

---

## Primeros pasos con R y RStudio

---

Métodos Numéricos y Estadísticos

Eva María Mazcuñán Navarro



---

## Contenidos

---

	Página
<b>1 Requisitos previos</b>	<b>2</b>
<b>2 Interfaz de RStudio</b>	<b>2</b>
<b>3 Escribir y ejecutar código de R</b>	<b>3</b>
3.1 Consola . . . . .	4
3.2 Scripts de R . . . . .	5
3.3 Documentos R Markdown . . . . .	7
<b>4 Plantilla de R Markdown</b>	<b>15</b>
4.1 Primeros contenidos . . . . .	15
4.2 Algunas opciones . . . . .	18
4.3 Opciones globales . . . . .	18
4.4 Tabla de contenidos flotante . . . . .	19
4.5 Plantilla final . . . . .	20
<b>5 Flujo de trabajo</b>	<b>21</b>
<b>6 Paquetes</b>	<b>21</b>
6.1 Instalar un paquete . . . . .	22
6.2 Cargar un paquete . . . . .	22
<b>7 Objetos en R</b>	<b>23</b>

7.1	Crear un objeto . . . . .	23
7.2	Vectores . . . . .	24
7.3	Hojas de datos . . . . .	25
7.4	Funciones . . . . .	27
<b>8</b>	<b>Visualización de datos</b>	<b>29</b>
<b>9</b>	<b>Crear hojas de datos</b>	<b>32</b>
9.1	La función <code>tibble()</code> . . . . .	33
9.2	Importar datos . . . . .	34

En este documento se da una breve introducción al lenguaje de programación R y a la IDE (*Integrated Development Environment*) RStudio.

Se trata de una práctica introductoria, en la que no hay que entregar ningún material, y no hay ninguna tarea evaluable asociada. Pero es necesario que estudies este material como previo a la realización de las siguientes prácticas y tareas.

## 1. Requisitos previos

Antes de realizar esta práctica tienes que instalar R y RStudio en tu equipo.

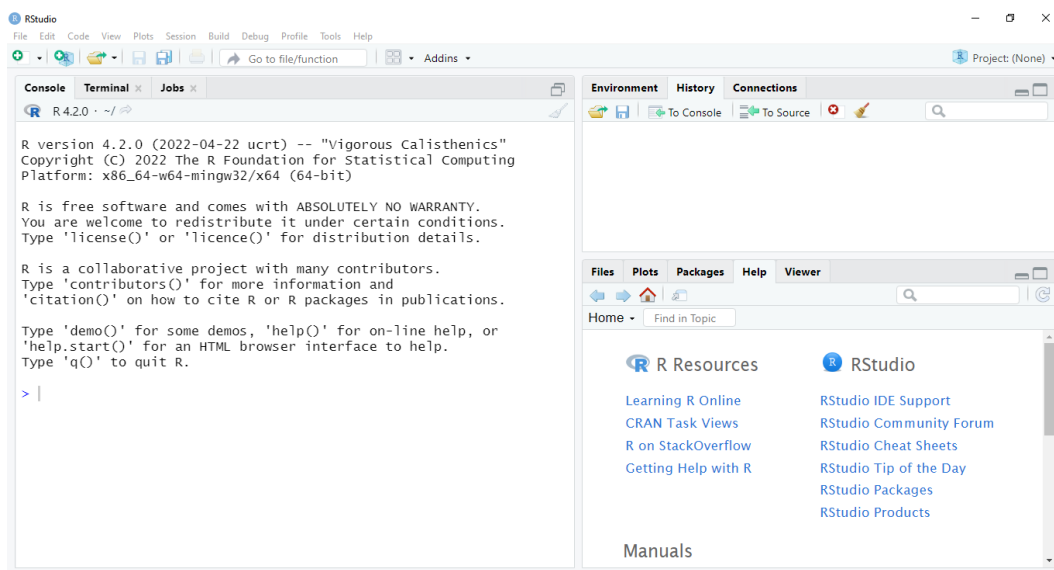



En el documento [Instalación de R y RStudio](#) encontrarás las instrucciones para hacerlo.

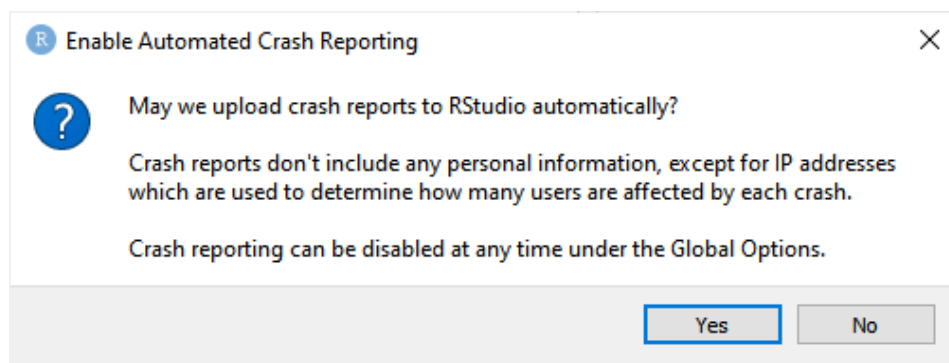
## 2. Interfaz de RStudio

Al abrir RStudio verás una pantalla como la de la imagen a continuación.

La interfaz está dividida en varias regiones con varias pestañas o paneles. Empezaremos usando la consola, que es el panel en la parte izquierda de la pantalla de nombre **Console**. Más adelante exploraremos nuevos paneles y veremos su utilidad.

**Figura 1:** Interfaz de RStudio

 La primera vez que abras RStudio aparecerá un cuadro de diálogo preguntando si deseas habilitar el envío automático de informes de fallos.

**Figura 2:** Solicitud de envío automático de informes de fallos de RStudio

Puedes realizar la elección que desees, teniendo en cuenta que no perderás ninguna funcionalidad si no lo habilitas.

### 3. Escribir y ejecutar código de R

Para escribir y ejecutar código de R desde RStudio podremos utilizar bien la consola o bien escribir nuestro código en un archivo.

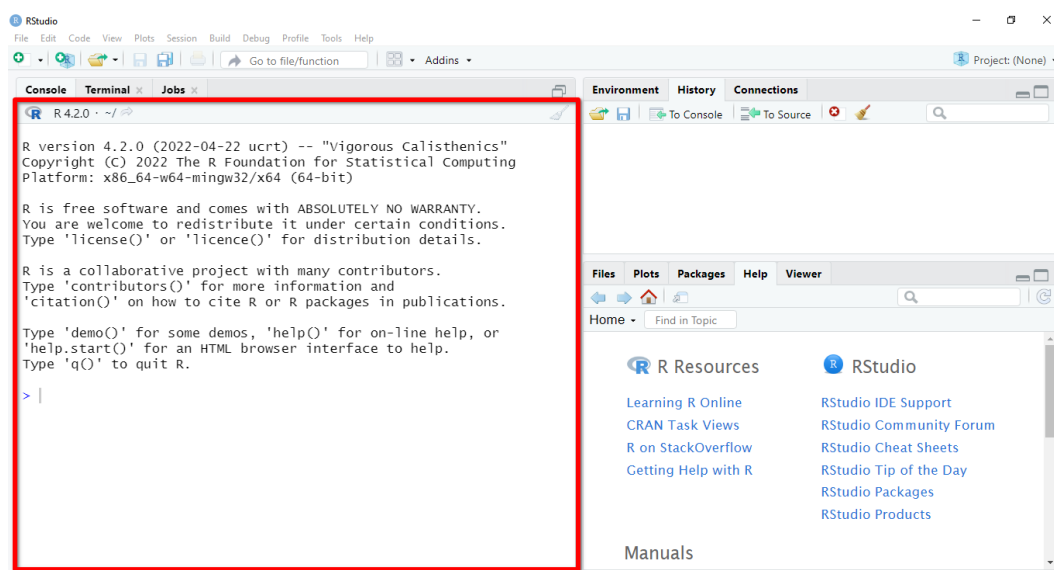
La **consola** se utiliza para ejecutar instrucciones sueltas, que no tenemos interés en conservar, por ejemplo para realizar cálculos auxiliares o acciones que se realizan una única vez como instalar un determinado paquete.

Cuando el código que queremos escribir sea un conjunto de instrucciones que queramos conservar, de forma que podamos reutilizarlo posteriormente o compartirlo con otras personas, escribiremos nuestro código en un archivo. Entre los tipos de archivos que podemos crear desde RStudio para escribir y ejecutar código R están los **scripts** y los documentos **R Markdown**.

En los siguientes apartados exploraremos las tres opciones presentadas para escribir código R: ejecutar instrucciones desde la consola, escribir código en un script, y elaborar documentos R Markdown. Nuestra elección final para realizar las prácticas y tareas de esta asignatura serán los documentos R Markdown.

### 3.1. Consola

En RStudio encontrarás la consola en el panel de nombre Console en la ventana a la izquierda de la pantalla.



**Figura 3:** Consola de RStudio

Para calcular  $\sqrt{2}$  desde la consola, sitúa el cursor al lado del símbolo `>` en la consola, escribe la instrucción `sqrt(2)` y presiona **Enter**. Verás la salida debajo:

```
> sqrt(2)
```

```
[1] 1.414214
```

El símbolo [1] antes del valor 1.4142136 no es importante por ahora, simplemente indica que la salida de nuestra instrucción está compuesta por un único elemento.



Si quieres borrar las instrucciones de la consola, utiliza el botón con icono



en la barra de título del panel (o el atajo **Ctrl + L**).

## 3.2. Scripts de R

Los scripts son el tipo de archivo más simple para escribir y ejecutar código de R.

### 3.2.1. Creación

Para crear un script, utiliza el menú

**File > New File > R Script**

El script se abrirá en una pestaña de una nueva ventana sobre la ventana con el panel de la consola. Este script no es más que un archivo de texto, que se guardará con la extensión **.R**.

### 3.2.2. Ejecución

Escribe en la primera línea del script la misma instrucción que escribimos antes en consola para calcular  $\sqrt{2}$ :

```
sqrt(2)
```

Para ejecutarla, sitúa el cursor sobre cualquier punto de la línea y presiona **Ctrl + Enter**. Verás la salida en la consola.



Si en un script queremos incluir varias instrucciones, cada nueva instrucción debe comenzar en una nueva línea.

Añade dos nuevas líneas al script: `5^3` para calcular  $5^3$ , y `exp(3)` para calcular  $e^3$ , de forma que el contenido del script quede:

```
sqrt(2)
5^3
exp(3)
```



Notar que para calcular  $5^3$  hemos escrito `5^3` y que para calcular  $e^3$  hemos escrito `exp(3)`. Esto es así porque R no tiene predefinida una constante con el valor del número  $e$ , pero sí la función exponencial `exp()`, que calcula  $e$  elevado a su argumento.

Si necesitamos el número  $e$  se obtendría con `exp(1)`. De hecho serían equivalentes `exp(3)` y `exp(1)^3`.

Para ejecutar las tres instrucciones al mismo tiempo, selecciona las tres líneas y presiona de nuevo **Ctrl + Enter**. En la consola, verás las tres instrucciones y su salida correspondiente.



Pueden dejarse tantas líneas en blanco como se quiera entre diferentes instrucciones, y también incluir espacios alrededor de los argumentos de una función e incluso separarlos en varias líneas (tiene sentido para funciones con más de un argumento). Por ejemplo:

```
sqrt(
  2
)

5^3

exp( 3 )
```

Notar que si situamos el cursor sobre cualquiera de las tres líneas que componen la primera instrucción para calcular la raíz de 2 y presionamos **Ctrl + Enter**, RStudio reconoce que la línea en la que tenemos el cursor forma parte de una instrucción compuesta por varias líneas y ejecuta todas ellas.

### 3.2.3. Comentarios

Para añadir comentarios en un script, se utiliza el carácter `#`: Al ejecutar una línea de código, todo el texto escrito en la línea después del carácter `#` será ignorado. Puedes escribir por ejemplo

```
# calcular raíces  
sqrt(2)
```

o

```
sqrt(2) # calcular la raíz de  
5^3     # calcular 5 elevado al cubo  
exp(3)  # calcular e elevado al cubo
```



Para comentar varias líneas hay que comentar cada una de ellas añadiendo `#` al principio de cada una (no hay sintaxis especial para comentar varias líneas).

Ahora, en lugar de hacerlo manualmente línea por línea, puedes seleccionar las líneas que quieras comentar y usar el menú

**Code > Comment/Uncomment Lines**

### 3.2.4. Guardado

Crea ahora una carpeta, de nombre `IntroR`, para guardar el script que acabas de escribir y otros documentos que generaremos a lo largo de la práctica.

Para guardar el script que acabas de escribir, presiona `Ctrl + S`. También



puedes utilizar el icono en la barra de herramientas del archivo o el menú

**File > Save As ...**

Si aparece un cuadro de diálogo preguntando por la codificación del archivo, selecciona la codificación que aparezca listada en primer lugar como defecto para tu sistema operativo (verás el texto (System default) al lado de su nombre).

En el selector de archivos que se abrirá a continuación, navega hasta la carpeta `IntroR` que has creado antes e indica `script` como nombre del archivo (la extensión `.R` se añadirá automáticamente). Verás entonces que la etiqueta de la pestaña del script en el panel de RStudio cambia de `Untitled1` a `script.R`.

## 3.3. Documentos R Markdown

Nuestra elección para realizar las prácticas y tareas de esta asignatura será utilizar documentos R Markdown. En un documento R Markdown podremos

escribir tanto código R como texto. Y al compilarlo obtendremos un documento que incluirá el código, la salida resultante de ejecutar dicho código, y el texto explicativo que hayamos escrito.

- En el texto podremos utilizar la sintaxis propia del lenguaje de marcado [Markdown](#) (independiente de R). Por ejemplo, para escribir texto negrita se escribe **\*\*** al comienzo y al final del texto en cuestión:

```
El resultado anterior permite extraer una conclusión muy
**importante**.
```

- Y el código de R se incluirá en unos bloques especiales, que en su forma más simple tendrán la estructura

```
```{r}
<código R>
```
```

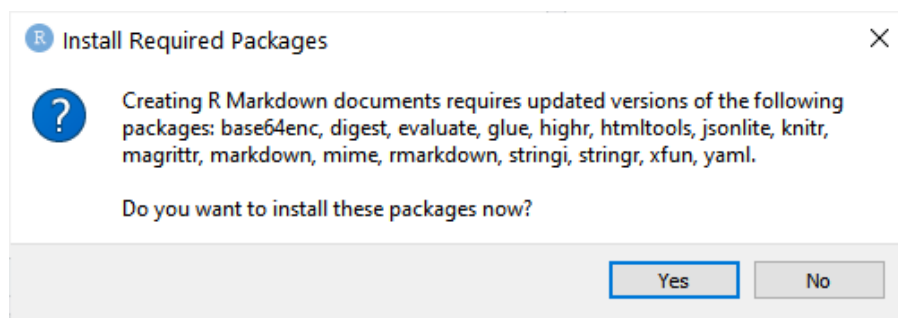
### 3.3.1. Creación

Para crear tu primer documento R Markdown utiliza el menú

**File > New File > R Markdown ...**



Es probable que la primera vez que crees un documento R Markdown, aparezca una ventana como la de la siguiente imagen, solicitando permiso para instalar varios paquetes requeridos para usar este tipo de documentos.

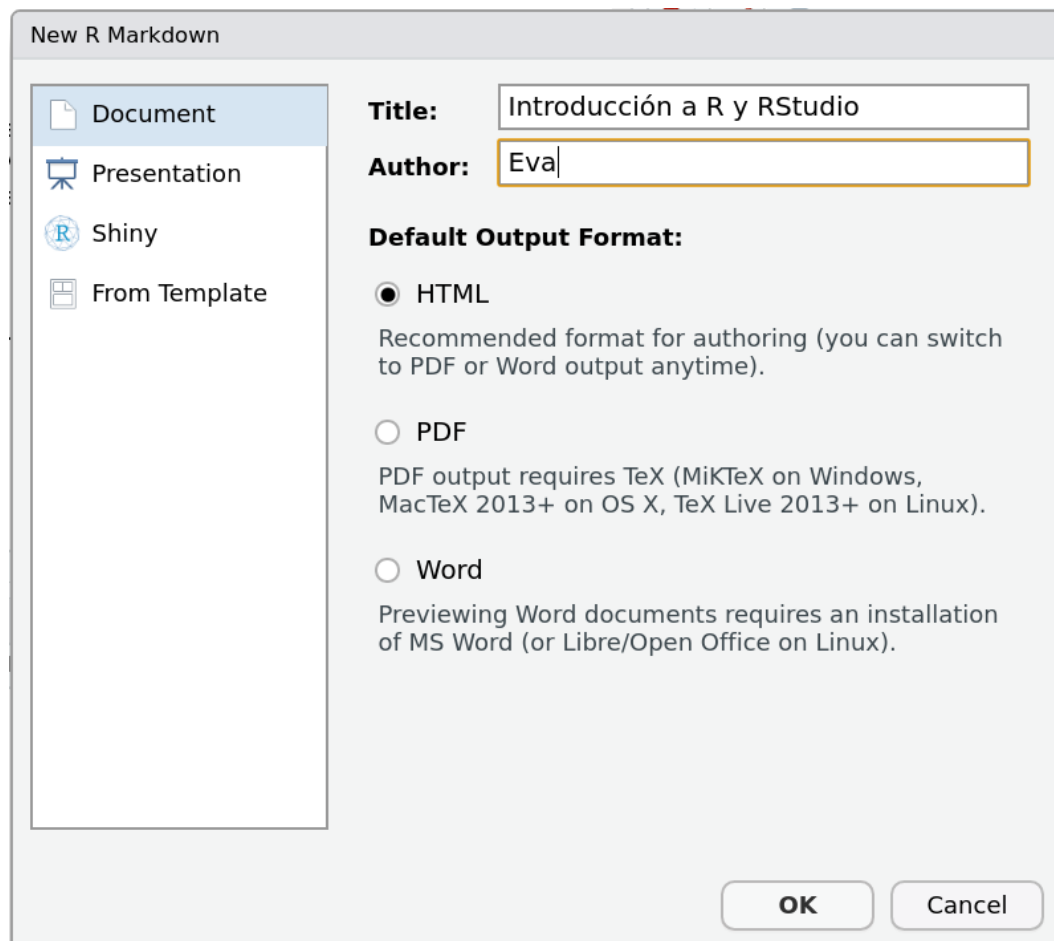


**Figura 4:** Instalar paquetes requeridos para R Markdown

Presiona el botón **Yes** para que se inicie el proceso de instalación de las dependencias indicadas y espera hasta que se complete la tarea. Tomará foco el panel de nombre **Jobs**, al lado de la consola y verás una barra indicando el progreso de la instalación.



Aparecerá entonces un cuadro de diálogo de nombre **New R Markdown**.



**Figura 5:** Nuevo documento R Markdown

Rellena ‘Introducción a R y RStudio’ en el campo **Title** y tu nombre en el campo **Author**. Para el campo **Default output format**, conserva la elección ‘HTML’ que aparece por defecto.

Al presionar el botón **OK** se abrirá una nueva pestaña en el panel de RStudio con el nuevo documento R Markdown.

### 3.3.2. Guardado

Presiona **Ctrl + S** para guardarlo, en la carpeta **IntroR** que creaste antes para la práctica, con el nombre **intro-r**. Verifica que la etiqueta de la pestaña del documento cambia de **Untitled1** a **intro-r.Rmd**.

### 3.3.3. Estructura

Las primeras líneas del archivo (1 a 6), delimitadas por tres guiones (---)

```
---  
title: "Introducción a R y RStudio"  
author: "Eva"  
date: "19/4/2021"  
output: html_document  
---
```

conforman la llamada **cabecera YAML** del documento. Incluye metadatos como el título, el autor y la fecha y el formato de salida del documento que se generará al compilar.

Las líneas siguientes (8 a 10)

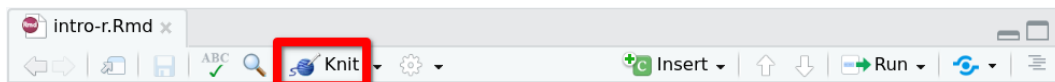
```
```${r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE)  
```
```

fijan opciones globales para los bloques de código de todo el documento. Las analizaremos con más detalle en la sección 4.3.

El resto de líneas (12 en adelante), son los contenidos propiamente dichos del documento. Se trata de unos contenidos de muestra, que enseguida reemplazaremos por nuestros contenidos propios. Pero antes de borrar estos contenidos de muestra, compilaremos el documento para ver el resultado inicial.

### 3.3.4. Compilación

Para compilar el documento presiona el botón **Knit** en la barra de herramientas de la pestaña del archivo.



**Figura 6:** Botón **Knit** para compilar un documento R Markdown

Tomará foco el panel de nombre **Render**, en la misma ventana que la consola. Al terminar el proceso, el documento compilado aparecerá en el panel **Viewer**, en la ventana de la zona derecha inferior.



Si al compilar el documento no se abre en el panel Viewer, sino en una ventana emergente, cierra esa ventana y modifica este comportamiento siguiendo los siguientes pasos:

1. Selecciona el menú

**Tools > Global Options**

2. Se abrirá una ventana de nombre **Options**. Selecciona la sección **R Markdown** en el menú lateral e indica

**Show output preview in: Viewer Pane**

como indica la imagen siguiente:

Cierra el diálogo presionando **OK** y vuelve a compilar.

Si miras los contenidos de la carpeta **IntroR** (puedes hacerlo desde el panel **Files** de RStudio) verás que, como resultado de la compilación del archivo fuente `intro-r.Rmd`, se ha creado el archivo de salida `intro-r.html`. Éste es el archivo que estamos visualizando en el visor de documentos. También podríamos abrirlo en el navegador web, pudiendolo hacer desde el propio visor, presionando el icono resaltado en la siguiente imagen:

A continuación compararemos el documento fuente `intro-r.Rmd` con el documento `intro-r.html` en el visor, para entender cómo se traducen los contenidos que escribimos en un archivo R Markdown en el formato de salida HTML. Nos fijaremos en particular en los siguientes elementos: encabezados y bloques de código, que se describen en los dos siguientes apartados.

### 3.3.5. Encabezados

Las líneas 12

```
## R Markdown
```

y 22

```
## Including Plots
```

se traducen en la salida como encabezados de secciones. Si inspeccionas el código del archivo `intro-r.html` verás que se crean elementos de tipo `<h2>`. En general,

```
# Título
```

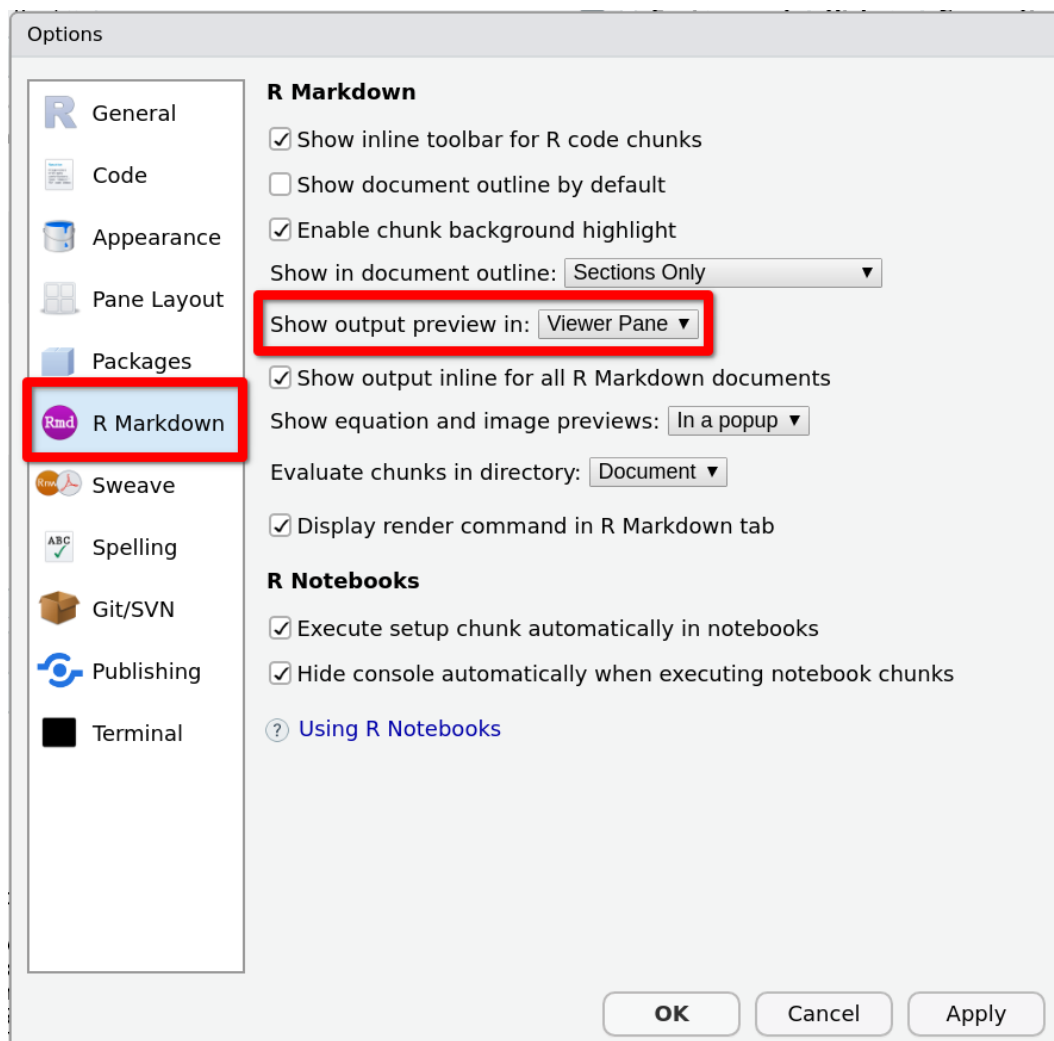


Figura 7: Ajuste para mostrar el documento compilado en el panel del visor



Figura 8: Abrir documento en visor externo

produce un encabezado de nivel 1,

```
## Título
```

un encabezado de nivel 2, y así sucesivamente.



Hay que indicar que es una casualidad la coincidencia del símbolo # para encabezados en el lenguaje Markdown (independiente de R ) y para comentarios en código R.

### 3.3.6. Bloques de código

Los elementos protagonistas del documento son los bloques de código R (*code chunks*). En el documento de muestra encontramos dos bloques de código: El primero en las líneas 18-20

```
```{r cars}
summary(cars)
```
```

y el segundo en las líneas 26-28

```
```{r pressure, echo=FALSE}
plot(pressure)
```
```

Si miras el documento compilado, verás que, para el primer bloque se muestra el código y a continuación la salida o resultado de su ejecución; mientras que para el segundo, se muestra solo la salida, y no el código. Esto es debido a la opción `echo=FALSE`.

La sintaxis general para incluir un bloque de código R en un documento R Markdown es la siguiente:

```
```{r [etiqueta], [lista de opciones]}
<código R>
```
```

Las etiquetas de los dos bloques de código del documento de muestra son `cars` y `pressure`. La etiqueta de un bloque de código sirve para identificarlo, podemos interpretarlo como su nombre, pero es opcional y puede omitirse.

El primer bloque de código no tiene ninguna opción. Y el segundo tiene la opción `echo=TRUE`, que como acabamos de decir inhibe la impresión del código

en el documento compilado.

Puesto que la etiqueta y la lista de opciones son opcionales, el esqueleto básico de un bloque de código R incluido en un documento R Markdown es

```
```${r}  
<código R>  
```
```

En el cuerpo del bloque podemos escribir instrucciones de R igual que si estuviéramos escribiendo en un script (incluidos comentarios precedidos por el carácter #).



Notar el cambio de enfoque al escribir en un documento R Markdown respecto a escribir en un script:

- En un script, se espera que escribamos código R, y para escribir texto ordinario hemos de usar comentarios utilizando el carácter #.
- Por el contrario, en un documento R Markdown, se espera que escribamos texto (con posibilidad de incluir la sintaxis propia del lenguaje Markdown), y para escribir código R hemos de incluirlo en un bloque especial delimitado por la línea ````${r}` al comienzo y la línea ````` al final.

### 3.3.7. Ejecución de instrucciones individuales

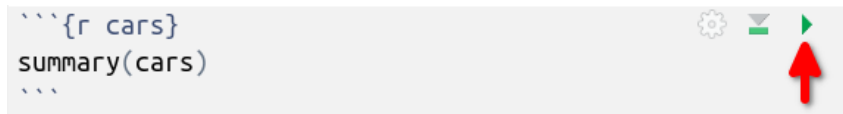
Cuando compilamos un documento R Markdown, se ejecutan todos los bloques de código que contenga, y en el documento compilado podemos visualizar, tanto el código como la salida o resultado (siempre que no hay opciones como `echo=FALSE` o `include=FALSE` que inhiban la impresión del código y/o de la salida).

Pero también podemos ejecutar determinadas instrucciones de forma individual, sin necesidad de compilar el documento completo. Para ello, podemos proceder exactamente igual que en el caso de los scripts, es decir:

- Para ejecutar una sola instrucción, situamos el cursor en cualquiera de las líneas que compongan la instrucción y presionamos **Ctrl + Enter**.
- Para ejecutar varias instrucciones, seleccionamos las correspondientes líneas y presionamos **Ctrl + Enter**.

Además, podemos ejecutar todas las instrucciones que componen un bloque de código utilizando el botón a la derecha del comienzo del bloque que se resalta

en la siguiente imagen:



**Figura 9:** Botón para ejecutar un bloque de código

La salida se mostrará en la consola, y también incrustada en el propio documento, justo debajo del correspondiente bloque de código. Para esto último ha de estar marcada la opción **Show output inline for all R Markdown documents** en las opciones para R Markdown en el menú

**Tools > Global options ... > R Markdown**

como muestra la siguiente imagen:

## 4. Plantilla de R Markdown

En el capítulo anterior hemos explorado los contenidos de muestra del archivo de R Markdown que hemos creado, y conocemos los dos elementos principales a incluir en este tipo de documentos:

- los **encabezados**, para estructurar nuestro documento en capítulos, secciones, subsecciones ...
- y los **bloques de código**, para incluir instrucciones de R.

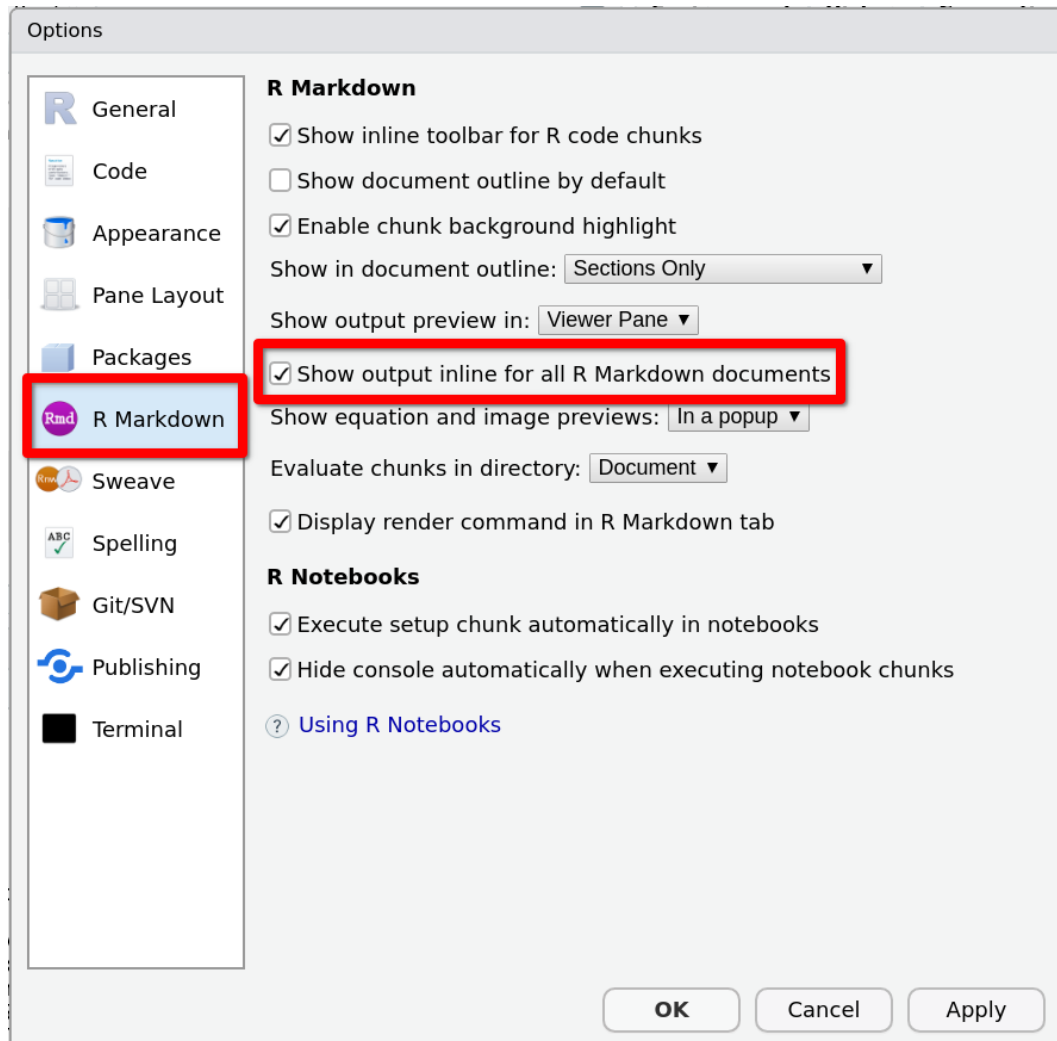
Ahora vamos a reemplazar los contenidos de muestra por nuestros propios contenidos. Crearemos un primer capítulo con un primer bloque de código, y personalizaremos algunas opciones.

Al final de este capítulo tendremos una plantilla para los documentos R Markdown que usaremos en las prácticas y tareas a lo largo del curso.

Y nuestro documento `intro-r.Rmd` quedará preparado para practicar el código de R que se presenta en los siguientes capítulos de esta práctica.

### 4.1. Primeros contenidos

Borra los contenidos de muestra (línea 12 en adelante) y añade la siguiente línea para crear un primer capítulo (encabezado de nivel 1) de título “Bloques de código”:



**Figura 10:** Ajuste para mostrar la salida de un bloque de código *\*inline\**



## # Bloques de código

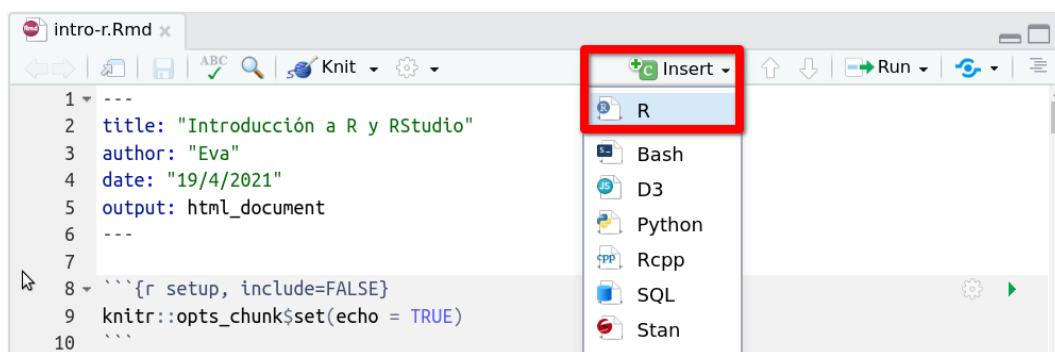
Ahora vamos a crear el primer bloque de código. Para escribir su esqueleto usa el atajo

Ctrl + Alt + I

(I de *Insert*) o, alternativamente, el menú

**Insert > R**

en la barra de herramientas de la pestaña del documento, que se muestra en la siguiente imagen:



**Figura 11:** Insertar bloque de código

En el bloque de código que acabas de crear añade las mismas instrucciones que habíamos incluido en el script:

```
sqrt(2)
5^3
exp(3)
```

de forma que los contenidos añadidos queden:

# Bloques de código

```
```{r}
sqrt(2)
5^3
exp(3)
```
```

Compila para ver el resultado.

## 4.2. Algunas opciones

Tras compilar el documento con los contenidos añadidos en la sección anterior, verás que en el documento HTML generado aparecen:

- El código de la primera instrucción `sqrt(2)`
- Su salida 1.4142136
- El código de la segunda instrucción `5^3`
- Su salida 125
- El código de la tercera instrucción `exp(3)`
- Su salida 20.0855369

Para que se muestre primero el código para las tres instrucciones y a continuación las dos salidas, añade la opción `results='hold'`.

Y para omitir los caracteres `##` al comienzo de las líneas de la salida, añade la opción `comment = ''`. Nuestro bloque de código con estas dos opciones quedaría:

```
```${r, results='hold', comment = ''}  
sqrt(2)  
5^3  
exp(3)  
```
```

Vuelve a compilar y observa el resultado.

## 4.3. Opciones globales

Al comienzo de nuestro documento, hemos conservado el bloque de código

```
```${r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE)  
```
```

que estaba incluido en el documento de muestra. Este bloque se identifica con la etiqueta `setup` y tiene la opción `include=FALSE`, que hace que, al compilar el documento, si bien se ejecutará la instrucción que contiene, no se incluirá en el formato HTML de salida.

Se explica a continuación el significado de la instrucción `knitr::opts_chunk$set(echo = TRUE)` en este bloque de código: Las opciones especificadas como argumentos de `knitr::opts_chunk$set` (por ahora `echo=TRUE`) aplicarán a todos los

bloques de código que se incluyan en el documento.

Las opciones `results='hold'` y `comment = ''` que hemos aplicado antes a nuestro bloque de código tendrían un efecto local, es decir, aplicarían solo al bloque en el que se han especificado, y si queremos usarlas en los nuevos bloques que creemos, habría que escribirlas de nuevo en todos ellos.

Para aplicar las opciones `results='hold'` y `comment = ''` a todos los bloques de código del documento, las añadiremos como argumentos de `knitr::opts_chunk$set`, de forma que quede:

```
knitr::opts_chunk$set(  
  echo = TRUE,  
  results='hold',  
  comment = ''  
)
```

Ahora, estas dos nuevas opciones aplicarán a todos los bloques, sin necesidad de repetirlas de forma individual en cada uno de ellos, así que puedes borrarlas del bloque de código que creaste antes.

## 4.4. Tabla de contenidos flotante

Ahora vamos a personalizar el formato de salida para que nuestro documento incluya una tabla de contenidos flotante con las secciones numeradas.

Para ello sustituimos la línea

```
output: html_document
```

en la cabecera YAML por

```
output:  
  html_document:  
    toc: true  
    toc_float: true  
    number_sections: true
```

Asegúrate de indentar las líneas conforme se indica, porque el indentado es fundamental para que los campos anidados se lean e interpreten correctamente en el proceso de compilación.

Para que nuestra tabla de contenidos tenga más de una entrada, añade al final del documento un segundo capítulo, con el título de otro capítulo de esta

práctica:

```
# Paquetes
```

Compila de nuevo y abre el resultado en el navegador. Verás la tabla de contenidos flotante a la izquierda del cuerpo del documento (o encima del título en pantallas de dimensiones reducidas).

Para apreciar la funcionalidad de la tabla de contenidos flotante, reduce la altura de la ventana del navegador hasta que sea inferior a los contenidos en el cuerpo del documento y aparezca la barra de *scroll* para recorrerlo. Verás que las entradas de la tabla de contenidos actúan como enlaces al inicio de cada capítulo.

## 4.5. Plantilla final

Después de los cambios que hemos ido haciendo en el documento de muestra, ha debido quedarte así:

```
---  
title: "Introducción a R y RStudio"  
author: "Eva"  
date: "19/4/2021"  
output:  
  html_document:  
    toc: true  
    toc_float: true  
    number_sections: true  
---  
  
```${r setup, include=FALSE}  
knitr::opts_chunk$set(  
  echo = TRUE,  
  results='hold',  
  comment = ''  
)  
---  
  
# Bloques de código  
  
```${r}  
sqrt(2)
```

```
5^3  
exp(3)  
...
```

```
# Paquetes
```

La primera parte del documento, con la cabecera YAML y el bloque de nombre `setup` con las opciones de configuración globales para los bloques de código, la repetiremos en todos los documentos R Markdown a lo largo del curso.

## 5. Flujo de trabajo

La idea es que, conforme vayas estudiando el resto de la práctica, continúes escribiendo en el documento R Markdown que tienes ahora, después de leer el capítulo anterior.

Así, al terminar de estudiar la práctica, tendrás un informe o resumen de la misma, incluyendo las instrucciones de R que hayas aprendido, los resultados obtenidos, y los comentarios explicativos que hayas añadido.

Incluye bloques de código para practicar el código de R que vayas encontrando en la práctica, así como el texto que creas oportuno para documentar el código que has escrito, de forma que puedas entenderlo cuando releas el documento. En cuanto al seccionado del documento, puedes crear un nuevo capítulo por cada capítulo de la práctica.

Experimenta, escribiendo el código que te apetezca para probar las ideas que te vayan surgiendo y escribiendo el texto que consideres para explicarlas.

Puedes ir compilando cada bloque o instrucción de forma individual, para ver su salida *inline*, y compilar el documento completo cada cierto tiempo.

## 6. Paquetes

Cuando instalamos R por primera vez estamos instalando el software **base R**, que contiene las funciones básicas que se usan regularmente en la mayoría de análisis estadísticos.

Un **paquete** de R es una colección de funciones, datos y documentación, que extiende las capacidades iniciales de la base de R. Existen multitud de paquetes

de R, en su mayoría desarrollados por la comunidad de usuarios de R, que añaden funciones para propósitos específicos.

**tidyverse** es una colección de paquetes para ciencia de datos que comparten una filosofía común y están diseñados para funcionar juntos de forma natural. La mayoría de paquetes que usaremos a lo largo del curso forman parte de **tidyverse**, de forma que instalando este *meta-paquete* tendrás acceso a todos ellos.

## 6.1. Instalar un paquete

Al instalar un paquete descargamos el código que compone dicho paquete en nuestro equipo. Para instalar un paquete nuevo se utiliza la función `install.packages()`. La instalación de un paquete se realiza una sola vez, así que la usaremos desde la consola. El paquete **tidyverse** se instalaría ejecutando la siguiente instrucción:

```
install.packages("tidyverse")
```

La instalación de este paquete puede llevar unos 5-10 minutos, hay que esperar hasta ver en la consola de RStudio que se ha completado el proceso.



**Nota para Ubuntu:** Si obtienes error al intentar instalar **tidyverse** en Ubuntu, y el mensaje de la salida contiene la siguiente línea:

```
Package libcurl was not found in the pkg-config search path.
```

puedes solucionarlo siguiendo los siguientes pasos:

1. Abre la terminal de Ubuntu y ejecuta

```
sudo apt install libcurl4-openssl-dev libssl-dev  
libxml2-dev
```

2. En la consola de RStudio vuelve a solicitar la instalación de **tidyverse** ejecutando

```
install.packages("tidyverse")
```

## 6.2. Cargar un paquete

Para cargar un paquete previamente instalado hay que usar la función `library()`. En particular, para cargar el paquete **tidyverse** escribimos

```
library(tidyverse)
```



Recuerda que en tu documento R Markdown las instrucciones de R se escriben en bloques de código delimitados por la línea ````\r{}` al comienzo y la línea ````` al final. Según el flujo de trabajo que se ha explicado antes, la idea es que, después de leer el párrafo anterior, escribas en tu documento algo como

Cargamos el paquete ``tidyverse``.

```
```\r{  
library(tidyverse)  
```
```

**Nota:** Al escribir ``tidyverse`` se consigue fuente monoespaciada en el documento de salida.

Puedes ignorar los conflictos que se listan en el mensaje que se muestra al cargar `tidyverse`, no tendrán efectos negativos sobre nuestro código.

Si el paquete está instalado, la instrucción anterior lo cargará y podremos hacer uso de sus funciones.

En caso de no estar instalado, recibiremos un mensaje de error indicando que no existe el paquete solicitado, y habrá que instalarlo primero, como se explicó en el apartado anterior.

Si bien instalación de un paquete (con `install.packages()`) se realiza una sólo vez, la carga (con `library()`) es necesaria en cada sesión en la que queramos usar funciones del paquete.

## 7. Objetos en R

En este capítulo veremos cómo crear un objeto en R, estudiaremos dos tipos de objetos destacados en R: los **vectores** y las **hojas de datos**, y aprenderemos a crear nuestras propias **funciones**.

### 7.1. Crear un objeto

La instrucción

```
x <- 2
```

crea una variable u objeto de nombre `x` y le asigna el valor 2. El operador de asignación en R es `<-` (compuesto por el símbolo menor `<` y un guión central `-`).

Puedes comprobar que, después de ejecutar la instrucción anterior, el objeto `x` aparece listado en el panel *Environment* de la ventana superior derecha de RStudio. En este panel podemos ver todos los objetos definidos en nuestro entorno o espacio de trabajo (*workspace*).

Cuando se ejecuta una instrucción como la anterior, en la que se asigna un valor o expresión a un objeto, la salida no muestra el valor asignado. Para verlo tenemos que *imprimir* el objeto en cuestión, simplemente escribiendo su nombre:

```
x # muestra el valor actual de x
```

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



Otra alternativa para mostrar el valor de un objeto es delimitar la instrucción con la asignación entre paréntesis:

```
(x <- 2) # asigna el valor 2 a la variable x y lo imprime
```

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

Una vez definida el objeto o variable `x` podremos referenciarlo en cálculos posteriores. Por ejemplo, al ejecutar la instrucción

```
x^2 - 3
```

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

obtendremos la salida 1 (resultado de  $2^2 - 3 = 1$ ).

## 7.2. Vectores

Para crear **vectores** en R usamos la función `c` (inicial de *concatenate* o *combine*). La instrucción

```
v <- c(1, 3, 5, 7, 9)
```



crea un vector de nombre `v` formado por los primeros 5 números impares. Observa la entrada que se ha creado para `v` en el panel Environment.

Para acceder al elemento número `i` del vector `v` se escribe `v[i]` (ojo, que el primer índice es 1, no 0). Por, ejemplo para extraer el segundo elemento de `v` escribimos

```
v[2]
```

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

La instrucción

```
v^2
```

```
## [1] 1 9 25 49 81
```

devuelve el vector cuyos elementos son los cuadrados de los elementos de `v`. La mayoría de funciones de R que reciben argumentos numéricos aplicadas a un vector numérico operan sobre cada elemento del vector (se dice que están vectorizadas).

## 7.3. Hojas de datos

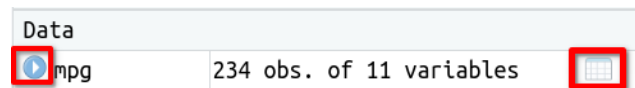
Las hojas de datos (en inglés *data frames*) son la clase de objetos que se usan en R para almacenar los datos obtenidos en un experimento y poder analizarlos.

Empezaremos explorando la hoja de datos de nombre `mpg` (**m**iles **p**er **g**allon), proporcionada por el paquete `ggplot2` (que ya hemos cargado al cargar el paquete `tidyverse`). Esta hoja de datos trata sobre el consumo de combustible de diferentes modelos de coches.

Para cargar la hoja de datos `mpg` en tu espacio de trabajo escribe

```
data("mpg")
```

Verás que `mpg` aparece listada en el panel Environment. Inicialmente está catalogada como `<Promise>`, pero en cuanto hagamos click en esa celda (o escribamos el nombre de la variable para usarla en nuestro código), se completará la carga y podremos leer que tiene 234 observaciones de 11 variables. Utilizando los iconos que se resaltan en la imagen siguiente, obtenemos más información sobre la hoja de datos.



- Presionando el icono de la izquierda, vemos los nombres de las 11 variables y sus valores para las primeras observaciones.
- Presionando el icono de la derecha, podremos visualizar la hoja de datos completa. El dato en la celda correspondiente a la fila  $i$  y la columna  $j$  es el valor observado para el coche número  $i$  de la variable número  $j$ .

En general, una hoja de datos es una especie de matriz u hoja de cálculo, donde las filas representan unidades experimentales, casos u observaciones (en nuestro ejemplo coches) y las columnas representan variables que describen características de interés de las unidades experimentales (en nuestro caso fabricante, modelo, cilindrada, consumo de combustible ...). Las celdas contienen los valores observados para cada variable en cada unidad experimental.

Para obtener una descripción de la hoja de datos, escribe `mpg` en el campo de búsqueda del panel **Help** de RStudio. También puedes escribir `?mpg` o `help(mpg)` en la consola, o poner el cursor sobre cualquier punto de la palabra `mpg` en la instrucción `data("mpg")` y presionar F1. La página de ayuda explica que la hoja de datos tiene 234 filas, donde cada fila representa un coche, y 11 variables, que describen diferentes características de los coches. Las variables que utilizaremos después son `displ`, `hwy` y `class`, que nos describen como:

- `displ`: *engine displacement, in litres* (cilindrada)
- `hwy`: *highway miles per gallon* (millas recorridas por galón de combustible en conducción por autopista)
- `class`: *"type" of car*

Para extraer las cilindradas (variable `displ`) de la hoja de datos `mpg` utilizamos la instrucción siguiente:

```
mpg$displ

##    [1] 1.8 1.8 2.0 2.0 2.8 2.8 3.1 1.8 1.8 2.0 2.0 2.8 2.8 3.1
##    [19] 5.3 5.3 5.3 5.7 6.0 5.7 5.7 6.2 6.2 7.0 5.3 5.3 5.7 6.5
##    [37] 3.6 2.4 3.0 3.3 3.3 3.3 3.3 3.3 3.8 3.8 3.8 4.0 3.7 3.7
##    [55] 4.7 5.2 5.2 3.9 4.7 4.7 4.7 5.2 5.7 5.9 4.7 4.7 4.7 4.7
##    [73] 4.7 4.7 5.2 5.2
```

```
## [73] 5.7 5.9 4.6 5.4 5.4 4.0 4.0 4.0 4.0 4.6 5.0 4.2 4.2 4.6
4.6 4.6 5.4 5.4
## [91] 3.8 3.8 4.0 4.0 4.6 4.6 4.6 4.6 5.4 1.6 1.6 1.6 1.6 1.6
1.8 1.8 1.8 2.0
## [109] 2.4 2.4 2.4 2.4 2.5 2.5 3.3 2.0 2.0 2.0 2.0 2.7 2.7 2.7
3.0 3.7 4.0 4.7
## [127] 4.7 4.7 5.7 6.1 4.0 4.2 4.4 4.6 5.4 5.4 5.4 4.0 4.0 4.6
5.0 2.4 2.4 2.5
## [145] 2.5 3.5 3.5 3.0 3.0 3.5 3.3 3.3 4.0 5.6 3.1 3.8 3.8 3.8
5.3 2.5 2.5 2.5
## [163] 2.5 2.5 2.5 2.2 2.2 2.5 2.5 2.5 2.5 2.5 2.5 2.7 2.7 3.4
3.4 4.0 4.7 2.2
## [181] 2.2 2.4 2.4 3.0 3.0 3.5 2.2 2.2 2.4 2.4 3.0 3.0 3.3 1.8
1.8 1.8 1.8 1.8
## [199] 4.7 5.7 2.7 2.7 2.7 3.4 3.4 4.0 4.0 2.0 2.0 2.0 2.0 2.8
1.9 2.0 2.0 2.0
## [217] 2.0 2.5 2.5 2.8 2.8 1.9 1.9 2.0 2.0 2.5 2.5 1.8 1.8 2.0
2.0 2.8 2.8 3.6
```

que mostrará el vector con las cilindradas de los 234 coches. En general, para extraer una de las variables de una hoja de datos se utiliza el formato `<nombre hoja>$<nombre variable>`.

Podemos ver los diferentes tipos de coches incluidos en la variable `class` con `unique(mpg$class)`

```
## [1] "compact"      "midsize"      "suv"          "2seater"
"minivan"
## [6] "pickup"       "subcompact"
```

El propósito de este apartado era presentar las hojas de datos en R y describir su estructura. En el capítulo [Crear hojas de datos](#) veremos cómo crear nuestras propias hojas de datos, bien de forma manual, bien importando los datos de un archivo.

## 7.4. Funciones

Hasta ahora hemos visto cómo utilizar algunas de las funciones predefinidas en R, pero en ocasiones necesitamos definir nuestras propias funciones. En este apartado veremos cómo hacerlo a través de un par de ejemplos.

El código a continuación define una función de nombre `add_two` que suma 2 al valor que recibe como argumento:

```
add_two <- function(x) {  
  x + 2  
}
```

Para utilizar la función que acaba de definirse escribiríamos

```
add_two(5)
```

```
## [1] 7
```

Veamos un segundo ejemplo. Consideremos la función

$$f(x) = a + be^{cx}$$

que depende de tres parámetros  $a$ ,  $b$  y  $c$ . Para evaluar está función en un determinado valor de la variable independiente  $x$  y valores concretos de los parámetros  $a$ ,  $b$  y  $c$  definiríamos la siguiente función de R:

```
myfun <- function(x, a, b, c) {  
  a + b * exp(c * x)  
}
```

Si queremos calcular  $f(4)$  siendo  $a = 1$ ,  $b = 2$  y  $c = 3$  usaríamos

```
myfun(4, 1, 2, 3)
```

```
## [1] 325510.6
```

Como puede verse en los ejemplos anteriores, para definir una función en R:

- Se indica el nombre de la función (como `add_two` o `myfun`) y se utiliza el operador de asignación `<-`, como para definir cualquier otro objeto.
- Se escribe la palabra clave `function` y entre paréntesis los nombres de los argumentos.
- Entre llaves, se escribe el cuerpo de la función, con el código que queremos ejecutar. En dicho código nos referiremos a los argumentos con los nombres que hayamos declarado antes entre paréntesis. El cuerpo de las dos funciones de nuestros ejemplos se limita a una única instrucción, pero en general puede contener un número arbitrario de líneas, donde cada

línea puede ser cualquier instrucción de R que puede hacer uso de los argumentos y de otros objetos definidos en instrucciones previas.

- La última expresión escrita en el cuerpo de la función será la salida de la función, que se imprimirá al llamarla. Nuestros dos ejemplos devuelven un valor numérico, pero una función puede devolver un objeto de cualquier otro tipo como un vector, una lista o una hoja de datos.



La llamada `myfun(4, 1, 2, 3)` asigna los valores a los argumentos en el orden en que se especificaron en la definición de la función. Como al definir `myfun` especificamos los argumentos en el orden `x`, `a`, `b`, `c`, la llamada `myfun(4, 1, 2, 3)` ejecuta la función con las asignaciones `x = 4`, `a = 1`, `b = 2` y `c = 3`.

Si escribimos explícitamente los nombres de los argumentos de una función, podemos indicarlos en cualquier orden. Por ejemplo:

```
myfun(a = 1, b = 2, c = 3, x = 4)
```

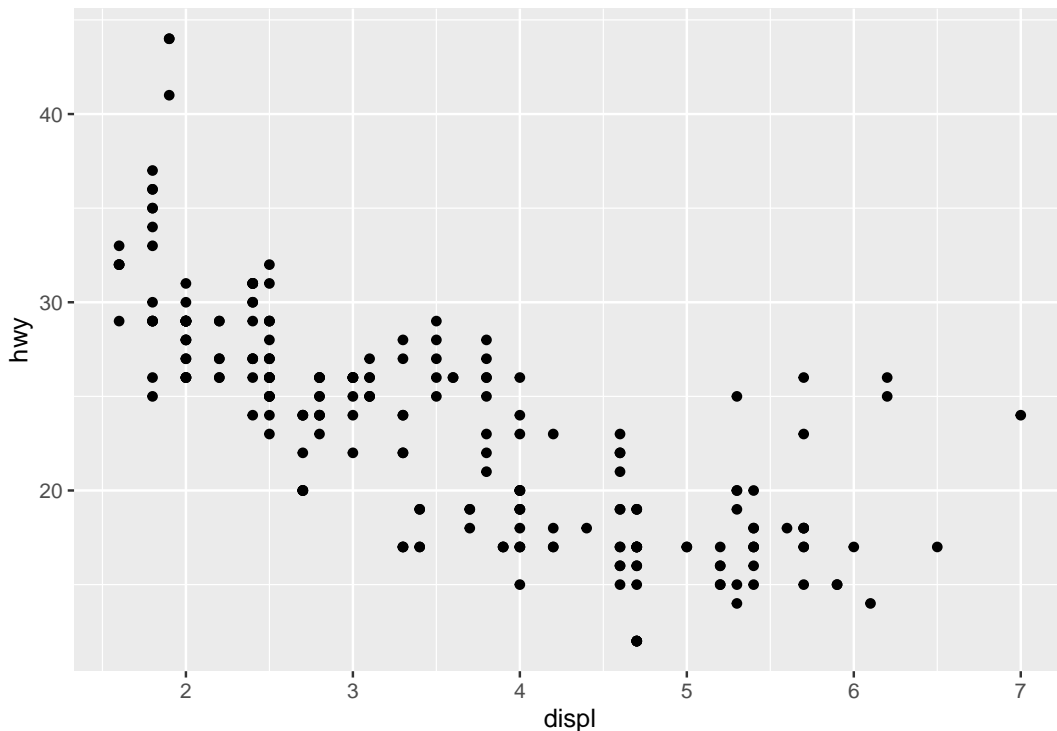
sería equivalente a `myfun(4,1,2,3)`.

## 8. Visualización de datos

En este capítulo seguimos trabajando con la hoja de datos `mpg` que presentamos antes. Nos planteamos en concreto si los coches con motor más grande (mayor `displ`) tienen mayor consumo de combustible (menor `hwy`).

Para responder a la pregunta anterior, vamos a realizar un gráfico de puntos para explorar la relación entre las variables `displ` (cilindrada) y `hwy` (millas por galón de combustible):

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))
```



Todos los gráficos se inician con la función `ggplot()`. En el código anterior, la línea

```
ggplot(data = mpg)
```

simplemente indica que en nuestro gráfico usaremos variables de la hoja de datos `mpg` y crea un gráfico vacío, al que podemos añadir capas utilizando funciones como `geom_point()`, que añadirá una capa de puntos. A lo largo del curso veremos otras funciones como `geom_bar()`, `geom_hist()` y `geom_line()` que añaden capas con otros objetos geométricos (barras, líneas ...).

Cada función para añadir un objeto geométrico como `geom_point()` necesita un argumento de la forma

```
mapping <- aes(...)
```

En nuestro ejemplo, ese argumento para la función `geom_line()` es

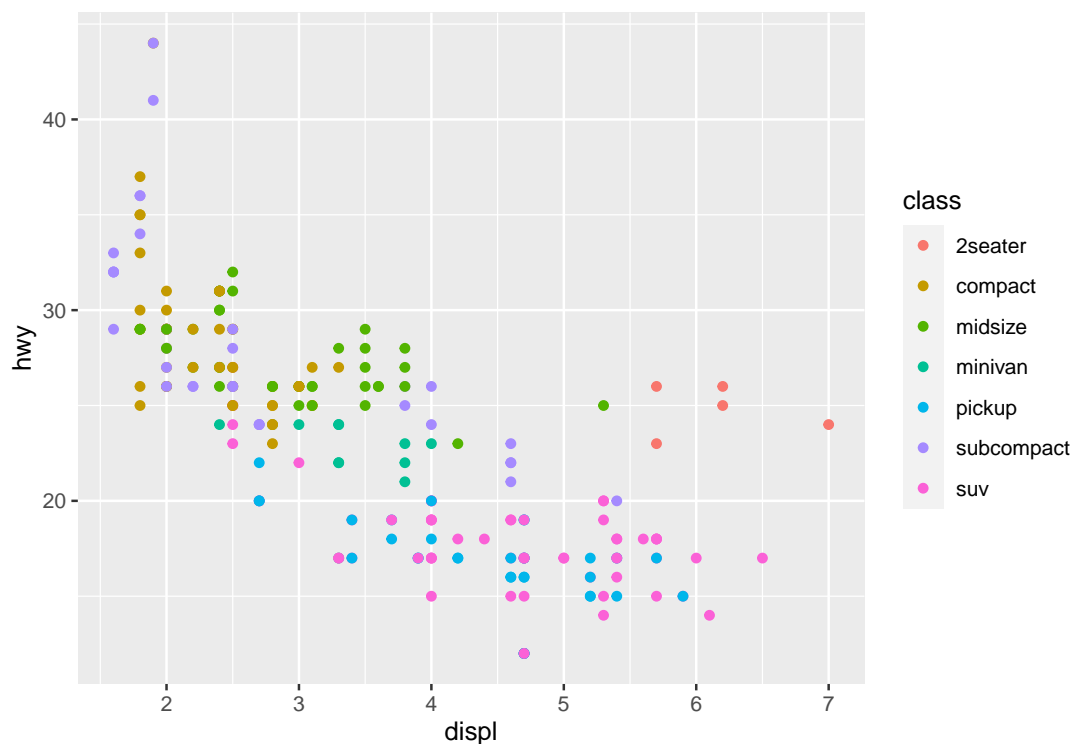
```
mapping <- aes(x = displ, y = hwy)
```

que especifica que, para cada punto en nuestro gráfico, su coordenada  $x$  vendrá definida por el valor de la variable `displ`, y su coordenada  $y$  se corresponderá con la variable `hwy`.

En general, en el argumento de la función `aes()` se especifica una lista de **correspondencias** (*mapping*) entre cada **estética** (*aesthetic*) o propiedad visual de los objetos geométricos a dibujar y las variables de nuestra hoja de datos.

Ahora vamos a incluir más información en nuestro gráfico estableciendo una correspondencia entre el color de los puntos (otra propiedad estética) y la variable `class`, que indica el tipo de coche.

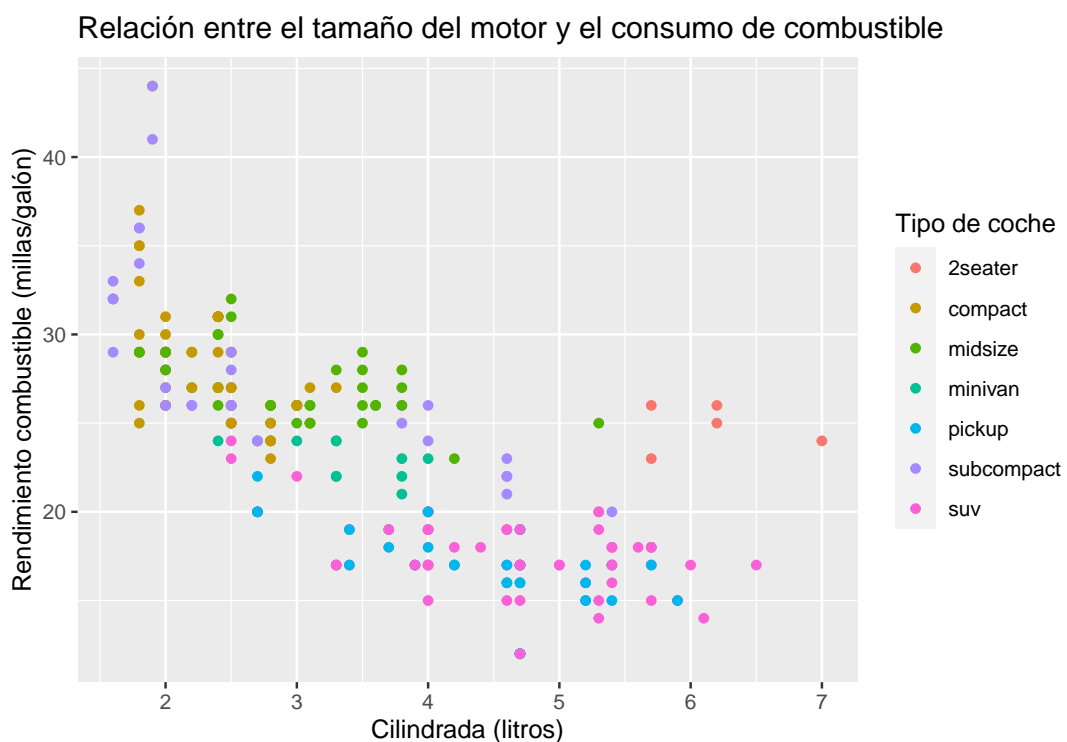
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



Notar que la inclusión de la correspondencia `color = class` ha hecho que `ggplot2` asigne un color diferente (nivel diferente de la estética `color`) a cada tipo de coche (valor diferente de la variable `class`), e incluya una leyenda explicando qué color corresponde a qué tipo de coche.

Acabamos personalizando los rótulos de nuestro gráfico con `labs()`:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class)) +
  labs(
    title = "Relación entre el tamaño del motor y el consumo de
    combustible",
    x = "Cilindrada (litros)",
    y = "Rendimiento combustible (millas/galón)",
    color = "Tipo de coche"
  )
)
```



## 9. Crear hojas de datos

Hasta ahora hemos trabajado con hojas de datos proporcionadas por paquetes de R, como `mpg` (de `ggplot2`). La colección de hojas de datos proporcionada por los diferentes paquetes de R es un buen recurso para enseñar y aprender diferentes tópicos. Pero en la práctica es necesario trabajar con nuestros propios datos o datos extraídos de diversas fuentes.

En este capítulo veremos cómo crear hojas de datos en R y cómo importar



datos desde un fichero externo.

## 9.1. La función `tibble()`

En este apartado veremos cómo crear hojas de datos en R con la función `tibble()`, del paquete del mismo nombre. El paquete `tibble` forma parte de `tidyverse` de forma que no es necesario cargarlo individualmente si ya has cargado `tidyverse`.

En el siguiente ejemplo se crea una hoja de datos sobre los [seis libros más vendidos de la historia](#) y se almacena en un objeto de nombre `best_sellers`.

```
best_sellers <- tibble(  
  title = c(  
    "A Tale of Two Cities",  
    "The Little Prince",  
    "Harry Potter and the Philosopher's Stone",  
    "And Then There Were None",  
    "Dream of the Red Chamber",  
    "The Hobbit"  
  ),  
  author = c(  
    "Charles Dickens",  
    "Antoine de Saint-Exupéry",  
    "J. K. Rowling",  
    "Agatha Christie",  
    "Cao Xueqin",  
    "J. R. R. Tolkien"  
  ),  
  lang = c(  
    "English",  
    "French",  
    "English",  
    "English",  
    "Chinese",  
    "English"  
  ),  
  year = c(  
    1859,  
    1943,  
    1997,
```

```
1939,  
1791,  
1937  
)  
)
```

Cada argumento de la función `tibble()` se corresponde con una variable en el formato `<nombre variable> = <vector de valores>`.

Para ver sólo los títulos de los libros usaríamos

```
best_sellers$title
```

```
## [1] "A Tale of Two Cities"  
## [2] "The Little Prince"  
## [3] "Harry Potter and the Philosopher's Stone"  
## [4] "And Then There Were None"  
## [5] "Dream of the Red Chamber"  
## [6] "The Hobbit"
```



La función usada tradicionalmente para crear hojas de datos en R es `data.frame()`. Las hojas de datos creadas con la función `tibble()` son una versión mejorada de las hojas de datos clásicas creadas con `data.frame()` y son el estandar actual. La mayoría de funciones del *tidyverse* trabajan con este tipo de hojas de datos, que a menudo se refieren como *tibbles*.

## 9.2. Importar datos

En este apartado veremos cómo importar datos desde un fichero externo con el paquete `readr`. El paquete `readr` forma parte de *tidyverse* de forma que no es necesario cargarlo individualmente si ya has cargado *tidyverse*. En concreto, veremos cómo importar los datos contenidos en un archivo CSV (*comma separated values*) con la función `read_csv()`. **strong**

### 9.2.1. Los datos

Trabajaremos con un archivo de nombre `temperaturas.csv`, que contiene las temperaturas máximas y mínimas registradas en el observatorio de la Virgen del Camino para todos los días del mes de abril del año 2018.

Descárgalo pinchando [aquí](#) y guárdalo en una carpeta de nombre **data** en el directorio del archivo R Markdown con el que estés trabajando.

### 9.2.2. Importar desde archivo local

Teniendo el archivo descargado en nuestro equipo, podemos importarlo especificando, como primer argumento de la función `read_csv()`, la ruta del archivo, que puede ser local respecto al directorio de trabajo actual.

Con la siguiente instrucción leemos el archivo que hemos descargado y lo almacenamos en un objeto de nombre **temps**.

```
temps <- read_csv("data/temperaturas.csv")
```

Si obtienes algún error indicando que no se encuentra el archivo, puede deberse a que el directorio de trabajo no es el correcto.

- Puedes saber cuál es el directorio de trabajo actual con `getwd()`.
- Para establecer que el directorio de trabajo sea aquel que contiene a tu archivo R Markdown, selecciona la pestaña de tu archivo `.Rmd` y utiliza el menú:

Session > Set Working Directory > To Source File Location

Si todo ha ido bien, el objeto **temps** aparecerá en el panel **Environment** y podrás visualizarlo.

Verás que hay 30 filas, una por cada día del mes, y 3 columnas con las variables:

- **Fecha**: fecha del día, en formato año-mes-día
- **Tmax**: temperatura máxima del día
- **Tmin**: temperatura mínima del día

### 9.2.3. Importar desde URL

Antes hemos realizado la descarga del archivo de datos de forma manual. También podríamos haberlo hecho desde R, con la función `download.file()`:

```
download.file(  
  url =  
  "https://raw.githubusercontent.com/EMazcunan/basics-r-rstudio/master/data/temperaturas.csv",  
  destfile = "data/temperaturas2.csv"  
)
```

Comprueba que se ha creado el archivo `temperaturas2.csv` en la carpeta `data`.

De hecho, el argumento para especificar el archivo en la función `read_csv`, admite una URL. Así que los dos pasos que hemos seguido antes, descargar e importar, pueden hacerse con una sola instrucción:

```
temps2 <-  
read_csv("https://raw.githubusercontent.com/EMazcunan/basics-r-rstudio/master/data/temperaturas2.csv")
```

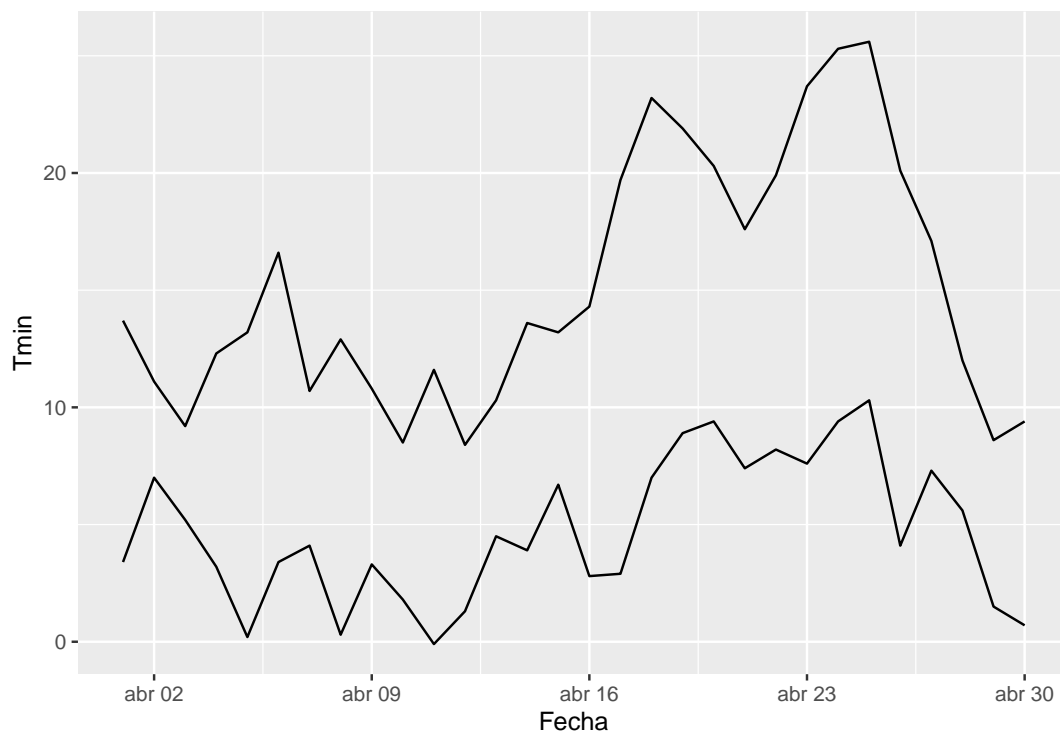
Después de ejecutar la instrucción anterior, puedes verificar que los objetos `temps` y `temps2` son idénticos.

#### 9.2.4. Usar datos importados

Una vez que hemos importado los datos de un fichero y los hemos almacenado en un objeto, podemos utilizar dicho objeto como si se tratara de una de las hojas de datos predefinidas en R.

Podemos por ejemplo representar la evolución de las temperaturas mínimas y máximas a lo largo de los días:

```
ggplot(data = temps) +  
  geom_line(mapping = aes(x = Fecha, y = Tmin, group = 1)) +  
  geom_line(mapping = aes(x = Fecha, y = Tmax, group = 1))
```



La función `geom_line()` actúa primero como si se dibujara un gráfico de puntos con `geom_point()` y luego conecta los puntos conforme indique el valor del argumento `group`. En el caso de nuestros dos diagramas de líneas, queremos que todos los puntos estén conectados, y lo indicamos usando `group=1`. Cuando hay más variables, la agrupación para las líneas suele hacerse por variable. Si por ejemplo tuviéramos cada día mediciones en varias ciudades, podríamos agrupar para la variable especificando la ciudad.

Cuando varias capas de un gráfico comparten estéticas, esas estéticas comunes pueden especificarse en el argumento de `ggplot()`. El ejemplo anterior puede acortarse a:

```
ggplot(data = temps, mapping = aes(x = Fecha, group = 1)) +  
  geom_line(mapping = aes(y = Tmin)) +  
  geom_line(mapping = aes(y = Tmax))
```

A continuación tenemos una versión un poco más elaborada del gráfico anterior. Utiliza el paquete `scales`, que ha de instalarse con `install.packages("scales")`.

```
library(scales)
```

```
ggplot(data = temps, aes(x = Fecha, group = 1)) +  
  geom_line(aes(y = Tmax), color = "red") +  
  geom_line(aes(y = Tmin), color = "blue") +  
  scale_x_date(  
    expand = c(0, 0),  
    breaks = date_breaks("1 day"),  
    labels = date_format("%d")  
  ) +  
  scale_y_continuous(breaks = seq(-5, 30, 5)) +  
  labs(  
    title = "Temperaturas máximas y mínimas en abril de 2018",  
    x = "Día",  
    y = "Temperatura ( °C )"  
  )
```

