivo de texto, un archivo de Excel o un archivo csv.

8.1.2 Leyendo Archivos `.csv`

Los archivos `.csv` tienen las mismas ventajas que los archivos `.txt` sobre los archivos de Excel. Este tipo de archivos se pueden leer con la función **read.csv()** . Empleemos el archivo *Employee.csv*[3] que tiene la misma estructura del archivo *Employee.txt* que ya cargamos. En este caso el código será el siguiente:

```
# leyendo el archivo .txt
employee2 <- read.csv("employee.csv", header = TRUE, sep = ";")
# verificando la clase del objeto
class(employee2)
```

```
## [1] "data.frame"
```

```
# inspeccionando los primeros 5 datos
head(employee2, 5)
```

```
##   mes vendedores unidades
## 1   1        322     44.2
## 2   2        317     44.3
## 3   3        319     44.4
## 4   4        323     43.4
## 5   5        327     42.8
```

```
# inspeccionando los últimos 3 datos
tail(employee2, 2)
```

```
##    mes vendedores unidades
## 59  59        392     49.2
## 60  60        396     48.1
```

```
# inspeccionando la clase de cada variable
str(employee2)
```

[3]El archivo se puede encontrar en la página web del libro: http://www.icesi.edu.co/editorial/empezando-usar

```
## 'data.frame':    60 obs. of  3 variables:
## $ mes : int 1 2 3 4 5 6 7 8 9 10 ...
## $ vendedores: int 322 317 319 323 327 328 325 326 330 334
##    ...
## $ unidades : num 44.2 44.3 44.4 43.4 42.8 44.3 44.4 44.8
##    44.4 43.1 ...
```

Leímos este segundo archivo correctamente, de hecho este último objeto leído es igual al primero. Podemos emplear una prueba lógica utilizando un operador de relación como lo vimos en la sección 4.2.2 de la siguiente manera:

```
# comparando los elementos de los
# dos objetos leídos
employee == employee2
```

Podemos emplear los mismo procedimientos por menús que vimos anteriormente para leer estos archivos en formato `.csv`.

8.1.3 Leyendo archivos de Excel

Para cargar datos de un archivos de Excel podemos emplear diferentes paquetes. Tal vez el más sencillo es el paquete *readxl* (Wickham & Bryan, 2019) De ese paquete usaremos la función **read_excel()**, la cual permite cargar de manera similar a los archivos planos archivos `.xls` y `.xlsx`. Una de las diferencias de esta función es que permite especificar de que hoja en específico queremos leer los datos.

Leamos el archivo *Employee.xlsx*[4] que tiene la misma estructura de los dos archivos anteriores. La única diferencia es que los datos están en la Hoja2, empieazan en la celda `C2` y terminan en la celda `E62` (abre el archivo de Excel para verificar esto). En este caso podemos especificar la hoja en la que están los datos que queremos leer (argumento **sheet**) y el rango de celdas que queremos leer (argumento **range** que se expresa entre comillas).

```
#instalar el paquete si no lo tienes aún instalado
 #install.packages("readxl")
# cargando el paquete
library(readxl)
# leyendo datos del archivo .xlsx que está en hoja2
employee3 <- read_excel("employee.xlsx", sheet = 2,
```

[4]El archivo se puede encontrar en la página web del libro: http://www.icesi.edu.co/editorial/empezando-usar

```
                        range = "C2:E62")
# verificando la clase del objeto
class(employee3)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
# inspeccionando los primeros 5 datos
head(employee3, 5)
```

```
## # A tibble: 5 x 3
##     mes vendedores unidades
##   <dbl>      <dbl>    <dbl>
## 1     1        322     44.2
## 2     2        317     44.3
## 3     3        319     44.4
## 4     4        323     43.4
## 5     5        327     42.8
```

```
# inspeccionando los últimos 3 datos
tail(employee3, 2)
```

```
## # A tibble: 2 x 3
##     mes vendedores unidades
##   <dbl>      <dbl>    <dbl>
## 1    59        392     49.2
## 2    60        396     48.1
```

```
# inspeccionando la clase de cada variable
str(employee3)
```

```
## tibble [60 x 3] (S3: tbl_df/tbl/data.frame)
## $ mes : num [1:60] 1 2 3 4 5 6 7 8 9 10 ...
## $ vendedores: num [1:60] 322 317 319 323 327 328 325 326
##    330 334 ...
## $ unidades : num [1:60] 44.2 44.3 44.4 43.4 42.8 44.3 44.4
##    44.8 44.4 43.1 ...
```

Así como en el caso de los archivos planos, los archivos de Excel pueden ser cargados empleados los menús de RStudio. El procedimiento es muy similar.

8.2 Extraer Datos de Paquetes

Como se discutió en el Capítulo 7, los paquetes de R contiene *datasets* (base de datos) que se emplean normalmente para mostrar como funciona el paquete. Otros paquetes son bases de datos, como por ejemplo, el paquete *gapminder* (Bryan, 2017). El paquete *gapminder* tiene datos para todos los países de población, pobreza y esperanza de vida entre otras variables. Un listado exhaustivo de paquetes que son exclusivamente bases de datos se puede encontrar en el siguiente enlace: https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html

Otros paquetes nos brindan una interfaz para descargar datos de entidades como el Banco Mundial (paquete *WDI* (Arel-Bundock, 2021)) o el Fondo Monetario Internacional[5] (paquete *imfr* (Gandrud, 2020)).

Veamos rápidamente cómo traer al *workspace* objetos de clase **data.frame** que vienen integrados en los paquetes. El primer paso es cargar el paquete del que se quieren extraer los datos. Sino sabemos con exactitud cuáles son los datos disponibles, o el nombre de los objetos con las bases de datos, podemos emplear la función **data()** sin ningún argumento. Esto desplegará una ventana con el listado de todas las bases de datos disponibles en los paquetes cargados y el *Core* de R.

```
#instalar el paquete si no lo tienes aún instalado
#install.packages("gapminder")
# cargando el paquete gapminder
 library(gapminder)
# listando las bases de datos disponibles
data()
```

O podemos ver los datos que hay en un solo paquete; por ejemplo, el paquete *gapminder*:

```
#
# listando las bases de datos disponibles
# en un paquete específico
data(package = 'gapminder')
```

En el Cuadro 8.1 se presenten las bases de datos presentes en el paquete **gapminder**.

[5]No discutiremos este tipo de paquetes pero puedes mirar su ayuda. Encontrarás que es relativamente fácil emplearlos y bajar datos de esos repositorios.

Tabla 8.1. Basses de datos en el paquete gapminder

Nombre	Descripción
continent_colors	Gapminder color schemes.
country_codes	Country codes
country_colors	Gapminder color schemes.
gapminder	Gapminder data.
gapminder_unfiltered	Gapminder data, unfiltered.

Una vez identifiquemos el objeto que tiene la base de datos que queremos, la podemos cargar al *workspace* empleando la misma función **data()**, empleando como argumento el nombre de la base. Si necesitamos ayuda para conocer a detalle de la base de datos, podemos emplear la función **help()** empleando como argumento el nombre de la base. Por ejemplo, supongamos que queremos cargar la base de datos que está en el objeto `gapminder` del paquete del mismo nombre pero no sabemos qué contiene. Por eso es importante consultar la ayuda de la siguiente manera:

```
# mirando la descripción de los datos en el objeto
help(gapminder)
```

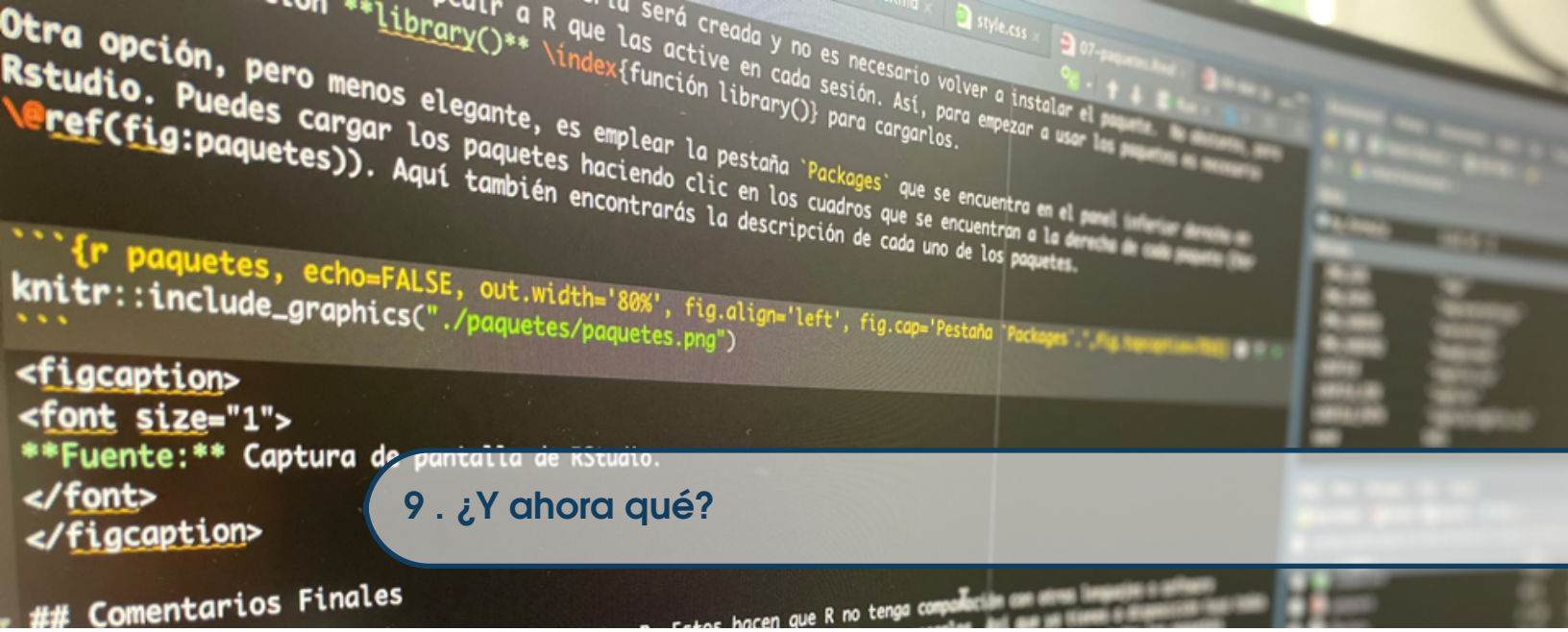
Una vez constatamos que esta base tiene la información que deseamos, podemos cargarla de la siguiente manera:

```
# cargando la base de datos al ws
data("gapminder")
# clase del objeto cargado
class(gapminder)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

8.3 Comentarios Finales

En este capítulo estudiamos cómo cargar bases de datos desde archivos y desde paquetes. Ahora estás listo para procesar tus datos y analizarlos. En el último capítulo discutiremos rápidamente cuáles pueden ser tus siguientes pasos en este universo de R.



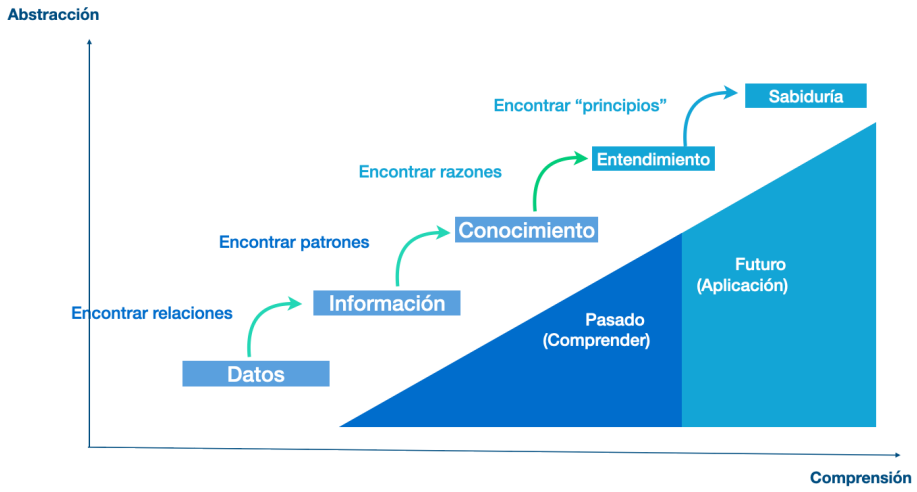
9 . ¿Y ahora qué?

A lo largo del libro discutimos cómo empezar a usar la herramienta R. Estudiamos desde su descarga, instalación, la descripción de sus componentes, hasta entender la sintaxis básica del lenguaje, la creación de los objetos y sus diferentes clases. Finalmente, vimos como cargar bases de datos en diferentes formatos.

R es una herramienta versátil que nos permite encontrar paquetes y material específico en diferentes disciplinas, en el siguiente enlace puedes encontrar información de estas diferentes temáticas: https://cran.r-project.org/web/views/. Desde epidemiología, psicometría, econometría y estadística, hasta paquetes relacionados con genética y métricas de medio ambiente, pasando por algoritmos de inteligencia artificial. En el universo de R encontrarás paquetes que te permitirán realizar cualquier actividad que te permita sacarle provecho a datos cuantitativos y cualitativos.

Después de cargar los datos nuestro objetivo es transformarlos en información, conocimiento, entendimiento y ojalá sabiduría (Ver Figura 9.1 y Alonso Cifuentes y Quintero Villarreal (2021) para una mayor discusión). Cuando encontramos relaciones en los datos generamos información que permite comprender lo ocurrido en el pasado. La información implica encontrar relaciones en un contexto de manera que se proporciona una historia útil detrás de los datos.

Figura 9.1. Pirámide del Conocimiento



Fuente: Adaptación de Alonso Cifuentes y Quintero Villarreal (2021).

Si además encontramos patrones en el comportamiento de los datos, generamos conocimiento sobre lo ocurrido en el pasado. El conocimiento es información que ha sido comprendida en su contexto de manera que se logran encontrar patrones de comportamiento. Esto nos permitirá encontrar aplicaciones a ese entendimiento.

Si adicionalmente se logra explicar el cómo y el por qué de algo o se encuentra una visión y comprensión de los datos generamos entendimiento. Finalmente, en algunas ocasiones se encuentra sabiduría, si el entendimiento puede enmarcarse en una estructura. Así se ubica al conocimiento en un marco teórico que permite aplicar el entendimiento a situaciones diferentes y no de manera intuitiva.

Para superar la etapa de tener datos y subir en esa pirámide de conocimiento, tenemos a nuestra disposición diferentes herramientas estadísticas y algoritmos de aprendizaje de máquina que se pueden en general agrupar en ocho categorías. Estas tareas son:

- Resumir
- Visualizar
- Agrupar o clustering
- Clasificar
- Detectar excepciones
- Asociar
- Estimar regresiones
- Pronosticar

Resumir implica simplificar la representación de los datos para generar información. Podemos emplear R para calcular estadísticas descriptivas y resumir grupos de variables empleando técnicas como el análisis de componentes principales[1].

La **visualización** facilita la comprensión y el descubrimiento de los datos por medio de gráficos. En muchas ocasiones una imagen dice mas que mil palabras. Las visualizaciones nos permiten tanto entender nuestros datos antes de analizarlos como comunicar los resultados de un análisis.

R tiene el paquete **ggplot2**(Wickham, 2016) que permite realizar visualizaciones de gran calidad[2]. En las Figuras 9.2, 9.3 y 9.4 se muestran diferentes ejemplos de visualizaciones realizadas en R. Por ejemplo, la Figura 9.2 presenta el resumen de los resultados tras aplicar una valoración de desempeño en práctica a un número grande de practicantes. Las filas representan a cada uno de los estudiantes, y las columnas cada una de las preguntas del instrumento (formulario) utilizado. Los colores simbolizan la valoración en una escala cualitativa. Nota que esta visualización permite resumir muchos datos y permite brindar información sobre el resultado.

La Figura 9.3 presenta una visualización de la distribución de la esperanza de vida al nacer en cada uno de los países disponibles en la base de datos **gapminder** del paquete con el mismo nombre (para una discusión de esta base de datos ver la sección 8.2) y empleando el paquete **ggstatsplot** (Patil, 2021). Esta visualización permite ver la espera de vida al nacer en el año 2007 (cada punto es un país). Además se observa cómo se distribuye la esperanza de vida al nacer por medio de gráficos de violines[3]. Pero la visualización tiene además información de pruebas estadísticas para comparar las medias de los diferentes continentes.

Y también podemos visualizar información cualitativa de textos como se muestra en la nube de palabras[4] de la Figura 9.4. En dicha nube se resumen los discursos del Presidente Juan Manuel Santos y Rodrigo Londoño (conocido como Timochenco), el día 24 de noviembre de 2016 cuando se firma el Acuerdo de Paz entre el Estado Colombiano y

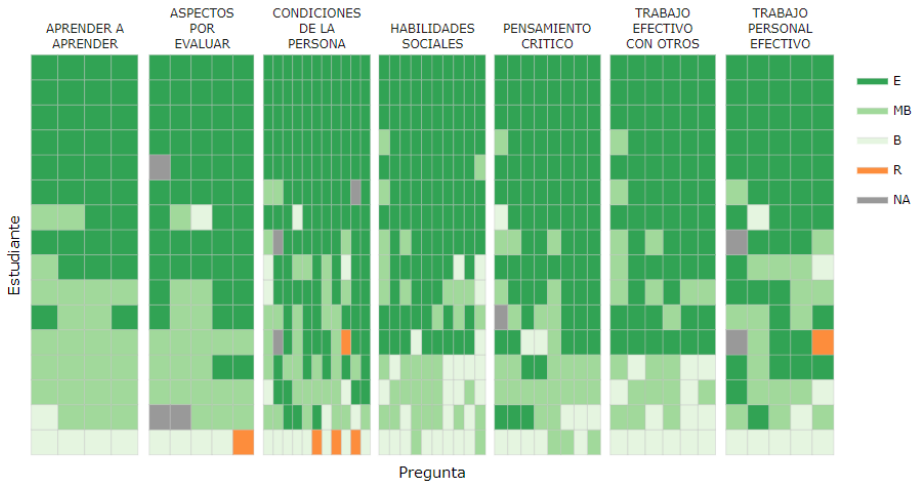
[1] En el siguiente enlace podrás encontrar una breve introducción a la lógica del PCA: https://youtu.be/-EQFB_iiqd4. Si quieres ver cómo implementar esta técnica en R puedes encontrar una introducción en Alonso Cifuentes (2020a) .

[2] En el siguiente enlace encontrarás un video con una breve introducción al paquete **ggplot2**: https://youtu.be/lVkn7spjZ1Q. Si deseas una introducción algo mas profunda a este paquete puedes consultar Alonso Cifuentes y González (2012) y Alonso Cifuentes y Montenegro (2012).

[3] Para una breve explicación de como interpretar los gráficos de violines puedes ver el siguiente video: https://youtu.be/FQZzjb2LiK8

[4] En el siguiente enlace puede observar una breve introducción a las nubes de palabras: https://youtu.be/EvDAbWPMqiQ . Y para una introducción de cómo construir las nubes de palabras en R puedes leer Alonso Cifuentes (2020b) .

Figura 9.2. Visualización de todas las preguntas del intrumento de valoración de estudiantes en práctica de la Universidad Icesi



Fuente: Cienfi.

las Fuerzas Armadas Revolucionarias de Colombia - Ejército del Pueblo (FARC-EP).

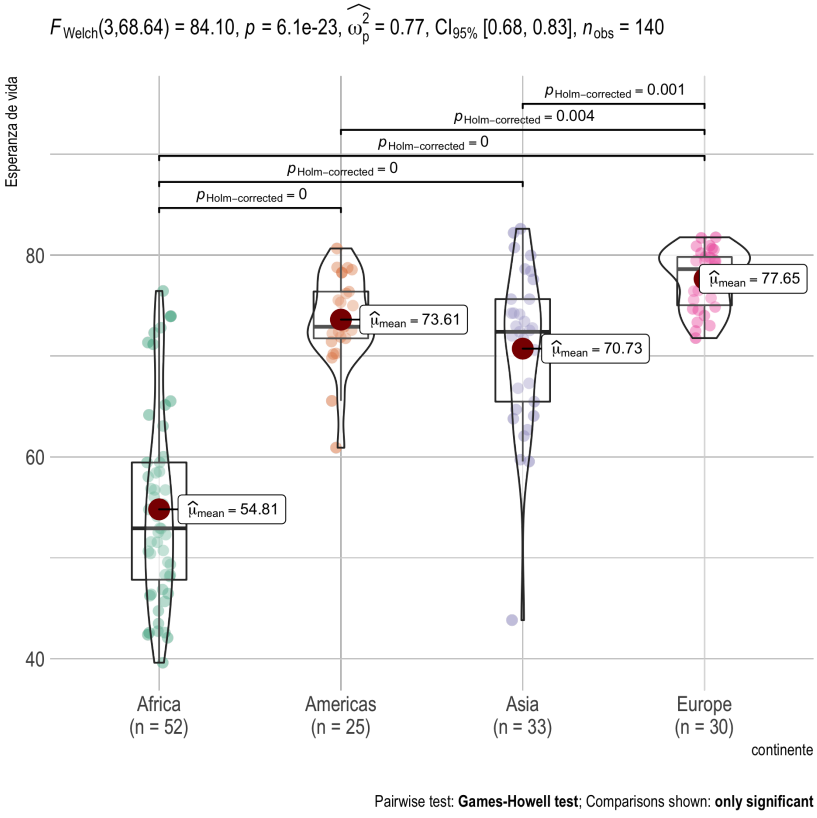
R también permite la construcción de visualizaciones interactivas[5] y animadas como los que se presentan en la Figuras 9.5 y 9.6, respectivamente (estas figuras solo son interactivas en la versión html del libro).

La Figura 9.5 presenta los datos del paquete **gapminer** del PIB per cápita (en escala logaítmica) y la esperanza de vida alrededor del mundo para 2007. Este tipo de visualizaciones le permiten al usuario interactuar con los datos. ¡Te invito a que juegues con el gráfico! se pueden apagar los continentes y hacer *zoom*. Pasa el cursor por encima de un punto para ver la información. (La interacción solo funciona en la versión Web del libro).

A diferencia de la Figura 9.5 la 9.6 no es interactiva , pero si permite ver la evolución en el tiempo de la misma relación representada anteriormente. Esta figura animada fue creada con el paquete **gganimate** (Pedersen & Robinson, 2020). La animación solo funciona en la versión web del libro. Aquí podemos ver cómo, por continente, han mejorado tanto la esperanza de vida como el PIB per cápita.

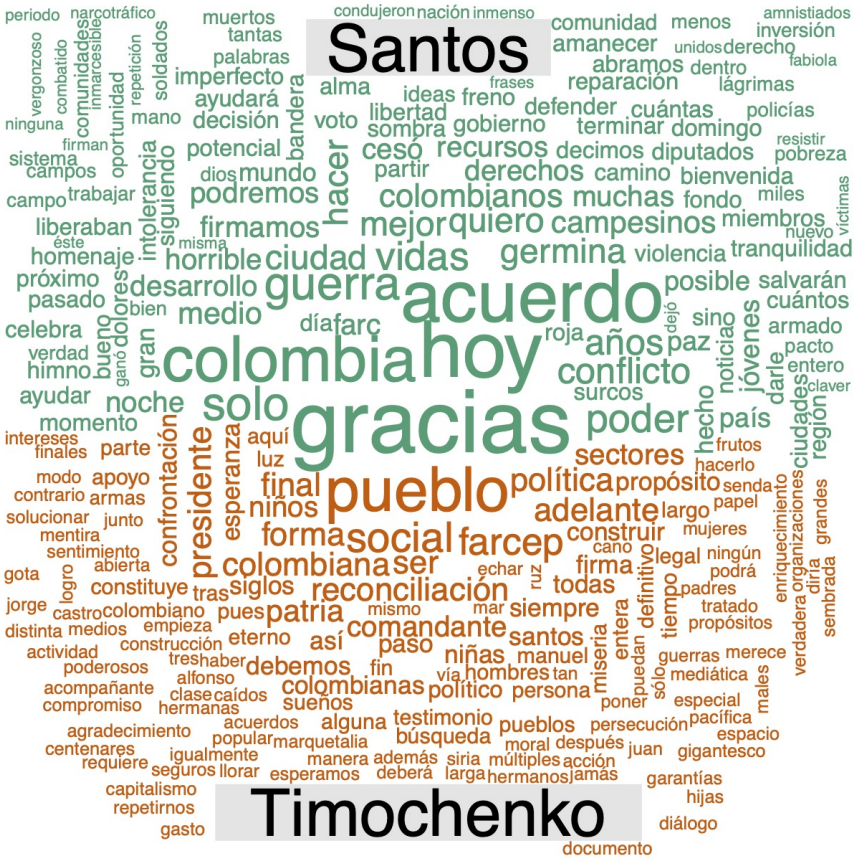
[5]Puedes encontrar una breve introducción a la construcción de gráficos interactivo con el paquete **plotly** (Sievert, 2020) en el siguiente enlace: https://youtu.be/EWjxic2ce9g .

Figura 9.3. Gráfico de violines y prueba de comparación de media por continentes para la esperanza de vida al nacer alrededor del mundo (2007)



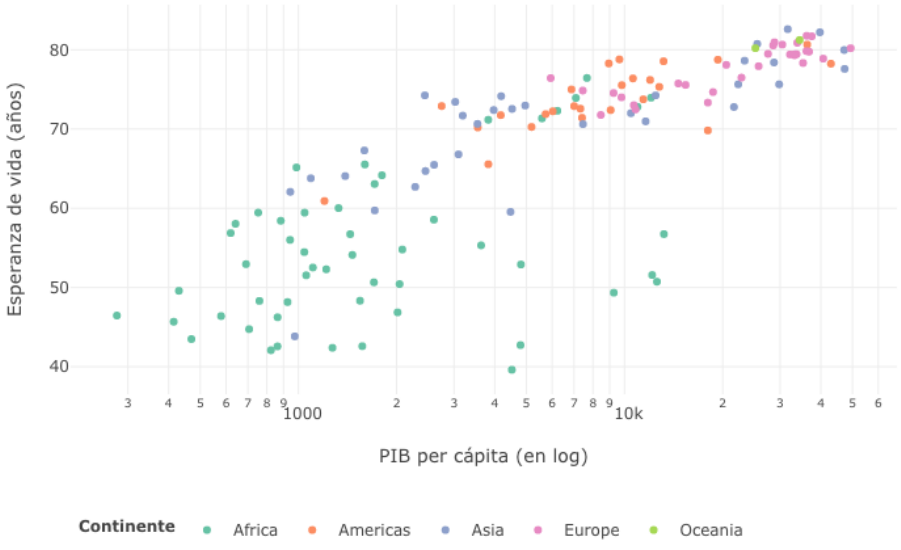
Fuente: Datos de Gapminder y cálculos propios empleando el paquete ggstatsplot.

Figura 9.4. Nube de palabras de los discursos al momento de la firma de del Acuerdo de Paz entre el Estado Colombiano y las FARC-EP (24 de noviembre de 2016)



Fuente: Tomado de Alonso Cifuentes (2020b).

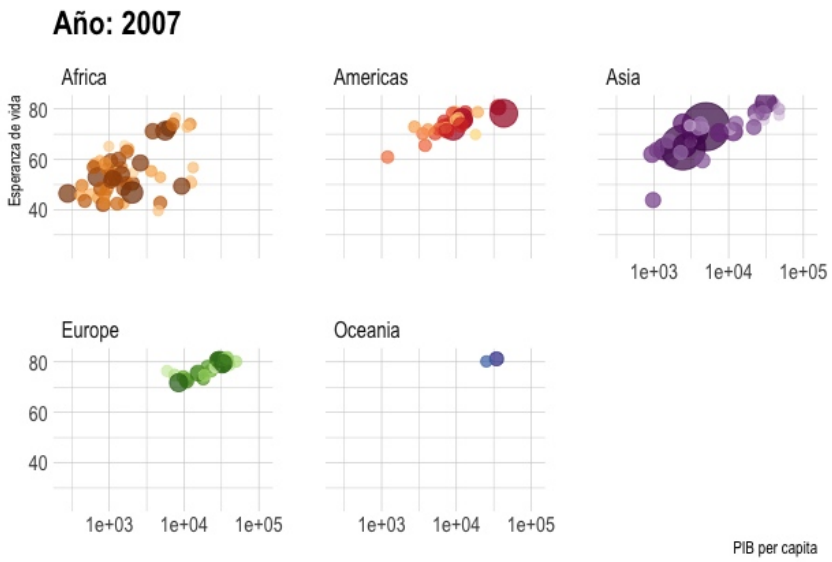
Figura 9.5. Gráfico interactivo de la relacion del PIB per cápita y la esperanza de vida al nacer alrededor del mundo (2007)



Fuente: Datos de Gapminder y cálculos propios empleando el paquete plotly

Podríamos llenar hojas y hojas con visualizaciones interesantes, pero ya te hiciste una idea de las potencialidades de emplear visualizaciones para comunicar mensajes.

Figura 9.6. Visualización animada de la evolución del PIB per cápita y la esperanza de vida alrededor del mundo (1952-2007).



Fuente: Datos de Gapminder y cálculos propios empleando el paquete gganimate

Regresando a las tareas que podemos realizar con los datos, el **clustering**[6] parte de una muestra para encontrar grupos de elementos similares. Por ejemplo, a partir de conjunto de clientes podemos, por medio de un modelo de *clustering*, crear tres grupos de clientes de acuerdo con ciertas características. Esto se conoce en el mercadeo como segmentación de clientes.

La tarea de **clasificación**[7] tiene como finalidad predecir la categoría de un individuo. Por ejemplo, en algunas situaciones se deseará determinar si un nuevo cliente comprará o no nuestro producto. En este caso las categorías son compra o no compra. Otro tipo de preguntas que

[6] Puedes encontrar una breve introducción a la tareas de clustering o agregación en el siguiente video: https://youtu.be/z0LX3sBSuXg .

[7] Puedes encontrar una breve introducción a la tarea de clasificación en el siguiente video: https://youtu.be/0K7ryP0uKGo .

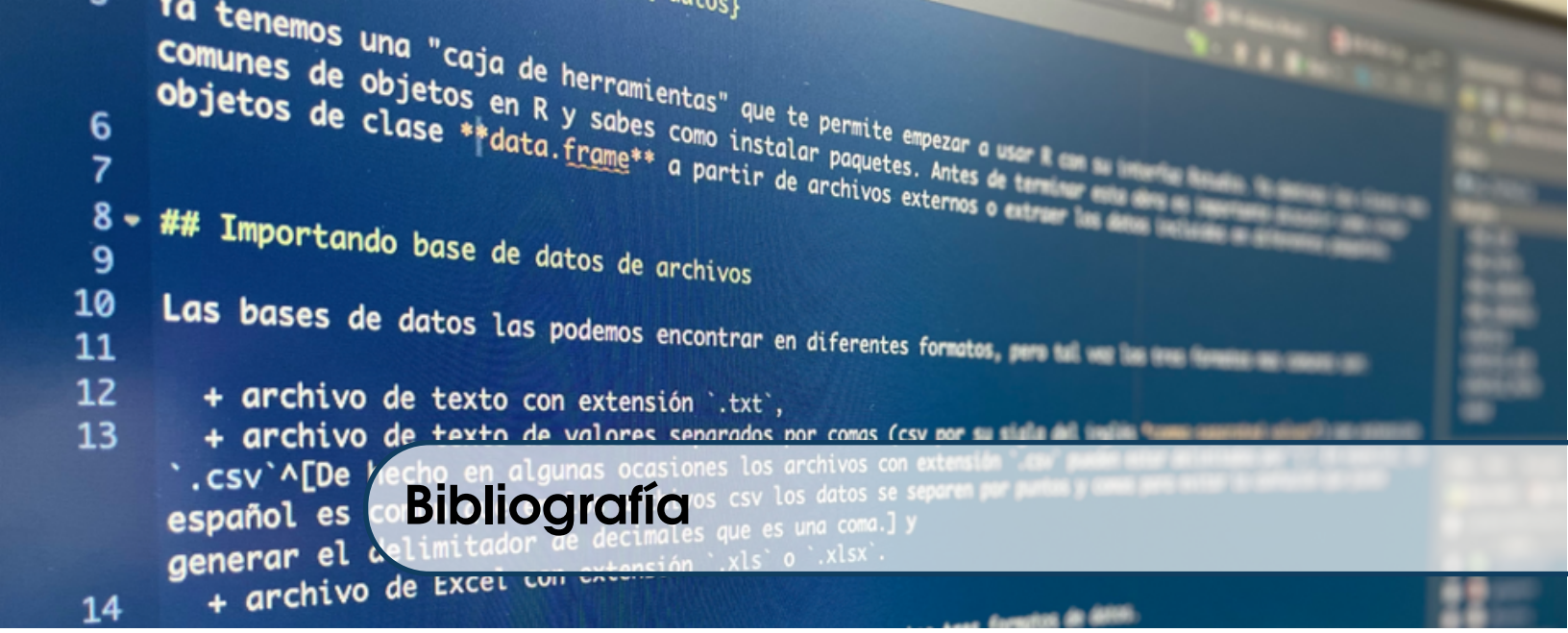
puede resolver esta tarea son ¿se irá el cliente?, ¿pagará el crédito? y, ¿será el individuo un buen *match* para la posición?

La tarea de **detección de excepciones** tiene como objetivo encontrar individuos con características o comportamiento diferentes. La tarea de encontrar **asociaciones** busca reglas de coocurrencia de productos en diferentes canastas. Es decir, busca cuáles productos son comprados regularmente al mismo tiempo que otros para poder sugerir composición de canastas. Estos modelos intentan encontrar la estructura de los datos sin la necesidad de enseñarle al algoritmo cuáles son las coocurrencias.

La tarea de **estimar regresiones** implica encontrar relaciones entre muchas variables y una variable cuantitativa de interés. Esto es tanto para entender qué variables están asociadas a un fenómeno, como para simular el comportamiento en diferentes escenarios.

Finalmente, la tarea de generar **pronósticos** implica predecir el comportamiento futuro de una variable cuantitativa. Para esto se emplean los patrones de comportamiento pasados para extrapolarlas al futuro.

Como ves el camino es largo y hay muchas tareas que podrás estudiar a tu ritmo. Todas podrán ser efectuadas con R. Esperamos que esta libro te motive a continuar tu camino de aprendizaje y unirte a la gran comunidad de R. En este universo de R, ¡la imaginación es el límite!



Bibliografía

Alonso Cifuentes, J. C. (2020a). *Herramientas del Business Analitycs en R: Análisis de Componentes Principales para resumir variables* (inf. téc.). Universidad Icesi. https://www.researchgate.net/publication/341829708_Herramientas_del_Business_Analitycs_en_R_Analisis_de_Componentes_Principales_para_resumir_variables

Alonso Cifuentes, J. C. (2020b). Una introducción a la construcción de Word Clouds (para economistas) en R. *Economics Lecture Notes*, (9), 1-28. https://www.researchgate.net/publication/341829699_Una_introduccion_a_la_construccion_de_Word_Clouds_para_economistas_en_R

Alonso Cifuentes, J. C., & González, A. (2012). Ggplot: gráficos de alta calidad. *Apuntes de Economía*, (33), 29. https://www.researchgate.net/publication/323202960_Ggplot_graficos_de_alta_calidad

Alonso Cifuentes, J. C., & Jaramillo, L. E. (2011). R-Commander: una puerta al análisis estadístico. *Apuntes de economía; No. 10*.

Alonso Cifuentes, J. C., & Montenegro, S. (2012). Visualización de información georeferenciada en ggplot2. *Apuntes de Economía*, (35), 16. https://www.researchgate.net/publication/306259435_Visualizacion_de_informacion_georeferenciada_en_ggplot2

Alonso Cifuentes, J. C., & Quintero Villarreal, L. M. (2021). Guía de buenas prácticas para la mitigación del riesgo de modelo de analítica.

Arel-Bundock, V. (2021). *WDI: World Development Indicators and Other World Bank Data* [R package version 2.7.4]. https://CRAN.R-project.org/package=WDI

Bryan, J. (2017). *gapminder: Data from Gapminder* [R package version 0.3.0]. https://CRAN.R-project.org/package=gapminder

Fox, J. (2017). *Using the R Commander: A Point-and-Click Interface for R*. Chapman; Hall/CRC Press. http://socserv.mcmaster.ca/jfox/Books/RCommander/

Gandrud, C. (2020). *imfr: Download Data from the International Monetary Fund's Data API* [R package version 0.1.9.1]. https://CRAN.R-project.org/package=imfr

Patil, I. (2021). Visualizations with statistical details: The 'ggstatsplot' approach. *Journal of Open Source Software*, *6*(61), 3167. https://doi.org/10.21105/joss.03167

Pedersen, T. L., & Robinson, D. (2020). *gganimate: A Grammar of Animated Graphics* [R package version 1.0.7]. https://CRAN.R-project.org/package=gganimate

R Core Team. (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. https://www.R-project.org/

Sievert, C. (2020). *Interactive Web-Based Data Visualization with R, plotly, and shiny*. Chapman; Hall/CRC. https://plotly-r.com

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wickham, H., & Bryan, J. (2019). *readxl: Read Excel Files* [R package version 1.3.1]. https://CRAN.R-project.org/package=readxl

Índice alfabético

Editorial Universidad Icesi

UNIVERSIDAD ICESI

Este libro tiene como propósito presentar una primera aproximación a R, que va desde su instalación, hasta presentar tips de buenas prácticas para escribir código. Si eres nuevo en el universo de R, este libro será un buen punto de arranque. En el caso de que hayas tomado un curso de R previamente y eres un usuario independiente de R, tal vez este libro no te aporte nuevos conocimientos, pero será una herramienta de consulta de algunos conceptos básicos.

Herramientas del BIG DATA y analytics