

Introducción a R y RStudio

---

# Introducción a R Markdown

# Índice

Ideas clave	3
7.1. Introducción y objetivos	3
7.2. Elementos básicos de RMarkdown	4
7.3. Formatos de salida	10
7.4. Bloques de código y código en línea.	14
7.5. Elementos de sintaxis	22
7.6. Escritura de expresiones matemáticas	39
7.7. Referencias bibliográficas	44
7.8. Cuaderno de ejercicios	44

## 7.1. Introducción y objetivos

El proceso habitual para hacer un informe (o unas transparencias) en el que aparezcan gráficos o tablas resumen de algún análisis estadístico consiste en:

- ▶ Escribir el texto en un programa (Word, Latex, Beamer, Powerpoint, Prezi, etc.)
- ▶ Realizar los cálculos estadísticos y gráficos en otro programa (R, Python, Stata, Excel, etc.)
- ▶ Pegar los gráficos y tablas en el documento de texto.

Este proceso tiene al menos dos desventajas evidentes: dificulta la investigación reproducible y puede ser tedioso de rehacer si por ejemplo cambian ligeramente los datos.

En R es posible realizar todo el informe, tanto la escritura del texto como la realización de los cálculos y gráficos, en un único documento: R Markdown.

En términos sencillos R Markdown es un procesador de texto que ofrece la posibilidad de incluir trozos de código desde R (u otros lenguajes). El principal beneficio de esta herramienta es que permite trabajar en un sólo documento tanto la redacción del contenido narrativo de reportes de investigación, como también la construcción y presentación formal de resultados de análisis estadísticos.


De este modo, podemos elaborar informes un sólo lugar (archivo R Markdown) donde se edita tanto el texto del reporte, los códigos para construir los resultados a incluir, así como el formato general del documento y la gestión bibliográfica. Si el documento debe actualizarse (por ejemplo, cambian los datos), se efectúan en un sólo lugar todos los cambios y se re-compila el informe en el formato final deseado

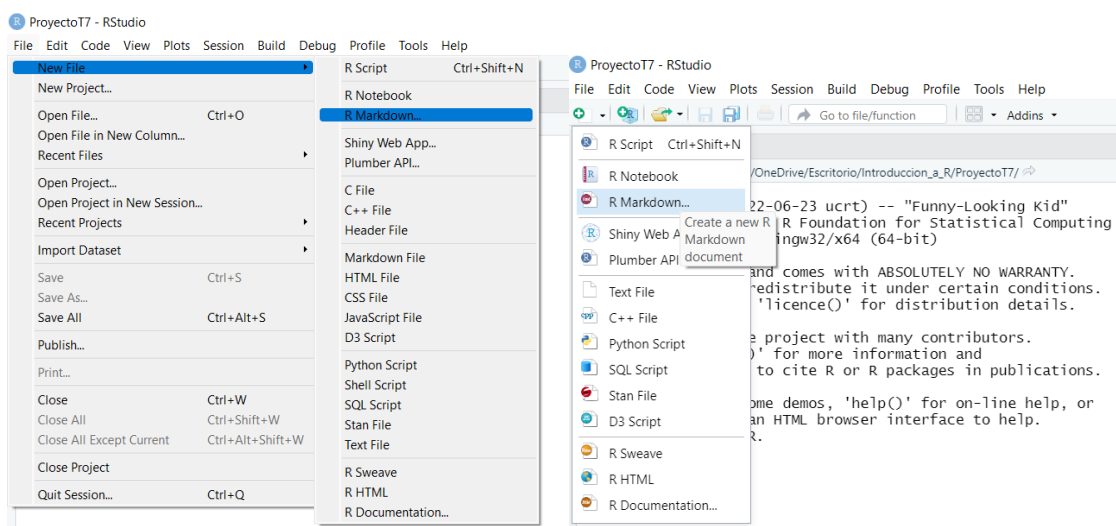
(PDF, Editor de Texto tipo Microsoft Word, diapositivas de presentación o un archivo dinámico tipo HTML).


Al finalizar este tema, esperamos que el hayas adquirido los fundamentos básicos para:

- ▶ Crear y editar un documento en R Markdown.
- ▶ Insertar tablas y gráficos en R Markdown.
- ▶ Elaborar un informe en las distintas opciones de salida (html, pdf, etc.)

## 7.2. Elementos básicos de RMarkdown

Un documento R Markdown es un archivo de texto simple guardado con extensión .Rmd. Podemos crearlos dentro de RStudio desde el mismo menú con el que abrimos los scripts: *File > New File > R Markdown* o, equivalentemente, clicando sobre el ícono  y eligiendo *R Markdown*).



Figuras 1 y 2: Abrir un nuevo archivo Rmd, dos opciones: desde el menú File (izquierda) desde el ícono  (derecha),

Al clicar en R Markdown, se abrirá un cuadro de diálogo (ver Figura 3) en el que podremos especificar el título, el autor y la fecha del documento, así como el formato

de salida predeterminado (es posible modificar fácilmente estos elementos más adelante) o elegir crear un nuevo documento vacío. La diferencia está en que, este último, crea un archivo .Rmd sin ningún contenido, y nosotros debemos especificar todo desde cero, mientras que de la otra forma (clicando sobre OK luego de especificar título, autor y fecha) genera una plantilla con contenido mínimo demostrativo (que por supuesto deberemos eliminar luego de ver como funciona pues no tendrá nada que ver con nuestro trabajo).

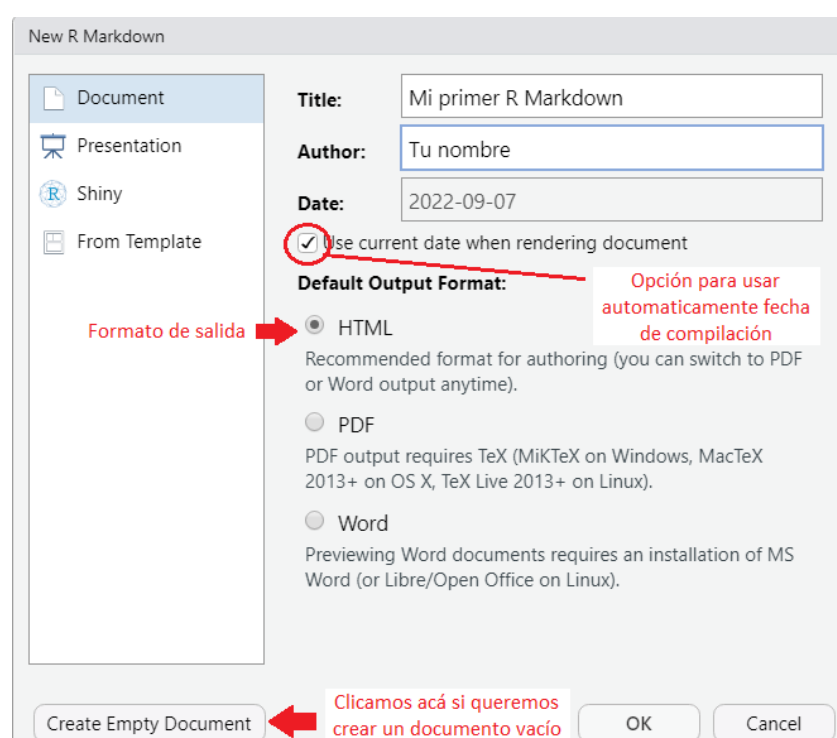



Figura 3: Cuadro de diálogo de creación de un nuevo .Rmd.

Cuando aceptemos (clicando en *OK*) nos generará un documento/plantilla para nuestro .Rmd. Si queremos procesarlo o compilarlo, tendremos que hacer clic en el icono *Knit*  (ver Figura 4).

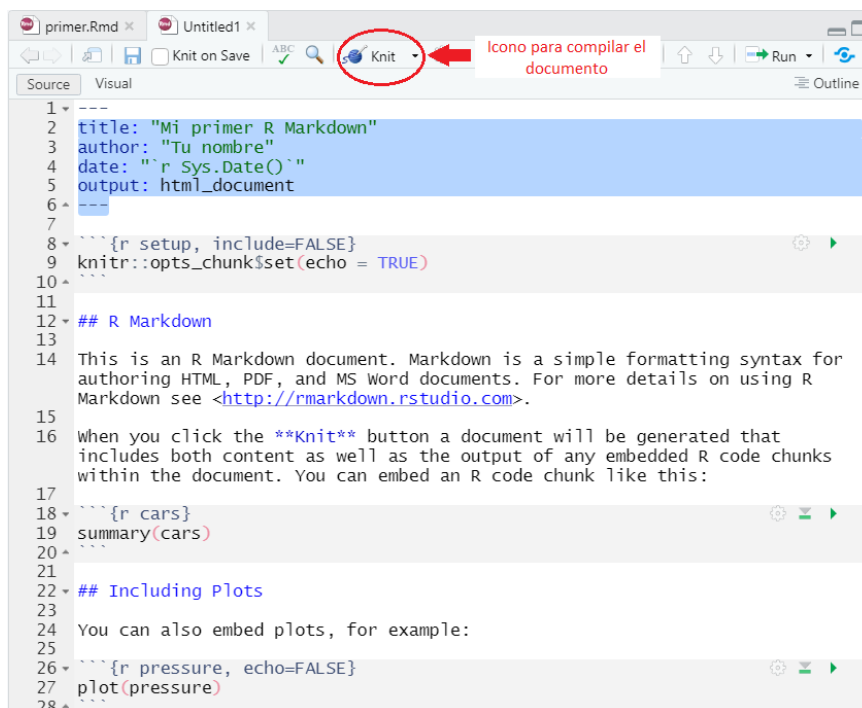



Figura 4: Plantilla para el .Rmd.

En R Markdown, el texto se escribe en *Markdown* y el código R se coloca en distintos bloques de código (o fragmentos de código). Los fragmentos de texto y R juntos se procesan con el paquete {knitr} en el documento final.

### Proceso para convertir los .Rmd a otro(s) formatos.

Como trabajamos con RStudio, en la práctica, procesar los ficheros .Rmd consistirá solamente en clicar en el icono  Knit.

No es necesario, pero quizá te interese saber cómo se procesan realmente los ficheros .Rmd para acabar convirtiéndose en html, pdf, etc.

La respuesta es que se ocupa de ello el paquete {rmarkdown} que llama otro paquete de R, {knitr} y a un programa llamado pandoc.

{knitr} se ocupa de ejecutar todos los trozos con código R que haya en el fichero .Rmd, después de ejecutar el código, pegará los resultados de la evaluación del código (gráficos, tablas etc...) junto con el texto en un documento intermedio (con extensión

.md), para después transferir, con la ayuda del paquete {rmarkdown}, este documento .md a pandoc que se encargará de traducirlo al formato elegido (html, pdf, ...).

Visualmente:

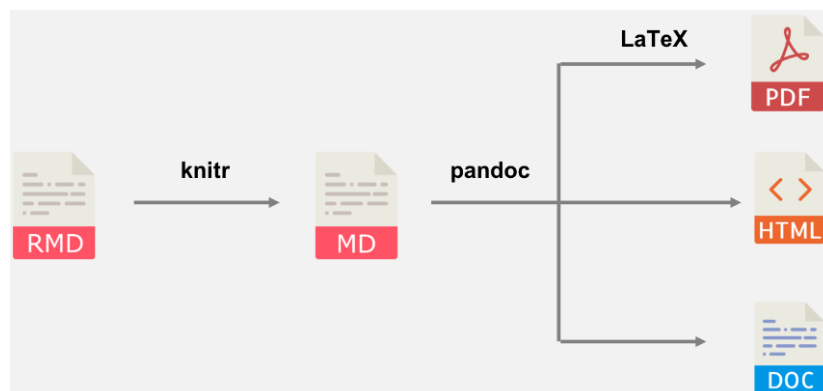


Figura 5: Proceso de conversión de un documento .Rmd en documento de salida final. Fuente: R Markdown from RStudio.

A continuación, presentamos el código del ejemplo (de un documento R Markdown mínimo):

```
---
title: "Mi primer R Markdown"
author: "Tu nombre"
date: "`r Sys.Date()`"
output: html_document
---

*R Markdown* permite combinar:

- texto formateado
- bloques de código R

Los bloques de código se ejecutan y se muestra su resultado, por ejemplo:

```{r}
mean(mtcars$mpg)
```

## Gráficos

También podemos incluir gráficos:

```{r}
plot(mtcars$hp, mtcars$mpg)
```
```

Si compilamos el documento anterior en formato HTML, el resultado que obtenemos es el siguiente:

## Mi primer R Markdown

Tu nombre

2022-09-07

R Markdown permite combinar:

- texto formateado
- bloques de código R

Los bloques de código se ejecutan y se muestra su resultado, por ejemplo:

```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

## Gráficos

También podemos incluir gráficos:

```
plot(mtcars$hp, mtcars$mpg)
```

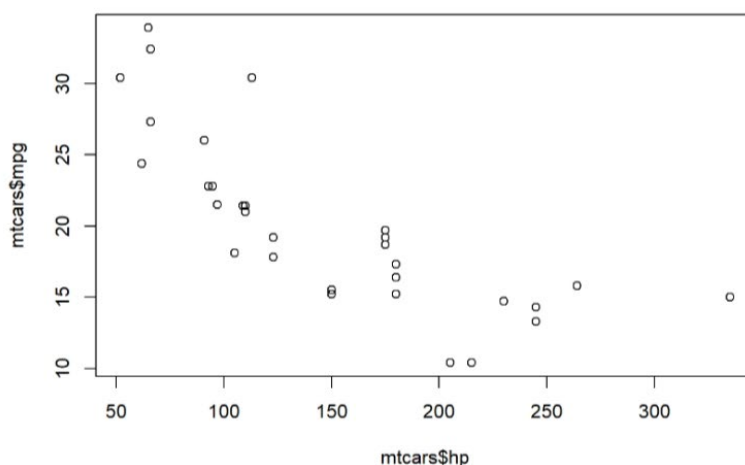


Figura 6: Resultado de compilar archivo .Rmd a formato HTML.

En la próxima sección estudiaremos los diferentes formatos de salida entre los que podemos elegir.

Los documentos R Markdown tienen 3 partes o componentes básicos:

1. Encabezamiento o YAML header: Se encuentra al principio del documento y está delimitado por el par de tres guiones (---)

```
---  
title: "Mi primer R Markdown"  
author: "Tu nombre"
```



```
date: "`r Sys.Date()`"  
output: html_document  
---
```

Este encabezado contiene los **metadatos** del documento, como su título, autor, fecha, además de una gran cantidad de opciones posibles que le permitirán configurar o personalizar todo el documento y su representación. Aquí, por ejemplo, la línea `output: html_document` indica que el documento de salida generado debe estar en formato HTML.

2. Bloques de código R (*chunks*): Como su nombre lo indica, contiene código R. Para que {knitr} distinga las instrucciones de R del texto normal tenemos que poner los bloques de código de R dentro de unas marcas o identificadores: ````{r}` (donde `r` indica el nombre del lenguaje) al principio y ````` al final. Por ejemplo:

```
```{r}  
mean(mtcars$mpg)  
```
```

{knitr} interpreta este fragmento de texto como instrucciones de R porque van dentro de las marcas, y hará que R las ejecute y muestre los resultados en el documento final.

3. Texto (escrito en Markdown): El cuerpo del documento consta de texto que sigue la sintaxis de Markdown. Generalmente en un texto queremos resaltar ciertas palabras con negrita, o ponerlas en cursiva, o poner un título de sección y de subsecciones. Todo esto lo tendremos que hacer utilizando Markdown. Markdown es un lenguaje de marcado ligero que ayuda a establecer niveles de encabezado o dar formato al texto. Por ejemplo, el siguiente texto:

```
Este es un texto con *cursiva* y **negrita**.  
Puede definir listas con viñetas:  
- primer elemento  
- segundo elemento
```

Generará el siguiente texto formateado:

Este es un texto con *cursiva* y **negrita**.

Puede definir listas con viñetas:

- primer elemento
- segundo elemento

Figura 7: Formato final de texto escrito en markdown.

Como podemos observar, las palabras colocadas entre asteriscos se ponen en cursiva, las líneas que comienzan con un guion se transforman en una lista con viñetas, etc.

En las secciones siguientes estudiaremos distintos elementos de sintaxis necesarias para formatear el texto de un documento.

## 7.3. Formatos de salida

Como lo mencionamos antes, en el encabezamiento del documento debemos especificar el formato del documento final o de salida.

Esencialmente diremos que hay dos tipos de formatos de salida en el paquete `{rmarkdown}`: **documentos** y **presentaciones**. Dentro de esos dos grandes tipo, encontramos diferentes tipos de formatos finales. Todos los formatos disponibles (ordenados alfabéticamente) son:

Para crear documentos:

- ▶ `context_document`
- ▶ `github_document`
- ▶ `html_document`
- ▶ `latex_document`
- ▶ `md_document`
- ▶ `odt_document`

- ▶ pdf\_document
- ▶ rtf\_document
- ▶ word\_document

Para crear presentaciones:

- ▶ beamer\_presentation
- ▶ ioslides\_presentation
- ▶ powerpoint\_presentation
- ▶ slidy\_presentation

Es importante conocer el nombre de la función que genera el documento de salida para poder acceder a su ayuda. Por ejemplo, para saber qué parámetros podemos definir con `html_document`, buscamos en `?html_document`.

Cada formato de salida se implementa como una función en R. Por lo tanto, podemos personalizar la salida pasando argumentos a la función como **subvalores del campo** `output`. Por ejemplo, para generar un `html_document` con una tabla de contenido flotante de profundidad 2 niveles, necesitamos usar:

```
---
title: "Mi primer R Markdown"
author: "Tu nombre"
date: "`r Sys.Date()`"
output:
  html_document:
    toc: true # incluir una Table Of Content (toc)
    toc_float: true
    toc_depth: 2 # hasta 2 niveles en el toc (esto es # y ##)
---
```

El resultado en el html generado es:

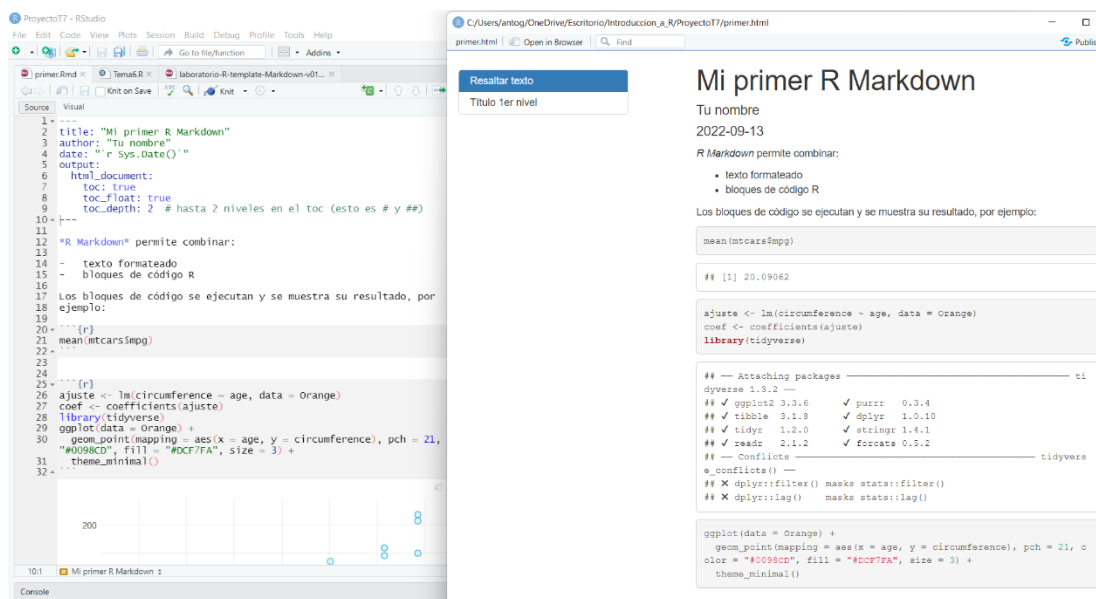


Figura 8: Modificar encabezado: toc flotante.

Los documentos .Rmd de R Markdown se convierten a PDF convirtiéndose primero en un archivo TeX y luego llamando al motor LaTeX para convertirlo a PDF. De manera predeterminada, este archivo TeX se elimina; sin embargo, podemos conservarlo (por ejemplo, para el envío de un artículo), especificando la opción `keep_tex`. Por ejemplo:

```
---
title: "Mi primer R Markdown"
author: "Tu nombre"
date: "`r Sys.Date() `"
output:
  pdf_document:
    toc: true
    number_sections: true
    keep_tex: true
---
```

El documento de salida es como se muestra en la Figura 9. Si buscamos entre los archivos que se crean luego de la compilación, encontraremos, además del fichero .pdf, uno de extensión .tex (ver Figura 10).

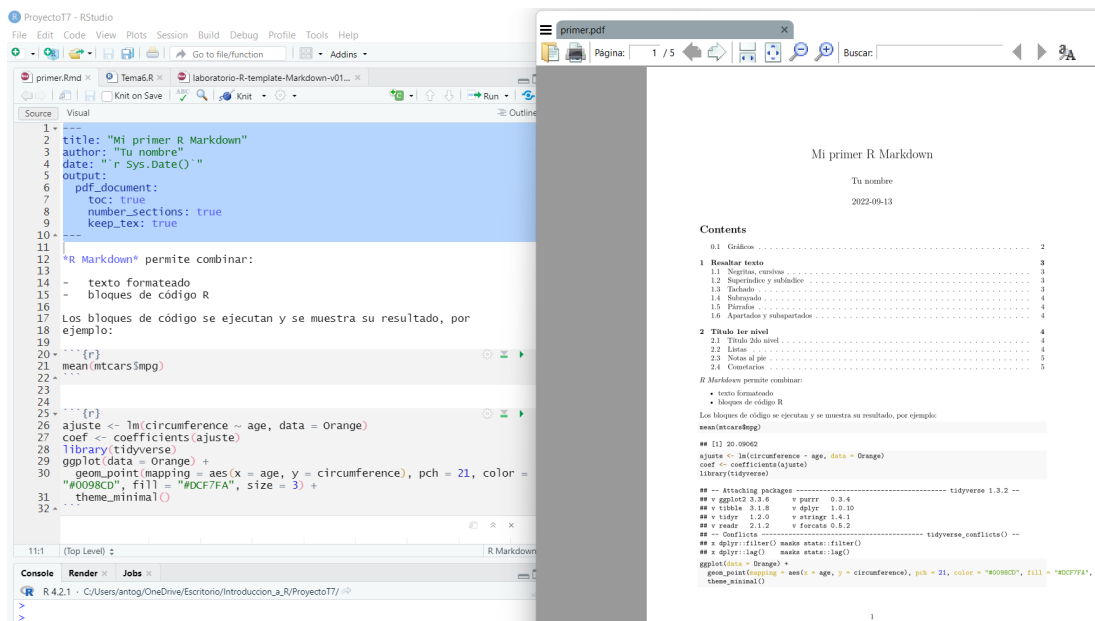


Figura 9: Documento de salida en formato PDF.

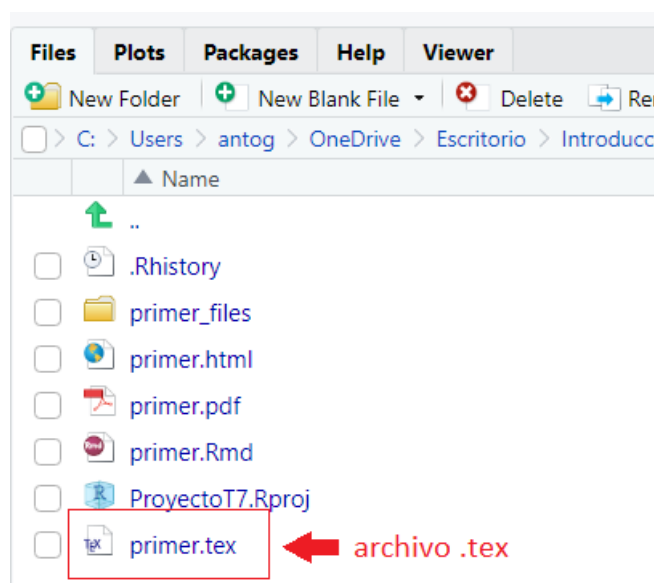


Figura 10: Fichero TeX generado con la opción keep\_tex: true.


Vale la pena destacar que, al hacer clic en el icono Knit  de RStudio, se compila al formato de salida indicado en el argumento output del encabezado (o al primero de la lista en caso de haber especificado más de un formato de salida). Sin embargo, podemos elegir otro formato al cual compilar del menú desplegable de dicho icono (ver Figura 11). Esto además de generar un documento de salida solicitado, añade al encabezado la instrucción correspondiente:



Figura 11: Menú desplegable de *Knit*.

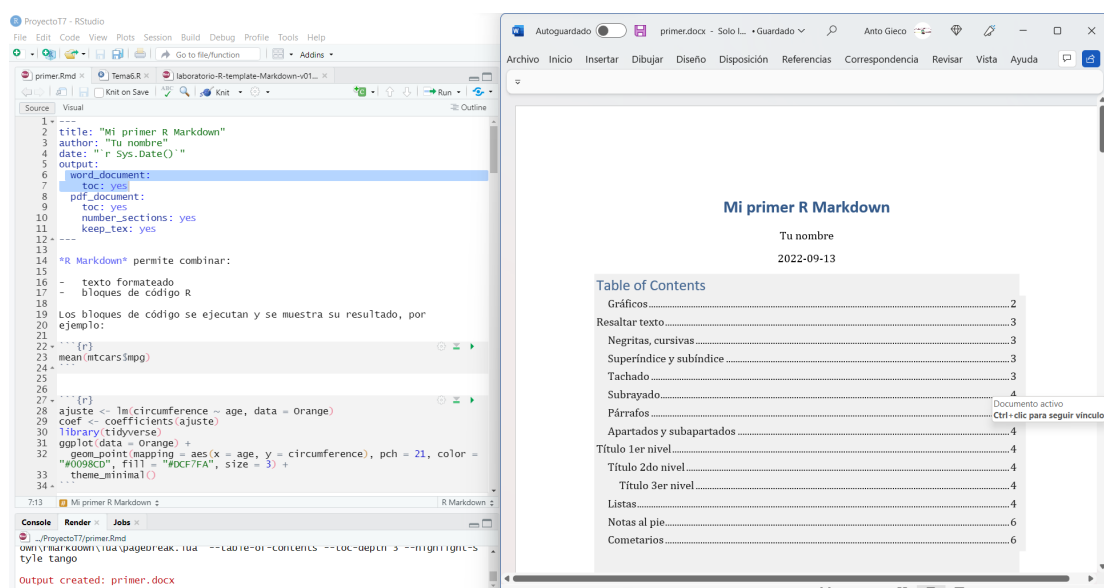


Figura 12: Documento de salida en formato Word.

Para más detalles sobre otros formatos de salida y las opciones que podemos incluir en cada uno, puedes visitar el apartado correspondiente (Output format) del sitio oficial de R Markdown ([aquí](#))

## 7.4. Bloques de código y código en línea.

Hay dos tipos de código en un documento Rmd: bloques de código (*chunks code*) y código R en línea. Para ver la diferencia entre ello, nada mejor que un ejemplo sencillo.

```

{r}
rad <- 5 # radio del círculo

```

Para un círculo de radio `rad`, su área es igual a `pi * rad^2`.

Figura 13: Código en línea y bloque de código.

Compilando el documento, obtenemos como resultado

```

rad <- 5 # radio del círculo

```

Para un círculo de radio 5, su área es igual a 78.5398163.

Figura 14: Resultado de Código en línea y bloque de código.

Como podemos ver, cuando hacemos *knit*, los resultados de los códigos en línea aparecen insertos en el texto. Esto resulta sumamente útil cuando en el texto hacemos referencia a características o valores calculados con los datos.

Veamos otro ejemplo más práctico. Supongamos que nos dan una muestra y queremos calcular su media, su desviación estándar y su tamaño muestral. Entonces, podemos cargar los datos y efectuar los cálculos en un bloque de código (que, como veremos enseguida, podemos ocultar completamente en el documento final con `include=FALSE`) y a continuación en un párrafo ir llamando los resultados mediante códigos en línea. Para ejemplificar concretamente, usemos la variable `mpg` (millas por galón) de los datos `mtcars`. Entonces, podríamos usar el bloque:

```

{r codigo-en-linea, include=FALSE}
res <- mtcars %>%
  summarize(media = mean(mpg), desvio = sd(mpg), n = length(mpg))

```

Figura 15: Bloque de código con los cálculos necesarios que incluiremos luego usando código en línea.

Y a continuación escribir el párrafo:

```

La muestra es de tamaño res$n, su media es res$media,
y su desviación estándar es res$desvio.

```

Figura 16: Código en línea para incluir resultados en texto.

Entonces, en el documento final, el bloque con los cálculos permanecería oculto, y este párrafo produciría:

```
La muestra es de tamaño 32, su media es 20.090625, y su desviación estándar es 6.0269481.
```

Figura 17: Resultado al compilar al utilizar código en línea.

Cuando necesitamos insertar números en el texto (como es el caso del ejemplo anterior), la función `round()` puede ser de gran ayuda, dado que podemos redondear a la cantidad de cifras decimales como le indiquemos con `digits`. Veamos en el ejemplo anterior, como hacemos para mostrar el área del círculo redondeado a dos decimales.



```
La muestra es de tamaño `r round(res$n, 2)`, su media es `r  
round(res$media, 2)`, y su desviación estándar es `r  
round(res$desvio, 2)`.
```

Figura 18: Usando `round()` para las salidas numéricas en texto.

La muestra es de tamaño 32, su media es 20.09, y su desviación estándar es 6.03.

Figura 19: Resultado al compilar de `round()` para las salidas numéricas en texto.

Pasemos ahora los bloques de código. Ya mencionamos que, para ejecutar código dentro de un documento R Markdown, necesitamos insertar un bloque (*chunk*). Hay tres maneras para hacerlo:

1. El atajo de teclado `Ctrl/Cmd + Alt + I`
2. El icono  (*Insert a new code chunk*) en la barra de edición (eligiendo del menú desplegable  (*Insert a new R chunk*))
3. Tippear manualmente los delimitadores de bloque ````\{r\}``` y `````.







En RStudio, los bloques de código R generalmente se muestran con un color de fondo ligeramente diferente para distinguirlos del resto del documento.



Cuando el cursor está en un bloque de código, podemos ingresar el código R que queramos, ejecutarlo, usar el autocompletado, tal como si estuviéramos en un script R.

## Ejecutar código


En R Markdown también podemos ejecutar código de distintas maneras:

- ▶ Si queremos ejecutar una línea de código individual, podemos utilizar el atajo de teclado `Ctrl/Cmd + Enter` (el cursor debe estar posicionado en la línea que queremos ejecutar).
- ▶ Si queremos ejecutar **todo** el código contenido en un bloque podemos hacerlo usando el atajo de teclado `Ctrl/Cmd + Shift + Enter` o utilizando el ícono “play” del bloque ( Run Current Chunk).
- ▶ También podemos ejecutar un bloque completo usando las opciones del menú  Run en el panel del archivo R Markdown, seleccionando de las opciones del desplegable  Run Current Chunk.
- ▶ Usando  Run del panel del archivo R Markdown, y seleccionando  Run All Chunks Above, RStudio ejecuta todos los bloques anteriores. Tenemos también la opción de ejecutar todos los bloques de código que siguen, eligiendo la opción  Run All Chunks Below.

En RStudio, por defecto, mostrará los resultados de ejecutar un bloque de código (texto, tabla o gráfico) directamente en la ventana de edición del documento, lo que permite verlos fácilmente y conservarlos durante la sesión<sup>1</sup>.

Cuando se compila el documento, cada bloque se ejecuta a su vez, y el resultado se incluye en el documento final, ya sea un texto, una tabla o un gráfico. Los bloques están vinculados entre sí, en el sentido de que los datos importados o calculados en

---

<sup>1</sup> Este comportamiento se puede cambiar haciendo clic en el ícono  de ajustes en la barra de herramientas y eligiendo *Chunk > Output en Consola*.

un bloque son accesibles a los bloques siguientes. Es por esto que también podemos ver un documento R Markdown como un script R en el que además insertamos texto en formato Markdown.

Es importante destacar que, antes de cada compilación, se inicia una nueva sesión de R que no contiene ningún objeto. Por esto, para que un R Markdown compile exitosamente tendremos que cuidar, obviamente, que no haya errores en el código ni en el texto, pero también vale la pena mencionar algunos lineamientos básicos. Por ejemplo, es necesario que el código esté en orden cronológico: no podemos calcular en la línea 5 una media de unos datos que solamente cargamos después, en la línea 10 o en un *chunk* posterior. Así mismo, en los bloques de código solo debe haber código (si hay texto, debe ir precedido de un # para marcarlo como comentario) y en el cuerpo del documento solo debe haber texto (si hay código, no se ejecutará como tal). Finalmente, es clave cargar todas las librerías que vamos a utilizar al inicio del archivo. Es práctica usual y recomendable, dedicar los primeros bloques de código del documento para importar datos, cargar librerías, realizar recodificaciones, etc.

## Nombre del bloque

Aunque no es obligatorio, podemos dar un nombre al bloque: {r nombre\_del\_bloque}. Esto puede ser útil en caso de un error de compilación, para identificar el bloque que causó el problema. Y ten en cuenta que no puedes tener dos bloques con el mismo nombre (en cuyo caso resultará un error de compilación).

## Opciones de los bloques

El comportamiento y salida de los bloques puede personalizarse agregando opciones (*options*), que son argumentos suministrados en el encabezado del bloque (es decir, que se colocan dentro de las llaves luego de r, separados entre comas). {knitr} provee casi 60 opciones que puedes usar para personalizar tus bloques de código. Aquí veremos las opciones de bloques más importantes y que usarás con mayor frecuencia. Para ver la lista completa puedes acceder [aquí](#).

El conjunto más importante de opciones controla si el bloque de código debe ejecutarse y/o mostrarse y si los resultados de la ejecución deben o no mostrarse en el reporte final:

- ▶ `eval`: para controlar si el código debe ser evaluado (`eval = TRUE`) o no (`eval = FALSE`). (Y, obviamente, si el código no es ejecutado no se generarán resultados). Esta opción es útil para mostrar códigos de ejemplo, o para deshabilitar un gran bloque de código sin comentar cada línea
- ▶ `include`: para decidir si incluir o no un fragmento de código en el documento de salida. Cuando `include = FALSE`, todo este fragmento de código se excluye en la salida, pero debemos tener en cuenta que aún se evaluará si `eval = TRUE`. Esta opción es la apropiada para los *chunk* de configuración o cuando tenemos trozos de código que necesitamos ejecutar, pero no deseamos mostrar (ni sus resultados).
- ▶ `echo`: para decidir si el código se mostrará (`echo = TRUE`) o no (`echo = FALSE`) en el reporte final. Esta opción es útil cuando queremos escribir reportes sin incluir el código subyacente de R (puede que al destinatario del reporte le interese ver los resultados, pero no el código R que los generó).
- ▶ `message`: permite indicar si los mensajes que R produce al ejecutar el código se han de mostrar en el documento final; su valor por defecto es `TRUE`, y si lo igualamos a `FALSE` no los muestra.
- ▶ `warning`: permite indicar si se han de mostrar en el documento final los mensajes de advertencia que a veces producen algunas funciones al ejecutarse. Al igual que `message`, su valor por defecto es `TRUE`, y si lo igualamos a `FALSE` no los muestra.
- ▶ `results`: Controla cómo mostrar los resultados de la ejecución del código en el documento final. Sus posibles valores son (y prestar atención a que hay que entrarlos entre comillas, porque son palabras *-strings-*):
  - `'hide'`: hace que no se muestre el resultado en el documento final.
  - `'asis'`: devuelve los resultados en el documento final línea a línea y el programa con el que se abre el documento final los interpreta como texto y los formatea adecuadamente.
  - `'hold'`: *retrasa* la visualización de todas las partes de salida hasta el final del bloque.

- 'markup': es el valor por defecto, muestra los resultados en el documento final línea a línea, encabezados con dos marcas de comentario ##, y literales, sin que el programa que abre el documento final los interprete como código.

Notar que no es lo mismo especificar que no se evalúe el código (`eval=FALSE`) que hacer que se evalúe, pero no mostrar el resultado (`results="hide"`): en el segundo caso, el resultado del cálculo no aparecerá en el documento final, pero se podrá usar en bloques de código posteriores.

- ▶ `error = TRUE` causa que la compilación continúe incluso si el código devuelve un error. Por defecto, `error = FALSE` provoca que el proceso de compilado falle si hay incluso un error en el bloque de código.

## Cambiar opciones

Es posible modificar las opciones manualmente editando el encabezado del bloque de código (es decir, agregándolas dentro de `{r}`), pero también podemos usar una pequeña interfaz gráfica que ofrece RStudio. Para hacer esto, simplemente hacemos clic en el icono de ajustes ubicado a la derecha de la línea de encabezado de cada bloque:

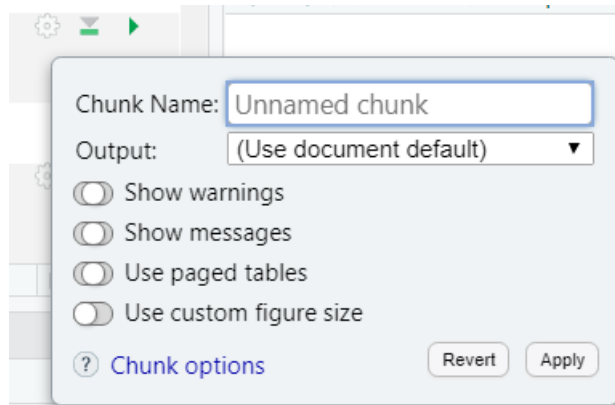


Figura 20: Configuración de bloque de código utilizando la IDE de RStudio.

Notarás que, al ir eligiendo las opciones, R irá añadiendo las opciones correspondientes al encabezado el bloque. Para finalizar la elección, hacemos clic en *Apply*.

## Opciones globales

Es posible que queramos aplicar una opción a **todos los bloques** de un documento. Por ejemplo, uno puede desear que, por defecto, no se muestre el código R de cada bloque en el documento final.

Esas opciones que llamamos globales (que queremos que apliquen sobre todos los bloques del documento), podemos configurarlas usando la función `knitr::opts_chunk$set()`. Por ejemplo, la instrucción `knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE)` añadida en un bloque de código establecerá las opciones `echo = FALSE`, `warning = FALSE`, `message = FALSE` de forma predeterminada para todos los bloques posteriores (de modo que, si la instrucción la colocamos en un bloque al inicio del documento, aplica sobre todos los bloques de código del documento).

En general, todas estas modificaciones globales las colocamos en un bloque especial llamado `setup` y que es el primer bloque del documento (inmediatamente a continuación del encabezado YAML). También es una buena idea cargar en este *chunk* todos los paquetes que se van a usar en el documento. Y como seguramente no querrás que se note su existencia en el documento final, usaremos la opción `include=FALSE`. Así, por ejemplo, incluyendo al principio del fichero R Markdown el *chunk*:

```
```${r} setup, include=FALSE}
library(tidyverse) # aquí ponemos todos los paquetes que necesitamos
library(flextable)
knitr::opts_chunk$set( # aquí la configuración global
  echo = FALSE,
  message = FALSE,
  warning = FALSE
)
```
```

especificamos que, por defecto, en el documento final no aparezcan ni los bloques de código ni los mensajes y advertencias que produce R al compilarlos.

Por defecto, RStudio ejecuta sistemáticamente el contenido del bloque `setup` antes de ejecutar el de otro bloque.

A diferencia de otros bloques de código, cuando se usa el menú de configuración del bloque en RStudio `setup` para cambiar sus opciones, estas cambian no las opciones de ese bloque sino las opciones globales, actualizando los argumentos en la llamada de función `knitr::opts_chunk$set()`.

## 7.5. Elementos de sintaxis

Ya hemos mencionado que el texto en los archivos `.Rmd` está escrita en Markdown, una colección simple de convenciones para dar formato a archivos de texto plano. Markdown está diseñado para ser fácil de leer y fácil de escribir, siendo también muy fácil de aprender.

La funcionalidad para editar la presentación de texto es limitada, pero es más que suficiente para el 99% de las cosas que necesitamos al escribir. Si quieres, puedes insertar elementos HTML en un documento de Markdown para ampliar el rango del lenguaje (veremos algunos ejemplos de esto).

Todos estos formatos (negrita, cursiva, ...) se introducen en R Markdown con marcas; por ejemplo, si quieres que una palabra se resalte en negritas, tienes que escribirla enmarcada en `**`: **esto se mostraría en negrita**

A continuación, veremos los principales elementos de sintaxis necesarios para formatear nuestro texto en el informe final. En [este enlace](#), encontrarás una guía tutorial de elementos de sintaxis básica de Markdown.

## Resaltar texto

Veremos en este apartado las distintas marcas que permiten introducir diferentes énfasis en el texto. Los siguientes ejemplos muestran cómo se escriben en R Markdown y cómo es el formato que se obtiene al compilar.

### Negrita y cursiva

Crear negritas y cursivas con Markdown es particularmente fácil, ya que solo necesitamos los asteriscos. Para poner una palabra o grupo de palabras en **cursiva**, inserta un asterisco antes y después de ellas. Para la **negrita**, utiliza dos. Si deseas remarcar un área de texto en **negrita y cursiva**, pon tres asteriscos. Como alternativa, también puedes utilizar guiones bajos.

Texto plano

*\*cursiva\** y \_cursiva\_

**\*\*negrita\*\*** y \_\_negrita\_\_

**\*\*\*cursiva y negrita\*\*\*** o \_\_\_cursiva y negrita\_\_\_ o **\_\_\*\*cursiva y negrita\*\*\_\_**  
o **\*\_cursiva y negrita\_\***

Figura 21: Guía de sintaxis para énfasis de texto: negrita, cursiva.

Texto plano

*cursiva y cursiva*

**negrita y negrita**

***cursiva y negrita*** o ***cursiva y negrita*** o ***cursiva y negrita*** o ***cursiva y negrita***

Figura 22: Compilación de documento R Markdown con énfasis de texto: negrita, cursiva.

## Tachado

Para representar un texto tachado en Markdown, escribe dos virgulillas seguidas; después, escribe el texto correspondiente y ciérralo con otras dos virgulillas.

```
~~Este texto está tachado,~~ pero este no.
```

Figura 23: Guía de sintaxis para énfasis de texto: tachado.

~~Este texto está tachado, pero este no.~~

Figura 24: Compilación de documento R Markdown con énfasis de texto: tachado.

## Subrayado

No se puede subrayar texto en Markdown. En HTML, esto es posible con la etiqueta `<u>`, aunque, por lo general, no se recomienda hacerlo para evitar confusiones, porque los hipervínculos se muestran como texto subrayado.

```
<u>Uso la etiqueta HTML para subrayar este texto</u> y sigo sin subrayar.
```

Figura 25: Etiqueta HTML para subrayado.

Uso la etiqueta HTML para subrayar este texto y sigo sin subrayar.

Figura 26: Resultado al compilar.

## Párrafos

En el lenguaje Markdown, utilizamos líneas vacías para separar los párrafos. Para crear un bloque de texto completamente nuevo (etiqueta), simplemente se introduce una línea en blanco. Es importante tener en cuenta que para Markdown es suficiente si la línea está visualmente vacía. Por lo tanto, si contiene caracteres en blanco, como tabulaciones o espacios, el analizador los ignorará e interpretará que la línea está vacía. También podemos insertar un salto de línea introduciendo dos espacios al final de la línea.



Los párrafos son una o múltiples líneas de texto adyacentes separadas por una o múltiples líneas en blanco.

Termina línea con dos espacios para nuevo párrafo.  
Este texto comienzo en un párrafo nuevo.  
Pero sin añadir los espacios en blanco al final, seguimos en el mismo párrafo.

También este es un nuevo párrafo pues dejamos una línea en blanco

Figura 27: Sintaxis para párrafos.

Los párrafos son una o múltiples líneas de texto adyacentes separadas por una o múltiples líneas en blanco.

Termina línea con dos espacios para nuevo párrafo.

Este texto comienzo en un párrafo nuevo. Pero sin añadir los espacios en blanco al final, seguimos en el mismo párrafo.

También este es un nuevo párrafo pues dejamos una línea en blanco

Figura 28: Resultado de compilación de sintaxis para párrafos.

También podemos insertar un break HTML `<br/>`, separar dos párrafos con dos o más líneas en blanco (agregar líneas en blanco no resulta en mayor espaciado entre párrafos)

Quiero separar este párrafo

de este, pero dejar líneas en blanco no funciona.

Quiero separar este párrafo

`<br/>`

`<br/>`

`<br/>`

de este, lo logro con la etiqueta HTML.

Figura 29: Etiqueta HTML `<br/>` para separar párrafos.

Quiero separar este párrafo

de este, pero dejar líneas en blanco no funciona.

Quiero separar este párrafo

de este, lo logro con la etiqueta HTML.

Figura 30: Resultado de usa la etiqueta HTML `<br/>` para separar párrafos.

## Citas en bloque

Para marcar un fragmento de texto como una cita en Markdown, puedes crear las llamadas **citas en bloque**, utilizando el **signo de mayor (>)**. Tienes dos opciones: marcar cada línea individual con este carácter o insertar únicamente uno al principio del párrafo de la cita sangrada y marcar el fin de la cita introduciendo una línea en blanco. La cita en bloque, a su vez, puede ser formateada con otros elementos.

```
> Esta es una *cita con *formato*.  
> La cita continúa aquí.  
  
> Este es otra *cita con *formato*.  
Este fragmento continúa en la siguiente línea.  
  
Esta línea ya no está sangrada.
```

Figura 31: Sintaxis para citas en bloque.

Esta es una **cita con *formato***. La cita continúa aquí.

Este es otra **cita con *formato***.  
Este fragmento continúa en la siguiente línea.

Esta línea ya no está sangrada.

Figura 32: Resultado de citas en bloque.

## Apartados y subapartado

Para crear un título en Markdown, se utiliza de forma predeterminada una almohadilla (numeral o *hashtag*), que se pone delante del texto, separada por un espacio en blanco. Para crear subtítulos y, por lo tanto, en letra más pequeña, se insertan más almohadillas (por ejemplo dos almohadillas creará un subtítulo o título de 2do nivel, tres almohadillas, creará un sub-subtítulo o título de 3er nivel, y así sucesivamente). De esta manera, se pueden crear hasta seis niveles de títulos, como en HTML.

```
# Título 1er nivel
## Título 2do nivel
### Título 3er nivel
#### Título 4to nivel
##### Título 5to nivel
##### Título 6to nivel
```

Figura 33: Sintaxis para títulos y subtítulos.

# Título 1er nivel

## Título 2do nivel

### Título 3er nivel

#### Título 4to nivel

##### Título 5to nivel

###### Título 6to nivel

Figura 34: Títulos y subtítulos.

## Listas

Para crear una **lista no ordenada** en Markdown, puedes utilizar el signo más (+), un guion (-) o un asterisco (\*). Con las tres opciones obtendrás el mismo resultado.

```
+ Primer ítem
+ Segundo ítem
+ Tercer ítem

o

- Primer ítem
- Segundo ítem
- Tercer ítem

o

* Primer ítem
* Segundo ítem
* Tercer ítem
```

Figura 35: Sintaxis para generar listas no ordenadas.

- Primer ítem
  - Segundo ítem
  - Tercer ítem
- o
- Primer ítem
  - Segundo ítem
  - Tercer ítem
- o
- Primer ítem
  - Segundo ítem
  - Tercer ítem

Figura 36: El resultado es el mismo con las tres opciones.

Si lo que quieres es crear una **lista ordenada**, deberás introducir un número o una letra con un punto directamente después.

- ```
1. Item uno
2. Item dos
3. Item tres

o

a. Item a
b. Item b
c. Item c
```

Aunque Markdown mostrará los items correctamente en orden, esto no es una buena idea:

- ```
1. Item uno
1. Item dos
1. Item tres
```

Figura 37: Sintaxis para listas ordenadas.

1. Item uno
2. Item dos
3. Item tres

o

- a. Item a
- b. Item b
- c. Item c

Aunque Markdown mostrará los items correctamente en orden, esto no es una buena idea:

1. Item uno
2. Item dos
3. Item tres

Figura 38: Listas ordenadas.

Finalmente, también puedes crear listas anidadas, con cualquier combinación entre ordenadas y no ordenadas:

También puedes usar listas anidadas

1. Item uno
2. Item dos
3. Item tres
  - Sub-item
  - Sub-item
4. Item cuatro

o también

1. Item uno
2. Item dos
3. Item tres
  - i. Sub-item
  - ii. Sub-item
4. Item cuatro

o también |

- + Item uno
- + Item dos
- + Item tres
  1. Sub-item
  2. Sub-item
- + Item cuatro

Figura 39: Sintaxis para listas anidadas.

También puedes usar listas anidadas

1. Item uno
2. Item dos
3. Item tres
  - Sub-item
  - Sub-item
4. Item cuatro

o también

1. Item uno
2. Item dos
3. Item tres
  - i. Sub-item
  - ii. Sub-item
4. Item cuatro

o también

- Item uno
- Item dos
- Item tres
  - 1. Sub-item
  - 2. Sub-item
- Item cuatro

Figura 40: Listas anidadas.

## Notas al pie

Markdown ofrece la posibilidad de incorporar notas al pie. Esto significa que escribes un número de nota en el texto y lo refiere a una nota al pie al final de la página. El número de nota también se formatea como un hipervínculo, que nos lleva directamente a la nota al pie correspondiente al hacer clic en él. Para utilizar esta función automática, deberás insertar el número de la nota detrás de la palabra que desees. Para ello, escribes entre corchetes un acento circunflejo y, después, el número.

El número que utilices (se admiten también otros elementos) es irrelevante. Al igual que con la creación de listas, Markdown realiza el conteo automáticamente. Sin embargo, es importante que vuelvas a introducir la misma denominación. Para ello, pon el mismo número en una nueva línea, de nuevo entre corchetes y precedido por un acento circunflejo, inserta dos puntos y, luego, escribes el texto de la nota. Este, a

su vez, también podrá formatearse de todas las maneras posibles y ocupar varias líneas.

## ## Notas al pie

Colocar notas en el pie de página<sup>[^1]</sup> es muy fácil<sup>[^2]</sup>.

[^1]: Aquí encuentras el texto de la nota al pie de página.

[^2]: **Las notas de pie de página** pueden *formatearse* también. Estas pueden ocupar varias líneas.

Figura 41: Sintaxis para notas al pie.

## Notas al pie

Colocar notas en el pie de página<sup>1</sup> es muy fácil<sup>2</sup>.

## Comentarios

En el documento copiado no se muestra las líneas comentadas.

Para ingresar un enlace asociado a una palabra, se debe encerrar la palabra o palabras que [queremos sea un enlace](#)

---

1. Aquí encuentras el texto de la nota al pie de página.↩

2. **Las notas de pie de página** pueden *formatearse* también. Estas pueden ocupar varias líneas.↩

Figura 42: Las notas al pie se muestran al final de la página.

## Comentar texto

Al igual que con los scripts, resulta útil incluir comentarios el texto en el documento de origen, sin que se muestren en el documento de salida final. Para ello, podemos utilizar la sintaxis HTML `<!--el comentario -->`. Los comentarios no se mostrarán en **ningún formato de salida**.

Los comentarios pueden abarcar una sola línea o varias líneas.

En RStudio, podemos usar el atajo de teclado `Ctrl + Shift + C` (o `Command + Shift + c` en macOS) para comentar líneas de texto seleccionadas.

## Tablas

Por defecto, R Markdown imprime *data frames* y matrices tal como se ven en la consola:

```
mtcars[1:7, 1:5]
```

```
##           mpg cyl  disp  hp drat
## Mazda RX4    21.0   6  160  110 3.90
## Mazda RX4 Wag 21.0   6  160  110 3.90
## Datsun 710    22.8   4  108   93 3.85
## Hornet 4 Drive 21.4   6  258  110 3.08
## Hornet Sportabout 18.7   8  360  175 3.15
## Valiant      18.1   6  225  105 2.76
## Duster 360    14.3   8  360  245 3.21
```

Figura 43: Impresión de tablas por defecto en R Markdown.

Bien podría interesarnos darle algún formato adicional. Esto puede hacerlo utilizando la función `knitr::kable()`.

```
knitr::kable(
  mtcars[1:7, 1:5],
  caption = "Un kable de knitr."
)
```

Un kable de knitr.

|                   | mpg  | cyl | disp | hp  | drat |
|-------------------|------|-----|------|-----|------|
| Mazda RX4         | 21.0 | 6   | 160  | 110 | 3.90 |
| Mazda RX4 Wag     | 21.0 | 6   | 160  | 110 | 3.90 |
| Datsun 710        | 22.8 | 4   | 108  | 93  | 3.85 |
| Hornet 4 Drive    | 21.4 | 6   | 258  | 110 | 3.08 |
| Hornet Sportabout | 18.7 | 8   | 360  | 175 | 3.15 |
| Valiant           | 18.1 | 6   | 225  | 105 | 2.76 |
| Duster 360        | 14.3 | 8   | 360  | 245 | 3.21 |

Figura 44: Impresión de tablas con formato, utilizando la función `kable()`.



Bastante mejor, ¿no les parece?

La sintaxis para la función `kable()` del paquete `{knitr}` es

```
kable(x, format, digits = getOption("digits"), row.names = NA,  
      col.names = NA, align, caption = NULL, label = NULL,  
      format.args = list(), escape = TRUE, ...)
```

donde `x` es el objeto de datos (que debe ser de formato rectangular, es decir, un *data frame* o una matriz), y `format` especifica el formato de la tabla de salida.

### Formatos de tabla admitidos

En la mayoría de los casos, `knitr::kable(x)` es suficiente si solo necesitamos una tabla simple para el objeto de datos `x`. El argumento `format` se establece automáticamente de acuerdo con el formato del documento de origen (el valor dado en output en el encabezado YAML). No obstante, podemos cambiar el formato de salida de la tabla, asignándole al argumento `format` alguno de los siguientes valores posibles:

- ▶ `pipe` (tablas Pandoc con columnas separadas por pipes),
- ▶ `simple` (tablas simples de Pandoc),
- ▶ `latex` (tablas LaTeX),
- ▶ `html` (tablas HTML)
- ▶ `rst` (tablas reStructuredText).

Para los documentos de R Markdown, `kable()` usa el formato `pipe` para tablas de forma predeterminada.

Debemos tener en cuenta que únicamente los formatos `pipe` y `simple` son portátiles, es decir, funcionan para cualquier formato de documento de salida (html, pdf, etc.). Otros formatos de tabla solo funcionan para formatos de salida específicos, por ejemplo, `format = 'latex'` solo funcionan para documentos de salida LaTeX. El uso de un formato de tabla específico brinda más control, al precio de sacrificar la portabilidad.

Hay muchos otros paquetes de R que se pueden usar para generar tablas. Algunos de ellos: {flextable} y {huxtable}: si está buscando un paquete de tablas que admita la más amplia gama de formatos de salida, {flextable} y {huxtable} son probablemente las dos mejores opciones. Todos los formatos son compatibles con los formatos HTML, LaTeX y Office, y contienen las funciones de tabla más comunes (p. ej., formato condicional). Puede encontrar más información sobre flextable en <https://ardata-fr.github.io/flextable-book/>, y la documentación de huxtable está en <https://hughjonesd.github.io/huxtable/>.

Veamos dos ejemplos sencillos (sobre los mismos datos que venimos trabajando) de algunas funcionalidades del paquete {flextable}.

Ejemplo 1: forma básica:

```
```{r tabla_flex}
ftab <- flextable(mtcars[1:7,1:5])
ftab <- set_caption(ftab, caption = "Datos mtcars")
ftab
```
```

Ejemplo 2: personalizando la tabla resultante:

```
```{r}
flex_data <- flextable(mtcars[1:7,1:5]) %>%
  border_remove()%>%
  flextable::align(align = 'center', part = "all")

big_border <- fp_border(color = "#0098CD", width = 2)
std_border <- fp_border(color = "#0098CD", width = 1)

flex_data <- hline_top(flex_data,part="body", border = std_border)%>%
  vline(border = std_border, part = "body")%>%
  vline_left(border = std_border, part = "body")%>%
  hline_bottom(part="body", border = std_border)%>%
  fontsize(size = 11, part = "all")%>%
  bold(i=1,part='header')%>%
  bg(i =1,bg='#0098CD', part = "header")%>%
  color(i=1,color='white',part='header')%>%
  bg(i = c(1, 3, 5,7), j = seq(1,5,2), bg='#d5e7fb', part = "body")%>%
  font(fontname = "Calibri", part = "all")%>%
  width(j=2,width = 2)%>%
  width(j=1,width = 0.5)
flex_data <- fix_border_issues(flex_data)
flex_data
```

Las tablas se muestran en el documento de salida HTML de la siguiente manera:

Ejemplo 1

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08
18.7	8	360	175	3.15
18.1	6	225	105	2.76
14.3	8	360	245	3.21

Ejemplo 2

mpg	cyl	disp	hp	drat
21.0	6	160	110	3.90
21.0	6	160	110	3.90
22.8	4	108	93	3.85
21.4	6	258	110	3.08
18.7	8	360	175	3.15
18.1	6	225	105	2.76
14.3	8	360	245	3.21

Figura 45: Tablas utilizando funcionalidades del paquete `{flextable}`.

## Imágenes

Markdown posee una sintaxis especial para colocar imágenes de manera sencilla. Por ejemplo, queremos colocar la siguiente [imagen](#)



Figura 46: Ejemplo de imagen a incluir en texto. Fuente: [https://commons.wikimedia.org/wiki/File:Logo\\_UNIR.png](https://commons.wikimedia.org/wiki/File:Logo_UNIR.png).

Entonces, para esto, necesitamos escribir en nuestro documento el siguiente código:

```
![Logo_UNIR](https://upload.wikimedia.org/wikipedia/commons/d/df/Logo_UNIR.png){width='100px'}
```

que produce la siguiente salida

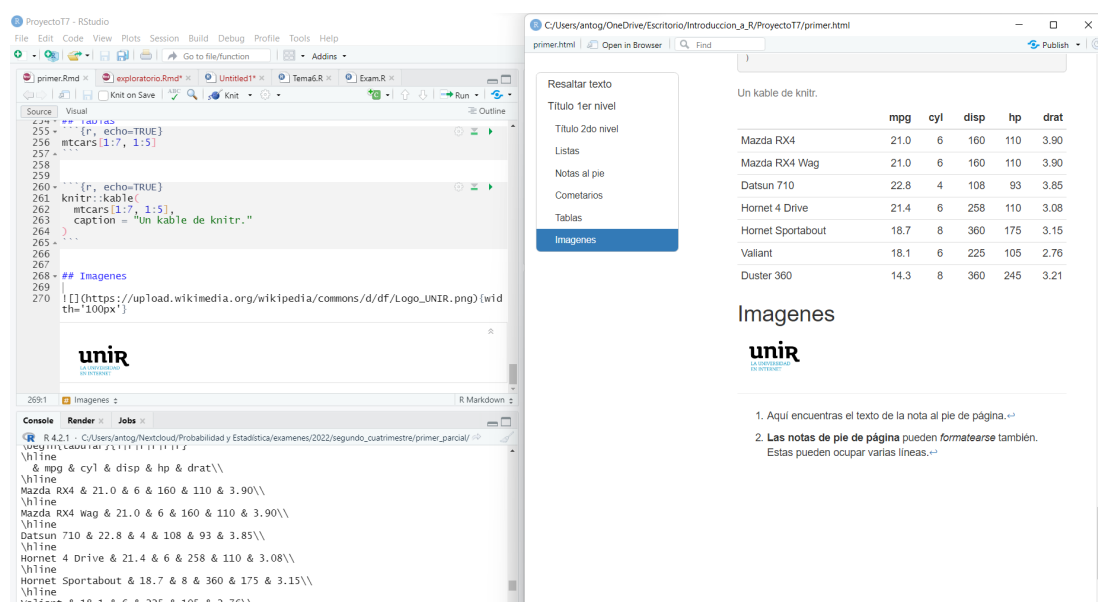


Figura 47: Insertar figura con la sintaxis de Markdown.

Notar que la imagen se previsualiza en la ventana de edición del documento.

La sintaxis general la siguiente:

```
![texto_alternativo](url/ubicacion_de_la_imagen){width=width height=height}
```

### Instrucciones para incluir una imagen en R Markdown:

- Primero escribimos `![]()`.

- Después escribimos, dentro de los corchetes, el texto alternativo. Este es **opcional** y solo entra en acción cuando no se puede cargar la imagen correctamente.
- Después escribimos, dentro de los paréntesis, la ubicación del archivo (ya sea una url o una ubicación en nuestra computadora).

Otra opción es usar la función `include_graphics()` del paquete `{knitr}`, que brinda más control sobre los atributos de la imagen que la sintaxis de Markdown (por ejemplo, puede especificar el ancho de la imagen a través de `out.width`).

Retomando el ejemplo anterior, usando la función `include_graphics()` con argumento personalizados, resulta:

```
```${r, out.width='50%', fig.align='center', fig.cap='Caption o leyenda para la imagen'}
knitr::include_graphics('https://upload.wikimedia.org/wikipedia/commons/d/df/Logo_UNIR.png')
```
```

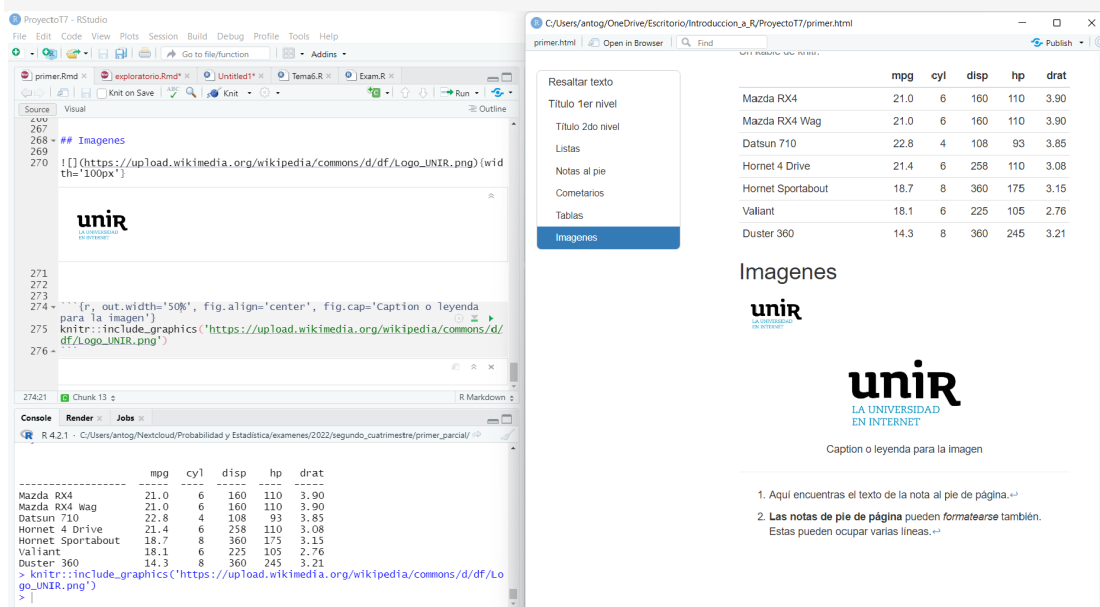


Figura 48: Insertar figura a través de una bloque de código.

Notemos que la figura se muestra centrada (`fig.align='center'`), modificamos su tamaño (`out.width = '50%'`) y le agregamos una leyenda (`fig.cap = 'Caption o leyenda para la imagen'`).

Es importante señalar que, la primera opción (que es sintaxis de Markdown) la incluimos directamente en el cuerpo del texto, mientras que la segunda opción, la imagen es el resultado de ejecución de código de R, por eso está incluida dentro de un bloque de código.

## Opciones del bloque de código para figuras

Ya sea que se trate de una figura generada en el mismo documento (por ejemplo usando `plot()`) o de una imagen que importemos (usando la función `knitr::include_graphics()`), podemos controlar el tamaño, la posición etc. de los gráficos en el documento final (html, pdf o Word) por medio de opciones en el encabezamiento de los *chunks* que los producen. Veamos algunas de las opciones básicas más útiles:

- ▶ `fig.height` y `fig.width` sirven para especificar la altura y la anchura, respectivamente, del gráfico **real**. Sus posibles valores son números enteros y están medidos en pulgadas (no pueden usarse otras unidades). Por defecto, ambos valen 7.
- ▶ `out.height` y `out.width` sirven para especificar la altura y la anchura, respectivamente, del gráfico **en el documento final**. Estas opciones son útiles si se quieren indicar las dimensiones del gráfico en tamaños relativos respecto de las medidas de la caja de texto. Así, por ejemplo, al compilar con *Knit PDF* o *Knit HTML*, si se especifican `fig.height` y `fig.width` (o se dejan con sus valores por defecto) y se incluye la opción `out.width = "60%"` (el entrecomillado es obligatorio), el gráfico aparecerá en el documento escalado de manera que su ancho sea un 60% del ancho del texto.
- ▶ `fig.asp` permite fijar la proporción altura/anchura de un gráfico si solo se especifica su `fig.width`.
- ▶ `fig.align` permite establecer la alineación de las figuras. Por ejemplo, puede centrar las imágenes con `fig.align = 'center'` o alinearlas a la derecha con `fig.align = 'right'`. Esta opción funciona tanto para la salida HTML como para LaTeX, pero es posible que no funcione para otros formatos de salida (como Word, desafortunadamente).

- ▶ `fig.show` sirve para controlar cómo se incluyen en el documento final los gráficos producidos en el *chunk* cuando hay más de uno o bien, para que no se incluyan. Su valor por defecto es `"as.is"`, que los va incluyendo a medida que se generan. Con `fig.show="hold"` se dibujan todos los gráficos de golpe al final del *chunk* mientras que, `fig.show="hide"`, no mostrará los gráficos en el reporte final.
- ▶ `dev` sirve para especificar el tipo de fichero gráfico que se genera. Resulta útil usar esta opción cuando necesitemos publicar el documento html como una página web. En tal caso conviene usar `dev="svg"` y el gráfico se creará en formato “*Scalable Vector Graphics*”, con lo que se escalará de manera correcta y sin perder calidad al aumentar o disminuir el ancho de la página web en el navegador.

## Enlaces

Para poner enlaces o *hyperlinks* podemos escribir la url entre los signos de mayor y menor (`<>`). Por ejemplo: `<https://www.unir.net>` y se mostrará así:

<https://www.unir.net>

UNIR

Pero es mejor ponerlo así: `[UNIR](https://www.unir.net)` y se mostrará así:

Si quieres que el enlace se abra en el navegador en una página nueva debes añadir `{target="_blank"}`, es decir:

`[UNIR](https://www.unir.net){target="_blank"}`

## 7.6. Escritura de expresiones matemáticas

Si necesitamos incluir expresiones matemáticas en el documento, la manera de hacerlo se basa en la sintaxis del sistema de composición de textos científicos [LaTeX](#). Incluir fórmulas en un texto R Markdown no tiene ningún misterio. Solo hay que introducir el código que representa la fórmula de una de las dos formas siguientes:

- ▶ Para las fórmulas o ecuaciones dentro del mismo párrafo (en línea con el texto), se escribe el código entre dos signos \$: \$código\_de\_la\_expresión\$.
- ▶ Para las fórmulas o ecuaciones que queramos que aparezcan centradas en una línea aparte, se escribe el código entre dos dobles signos \$: \$\$ código\_de\_la\_expresión \$\$.

Al crear una fórmula a partir del código, RStudio ignora los espacios en blanco que hayamos escrito en ella (excepto si el espacio inmediatamente antes del signo \$ de cierre), y añade los espacios en blanco a partir del significado lógico de sus elementos. Por ejemplo (y dejamos algunos espacios en blanco innecesarios para que observemos que no tienen ningún efecto en el resultado), el código siguiente:

```
## Ecuaciones latex
Las raíces de la ecuación  $x^2 = 2$  son  $x = \sqrt{2}$  y
 $x = -\sqrt{2}$ ;
en general, las raíces de  $ax^2 + bx + c = 0$ , con  $a \neq 0$ , vienen
dadas
por la fórmula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

```

Figura 49: Ejemplo de escritura de expresiones matemáticas usando funcionalidades de LaTeX.

## Ecuaciones latex

Las raíces de la ecuación  $x^2 = 2$  son  $x = \sqrt{2}$  y  $x = -\sqrt{2}$ ; en general, las raíces de  $ax^2 + bx + c = 0$ , con  $a \neq 0$ , vienen dadas por la fórmula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Figura 50: Resultado en documento final compilado.

Como podemos observar, los espacios en blanco dentro de las ecuaciones se ignoran, a excepción de la expresión  $x = -\sqrt{2}$ , que no se compila correctamente dado que hay un espacio en blanco inmediatamente antes del signo \$ de cierre de la expresión.



Notemos que:

- ▶ Las potencias, y en general los superíndices, se indican con `^`. Si el exponente consta de más de un carácter, lo encerramos entre llaves. Por ejemplo: `$2^{x+1}$`.
- ▶ La raíz cuadrada de algo se indica con `\sqrt{numero}` (de *square root*).
- ▶ Una fracción se indica con `\frac{numerador}{denominador}` (de *fraction*).
- ▶ Los signos  $\pm$  y  $\neq$  se indican con las marcas `\pm` (de *plus-minus*) y `\neq` (de *not equal*), respectivamente.

Como vemos, las marcas de LaTeX que definen los diferentes elementos de las fórmulas matemáticas tienen nombres intuitivamente claros y (salvo las que corresponden a signos usuales como la suma o la resta) empiezan con el signo `\`.

También podemos incluir matrices, o, más en general, tablas, en las fórmulas matemáticas. Una tabla se define empezando con `\begin{array}{formato}` y acabando con `\end{array}`. `formato` es una secuencia de letras `l` (de izquierda, *left*), `r` (de derecha, *right*) o `c` (de centrada): el número de letras indica el número de columnas, y cada letra indica el tipo de alineamiento de la columna correspondiente. Así, por ejemplo, `\begin{array}{rcccl}` define una tabla de cuatro columnas: la primera alineada a la derecha, la segunda y la tercera centradas, y la cuarta alineada a la izquierda.

Entre el `\begin{array}{formato}` y el `\end{array}` se introducen por filas los valores de la tabla: los elementos de cada fila se separan con el signo `&` y el cambio de fila se indica con `\\`.

Para definir una matriz, hemos de envolver la tabla con los delimitadores `\left(` y `\right)`. Sin embargo, existen otros entornos predefinidos que incluyen delimitadores automáticamente:

| Entorno              | Delimitador        |
|----------------------|--------------------|
| <code>pmatrix</code> | <code>()</code>    |
| <code>bmatrix</code> | <code>[]</code>    |
| <code>Bmatrix</code> | <code>{}</code>    |
| <code>vmatrix</code> | <code>  </code>    |
| <code>Vmatrix</code> | <code>     </code> |

Tabla 1: Entornos disponibles para crear matrices.

Veamos algunos ejemplos.

Si escribimos el código:

```


$$\begin{matrix}
\text{\textcolor{violet}{\$}} \\
A_{\{m,n\}} = \\
\backslash\text{begin}\{pmatrix\} \\
a_{\{1,1\}} \ \& \ a_{\{1,2\}} \ \& \ \text{\textbackslash cdots} \ \& \ a_{\{1,n\}} \ \backslash \\
a_{\{2,1\}} \ \& \ a_{\{2,2\}} \ \& \ \text{\textbackslash cdots} \ \& \ a_{\{2,n\}} \ \backslash \\
\text{\textbackslash vdots} \ \& \ \text{\textbackslash vdots} \ \& \ \text{\textbackslash ddots} \ \& \ \text{\textbackslash vdots} \ \backslash \\
a_{\{m,1\}} \ \& \ a_{\{m,2\}} \ \& \ \text{\textbackslash cdots} \ \& \ a_{\{m,n\}} \\
\backslash\text{end}\{pmatrix\} \\
\text{\textcolor{violet}{\$}}
\end{matrix}$$


```

Figura 51: Crear matrices usando funcionalidades de LaTeX.

El resultado al compilar es:

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Figura 52: Resultado al compilar de matrices usando funcionalidades de LaTeX.

```


$$A_{m,n} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}, \quad A_{m,n} = \left\{ \begin{array}{cccc} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{array} \right\}$$


```

Figura 53: Ejemplos de las funcionalidades de LaTeX para crear matrices.

Al escribir matrices de tamaño arbitrario, es común usar tres puntos (puntos suspensivos) horizontales, verticales y diagonales para completar ciertas columnas y filas. Como observamos en los ejemplos anteriores, estos se pueden especificar usando `\cdots`, `\vdots` y `\ddots` respectivamente.

Si sólo queremos usar un delimitador a un lado de la tabla, tenemos que incluir al otro lado `\left.` o `\right.`, según corresponda dado que las marcas `\left` y `\right` siempre han de ir en parejas. Por ejemplo:

```


$$|x| = \left\{ \begin{array}{cl} x & \text{si } x \geq 0 \\ -x & \text{si } x < 0 \end{array} \right.$$


```

Figura 54: Ejemplo del uso de array con un solo delimitador.

En el último ejemplo, vemos que, si en una entrada en modo matemático queremos introducir texto, lo tenemos que incluir en una caja de texto definida con la instrucción `\text{...}`. Por ejemplo, `\text{si}`. Observar que hemos dejado un espacio en blanco dentro de `\text{si}` para separar el texto “si” de las fórmulas

que los siguen. Esto significa que el contenido de un `\text{...}` se transforma en texto, y por lo tanto se tienen en cuenta los espacios en blanco igual que en el texto normal.

Si necesitas escribir expresiones matemáticas de manera regular en tus documentos R Markdown, te recomendamos que consultes la sección de Matemáticas del [Wikibook de LaTeX](#).

## 7.7. Referencias bibliográficas

Xie, Y., Dervieux, C. y Riederer, E. (2020). *R Markdown Cookbook*. First edition, C&H/CRC Press, 2021.

<https://bookdown.org/yihui/rmarkdown-cookbook/rmarkdown-process.html>

Wickham, H. and Grolemund, G. (2019). *R para Ciencia de Datos*.

<https://es.r4ds.hadley.nz/>

## 7.8. Cuaderno de ejercicios

### Ejercicio 1

Practica lo que has aprendido creando un informe breve sobre un tema de tu interés.

El documento debe incluir:

- ▶ Un título y encabezados para (por lo menos) 2 secciones. Cada una de las secciones debe incluir una lista con viñetas y/o numerada.
- ▶ Al menos una nota al pie.
- ▶ Una línea horizontal.
- ▶ Una cita en bloque.

## Ejercicio 2

A partir del fichero `diamantes.Rmd` proporcionado con el material del curso:

- ▶ Revisa que puedes ejecutarlo. Agrega los datos de autor y fecha del documento.
- ▶ Agrega texto después del polígono de frecuencias que describa sus características más llamativas.
- ▶ Incluye una sección que explore cómo los tamaños de diamantes varían por corte, color y claridad. Asume que escribes un reporte para alguien que no conoce R, por lo que en el informe final no debería mostrarse el código R. Utiliza para esto una opción global.

## Ejercicio 3

Modifica `diamantes.Rmd` de modo que en el reporte final:

- ▶ se muestre también el porcentaje de diamantes que son mayores a 2.5 quilates.
- ▶ incluya una sección que describa los 20 diamantes más grandes, incluyendo una tabla que muestre sus atributos más importantes.

## Ejercicio 4

Modifica el fichero `Anova.Rmd` (proporcionado como material del curso) de modo tal que el resultado final sea el siguiente:

# ANOVA

Tu nombre

La fecha

- Análisis de la varianza
  - El modelo ANOVA

## Análisis de la varianza

El análisis de varianza o ANOVA por sus siglas en inglés (*ANalysis Of VAriance*), es un modelo estadístico que evalúa si las medias de distintos grupos son iguales. Su versión más simple es aquella en la cual dos grupos distintos son comparados considerando solo los efectos fijos de los mismos. Este caso particular es llamado comúnmente Test de t (*t*—test).

## El modelo ANOVA

El modelo estadístico clásico del análisis de varianza con un solo factor par los datos es:

$$Y_{ij} = \mu_i + \varepsilon_{ij} \quad i = 1, \dots, I, \quad j = 1, \dots, n_i$$

donde  $Y_{ij}$  es la  $j$ -ésima observación del grupo  $i$  (notar que cada grupo es un nivel de un factor o **tratamiento**),  $\mu_i$  es la media del grupo  $i$  y  $\varepsilon_{ij}$  es el error aleatorio correspondiente a esta observación.  $n_i$  es la cantidad de observaciones en el grupo  $i$  e  $I$  es la cantidad de grupos o niveles del factor (*cantidad de tratamientos*).

Otra forma de plantear el modelo la siguiente. Si se considera  $\mu$  como el valor esperado para una observación cualquiera de la población (la media global sin tener en cuenta los diferentes niveles del factor), y  $\alpha_i$  el efecto introducido por el nivel  $i$  del factor. Entonces, la media de un determinado nivel ( $\mu_i$ ) se puede definir como:

$$\mu_i = \mu + \alpha_i$$

y por tanto el modelo en los datos resulta

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij} \quad i = 1, \dots, I, \quad j = 1, \dots, n_i$$

donde  $\mu$  es la media global,  $\alpha_i$  es el efecto del  $i$ -ésimo tratamiento, sujeto a la restricción  $\sum_{i=1}^I \alpha_i = 0$  y  $\varepsilon_{ij}$  es el error aleatorio correspondiente a esta observación que suponemos con distribución  $\mathcal{N}(0, \sigma)$  e independientes.

Figura 55: Resultado final para ejercicio 4.

---

Puedes consultar distintas alternativas de personalización de documentos de salida en formato HTML en los siguientes enlaces:

- R Markdown: The Definitive Guide: <https://bookdown.org/yihui/rmarkdown/html-document.html#advanced-customization> (esta es una guía de usuario completa para el ecosistema completo de R Markdown para la creación de documentos, escrita por los mismos autores del paquete {rmarkdown}).
  - bookdown: Escritura de libros y documentos técnicos con R Markdown (traducción al español del libro “bookdown: Authoring Books and Technical Documents with R Markdown”) [https://mamaciasq.github.io/libro\\_bookdown/personalizacion.html](https://mamaciasq.github.io/libro_bookdown/personalizacion.html)
  - rstudio4edu: A Handbook for Teaching and Learning with R and RStudio <https://rstudio4edu.github.io/rstudio4edu-book/>
-

En el siguiente vídeo podrás acceder a la resolución del ejercicio 4 del cuaderno de ejercicios:



Accede al vídeo

---