

Introducción al Análisis de Datos con R

Agustin Alesso

Patricia Acetta

Tabla de contenidos

Bienvenidos!	4
Objetivos del curso	4
Contenidos	4
1 Introducción	6
1.1 La necesidad de analizar datos	6
1.2 ¿Cómo es el flujo de análisis de datos?	7
1.3 ¿Que es un análisis reproducible?	8
2 Comenzando con R	10
2.1 ¿Qué es R y RStudio ?	10
2.2 ¿Cómo instalar R y RStudio ?	14
2.2.1 Instalación de R	14
2.2.2 Instalación de RStudio	15
2.3 Primera sesión	18
2.3.1 La consola	21
2.3.2 El script	21
2.3.3 Directorio de trabajo y proyectos	23
2.3.4 Ayuda!!!	26
3 Aspectos básicos de R	27
3.1 Operadores matemáticos y lógicos	27
3.2 Variables y objetos	29
3.2.1 Vectores	30
3.2.2 Funciones y argumentos	31
3.2.3 Creando funciones	32
3.3 Tipos de datos	34
3.3.1 Numéricos (numeric)	34
3.3.2 Texto (character)	35
3.3.3 Lógicos (logic)	35
3.3.4 Factores (factor y ordered)	36
3.3.5 Otros tipos de datos	37
3.4 Estructura de datos	38
3.4.1 Matriz (matrix)	38
3.4.2 Listas (list)	39

3.4.3	Hoja de datos (<code>data.frame</code>)	39
4	Paquetes de R	42
4.1	¿Qué son los paquetes?	42
5	Importar datos en R	44
5.1	Función nativa para importar datos	44
5.2	Paquetes para importar datos	46
5.2.1	<code>readr</code>	46
5.2.2	<code>rio</code>	47
5.3	Formas de importar datos	48
5.3.1	Desde la consola (recomendado)	48
5.3.2	Desde el importador de datos de RStudio	49
5.3.3	Desde el portapapeles	50
	References	51

Bienvenidos!

¡Bienvenidos al curso de *Introducción al Análisis de Datos con R*!

Objetivos del curso

- Reconocer al lenguaje R, y su entorno de desarrollo RStudio, como herramienta para el procesamiento y análisis de datos.
- Identificar y explicar los tipos y estructuras de datos representados en R.
- Identificar y emplear funciones y librerías útiles para el procesamiento, visualización y análisis de datos en R.
- Aplicar técnicas estadísticas básicas para analizar datos en R.
- Crear visualizaciones efectivas para comunicar resultados.
- Aplicar los principios de reproducibilidad para procesar y analizar datos y comunicar resultados

Contenidos

- **Unidad 1:** ¿Qué es R y RStudio? Instalación de R y RStudio. Características RStudio: menús, paneles, etc. Sintaxis de R, convenciones y símbolos de R. Sistema de librerías: instalación y carga. Sistema de ayuda. Tipos de datos: numérico, carácter, lógico. Funciones. Estructuras de datos: vectores, listas, hoja de datos (data frame). El flujo de trabajo en un proyecto de análisis de datos: proyectos, scripts y notebooks. Análisis reproducible.
- **Unidad 2:** Manipulación de datos con R. Importación y exportación de datos. Librería tidyverse. Gramática de manipulación de datos. Limpieza, normalización, combinación y resumen de datos numéricos y categóricos.
- **Unidad 3:** Visualizaciones, gráficos. Nociones generales sobre tipos de gráficos y sus usos. Introducción a ggplot2. La gramática de gráficos: estéticas, geometrías, escalas, temas, etc. Gráficos condicionales y multi-paneles.

- **Unidad 4:** Programación literaria. Generación de informes o reportes con Quarto/RMarkdown.

Sin más preámbulos... comencemos!

1 Introducción

1.1 La necesidad de analizar datos

Hoy en día los datos abundan y están en todos lados. Esto desafía nuestra capacidad para analizarlos y extraer significado de los mismos para tomar decisiones.

La ciencia de datos (data science) es una nueva disciplina que emerge de la combinación de disciplinas existentes (diagrama) y permite convertir datos sin procesar en entendimiento, comprensión y conocimiento.

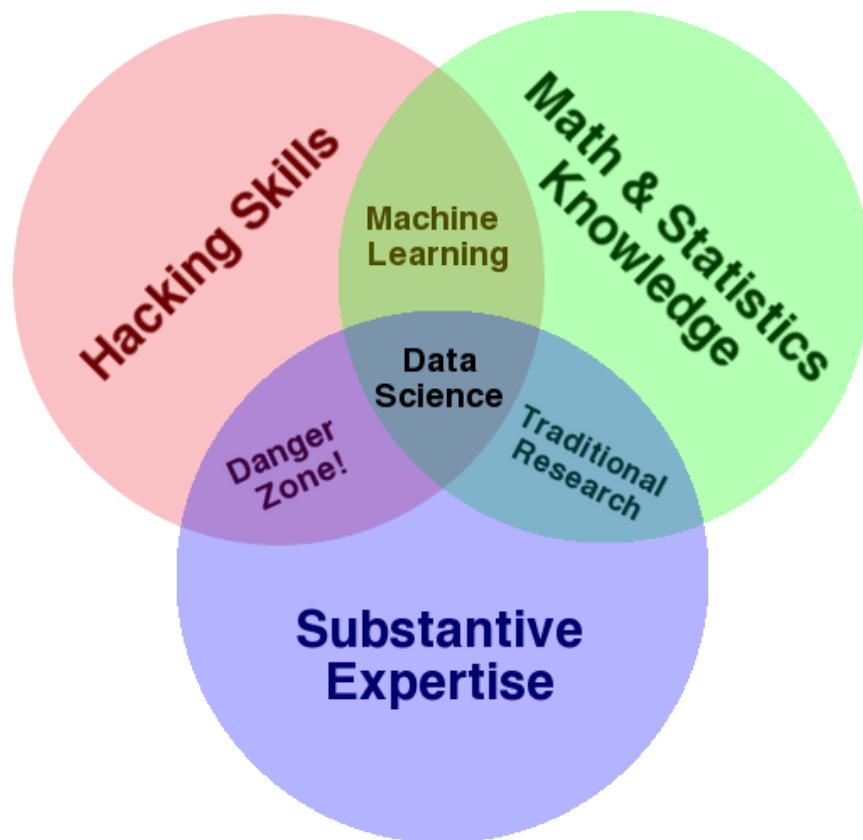


Figura 1.1: Diagrama de Venn de la Ciencia de Datos según Conway

Para poder analizar los datos de manera efectiva, es necesario tener conocimientos de disciplinas como ciencias de la computación (programación y más), matemática y estadística. Pero también tenemos que tener conocimientos para lograr el entendimiento del problema en estudio. La combinación de estas áreas nos lleva al concepto de Ciencia de Datos.

1.2 ¿Cómo es el flujo de análisis de datos?

Si bien el análisis de datos es un proceso no lineal que varía en cada situación, existe consenso en cuanto a las principales actividades que se deben desarrollar. El siguiente gráfico resume el flujo de trabajo o *workflow*.

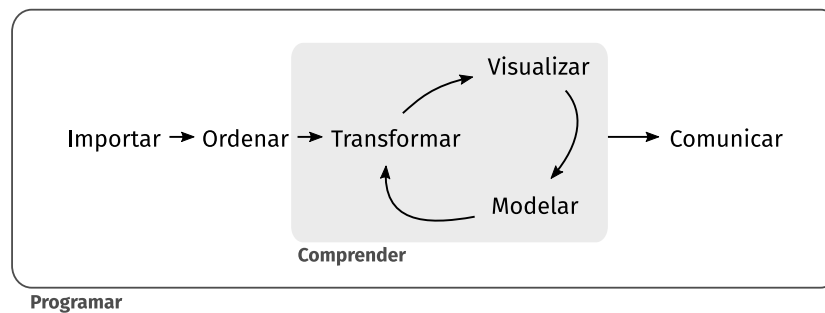


Figura 1.2: Diagrama de Venn de la Ciencia de Datos según Conway

Todo análisis comienza con importar datos a la herramienta que estemos utilizando, **R** en este caso. Los datos pueden estar almacenados de diferentes formas, como archivos, dentro de una base de datos o una API (*application programming interface*). En el curso veremos distintas formas de importar datos dentro de **R** en un formato compatible para su análisis.

Salvo excepciones, una vez importados los datos, hay que ordenarlos de alguna forma que permita realizar el análisis que queremos. En la mayoría de los casos necesitaremos que los datos estén almacenados de manera **rectangular** tal que cada columna represente una variable o atributo de los datos y cada fila una o más observaciones o sujetos (formato *apaisado*).

pais	año	casos	poblacion
Afganistán	1999	745	19987071
Afganistán	2000	2666	20595360
Brasil	1999	37737	172006362
Brasil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

variables

pais	año	casos	poblacion
Afganistán	1999	745	19987071
Afganistán	2000	2666	20595360
Brasil	1999	37737	172006362
Brasil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

observaciones

pais	año	casos	poblacion
Afganistán	1999	745	19987071
Afganistán	2000	2666	20595360
Brasil	1999	37737	172006362
Brasil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

valores

A veces cuando algunas columnas representan valores de una misma variable hay que modificar este formato y *pivotar* a un formato *longitudinal*.

Con los datos ordenados podemos concentrarnos en comprender su estructura y empezar a jugar con ellos. Esta etapa no es para nada lineal y generalmente implica un proceso iterativo donde los datos se visualizan, se transforman y se modelan.

La transformación de los datos implica crear subconjuntos o combinar con otros datos, alterar las variables existentes (e.g. cambiar unidades) o generar nuevos atributos combinando información de otros, resumir los datos agrupando observaciones, etc. El manejo de datos de datos (*data wrangling*) es la combinación de técnicas de ordenamiento y transformación.

La visualización es el arte de convertir los datos de forma tabular en gráficos o diagramas donde los distintos atributos de los datos se relacionan a características gráficas tales como ejes, colores, formas, etc. Una buena visualización permitirá revelar patrones inesperados, confirmar algunas preguntas o sugerir nuevas. Es muy útil para comunicar.

El modelado es una forma de cuantificar lo podemos ver en una visualización. Al igual que las visualizaciones, los modelos son abstracciones de los datos y nos permiten resumir la variación quedándonos con la generalidad. Los modelos son útiles para contestar preguntas sobre los datos. Lo clave es hacer la pregunta correcta! Por otro lado los modelos son tan buenos como los datos de entrada y los supuestos utilizados.

El último paso de un proyecto de ciencia de datos es la comunicación. Esta etapa resume todo nuestro trabajo y determina como podemos transmitir nuestras conclusiones y descubrimientos a otros que no participaron del análisis.

Finalmente, todas y cada una de las etapas de este proceso se desarrollan con la ayuda de la programación. No es necesario ser un hacker, hay que saber pensar como un programador y saber programar ayuda a automatizar tareas.

1.3 ¿Que es un análisis reproducible?

En las Ciencias Experimentales, poder replicar los experimentos es un componente muy importante del método científico ya que le da validez a los descubrimientos. Es por ello que en los trabajos científicos, una sección importante es la de *Materiales y Métodos* donde se describen los pasos que se deben dar para poder *replicar* de los resultados del estudio. Estos pasos incluyen las instrucciones para replicar el experimento físico y de como *reproducir* el análisis de los datos que llevó a las conclusiones.

Así la **replicabilidad** implica que, siguiendo los *materiales y métodos* un experimento independiente puede llegar a los mismos resultados con datos distintos. En cambio la **reproducibilidad** significa que con la misma persona u otra persona con los mismos datos llegue a los mismos resultados, es decir, que pueda reproducir el análisis.

La Ciencia de Datos la **reproducibilidad computacional** es clave ya que en el procesamiento de datos se toman una serie de decisiones que afectan el resultado. ¿Qué significa esto? Dada una serie de datos de entrada, el flujo de trabajo que se aplica para *importar*, *ordenar*, *transformar*, *visualizar* y *modelar* los datos, es decir, convertirlos en información, debe estar correctamente **documentado** para que cualquier persona pueda entender la lógica y eventualmente reproducir los mismos resultados.

R, como cualquier lenguaje de programación, es una herramienta ideal para aplicar el concepto de **reproducibilidad** ya que mediante en el código o *script* quedan plasmados todos los pasos que se aplicaron en el análisis de datos. Más aun, el uso del paradigma de **programación literaria** permite generar documentos donde se narran todos los pasos y los resultados obtenidos.

2 Comenzando con R

En esta sección vamos a ver que es cómo instalar y cómo empezar a usar R y RStudio creando nuestro proyecto de análisis reproducible de datos.

2.1 ¿Qué es R y RStudio?

R es un lenguaje y entorno para el procesamiento, visualización y análisis estadístico de datos. Fue creado en 1993 por R. Gentleman y R. Ihaka, ambos científicos del Departamento de Estadística de la Universidad de Auckland (Nueva Zelanda). Actualmente su desarrollo y mantenimiento está a cargo del R Core Team (2023). El sitio oficial del proyecto es www.r-project.org.

Hoy en día, **R** es la *lingua franca* del procesamiento y análisis de datos, tanto en el ámbito académico como comercial dado que es gratuito, multiplataforma, de código abierto (*open source*, liberado con licencia GNU/GPL). Esto y el ecosistema de paquetes contribuidos por la comunidad de usuarios lo convierte en un software muy potente ya que expresa el estado del arte de los métodos estadísticos.

La flexibilidad y potencia de **R** se basa en su interfaz de comandos (CLI, del inglés *command line interface*) que permite la ejecución de comandos de manera interactiva (en consola) o estructurada mediante scripts.

Existen algunos desarrollos de interfaces gráficas (GUIs, del inglés *graphical user interface*), e.g. RCommander, Deducer, etc., que ofrecen la posibilidad de, mediante menús y botones dedicados, ejecutar algunos análisis relativamente simples minimizando la necesidad de escribir código.

Los entornos de desarrollo integrados (IDE por sus siglas en inglés *integrated development environments*) ofrecen un enfoque intermedio: los menús o funciones asistentes facilitan algunas tareas generales (abrir archivos, carga de datos, exportar gráficos y resultados, etc.) pero dejan la escritura del código y ejecución del análisis estadístico en manos del usuario. Entre estas alternativas se destaca **RStudio** desarrollado por la empresa [posit](https://posit.co/) el cual también es de código abierto (licencia GNU/GPL), multiplataforma y ofrece una versión gratuita.



[\[Home\]](#)

Download

[CRAN](#)

R Project

[About R](#)

[Logo](#)

[Contributors](#)

[What's New?](#)

[Reporting Bugs](#)

[Conferences](#)

[Search](#)

[Get Involved: Mailing](#)

[Lists](#)

[Developer Pages](#)

[R Blog](#)

R Foundation

[Foundation](#)

[Board](#)

[Members](#)

[Donors](#)

[Donate](#)

Help With R

[Getting Help](#)

Documentation

[Manuals](#)

[FAQs](#)

[The R Journal](#)

[Books](#)

[Certification](#)

[Other](#)

Links

[Bioconductor](#)

[R-Forge](#)

[R-Hub](#)

[GSoC](#)

The R Project for Statistical Computing

Getting Started

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

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

News

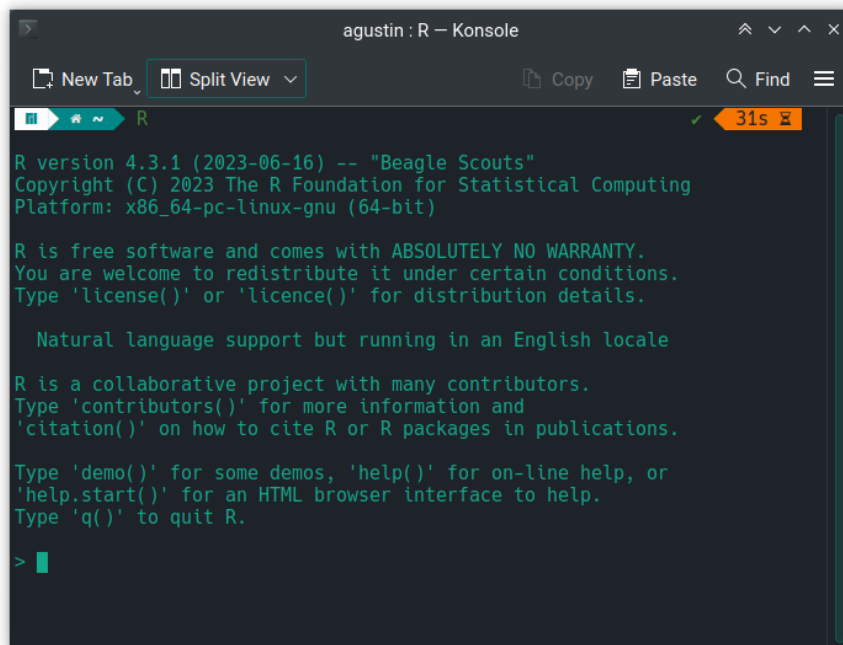
- [R version 4.1.1 \(Kick Things\)](#) has been released on 2021-08-10.
- [R version 4.0.5 \(Shake and Throw\)](#) was released on 2021-03-31.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the [R Consortium YouTube channel](#).
- You can support the R Foundation with a renewable subscription as a [supporting member](#)

News via Twitter

[News from the R Foundation](#)

© The R Foundation. For queries about this web site, please contact [the webmaster](#); for queries about R itself, please consult the [Getting Help](#) section.

Figura 2.1: Página oficial de R Project



```
agustin : R - Konsole
New Tab Split View Copy Paste Find
R
R version 4.3.1 (2023-06-16) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> █
```

Figura 2.2: Ejemplo de consola o terminal de Linux y Windows corriendo la última versión estable de **R**

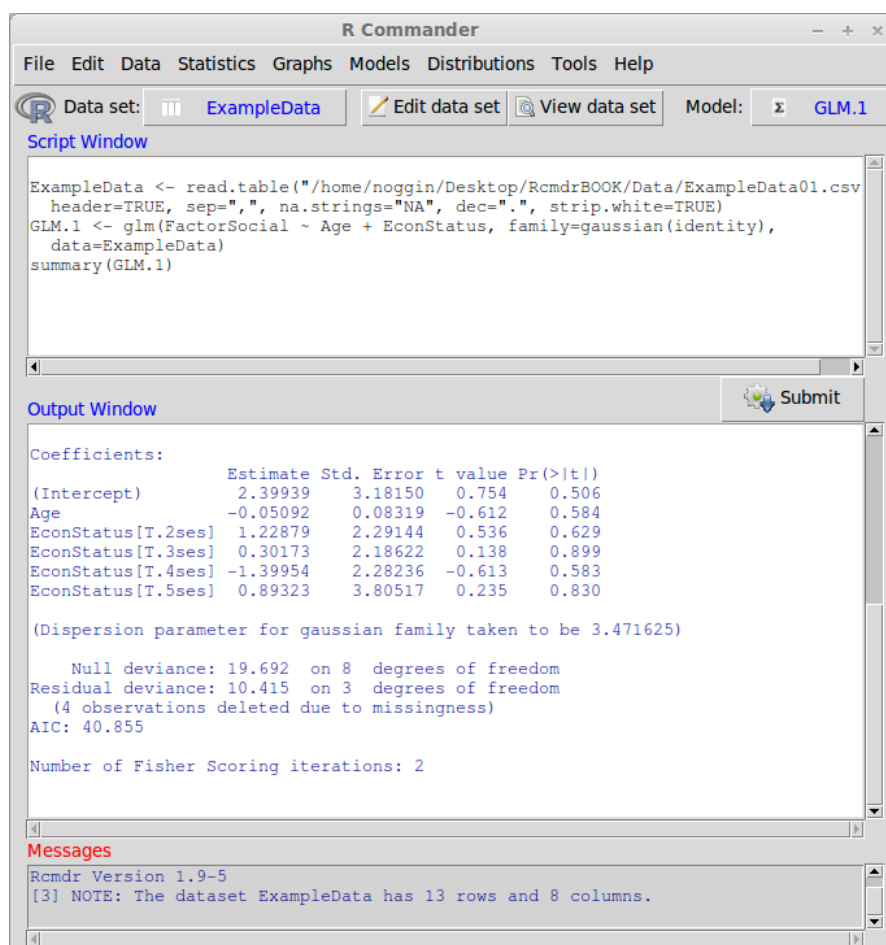


Figura 2.3: Interfase de R Commander

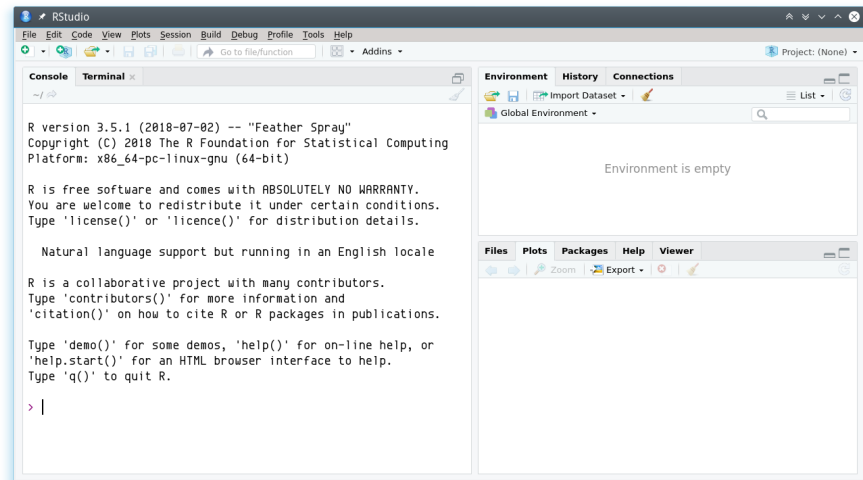


Figura 2.4: Interfase de **RStudio**

2.2 ¿Cómo instalar R y RStudio?

R y **RStudio** se instalan por separado. Ambos softwares son multiplataforma y pueden ser ejecutados en sistemas operativos Windows, Mac OS X y Linux.

R puede funcionar sin **RStudio**, en cambio **RStudio** necesita que al menos una versión de **R** esté instalada en el sistema.

La página de descarga de **posit** <https://posit.co/download/rstudio-desktop/> ofrece un excelente punto de partida para instalar ambos programas.

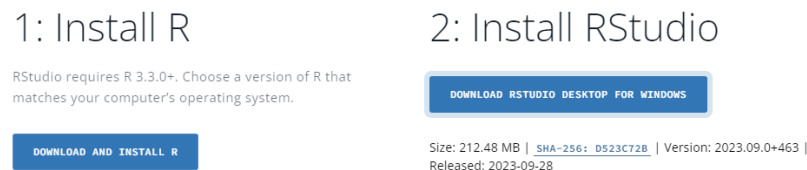


Figura 2.5: Página de descarga de **R**

A continuación se describe el procedimiento para instalar **R** y **RStudio** bajo Windows.

2.2.1 Instalación de **R**

- 1) Click en el botón **DOWNLOAD AND INSTALL R**:

1: Install R

RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

DOWNLOAD AND INSTALL R

Figura 2.6: Página de descarga de **R**

Descargar el archivo instalador correspondiente a la última versión estable de **R** desde el CRAN¹ (del inglés, *Comprehensive R Archive Network*) visitando el siguiente [link](#).

- 2) Ejecutar el archivo descargado ² y seguir el asistente de instalación con todas las opciones por defecto.
- 3) Si la instalación ha sido exitosa en el menú *Inicio* podrá encontrarse la carpeta *R* que contendrá dos accesos directos a la interfase de usuario mínima que viene con la versión de **R** para Windows.

2.2.2 Instalación de RStudio

- 1) Click en el botón DOWNLOAD RSTUDIO DESKTOP FOR:

Descargar el archivo de instalación correspondiente a nuestra plataforma o sistema operativo. Al momento de escribir estas instrucciones la última versión estable de **RStudio** era RStudio-2023.09.0-463.exe que se encuentra en este [link](#)

En el caso que haya una nueva versión, ir al sitio web de descarga de **RStudio** <https://posit.co/download/rstudio-desktop/>

- 2) Ejecutar el archivo .exe y seguir el asistente de instalación con todas las opciones por defecto.

¹Si bien <- funciona igual que = en la mayoría de los casos, por convención se usa <- para asignar y = para argumentos dentro de las funciones.

²Al momento de escribir estas instrucciones la última versión estable de **R** era la 4.3.1 *Beagle Scouts*, por lo tanto el link apuntará al archivo R-4.3.1-win.exe.



CRAN
[Mirrors](#)
[What's new?](#)
[Task Views](#)
[Search](#)

About R
[R Homepage](#)
[The R Journal](#)

Software
[R Sources](#)
[R Binaries](#)
[Packages](#)
[Other](#)

Documentation
[Manuals](#)
[FAQs](#)
[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

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

- [Download R for Linux \(Debian, Fedora/Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

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

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2021-08-10, Kick Things) [R-4.1.1.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

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

What are R and CRAN?

R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the [R project homepage](#) for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN [mirror](#) nearest to you to minimize network load.

Submitting to CRAN

To "submit" a package to CRAN, check that your submission meets the [CRAN Repository Policy](#) and then use the [web form](#).

If this fails, send an email to CRAN-submissions@R-project.org following the policy. Please do not attach submissions to emails, because this will clutter up the mailboxes of half a dozen people.

Note that we generally do not accept submissions of precompiled binaries due to security reasons. All binary distribution listed above are compiled by selected maintainers, who are in charge for all binaries of their platform, respectively.

For queries about this web site, please contact [the webmaster](#).

Figura 2.7: Página de descaga de R



CRAN
[Mirrors](#)
[What's new?](#)
[Search](#)
[CRAN Team](#)

About R
[R Homepage](#)
[The R Journal](#)

[Software](#)

R for Windows

Subdirectories:

[base](#) Binaries for base distribution. This is what you want to [install R for the first time](#).
[contrib](#) Binaries of contributed CRAN packages (for R >= 3.4.x).
[old.contrib](#) Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).
[Rtools](#) Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

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

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

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

Figura 2.8: Dirigirse a Install R for first time

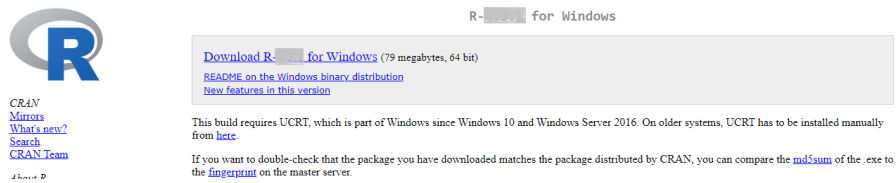


Figura 2.9: Dirigire a “Download R-X.X.X for Windows”

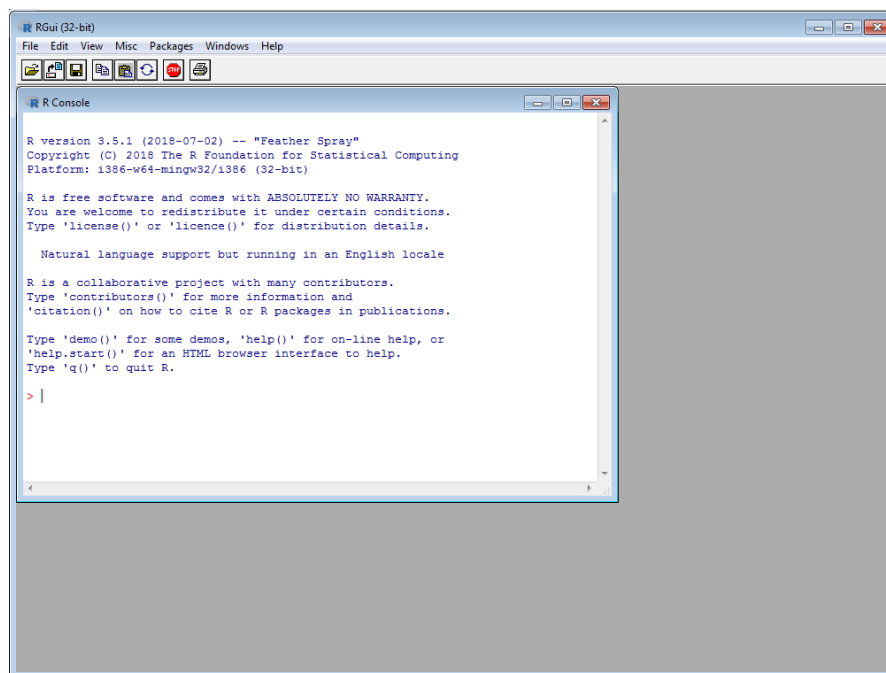


Figura 2.10: R GUI para Windows

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

Size: 212.48 MB | [SHA-256: D523C72B](#) | Version:
2023.09.0+463 | Released: 2023-09-28

Figura 2.11: Página de descarga de **R**

- 3) Si la instalación ha sido exitosa en el menú *Inicio* dentro de la carpeta *RStudio* se encontrará el acceso directo a **RStudio** el cual, mediante el menu contextual (botón derecho del ratón) puede enviarse al Escritorio como acceso directo o bien anclar al menu de Inicio o barra de acceso rápido.

Ahora sí, ya tenemos listo **R** y **RStudio** para empezar a trabajar!!

ACTIVIDAD

1. Instalar **R** y **RStudio**

2.3 Primera sesión

El entorno de trabajo de **RStudio** se divide en cuatro paneles. La disposición y contenido de los paneles se puede personalizar yendo a **View > Panes > Panes Layout...** A continuación la descripción de los paneles principales:

1. **Editor**. Es donde se editan los *scripts* que son archivos de texto plano con los comandos para ejecutar en **R**. Este panel no aparece a menos que se cree un nuevo script o se abra uno previamente guardado.
2. **Console** (consola). Es donde vive **R** propiamente dicho. Allí se ejecutan los comandos y se obtienen las salidas de **R**.
3. **Environmnet/History/Connections**. En la primera pestaña se visualizan los objetos (variables, funciones o datos cargados) que están disponibles en el entorno de **R**, i.e. en

RStudio Desktop 1.4.1717 - [Release Notes](#)

1. Install R. RStudio requires [R 3.0.1+](#).
2. Download RStudio Desktop. Find your operating system in the table below.



All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy.

RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).

OS	Download	Size	SHA-256
Windows 10	RStudio-1.4.1717.exe	156.18 MB	71b36e64
macOS 10.14+	RStudio-1.4.1717.dmg	203.06 MB	2cf2549d
Ubuntu 18/Debian 10	rstudio-1.4.1717-amd64.deb	122.51 MB	e27b2645
Fedora 19/Red Hat 7	rstudio-1.4.1717-x86_64.rpm	138.42 MB	648e2be0
Fedora 28/Red Hat 8	rstudio-1.4.1717-x86_64.rpm	138.39 MB	c76f620a
Debian 9	rstudio-1.4.1717-amd64.deb	123.29 MB	e4ea3a60
OpenSUSE 15	rstudio-1.4.1717-x86_64.rpm	123.15 MB	e69d55db

Zip/Tarballs

OS	Zip/tar	Size	SHA-256
Windows 10	RStudio-1.4.1717.zip	227.77 MB	84b1dc1a
Ubuntu 18/Debian 10	rstudio-1.4.1717-amd64-debian.tar.gz	177.14 MB	ba24900c
Fedora 19/Red Hat 7	rstudio-1.4.1717-x86_64-fedora.tar.gz	177.24 MB	4c05ddca
Debian 9	rstudio-1.4.1717-amd64-debian.tar.gz	177.48 MB	cd6d8462

Source Code

A tarball containing source code for RStudio 1.4.1717 can be downloaded from [here](#).

Figura 2.12: Página de descarga de **RStudio**



Figura 2.13: Icono de **RStudio**

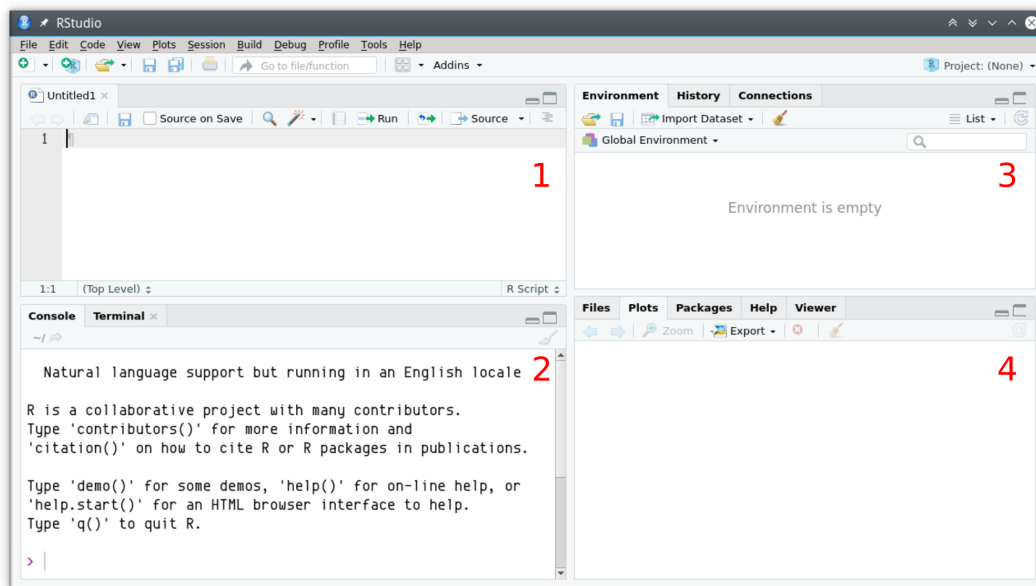


Figura 2.14: Interfase principal de **RStudio**

la memoria. En la segunda se puede ver el historial de comandos ingresados o enviados a la consola. La tercera pestaña visualiza las conexiones establecidas con diferentes base de datos.

4. **Files/Plots/Packages/Help/Viewer.** Allí se puede manejar los archivos del directorio de trabajo, visualizar los gráficos generados en **R** con posibilidad de exportarlos en varios formatos, administrar los paquetes o complementos, buscar o explorar el manual de ayuda y previsualizar archivos HTML.

2.3.1 La consola

La línea de comandos o **consola** es el modo interactivo mediante el cual podemos ejecutar comandos directamente en el intérprete de **R**. El símbolo o *prompt* `>` indica que **R** está disponible esperando una orden. Si la orden no está completa el símbolo se transforma en `+`. Por ejemplo: si tipeamos `2 + 2` y luego **ENTER**:

```
2 + 2
```

```
[1] 4
```

Obtenemos inmediatamente el resultado. Otro ejemplo: el promedio de los números 1, 3 y 4

```
(1 + 3 + 4) / 3
```

```
[1] 2.666667
```

ACTIVIDAD

1. Escribir una operacion matemática, por ejemplo: `3*4`
2. Escribir algo en la consola. ¿Que sucede?
3. Escribir lo anterior entre comillas " ".

2.3.2 El script

El **editor de scripts** (panel #1) es un editor de texto plano que está conectado con la consola (panel #2). Tiene algunas funcionalidades que facilitan la edición del código:

- Resaltado sintaxis: mediante colores resalta las funciones, variables, comandos o palabras claves del lenguaje **R**
- Sangrado automático: agrega espacios en blanco para mantener la sangría de los bloques de código.
- Plegado de código: permite colapsar bloques de código
- Completado automático y ayuda en línea: muestra sugerencias para completar el comando o argumentos usando la tecla **TAB**.

Para crear un nuevo script se puede usar uno de los siguientes métodos:

- Ir a al menu **File > New File > R Script**
- Usar el atajo de teclado **CTRL + SHIFT + N**
- Clickear en el primer ícono de la barra de menu

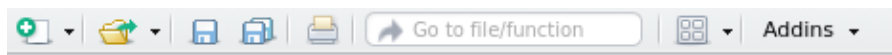


Figura 2.15: Barra de herramientas de **RStudio**

Una vez abierto el script en blanco, se pueden empezar a escribir los comandos de **R**. Por ejemplo podemos escribir lo siguiente:

```
"Hola Mundo!" # Clásico mensaje "Hola mundo!"

# Calcular el promedio de estos números
(1 + 3 + 4) / 3
```

Estos comandos no se van a ejecutar automáticamente ya que sólo los hemos escrito en el *script*. Para ejecutar los comandos en la consola hay que posicionar el cursor en la línea deseada o bien seleccionar si queremos ejecutar varias a la vez y luego enviarlo a la consola con una de las siguientes opciones:

- Ir al menu **Code > Run Selected Line(s)**
- Usar el atajo de teclado **CTRL + ENTER** o **CTRL + R**
- Usar el ícono **Run** de la barra de herramientas de la pestaña del script



Figura 2.16: Barra de herramientas del panel Editor

El simbolo **#** indica que lo que sigue es un **comentario** y por lo tanto **R** lo ignora cuando es enviado a la consola. Los comentarios pueden ir solos en una línea separada o bien dentro de una línea que tenga algún comando. Si bien no son necesarios para correr el código, los

comentarios son muy útiles para estructurar el script y hacer anotaciones para que otros, o nosotros en un futuro, entiendan lo que hace esa parte del script.

Para guardar el script:

1. Ir al menu **File > Save** o usar el atajo de teclado **CTRL + S** o bien el ícono con el diskette de la barra de herramientas global o de la pestaña del script activo.
2. Elegir la carpeta destino y el nombre de archivo. Automáticamente se agregará la extensión **.R** que corresponde a los scripts.

ACTIVIDAD

1. Abrir un script nuevo
2. Escribir un comentario
3. Escribir un texto o un comando numérico
4. Guardar el script con el nombre `mi_primer_script.R`.

2.3.3 Directorio de trabajo y proyectos

R trabaja con un directorio de trabajo o *working directory* que es la dirección o *path* que figura en el título del panel **Console**. Esto se puede averiguar con `getwd()`

```
getwd()
```

Por defecto es el directorio base del usuario que depende de cada plataforma. En linux es el `/home/usuario` en cambio en Windows es `C:/Users/usuario/Documents`.

A menos que se especifique lo contrario, se asume que los archivos de entrada o salida se ubican en esa. Esto se puede modificar en cualquier momento con la función `setwd()`.

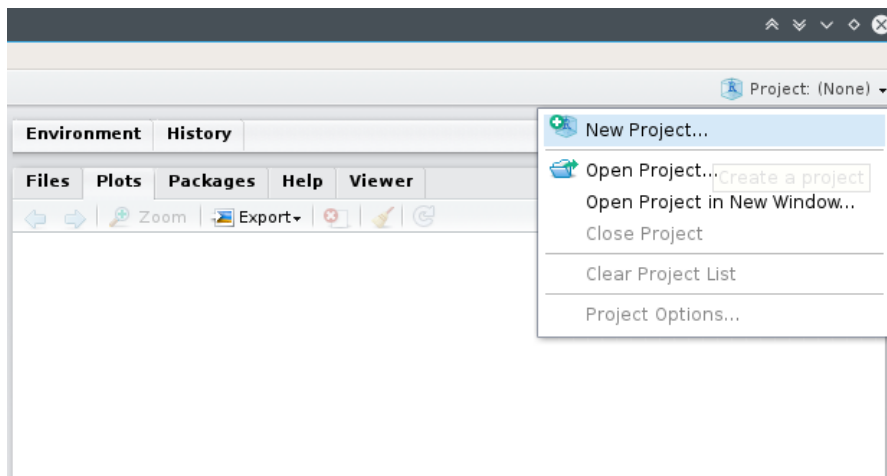
```
setwd("ruta/a/otra/carpeta")
```

RStudio extiende esta característica a través de los proyectos o *projects*. Cada proyecto es una carpeta o *folder* que contienen un archivo `.RProj` con algunas configuraciones específicas.

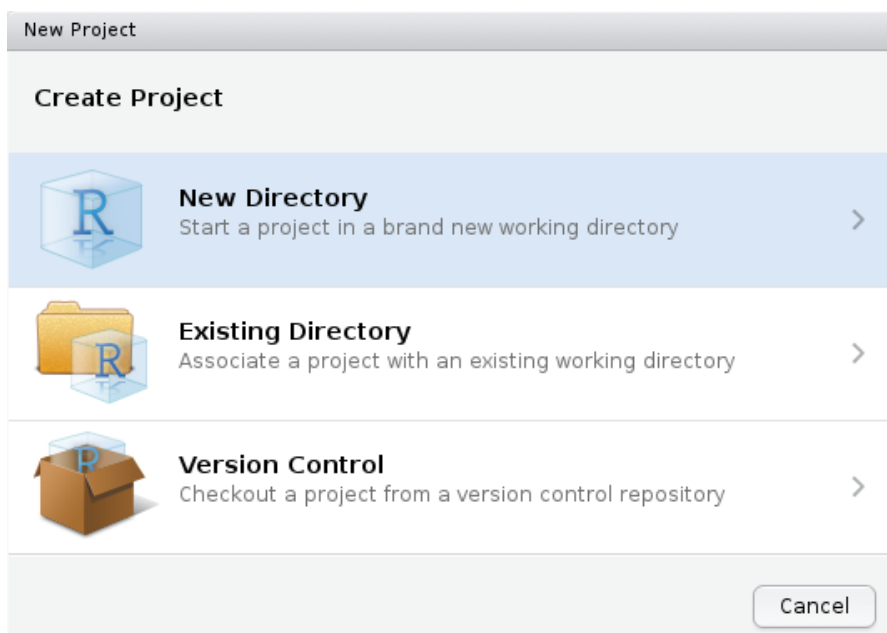
Al abrir un proyecto, automáticamente se cambia el directorio de trabajo a esta carpeta. Esto permite organizar los archivos de datos, las salidas, los scripts, etc., dentro de un directorio de trabajo (*working directory*) y volver a ellos de manera más rápida, eficiente, y portable.

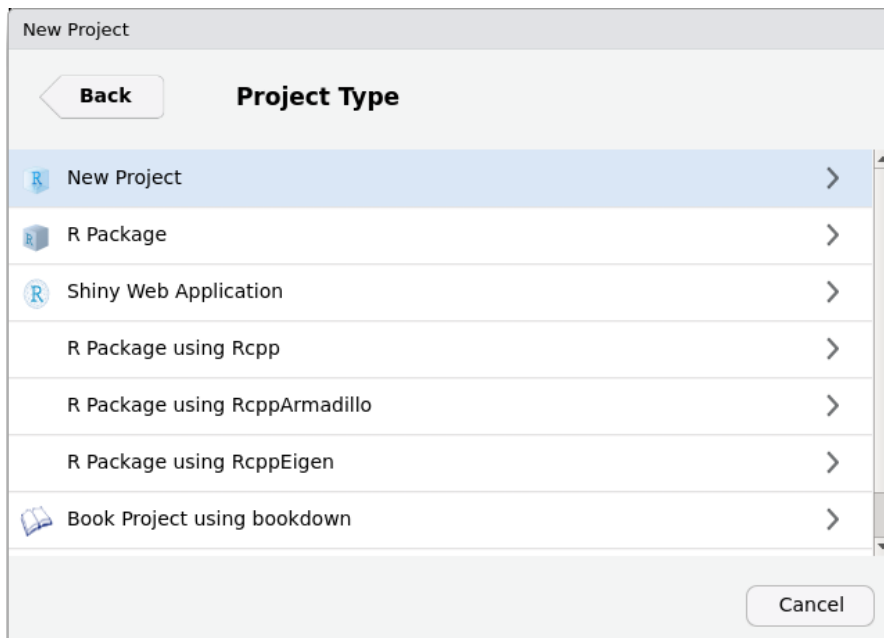
Para crear un proyecto en **RStudio**:

1. Ir a File > New project... o bien el ícono Create project de la barra de herramientas.



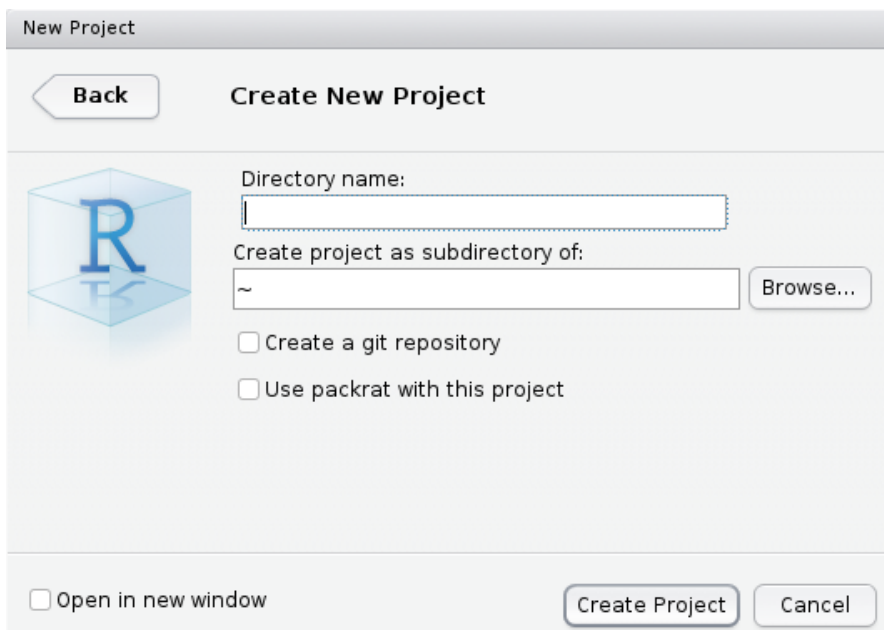
2. Seleccionar New Directory y en Project type seleccionar New project.





- Una vez en el cuadro de diálogo **Create new project** ingresar el nombre del proyecto (e.g. `mi_proyecto`) en **Directory name** que será a su vez el nombre de la carpeta que **RStudio** va a crear por nosotros.

Luego en **Create project as subdirectory of** indicar *donde* queremos que **RStudio** cree la carpeta.



4. Si todo sale bien, se crea la carpeta con el nombre que indicamos y dentro de ésta un archivo con extensión `.Rproj`. Este archivo solamente se usa para abrir el directorio. No se debe sobrescribir con el script.

ACTIVIDAD

1. Crear un proyecto nuevo con el nombre “Intro_R” en el escritorio o lugar de preferencia.
2. Cerrar **Rstudio**.
3. Abrir **Rstudio** desde el proyecto.
4. Copiar el script creado en la actividad anterior dentro del nuevo proyecto.
5. Abrir el script.

2.3.4 Ayuda!!!

Por último, y no menos importante, **R** y **RStudio** cuentan con un completo sistema de ayuda.

Desde la consola se puede acceder usando la función `?` seguida del nombre de la función o bien `help("nombre")`

```
# Pedir ayuda de la función mean
?mean
help(mean)
```

Una de las ventajas de **RStudio** es que dispone de un panel (Panel #4) dedicado a visualizar las páginas de ayuda. Allí se puede navegar por las páginas utilizando los links, realizar búsquedas, etc. Leer la documentación nunca viene mal y generalmente ahorra dolores de cabeza.

3 Aspectos básicos de R

En esta sección vamos a aprender nociones básicas de la sintaxis de R, tipos de datos y estructuras.

3.1 Operadores matemáticos y lógicos

Como vimos antes, las operaciones matemáticas básicas se realizan usando los símbolos convencionales:

- suma (+)
- resta (-)
- división (/)
- producto (*)
- potencia (^)

Por ejemplo, $1 + \left(3 \times 4 + \frac{5-2}{3}\right)^2$ en **R** es:

```
1 + (3 * 4 + (5 - 2)/3)^2
```

```
[1] 170
```

También se pueden evaluar expresiones lógicas:

- *igual que* (==)
- *distinto que* (!=)
- *mayor que* (>)
- *menor que* (<)
- *mayor o igual que* (>=)
- *menor o igual que* (<=)

El resultado es **TRUE** (verdadero) o **FALSE** (falso). Por ejemplo, podemos evaluar si 3 es igual a 4

```
3 == 4
```

```
[1] FALSE
```

O si 5 es mayor o igual a 3

```
5 >= 3
```

```
[1] TRUE
```

También se pueden combinar con los operadores *intersección* (&), *unión* (|) y *negación* (!).

Por ejemplo, evaluar si se cumplen las dos cosas anteriores a la vez

```
3 == 4 & 5 >= 3
```

```
[1] FALSE
```

Devuelve **FALSE** porque `3 == 4` no es verdadero. Si reemplazamos & por | va a devolver evaluar si una de las dos se cumple:

```
3 == 4 | 5 >= 3
```

```
[1] TRUE
```

También se pueden combinar con operaciones matemáticas...

```
4 * 2 == 8
```

```
[1] TRUE
```

En este caso primero evalúa `4 * 2` y luego compara el resultado con 8

ACTIVIDAD

1. Hallar el resultado de la siguiente expresión $(2 + 4)^2$ y $2^2 + 4^2$.
2. Comparar ambos resultados en una línea de comando.

3.2 Variables y objetos

Un objeto es un espacio de la memoria que almacena un pedazo de información (una cifra, un conjunto de números, el resultado de un análisis, etc). También se denomina *variables* ya que su contenido puede cambiar. En **R** prácticamente todo puede representarse como un *objeto*.

Los objetos o variables se crean *asignándoles* información (números, letras, resultados de operaciones, etc), con el símbolo `<-` (ALT + -) o `=`¹. Esta información se puede recuperar, modificar o utilizar para otros cálculos.

Supongamos que queremos asignar el valor 2 a la variable `x`.

```
x <- 2
```

En la consola vuelve a aparecer el símbolo `>` y nada más. En el ambiente se ve una entrada que dice `x` y el valor. Podemos recuperar el valor en la consola tipeando el nombre del objeto:

```
x
```

```
[1] 2
```

También podemos reusarlo en otro cálculo, por ejemplo obtener 2 veces `x`.

```
2 * x
```

```
[1] 4
```

O bien obtener una nueva variable:

```
y <- 2 * x + 1  
y
```

```
[1] 5
```

Los nombres de las variables no deben empezar con números ni contener espacios. No pueden usarse operadores (`*+-/%%`) en los nombres pero puede usarse `.` o `_`.

¹Si bien `<-` funciona igual que `=` en la mayoría de los casos, por convención se usa `<-` para asignar y `=` para argumentos dentro de las funciones.

```
# Mal
2x <- 3
mi variable <- 3

# Bien
x_2 <- 3
x.2 <- 3
x2 <- 3
```

También **R** es sensible a mayúsculas

```
# Definir 'A' y 'a'
A <- 3
a <- 5

# Verificar si 'A' y 'a' son lo mismo
A == a
```

[1] FALSE

ACTIVIDAD

¿Por qué este código no funciona?

```
my_variable <- 10
my_variable
```

3.2.1 Vectores

Son los objetos más simples a partir de los cuales se construyen otros tipos de objetos. Se crean utilizando la función `c()` (*combine*) para “combinar” **datos del mismo tipo**

```
x <- c(13, 45, 67, 45)
x
```

[1] 13 45 67 45

En el caso de mezclar de datos, **R** los va a *convertir* al tipo de datos más simple.

Por ejemplo: si queremos crear un vector con 3 valores: lógico, numérico y texto, **R** va a asumir que todos los elementos son de tipo *texto*

```
y <- c(TRUE, 34, "hola")
y
```

```
[1] "TRUE" "34"   "hola"
```

Los vectores están indexados. Se puede acceder a sus elementos usando el operador `[]` e indicando el número de orden.

Por ejemplo: para recuperar el 3er elemento del vector **x**

```
y[3]
```

```
[1] "hola"
```

Veremos más adelante los distintos tipos.

3.2.2 Funciones y argumentos

Para crear los vectores sin pensarlo utilizamos una función `c()`. Las funciones son series de comandos que hacen alguna acción y producen un resultado.

Generalmente tienen la siguiente la siguiente forma:

```
nombre_funcion(arg1, arg2, ...)
```

donde **arg** son los argumentos (valores de entrada u opciones). Algunos argumentos toman valores por defecto otros hay que declararlos.

Por ejemplo, la función `round()`⁴ tiene los argumentos:

- **x**, para pasar el número o vector numérico que queremos redondear
- **digits = 0** para indicar el número de dígitos a usar, por defecto 0.

Supongamos que queremos redondear el número 3.141593 a 3 dígitos. Para eso usamos la función `round()` (buscar en la ayuda `?round`)

```
# Indicando los argumentos
round(x = 3.141593, digits = 3)
```

```
[1] 3.142
```

```
# Sin indicar los argumentos  
round(3.141593, 3)
```

```
[1] 3.142
```

En este último caso, el orden de los argumentos es clave ya que **R** asigna los valores en función de la posición.

```
# Sin indicar los argumentos  
round(3, 3.141593)
```

```
[1] 3
```

Devuelve 3 por considera que queremos redondear el número 3

3.2.3 Creando funciones

Así como **R** tiene viene con funciones ya definidas, nosotros podemos crear nuestras propias funciones para poder simplificar nuestro análisis encapsulando tareas repetitivas.

Por ejemplo, supongamos que queremos calcular el área de 3 rectángulos cuyas dimensiones son:

- Rectángulo 1: base = 10, altura = 20
- Rectángulo 1: base = 15, altura = 35
- Rectángulo 1: base = 20, altura = 5

Sabemos que la fórmula del área es: $A = b \times a$, donde A es el area, b es la base y a es la altura.

Podríamos hacer:

```
10 * 20
```

```
[1] 200
```

```
15 * 35
```

```
[1] 525
```



```
20 * 5
```

```
[1] 100
```

Como vimos antes podemos usar vectores y operar con ellos:

```
bases <- c(10, 15, 20)
alturas <- c(20, 35, 5)
bases * alturas
```

```
[1] 200 525 100
```

Yendo un poco mas lejos, en vez de repetir siempre la operación podríamos encapsularla en una *función*. Las funciones se crean con `function()` o `\()` y adentro se declaran los argumentos. Por defecto la función devuelve el ultimo comando o bien lo que se indique en `return()`

```
area_rectangulo <- function(base, altura) {

  # Calculo superficie
  area <- base * altura

  return(area)

}
```

R no devuelve nada porque lo que hizo fue crear la función, a continuación podemos usarla:

```
area_rectangulo(40,50)
```

```
[1] 2000
```

```
area_rectangulo(bases, alturas)
```

```
[1] 200 525 100
```

ACTIVIDAD

1. Crear un vector llamado `diametro` que contenga 10 valores numéricos
2. Calcular otro vector que se llame `radios` en función del anterior
3. Crear una función que calcule el área de un círculo. Tip: la constante π en **R** está en el objeto `pi`.
4. Avanzado: crear una función más avanzada llamada `area` que calcule el área de un rectángulo o círculo indicando en el argumento `figura` y suministrando el dato de base y altura o radio, según corresponda.
5. ¿Cómo podríamos expandirla para controlar el número de decimales de la respuesta?

3.3 Tipos de datos

3.3.1 Numéricos (`numeric`)

Números racionales (enteros o con coma).

```
x <- c(3, 4, 5)
class(x)
```

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

Los números enteros se tratan como `numeric` a menos que se los convierta con `as.integer()`.

```
y <- as.integer(x)
class(y)
```

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

Los datos numéricos permiten todas las operaciones algebraicas

```
mean(x)
```

```
[1] 4
```

```
mean(y)
```

```
[1] 4
```

3.3.2 Texto (character)

Cadenas de texto o número delimitadas por comillas (simples o dobles, nom mezclar).

```
x <- c("hola", '3')  
class(x)
```

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

Lógicamente no se pueden realizar operaciones numéricas. **R** avisa y devuelve NA

```
mean(x)
```

Warning in mean.default(x): argument is not numeric or logical: returning NA

```
[1] NA
```

3.3.3 Lógicos (logic)

Condición verdadero (TRUE o T) o falso (FALSE o F)

```
logico <- c(T, F, T, TRUE, FALSE, F)  
logico
```

```
[1] TRUE FALSE TRUE TRUE FALSE FALSE
```

Otro ejemplo: ¿cuáles de los siguientes números son mayores a 30?

```
x <- c(23, 43, 21, 34, 56, 3, 23, 3)  
x > 30
```

```
[1] FALSE TRUE FALSE TRUE TRUE FALSE FALSE FALSE
```

3.3.4 Factores (factor y ordered)

Si los elementos de vector de tipo texto (`character`) y representan niveles nominales (categorías), el objeto puede convertirse a `factor` de modo tal que los valores son reemplazados por un número que se asocia a los niveles del factor (ordenados alfabéticamente, a menos que se indique otra cosa).

Un ejemplo de un vector tipo `character`.

```
x <- c('bajo', 'medio', 'alto', 'alto', 'bajo', 'bajo')
x
```

```
[1] "bajo" "medio" "alto" "alto" "bajo" "bajo"
```

Sólo se muestran los valores (bajo, medio y alto). No hay información de niveles. Ahora si aplicamos `factor(x)`:

```
y <- factor(x)
y
```

```
[1] bajo medio alto alto bajo bajo
Levels: alto bajo medio
```

Los valores pasaron al atributo `levels` y los datos fueron reemplazados por los identificadores 2, 3, y 1 según el orden alfabético de los niveles.

```
as.numeric(y)
```

```
[1] 2 3 1 1 2 2
```

Cuando los niveles tienen una jerarquía u orden, se puede especificar este tipo de relación mediante `as.ordered()` que convierte el `factor` en uno especial `ordered` agregando la relación entre los niveles

```
z <- factor(x, levels = c('bajo', 'medio', 'alto'))
z <- as.ordered(z)
z
```

```
[1] bajo medio alto alto bajo bajo
Levels: bajo < medio < alto
```

Los factores como cualquier vector también se indexan con `[]`.

3.3.5 Otros tipos de datos

Los valores faltantes se simbolizan en **R** con **NA** (*not available*). Indican que debería haber un valor pero que está faltando.

```
x <- c(1, 2, 3, NA, 4)
is.na(x)
```

```
[1] FALSE FALSE FALSE  TRUE FALSE
```

A diferencia del **NA**, un valor de tipo **NULL** indica que no hay información y que tampoco se esperaba que la haya.

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

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

Algunas operaciones matemáticas devuelven valores **NaN** (*not a number*) cuando no están definidas, por ejemplo:

```
0/0
```

```
[1] NaN
```

O bien valores infinitos (**Inf**):

```
1/0
```

```
[1] Inf
```

ACTIVIDAD

1. Ingrese lo siguiente en un vector con el nombre **color**: púrpura, rojo, amarillo, marrón
2. Mostrar el segundo elemento en el vector (rojo) en la consola.
3. Ingrese lo siguiente en un vector con el nombre **peso**: 23, 21, 18, 26

3.4 Estructura de datos

A partir de los tipos de datos que vimos antes, se pueden construir objetos más complejos.

3.4.1 Matriz (`matrix`)

Colección de vectores de *igual longitud y mismo tipo de datos*. Se crea con la función `matrix()`, o combinando filas o columnas de igual longitud con `rbind()` o `cbind()`.

Por ejemplo la matriz:

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

en **R** se representa así:

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

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

Se puede indexar usando `[n, p]` donde **n** es el numero de fila y **p** numero de columna. Por ejemplo para obtener el elemento m_{12}

```
M[1,2]
```

```
[1] 4
```

O todos los elementos de la columna 2

```
M[, 2]
```

```
[1] 4 5 6
```

3.4.2 Listas (list)

Es una generalización de los vectores ya que *los elementos pueden ser de igual o diferente tipo de datos*

```
lst <- list(23, "hola", TRUE)
lst
```

```
[[1]]
[1] 23
```

```
[[2]]
[1] "hola"
```

```
[[3]]
[1] TRUE
```

Se pueden indexar usando `[[]]`

```
# El segundo elemento de l
lst[[2]]
```

```
[1] "hola"
```

Cada elemento a su vez puede ser cualquier objeto de los vistos anteriormente.

3.4.3 Hoja de datos (data.frame)

Similares a las matrices pero cada columna puede ser de un tipo de dato diferente. Útil para guardar datos donde cada fila es un caso y cada columna una variable.

Supongamos que tenemos la tabla de datos:

Lote	Variedad	Rendimiento
1	Escorpion	34
2	Escorpion	36
3	Yarara	40
4	Baguette11	28
5	Tijetera	31

En **R** podemos representarla así:

```
trigo <- data.frame(  
  lote = 1:5,  
  variedad = c('Escorpion', 'Escorpion', 'Yarara', 'Baguette 11', 'Tijetera'),  
  rendimiento = c(34, 36, 40, 28, 31)  
)  
trigo
```

	lote	variedad	rendimiento
1	1	Escorpion	34
2	2	Escorpion	36
3	3	Yarara	40
4	4	Baguette 11	28
5	5	Tijetera	31

Al igual que las matrices, un `data.frame` se puede indexar con `[]`. Por ejemplo, si quisieramos El nombre de la variedad de la fila 2

```
trigo[2, 3]
```

```
[1] 36
```

O todos los nombres de la fila 2

```
trigo[2, ]
```

	lote	variedad	rendimiento
2	2	Escorpion	36

También podemos hacer consultas más específicas: “Lotes con rendimiento mayor a 35 qq/ha”

```
trigo[trigo$rendimiento > 35, ]
```

	lote	variedad	rendimiento
2	2	Escorpion	36
3	3	Yarara	40

Las variables o columnas se pueden acceder individualmente usando o el operador `$` seguido de nombre de la columna o `[, "nombre"]`, o `[, posicion]`. Ejemplo: extraer la columna `rendimiento` que es la número 3

```
trigo$rendimiento
```

```
[1] 34 36 40 28 31
```

```
trigo[, "rendimiento"]
```

```
[1] 34 36 40 28 31
```

```
trigo[, 3]
```

```
[1] 34 36 40 28 31
```

Otras estructuras

Las estructura listadas arriba son nativas de **R**. Los paquetes o complementos pueden agregar nuevas estructuras o redifinar las existentes.

ACTIVIDAD

1. Unir los 2 vectores de la actividad anterior usando la función `data.frame` para crear un marco de datos llamado `info` con 2 columnas y 4 filas. Llame a la primera columna `'color'` y a la segunda `peso`.
2. Ver la estructura de datos en la consola.
3. Mostrar solo la fila 3 en la consola
4. Mostrar solo la columna 1 en la consola
5. Mostrar el elemento de datos en la fila 4, columna 1

4 Paquetes de R

En esta sección vamos a aprender como extender las funcionalidades de **R** mediante la instalación y utilización de paquetes o *packages*

4.1 ¿Qué son los paquetes?

R viene con un conjunto de librerías mínimo denominado *core* que permite realizar una amplia variedad de análisis y operaciones con los datos. La comunidad que desarrolla **R** provee un repositorio de librerías o paquetes complementarios (*packages*) que expanden notablemente las funcionalidades de **R**.

Los paquetes se deben *instalar* primero usando la función `install.packages()` por única vez y en cada sesión se deben *cargar* con `library()`. La siguiente figura resume el proceso:



Images sourced from <https://www.wikihow.com/Change-a-Light-Bulb>

Suponiendo que queremos instalar el paquete `foo`, se debe ejecutar por única vez:

```
install.packages("foo")
```

Luego, para acceder a todas las funciones que aporta `foo`, en cada sesión de trabajo ejecutar hay que ejecutar:

```
library("foo")
```

Alternativamente, si una vez instalado el paquete `foo` queremos usar la función `bar()` pero sin cargar el resto del paquete, entonces:

```
foo::bar(...)
```

El manejo de paquetes se puede simplificar enormemente con el paquete `pacman`. Entre otras funciones ofrece la función `p_load()` que carga los paquetes y si no están instalados los instala previamente.

Para instalar `pacman` por primera vez correr el siguiente comando:

```
# Instalar por unica vez  
install.packages("pacman")
```

Luego, cuando necesitemos podemos ejecutar `pacman::p_load()`. Por ejemplo, si queremos cargar el paquete `moments`

```
pacman::p_load(moments)
```

ACTIVIDAD

1. Instalar el paquete `pacman` usando `install.packages()`
2. Cargar/Instalar el paquete `tidyverse` usando `pacman::p_load()`

5 Importar datos en R

Un aspecto importante para cualquier análisis de datos es acceder a los **datos**!

Los datos pueden estar almacenados en diversos formatos: archivos de texto (`*.txt`, `*.dat`, etc), texto separado por comas (`*.csv`), planillas de cálculos (`*.xls` o `*.xlsx`), etc.

En este curso vamos a trabajar con archivos que ya se encuentran en planillas de cálculo tipo Excel o archivos de texto plano.

5.1 Función nativa para importar datos

R viene la función `read.table()` y derivados, que permiten leer datos desde formatos tipo **texto plano** (`plain text format`). El más popular entre estos es `*.csv`. Este formato asume que los datos están en formato de tabla o *rectangular* (e.g. variables en columnas y observaciones en filas) y devuelve un `data.frame`. En `?read.table` se detallan todos los argumentos, los más importantes son:

- `file` para indicar el nombre o ruta al archivo
- `header` para indicar si las columnas tienen encabezados que deben ser usados como nombre de las variables.
- `sep` para indicar el separador de columnas
- `dec` para indicar el símbolo decimal

Dependiendo de las combinaciones de estos 3 argumentos hay variantes (`read.csv()`, `read.csv2()`, `read.delim()`, `read.delim2()`) que son atajos de ‘`read.table()`’ (ver ayuda).

Mediante algún editor de textos (puede ser dentro de **RStudio**) conviene abrir el archivo y examinarlo para determinar:

- Tiene encabezados?
- Cómo están separadas las columnas?
- Cuál es el símbolo del decimal?

Supongamos que tenemos el archivo de texto `prueba.csv`, las alternativas podrían ser:

```
# Con encabezados, separado por tabulaciones y el decimal es el punto
prueba <- read.table("prueba.csv", header = T, sep = "\t", dec = ".")

# Con encabezados, separado por tabulaciones y coma como decimal
prueba <- read.table("prueba.csv", header = T, sep = "\t", dec = ",")

# Con encabezados, separado comas y punto como decimal
prueba <- read.table("prueba.csv", header = T, sep = ",", dec = ".")

# Con encabezados, separado punto y coma, y con coma como punto decimal
prueba <- read.table("prueba.csv", header = T, sep = ";", dec = ",")
```

En el caso que el archivo `prueba.csv` esté en otro directorio o ubicación que no sea el proyecto o `getwd()` hay que indicar la ruta completa al archivo.

Una vez importados los datos es conveniente verificar como han sido leídos en el **R**. Una alternativa es *imprimirlo* escribiendo el nombre del objeto directamente en la consola.

```
prueba
```

Otra alternativa es utilizar la función `View()` que muestra la tabla de datos en un formato de planilla interactiva de solo lectura.

```
View(prueba)
```

Si bien podemos inferir que tipo de datos se leyeron, una alternativa mejor es mirar la estructura con la función `str()`.

```
str(prueba)
```

ACTIVIDAD

1. Abrir un documento de texto en **RStudio**
2. Retomando el ejemplo del set de datos de trigo, crear un archivo separado por comas con los siguientes datos

```
Lote,Variedad,Rendimiento
1,Escorpion,34
2,Escorpion,36
3,Yarara,40
```

```
4,Baguette11,28
5,Tijetera,31
```

3. Guardar el archivo como `prueba_trigo.csv`.
4. Leer el set de datos dentro de **R** usando la función `read.table()` o alguna de sus variantes.
5. Verificar la importacion de los datos.

5.2 Paquetes para importar datos

Existen paquetes específicos que permiten leer virtualmente cualquier formato de archivos.

5.2.1 readr

Si bien la función `read.table()` y derivadas permiten leer datos *rectangulares* en formato texto, el paquete **readr** [link](#) provee una implementación más moderna (y rápidas) de estas funciones. Las funciones se llaman igual que las nativas pero se reemplaza `.` por `_` en el nombre. Ejemplo: `read_table()`

Este paquete ya está integrado a **tidyverse**. Para leer el set de datos que creamos en la actividad anterior:

```
pacman::p_load(tidyverse)
prueba_trigo <- read_csv("datasets/prueba_trigo.csv")
```

```
Rows: 5 Columns: 3
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (1): Variedad
```

```
dbl (2): Lote, Rendimiento
```

```
i Use `spec()` to retrieve the full column specification for this data.
```

```
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
prueba_trigo
```

```
# A tibble: 5 x 3
  Lote Variedad Rendimiento
<dbl> <chr>      <dbl>
1     1 Escorpion      34
2     2 Escorpion      36
3     3 Yarara         40
4     4 Baguette11     28
5     5 Tijetera       31
```

A diferencia de la función nativa, las funciones de **readr** devuelven un objeto llamado **tibble** que es una especie de **data.frame** pero con algunas propiedades extra.

5.2.2 rio

A diferencia de **readr** que es una reimplementación de funciones de **R**, hay un paquete llamado **rio** [link](#) que es una especie de *metapaquete* y permite simplificar la *importación*, *exportación*, y *conversión* de formatos en una sintaxis unificada.

Este paquete trabaja con una mayor variedad de formatos y, basado en la extensión del archivo, busca la función y/o paquete apropiado para leer o guardar los datos. En el caso de ser necesario, se pueden pasar argumentos a las funciones.

Retomando el ejemplo de trigo, podemos leer los datos con la función **import()**

```
pacman::p_load(rio)
prueba_trigo <- import("datasets/prueba_trigo.csv")
prueba_trigo
```

```
  Lote  Variedad Rendimiento
1     1 Escorpion      34
2     2 Escorpion      36
3     3 Yarara         40
4     4 Baguette11     28
5     5 Tijetera       31
```

A diferencia de **readr** siempre devuelve un **data.frame**. Si queremos que devuelva un **tibble** podemos usar el argumento **setclass**

```
prueba_trigo <- import("datasets/prueba_trigo.csv", setclass = "tibble")
prueba_trigo
```

```
# A tibble: 5 x 3
  Lote Variedad Rendimiento
<int> <chr>      <int>
1     1 Escorpion      34
2     2 Escorpion      36
3     3 Yarara         40
4     4 Baguette11     28
5     5 Tijetera       31
```

5.3 Formas de importar datos

A continuación vamos a detallar dos formas de abrir el archivo que contiene datos meteorológicos de la ciudad de Urbana (Illinois).

5.3.1 Desde la consola (recomendado)

Una vez que descargamos el archivo datos en la carpeta **datasets** dentro de nuestro directorio de trabajo o proyecto podemos leerlo en **R** usando la función `import()` del paquete **rio**. Esta función se encargará de llamar la función necesaria para leer el archivo que le suministremos.

```
urbana <- import("datasets/urbana_weather.xlsx", setclass = "tibble")
urbana
```

```
# A tibble: 240 x 4
  YEAR month precip temp
<dbl> <chr>  <dbl> <dbl>
1  2000 Jan    1.54  25.6
2  2001 Jan    1.32  25.4
3  2002 Jan    2.81   34
4  2003 Jan    0.79  21.1
5  2004 Jan    2.18   24
6  2005 Jan    6.2   27.8
7  2006 Jan    1.78  37.8
8  2007 Jan    3.03  29.7
9  2008 Jan    2.31  26.2
10 2009 Jan    0.68  18.8
# i 230 more rows
```

Si sólo estuviéramos interesados en el rango A1:C5 (primeros 4 registros de las 3 primeras columnas), podríamos usar:


```
urbana2 <- import(file = "datasets/urbana_weather.xlsx", range = "A1:C5")
urbana2
```

```
YEAR month precip
1 2000 Jan 1.54
2 2001 Jan 1.32
3 2002 Jan 2.81
4 2003 Jan 0.79
```

5.3.2 Desde el importador de datos de RStudio

RStudio cuenta con un asistente de importación de datos (**File > Import Dataset**) que brinda interfase a varias funciones especializadas en la importación de datos de paquetes específicos como **readr**, **readxl**, etc.

En el menú **File > Import Dataset** o bien el ícono del panel **Environment** despliega una lista con distintas opciones de importación: nos interesa **From Excel (readxl)**...



Figura 5.1: Importador de datos

Este menú tiene cuatro paneles:

1. Una barra de dirección para indicar la ruta al archivo o URL.
2. Una vista previa del contenido del archivo
3. Opciones de importación: aquí se puede especificar el nombre del objeto que se creará dentro de **R** (**Name**), la cantidad de líneas a leer, el número de la hoja, el rango de celdas, líneas a saltar (**skip**) y el identificador de datos **NA**.
4. Vista previa del código. En esta parte se puede visualizar como se construye el comando que se ejecutará al clickear en **Import**.

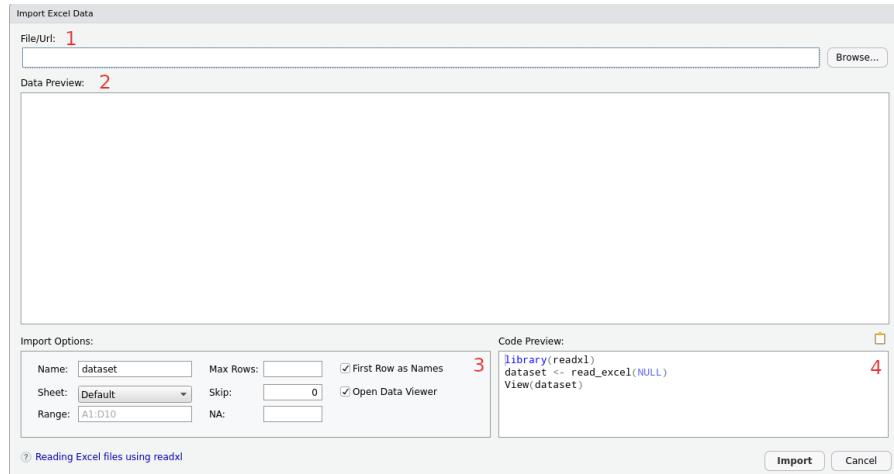


Figura 5.2: Importador de datos

i Aclaración

Si bien esta alternativa es intuitiva y amigable, no es reproducible a menos que el código generado por este asistente sea incluido en el script para futuras sesiones.

5.3.3 Desde el portapapeles

Una alternativa conveniente para acceder rápidamente a los datos es usando el portapapeles. Suponiendo que los datos están en una hoja de cálculos:

1. Seleccionar el rango de celdas **A1:C5** que nos interesa incluyendo los encabezados
2. Copiar en el porta papeles (CTRL + C)
3. Luego en R

```
urbana3 <- read.table("clipboard")
```

i Aclaración

Si bien esta alternativa es rápida, al no ser reproducible (no hay forma de plasmarla en el script para futuras sesiones), **no es recomendable** salvo para una exploración rápida.

References

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.