

# Introducción a R y RStudio

Christian Ballejo

Tamara Ricardo



Figure 1: Artwork por @allison\_horst

## ¿Qué es R?

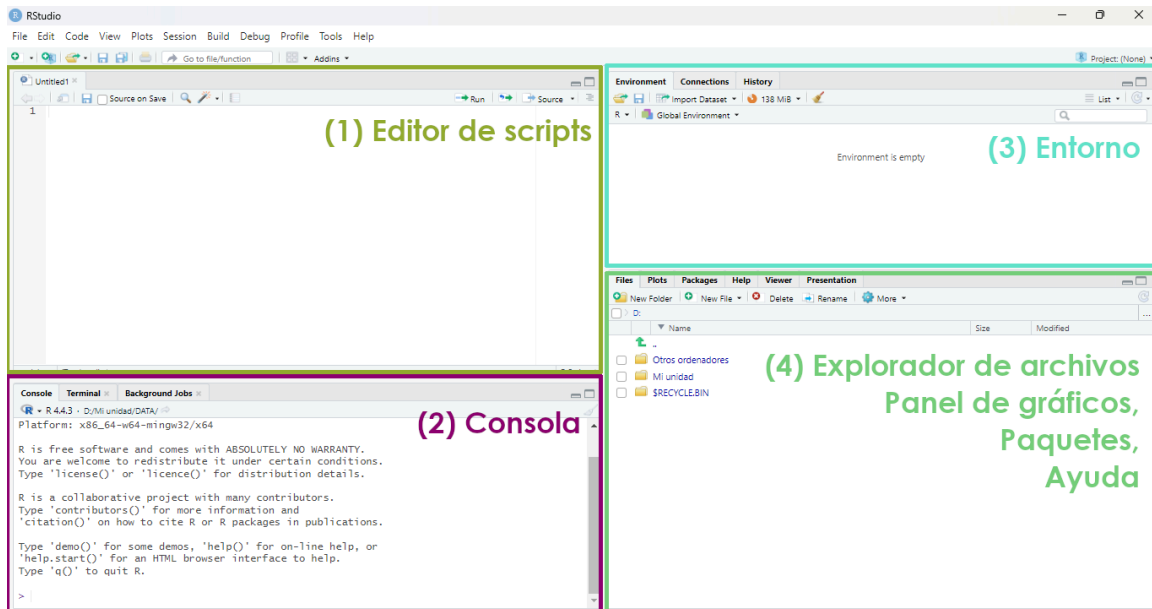
R [1] es un lenguaje de programación interpretado, orientado a objetos, multiplataforma y de código abierto, diseñado específicamente para el análisis estadístico de datos. Cuenta con estructuras y sintaxis propias, y una extensa colección de funciones desarrolladas para aplicaciones estadísticas.

- Como lenguaje **orientado a objetos**, todo lo que manipulamos —variables, funciones, conjuntos de datos, resultados— se considera un objeto, lo que aporta flexibilidad y simplicidad al trabajo con información.
- Al ser un lenguaje **interpretado**, los scripts se ejecutan directamente sin necesidad de compilación, lo que favorece la exploración interactiva.
- R es **multiplataforma**: se puede instalar y ejecutar en Linux, Windows y macOS con un comportamiento consistente.
- Además, es **software libre** distribuido bajo licencia GNU-GPL, lo que permite su uso, modificación y redistribución sin restricciones.

Para instalarlo en Windows, se debe descargar el instalador desde el **sitio oficial del proyecto R (CRAN)** y seguir los pasos guiados. Una vez finalizada la instalación, R estará listo para usarse desde cualquier entorno compatible. Sin embargo, si no se cuenta con experiencia previa en programación, no se recomienda utilizar R directamente desde su consola nativa.

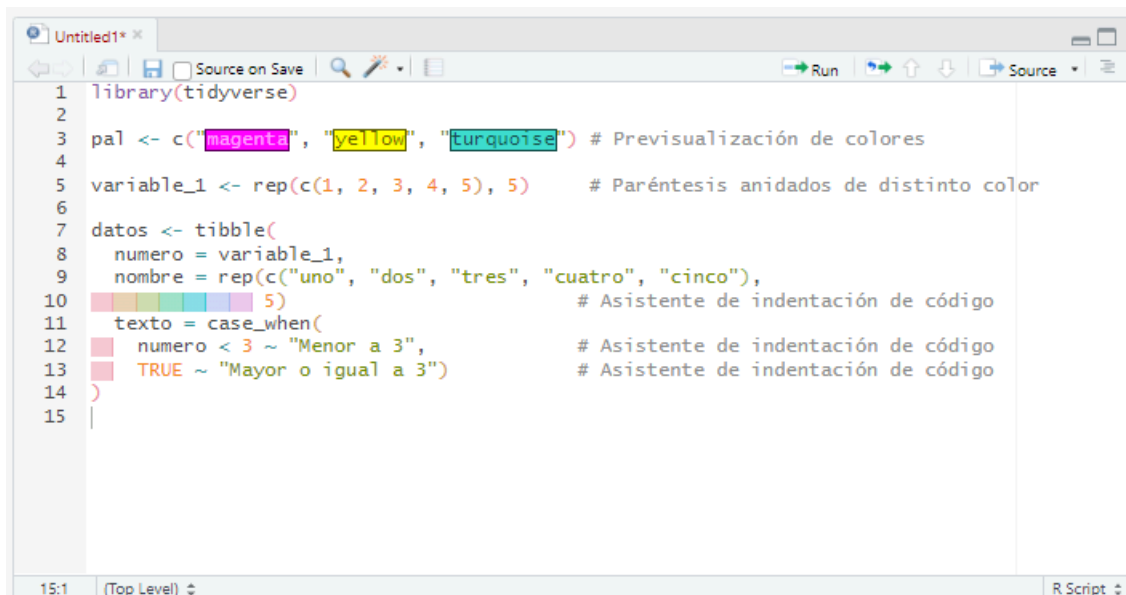
## ¿Qué es RStudio?

RStudio Desktop (2025, Posit Software) es un entorno de desarrollo integrado (IDE) diseñado específicamente para facilitar el trabajo con R. Proporciona una interfaz unificada que incluye editor de scripts, consola de R, entorno, explorador de archivos, panel de gráficos y ayuda, entre otros, optimizando el flujo de trabajo.

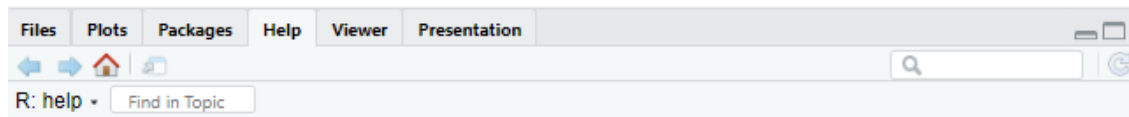


Entre sus principales ventajas se encuentran:

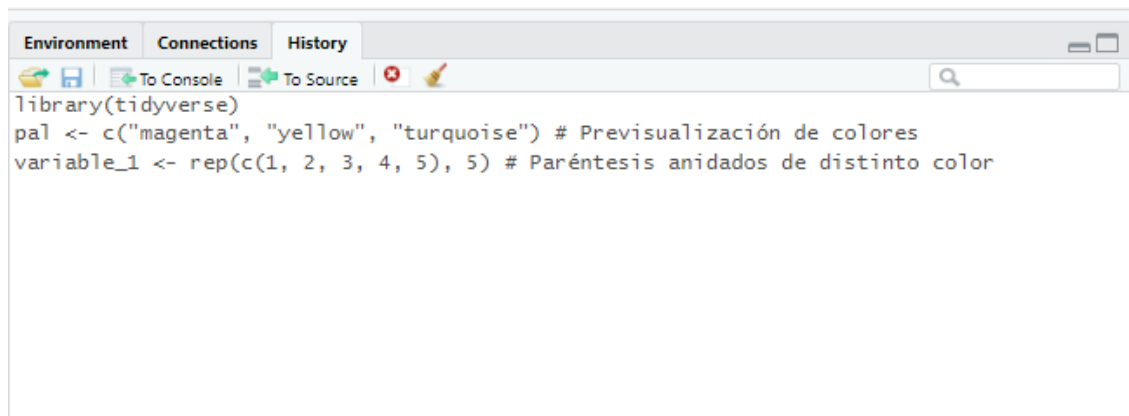
- **Asistente de código:** al escribir en el editor o la consola, la tecla **Tab** activa el autocompletado de funciones, nombres de objetos y argumentos, agilizando la escritura y reduciendo errores de sintaxis. En versiones recientes, el asistente también permite la previsualización de colores en los gráficos, resaltar los paréntesis de cierre en funciones anidadas con distintos colores y gestionar automáticamente la indentación del código.



- **Ayuda en línea:** al posicionar el cursor sobre el nombre de una función en el editor y presionar **F1**, se accede directamente a la documentación correspondiente en el panel **Help** (habitualmente ubicado en la esquina inferior derecha).



- **Historial de comandos:** en la consola, al usar las teclas de flecha arriba/abajo, se puede navegar por los comandos ejecutados durante la sesión actual. Además, el panel **History** (parte superior derecha) almacena los comandos de todas las sesiones previas, permitiendo reutilizarlos con un clic en **To Console** (Enter) o **To Source** (Shift + Enter), según se desee insertarlos en la consola o en el script activo.



RStudio es multiplataforma, de código abierto, y permite una integración fluida con herramientas del ecosistema R, como R Markdown, Quarto, control de versiones y manejo de proyectos. Para instalarlo, se puede acceder al siguiente enlace: <https://posit.co/download/rstudio-desktop/>.

#### **i** Note

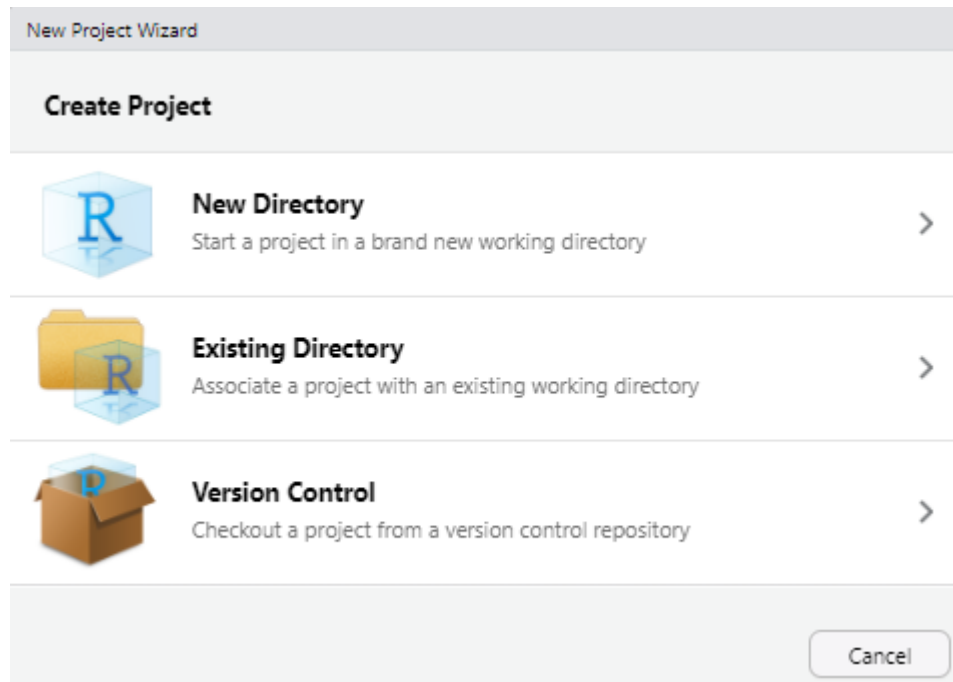
Una vez instalados R y RStudio, ya contamos con todo lo necesario para comenzar a trabajar. Aunque instalamos ambos programas, en la práctica sólo necesitamos abrir **RStudio**, que utiliza a R como motor de ejecución.

## **Proyectos en RStudio**

Los proyectos de RStudio permiten organizar de forma estructurada todo el material asociado a un análisis: scripts, informes, bases de datos, imágenes, etc. Cada proyecto se vincula a una carpeta


específica del sistema de archivos, y RStudio la utiliza como directorio de trabajo por defecto. Esto facilita la importación de datos y evita errores relacionados con rutas relativas o absolutas.

Para crear un nuevo proyecto, se puede utilizar el menú **File > New Project...** o el acceso directo **New Project...** ubicado en la esquina superior derecha de la interfaz. En ambos casos, se abre un asistente con tres opciones:






- **New Directory:** crea una nueva carpeta para el proyecto. Es la opción más habitual.
- **Existing Directory:** vincula el proyecto a una carpeta ya existente que contenga archivos previos.
- **Version Control:** permite clonar un repositorio (Git o SVN). Esta opción no se utilizará en este curso.

Trabajar con proyectos garantiza que, al importar archivos, RStudio los busque automáticamente dentro de la carpeta correspondiente. Además, cada proyecto mantiene su propio entorno de trabajo, lo que significa que al cerrar o cambiar de proyecto, se conserva la configuración previa sin interferencias.

Cuando un proyecto ya existe, dentro de la carpeta encontraremos un archivo con extensión `.Rproj` que al ejecutarlo abre una nueva sesión de RStudio con el proyecto activo. Otras opciones son abrir desde **File > Open Project...** o desde el ícono  **Project: (None)** en la esquina superior derecha de RStudio. Esta última opción también mantiene un historial de los proyectos abiertos recientemente, lo que permite acceder rápidamente a ellos mediante accesos directos.

## Scripts en RStudio

Un script es un archivo de texto plano que contiene instrucciones escritas en R. Permite guardar, reutilizar y compartir el código, favoreciendo la reproducibilidad del análisis.

- **Crear un nuevo script:** podemos crear un script desde el menú **File > New File > R Script** (acceso rápido: Ctrl + Shift + N) o haciendo clic en el ícono de la hoja (  ) con símbolo “+” en la barra de herramientas.
- **Ejecutar código:** la forma habitual de ejecutar un script es línea por línea, con Ctrl + Enter o el botón **Run** (  ). El cursor debe estar en cualquier punto de la línea a ejecutar. Tras la ejecución, el cursor avanza automáticamente a la siguiente línea de código.
- **Editar un script:** las líneas del script pueden editarse directamente. Cada vez que se realiza una modificación, es necesario volver a ejecutar esas líneas para actualizar los resultados.
- **Guardar un script:** Para guardar los cambios, se puede usar el ícono del diskette (  ), el menú **File > Save**, o el atajo Ctrl + S. Para guardar con otro nombre o ubicación, utilizar **File > Save As...**
- **Abrir un script existente:** Los archivos de script tienen extensión .R. Pueden abrirse desde el panel **File > Open File...**, el panel **Files** o usando el atajo de teclado Ctrl + O. Al abrirse, se muestran en una nueva pestaña del editor.

## Funciones

En R, los comandos básicos se denominan **funciones**. Muchas de ellas están incluidas en el núcleo del lenguaje (conocido como **R base**) y se denominan *integradas*, mientras que otras forman parte de paquetes adicionales.

Cada función tiene un **nombre** y suele requerir uno o más **argumentos** (también llamados *parámetros*), que se escriben entre paréntesis y separados por comas. Incluso las funciones que no requieren argumentos deben escribirse con paréntesis vacíos.

```
# Sintaxis general  
nombre_de_la_función(arg1, arg2, ...)
```

Las funciones siempre ejecutan una **acción** o **devuelven un valor**, que puede ser visualizado, almacenado o utilizado en otras operaciones.

### Reglas de sintaxis

Dado que R es un lenguaje interpretado, la **sintaxis debe ser estrictamente correcta**. Algunos puntos clave:

- Los argumentos pueden escribirse con el **nombre del parámetro seguido de un signo igual**:

```
funcion(arg1 = 32, arg2 = 5, arg3 = 65)
```

- También se pueden omitir los nombres y escribir directamente los valores. En ese caso, **el orden importa** y debe coincidir con el definido en la documentación de la función:

```
funcion(32, 5, 65)
```

### Tipos de argumentos

Los argumentos pueden ser:

- Valores numéricos: 3, 10.5
- Lógicos: TRUE, FALSE
- Especiales: NA (faltante), NULL, Inf
- Texto: debe escribirse entre comillas, por ejemplo "menos"
- Objetos: como variables previamente creadas (x, datos, etc.)

```
funcion(arg1 = 3, arg2 = NA, arg3 = TRUE, arg4 = "menos", arg5 = x)
```

### Paquetes

R se compone de un sistema base y de paquetes (*librerías*) que amplían sus funcionalidades. Un **paquete** es una colección de funciones, datos y documentación que extiende las capacidades del lenguaje para tareas específicas.

Existen distintos tipos de paquetes:


- **Base:** se instalan y activan junto con R.
- **Recomendados:** también se instalan por defecto, pero requieren ser cargados manualmente.
- **Adicionales:** más de 17.000 disponibles en el repositorio oficial CRAN, listos para ser instalados según necesidad. Además, algunos paquetes pueden descargarse desde otros repositorios como GitHub y Bioconductor.

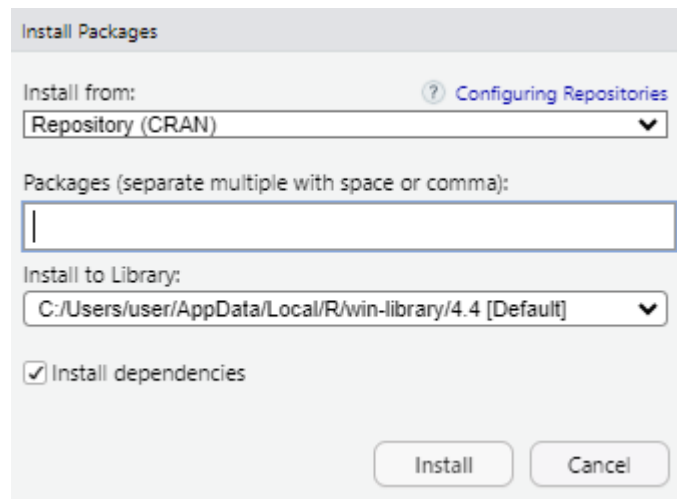
Al ser *open source*, cualquier persona puede desarrollar y publicar nuevos paquetes. Esto convierte a R en una herramienta en constante evolución.

### Instalación

Los paquetes pueden instalarse desde **R o RStudio** o (si no hay acceso a internet o trabajamos con conexiones de uso medido) desde **archivos locales** .zip o .tar.gz, descargados previamente desde CRAN u otros repositorios.

En RStudio, los paquetes se gestionan desde la pestaña **Packages** (bloque inferior derecho). Para instalar uno nuevo:

1. Hacer clic en , se abrirá una ventana emergente:



2. Especificar el nombre del paquete a instalar.
3. Marcar la opción **Install dependencies** para incluir automáticamente sus dependencias.
4. Al presionar el botón **Install**, R internamente traduce esta acción a la función `install.packages()`.

Los paquetes deben instalarse **una única vez por computadora** cuando se los va a utilizar por primera vez. A partir de entonces, sólo es necesario **cargarlos al inicio de cada sesión** mediante la función `library()`:

```
library(nombre_del_paquete)
```

## Dependencias

Muchos paquetes requieren funciones de otros paquetes para funcionar. Estos paquetes (**dependencias**) deben estar instaladas previamente, de lo contrario la ejecución de una función puede fallar por no encontrar otra interna. Por eso, es recomendable dejar seleccionada la opción *Install dependencies* al instalar.

## Paquetes a instalar

Para trabajar durante el curso, deberemos instalar los siguientes paquetes y sus dependencias:

```
# Manejo de datos
install.packages("tidyverse", dependencies = T)

install.packages("janitor", dependencies = T)

# Modelos de meta-análisis
install.packages("metafor", dependencies = T)

install.packages("meta", dependencies = T)

# Paletas aptas para daltonismo
install.packages("scico", dependencies = T)

# Visualización avanzada
remotes::install_github("daniellnoble/orchaRd")
```

## Objetos

En R, los datos, resultados, funciones y estructuras se almacenan en **objetos**, que constituyen la unidad fundamental de trabajo en el lenguaje.

Para **crear un objeto**, se utiliza el operador de asignación `<-` (también se acepta `=` aunque no se recomienda) para asignar un valor a un nombre:

```
x <- 10
```

En este ejemplo, el número 10 se asigna al objeto llamado x. A partir de ese momento, podemos utilizar x en otras operaciones:

```
x + 5 # devuelve 15
```

Los nombres de objetos:

- Deben comenzar con una letra y pueden incluir letras, números, puntos (.) y guiones bajos (\_).
- No deben coincidir con palabras reservadas a funciones del lenguaje.
- Son **sensibles a mayúsculas/minúsculas**: Edad y edad son objetos distintos.

Los objetos *contenedores de datos* más simples pertenecen a **cinco clases** que se denominan *atómicas* y que son los siguientes tipos de datos:

- **integer**: números enteros.



- **numeric**: números reales (también llamados “doble precisión”).
- **complex**: números complejos.
- **character**: cadenas de texto o caracteres.
- **logical**: valores lógicos (TRUE o FALSE).

```
número <- 25           # entero
decimal <- 3.14        # numérico
texto <- "Hola"        # carácter
logico <- TRUE         # lógico (booleano)
```

Además de los tipos atómicos, los datos pueden organizarse en **estructuras contenedoras** que permiten agrupar múltiples valores:

- **Vector**: conjunto de elementos del mismo tipo, ordenados linealmente. Se construye con la función `c()`.
- **Lista**: colección ordenada de objetos de distinto tipo o longitud, creada con `list()`.
- **Dataframe**: estructura bidimensional donde cada columna es un vector del mismo largo (generalmente del mismo tipo). Se construye con `data.frame()` o, en el tidyverse, con `tibble()`.

```
# Vector
vector <- c(1, 2, 3, 4)

# Lista
lista <- list(vector, "elemento_2") # lista

# Dataframe (R base)
dataframe <- data.frame(
  var1 = vector,
  var2 = vector + 5,
  var3 = vector * vector^2
)

# Dataframe (tidyverse)
tibble <- tibble(
  var1 = vector,
  var2 = vector + 5,
  var3 = vector * vector^2
)
```

## Archivos de datos

R permite importar tablas de datos desde diversos formatos, tanto utilizando funciones de **R base** como funciones provistas por paquetes específicos.

El formato más común es el **texto plano** (ASCII), donde los valores están organizados en columnas separadas por caracteres delimitadores. Los separadores más habituales incluyen:

- Coma (,)
- Punto y coma (;)
- Tabulación (\t)
- Barra vertical (|)

Estos archivos suelen tener una **cabecera** (header) en la primera fila con los nombres de las variables, y cada columna debe contener datos del mismo tipo (números, texto, lógicos, etc.).

Para importar correctamente un archivo es importante conocer su estructura:

- Si incluye o no cabecera.
- Qué carácter se usa como separador.
- El tipo de codificación (UTF-8, Latin1, etc.).

Dado que son archivos de texto, pueden visualizarse con editores simples como el Bloc de Notas o desde RStudio, lo que facilita su inspección previa.

Para cargar los datos desde un archivo de texto plano o una hoja de cálculo de Excel usamos el código:

```
datos <- read.xxx("mis_datos.txt")
```

(Se debe reemplazar `read.xxx()` por la función correspondiente: `read.table()`, `read.csv()`, `read_delim()`, `read_excel()`, etc., según el caso).

R también permite cargar bases de datos incluidas en paquetes instalados mediante:

```
data(nombre_datos)

datos <- nombre_datos
```

## Buenas prácticas

Adoptar buenas prácticas desde el inicio mejora la reproducibilidad, facilita el trabajo colaborativo y reduce errores. Algunas recomendaciones clave son:

- Trabajar siempre dentro de un **proyecto de RStudio** (`.Rproj`). Esto permite organizar los archivos, mantener rutas relativas consistentes y acceder a funcionalidades específicas como control de versiones o panel de archivos integrados.
- Incluir al comienzo de cada script las líneas de **activación de paquetes** necesarios, utilizando la función `library()`.

- **Cargar los datos** una vez activados los paquetes, para garantizar que todas las funciones requeridas estén disponibles.
- Documentar el código mediante **comentarios** iniciados con `#`. Esto permite entender qué hace cada bloque de código, facilitando futuras modificaciones o revisiones.
- **Usar espacios e indentación adecuada** para mejorar la legibilidad. Esto es especialmente importante en estructuras anidadas (como condicionales, bucles o funciones).

Una guía de estilo ampliamente recomendada —aunque no oficial— es la de `tidyverse`. Incluye ejemplos concretos de buenas y malas prácticas para nombrar variables, manejar líneas largas, usar sangrías, entre otros aspectos. Puede consultarse en: <https://style.tidyverse.org/>

#### Importante

Este apunte ofrece un resumen general para quienes deseen repasar los aspectos básicos de R y RStudio.

Si no cuentan con experiencia previa en R y necesitan una introducción más detallada, podés consultar los siguientes recursos:

- **Curso de Epidemiología Nivel Avanzado** - Unidad 1: Introducción a R.
- **EpiR Handbook** – secciones Aspectos básicos y Gestión de datos.

Ante cualquier duda específica, recuerden que pueden comunicarse con los/as docentes del curso.

## Bibliography

- [1] R. C. T., “R: A Language and Environment for Statistical Computing,” 2025, [Online]. Available: <https://www.r-project.org/>