Investigación reproducible con Quarto y R

Felipe Ortega María Jesús Algar Emilio López Cano

2024-10-24

Tabla de contenidos

Preface **3**

I Quarto **4**

1 Documentos científicos **5**
1.1 Programación literaria . . . 6
1.2 Investigación reproducible . . . 6
1.3 Quarto para publicaciones científicas . . . 6
1.4 Instalación de Quarto . . . 6

2 Tipos de documentos **7**
2.1 Documentos individuales . . . 7
2.2 Libros . . . 7
2.3 Artículos y publicaciones . . . 8
2.4 Presentaciones . . . 8
2.5 Sitios web . . . 8
2.6 *Dashboards* . . . 8

3 Proceso de trabajo **10**
3.1 Conjunto de herramientas . . . 10
3.2 Producir HTML . . . 10
3.3 Producir PDF . . . 10
3.3.1 Personalizar documentos PDF . . . 10

4 Documentos individuales **11**
4.1 Creación del documento con RStudio . . . 11
4.2 Estructura del documento . . . 13
4.2.1 El preámbulo . . . 14
4.2.2 Listado de opciones . . . 14
4.2.3 Sintaxis Markdown básica . . . 15
4.3 Creación del documento (*output*) . . . 15
4.3.1 Previsualización . . . 15
4.3.2 Seleccionar el tipo de documento . . . 16
4.3.3 Opciones básicas de configuración . . . 17
4.4 *Chunks* de código ejecutable . . . 18

4.5 Herramientas para el autor . . . 21
4.5.1 Secciones del documento . . . 23
4.5.2 Ecuaciones . . . 24
4.5.3 Tablas . . . 24
4.5.4 Llamadas . . . 24
4.5.5 Citas bibliográficas . . . 24
4.5.6 Maquetación del documento . . . 24

II Libros con Quarto **25**

5 Libros **26**
5.1 Herramientas de redacción . . . 26
5.1.1 Figuras . . . 26
5.1.2 Tablas . . . 26
5.1.3 Ecuaciones . . . 26
5.1.4 *Callouts* . . . 26
5.2 Gestión de referencias . . . 26
5.2.1 Referencias cruzadas en el documento . . . 26
5.2.2 Referencias bibliográficas . . . 26
5.3 Plantillas y personalización . . . 26

6 Taller: colección de apuntes **27**
6.1 Plantillas y herramientas . . . 27
6.2 Gestión del proyecto . . . 27
6.3 Publicación . . . 27

III Publicaciones **28**

7 Artículos y publicaciones científicas **29**
7.1 Replicabilidad en publicaciones científicas . . . 29
7.2 Figuras y gráficos para publicación . . . 29
7.3 EL paquete `rticles` . . . 29
7.4 Ejemplos y recomendaciones . . . 29

8 Principios FAIR **30**
8.1 Visión general . . . 30
8.2 Publicación del código fuente . . . 30
8.3 Publicación de conjuntos de datos . . . 30
8.4 Gestión de referencias . . . 30

9 Recursos adicionales **31**

Referencias **32**

Apéndices **33**

A Comandos de utilidad **33**
A.1 Comandos Quarto 33
A.2 Celdas de código en R 33

B Entornos de desarrollo para Quarto **34**
B.1 R Studio 34
B.2 Visual Studio 34

C Paquetes R de interés **35**
C.1 `rticles` 35

D Documentos PDF con LaTeX **36**
D.1 Salida en formato PDF 36
D.2 Acerca de LaTeX 36
D.3 Ejemplos prácticos 36

Referencias **37**

Preface

Este taller explica cómo utilizar Quarto, un software de creación de documentación científica, para crear publicaciones de calidad que integren contenido de texto formateado, gráficos, tablas, así como resultados de ejecución de código software en varios lenguajes, todo ello integrado en el propio documento.

Quarto se ha convertido en una herramienta muy versátil y potente en el conjunto de herramientas de los programadores científicos, en especial por proporcionar soporte para implementar buenas prácticas de **investigación reproducible**, incluyendo los principios FAIR. El intenso movimiento iniciado desde hace años por la comunidad científica para garantizar acceso en abierto no solo al producto final (e.g. una publicación) sino también a materiales adicionales (códig fuente, conjuntos de datos, figuras, procesos de trabajo, archivos de configuración, etc.) se ha convertido en un objetivo insoslayable para académicos y especialistas en muchos campos diferentes. Muchas publicaciones científicas de prestigio exigen ahora a los autores enviar estos materiales auxiliares junto con los borradores de sus manuscritos, para permitir que otros colegas reproduzcan y validen los resultados, repliquen sus estudios sobre nuevas cohortes de individuos o elementos o para contribuir a la interpretación de los resultados obtenidos.

Con la herramienta Quarto se puede combinar texto formateado (escrito en Markdown) junto con secciones de código ejecutable, todo integrado en un mismo documento. Estas secciones o *chunks* de código ejecutable pueden estar escritas en varios lenguajes: R, Python, Julia u Observable. Es incluso posible combinar en un mismo documento o colección de documentos secciones de código escritas en diferentes lenguajes de programación.

Este es un **taller práctico** que presenta ejemplos reales y comandos para crear paso a paso tus propios documentos con Quarto en poco tiempo. Además, junto a la explicación de los conceptos clave para entender este proceso también se ofrecen recomendaciones sobre buenas prácticas de trabajo, para guiar a los aprendices de Quarto en la dirección correcta.

Puedes aprender muchos más detalles sobre cuarto en la guía en línea https://quarto.org/docs/guide/. En particular, en https://quarto.org/docs/books se documenta en detalle cómo crear libros como este utilizando Quarto.

Parte I

Quarto

1 Documentos científicos

En su actividad diaria, los estudiantes, académicos y especialistas científicos producen gran cantidad de documentación de todo tipo: notas de laboratorio, apuntes, memorandos, informes técnicos y, sobre todo, artículos científicos para publicar sus descubrimientos y avances en un área de conocimiento. Normalmente, la creación de este tipo de documentos científicos conlleva una gran cantidad de tareas que involucran diferentes herramientas y posibles puntos de fallos.

La *figura X.X* muestra una descripción general esquemática de un proceso de trabajo clásico para la creación de documentos científicos. Con frecuencia, el elemento principal es un archivo maestro de procesador de textos (Word, OpenOffice/LibreOffice, etc.), una página web o un fichero LaTeX (si vamos a crear un documento PDF) que recoge todo el contenido.

Este archivo maestro se va llenando de contenido que procede de diversas fuentes, como por ejemplo:

- figuras y esquemas generados manualmente o mediante código software (como gráficos de visualización de datos);
- tablas y resúmenes que describen conjuntos de datos y resultados;
- resultados y evaluación del rendimiento de modelos o algoritmos; estadísticos o de aprendizaje automático;
- fórmulas y ecuaciones matemáticas;
- tablas de datos y otra información de utilidad;
- referencias bibliográficas (normalmente generadas con ayuda de algún programa de gestión de información bibliográfica).

Muchos de estos elementos fuerzan a los usuarios a ejecutar una y otra vez herramientas y programas externos, procedimientos y otras tareas para incorporar luego los nuevos resultados al fichero maestro. Debemos admitir que este proceso, en su mayor parte manual, además de tedioso puede ser muy propenso a que cometamos errores o descuidos. “¡Espera! He olvidado actualizar la figura 1”. “¿SEguro que estos son los últimos resultados de evaluación del modelo M?” “¿Has comprobado que hemos cargado la última versión del ficher de datos D?” Estas son preguntas comunes que surgen en el día a día de los equipos científicos.

Sin embargo, sería genial si no fuese necesario realizar todo ese proceso manual y, en ocasiones, muy frustrante de forma manual. ¿Tenemos alguna alternativa para evitarlo? Sí, la tenemos. La respuesta a nuestras necesidades nos la brinda un concepto muy poderoso: la **programación literaria.

1.1 Programación literaria

El concepto de programación literaria fue acuñado por el profesor Donald E. Knuth ya en 1984 Knuth (1984). Sí, no has leído mal, hace más de 40 años. Este concepto establece que debería ser posible integrar, en un solo documento científico, texto formateado y resultados de la ejecución de código software para componer dicho documento de forma dinámica. Entonces, ¿por qué hemos tardado tanto en poner en práctica esta idea? La visión de Knuth, aunque muy adelantada a su tiempo, era correcta, pero la tecnología de la época no permitía ponerla en práctica.

Sin embargo, hoy día contamos con todos los elementos indispensables para llevar esta idea a la práctica. Es más, contamos con una herramienta, Quarto, que nos va a permitir automatizar y gestionar todo el proceso de creación de documentos de programación literaria de forma rápida y fiable.

1.2 Investigación reproducible

1.3 Quarto para publicaciones científicas

Ahora que ya conocemos el concepto fundamental sobre el que se asienta el funcionamiento de Quarto y su aplicación para conseguir un mayor nivel de reproducibilidad y transparencia en nuestro proceso científico, vamos a explicar con más detalle el proceso que sigue Quarto para componer un documento. La figura X.X presenta un esquema con el proceso de creación del documento y los elementos y herramientas que entran en juego para conseguirlo.

- **Quarto engine**
- **Markdown** (contenido formateado):
- **Programming engine**: R, Python, Julia (lenguajes de programación que se pueden ejecutar en un entorno REPL).
- **Pandoc** (traductor universal de formatos documentales):
  - Primero se crea una salida en formato Markdown.
  - Después, usamos Pandoc para convertir el contenido Markdown en el tipo de salida seleccionado.
  - Opciones disponibles: HTML, PDF, Word.

1.4 Instalación de Quarto

https://quarto.org/docs/get-started/

Última versión disponible: `1.5.57`.

2 Tipos de documentos

En este capítulo, presentamos los principales tipos de documentos y colecciones de contenidos científicos que podemos generar con Quarto.

2.1 Documentos individuales

La forma más sencilla de trabajar con Quarto es crear un documento individual. Dicho documento podrá utilizar las secciones o *chunks* de código para leer datos de entrada o descargarlos de alguna fuente, procesarlos, analizarlos y mostrar los resultados. Se pueden añadir gráficos, tablas, ecuaciones, referencias bibliográficas y muchos otros elementos.

Los documentos tienen siempre una estructura estándar:

- *Preámbulo*: en el que se especifican opciones de configuración para la creación del documento con Quarto y sus herramientas asociadas.
- *Cuerpo*: la sección que alberga el contenido principal del documento, incluyendo secciones de texto formateado en Markdown y secciones de código ejecutable. El código software se podrá mostrar, si resulta de utilidad, o quedar oculto en el resultado final.
- *Referencias*: Al final del documento se incluyen las referencias bibliográficas, como es habitual en los textos científicos.

2.2 Libros

La evolución natural del caso anterior es reunir una colección de documentos individuales en un solo libro. *Quarto books* permite crear este tipo de documentos, estructurados en partes, capítulos y secciones. Las opciones de configuración permitirán confeccionar una portada de introducción para el sito web que contiene los capítulos (un documento por capítulo) o bien los elementos necesarios para crear un libro en PDF, semejante a los publicados por una editorial.

2.3 Artículos y publicaciones

Uno de los resultados clave en todo proceso científico es la producción de artículos y publicaciones (informes técnicos, etc.) que recojan los resultados y avances científicos conseguidos. En este caso, Quarto también nos podrá ayudar, con la colaboración de otros elementos indispensables como el paquete R `rticles`, que proporciona plantillas para generar artículos según las especificaciones de las principales publicaciones y editoriales científicas en multitud de campos de conocimiento.

2.4 Presentaciones

También es posible generar presentaciones (normalmente, en formato HTML) con diapositivas mediante Quarto. En este caso, tendríamos el soporte de varios paquetes y entornos de creación de presentaciones web a nuestra disposición, como `reveal.js` (HTML), Beamer (para LaTeX/PDF) o formato PPTX de MS Office.

Este caso no lo trataremos en este taller, pero se puede obtener más información en la guía online, disponible en https://quarto.org/docs/presentations/.

2.5 Sitios web

Otra opción que puede resultar interesante es crear sitios web personales (por ejemplo, para mostrar nuestro CV y una selección de trabajos destacados, publicaciones, etc.), blogs e incluso sitios web corporativos (organización, grupo de investigación) de forma rápida mediante Quarto. Existen numerosas plantillas gratuitas y de pago ya disponibles para crear sitios web con un aspecto armonizado, aunque necesitaremos aprender un poco de HTML y CSS para poder personalizar aún más nuestra web.

Aquí tenemos un ejemplo de sitio web de un investigador en tecnología medioambiental creado con Quarto: https://www.mm218.dev/. Más ejemplos de diferentes tipos de sitios web generados con Quarto: https://drganghe.github.io/quarto-academic-site-examples.html.

Se puede conseguir más información y tutoriales para crear sitios web con Quarto en https://quarto.org/docs/websites/.

2.6 *Dashboards*

Por último, es posible crear cuadros de mandos o *dashboards* personalizados para monitorización de datos, análisis de modelos y resultados o bien para ejemplos y aplicaciones docentes utilizando Quarto, tal y como se describen en la guía https://quarto.org/docs/dashboards/.

En este caso podemos incluir entre las herramientas Shiny, un paquete software para R (también disponible para Python) con el que crear aplicaciones interactivas basadas en datos de forma rápida y sencilla.

3 Proceso de trabajo

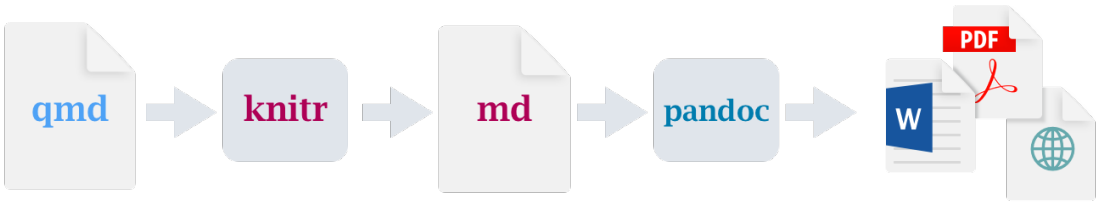


Figura 3.1: How it works. Fuente: RStudio.

3.1 Conjunto de herramientas

- Quarto
- Knitr
- Markdown
- Pandoc

Se necesitan programas para cargar los documentos de salida, según el formato: navegador web (HTML), MS Word (archivos DOC), visor PDF (archivos PDF).

3.2 Producir HTML

3.3 Producir PDF

3.3.1 Personalizar documentos PDF

Se pueden utilizar plantillas de documentos LaTeX predefinidas. Por defecto, Quarto utiliza varias plantillas de la colección de paquetes LaTeX koma-script.

4 Documentos individuales

La manera más sencilla de comenzar a utilizar Quarto es crear documentos individuales. Se trata de documentos autocontenidos, que incorporan texto formateado y código ejecutable en un único archivo.

Para crear un documento nuevo con Quarto, simplemente podemos usar las opciones del menu de RStudio o MS Visual Code, o bien crear un archivo con extensión `.qmd`.

4.1 Creación del documento con RStudio

Antes de empezar, comprueba que has instalado el software Quarto en tu máquina. Es un programa software independiente, que tiene que estar instalado para que el resto del proceso funcione (consulta la sección Sección 1.4).

Si ya tenemos instalada una versión reciente de RStudio, necesitaremos instalar los siguientes paquetes para el ejemplo:

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

Ahora, en RStudio creamos un nuevo proyecto eligiendo la opción *Quarto project*, tal y como aparece en la Figura 4.1.

Podemos nombrar el directorio de nuestro proyecto como `primer-ejemplo` y pulsamos `Create Project`.

Como resultado, nos debe aparecer un nuevo proyecto abierto en pantalla, con el aspecto que se muestra en la Figura 4.2.

En concreto, en el panel superior izquierdo podemos comprobar que, por defecto se ha abierto el editor *Visual*, que permite crear documentos Quarto de forma más intuitiva. Sin embargo, para empezar a familiarizarnos desde el principio con la estructura de un documento en quarto vamos a cambiar al editor *Source* para ver el código fuente, pulsando en el botón que se muestra en la figura Figura 4.3.

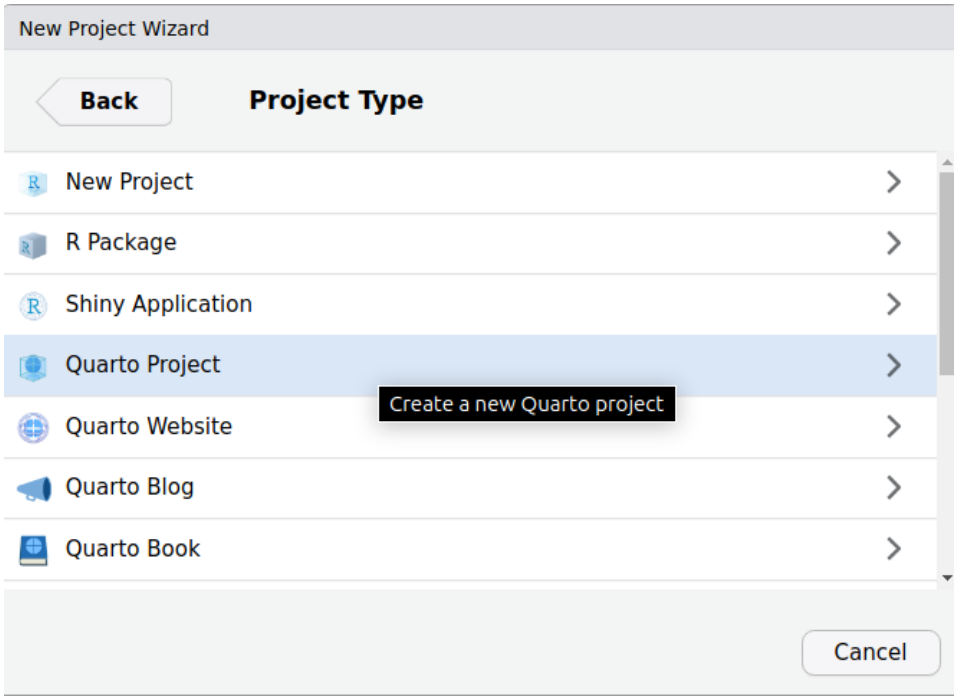


Figura 4.1: New Quarto project

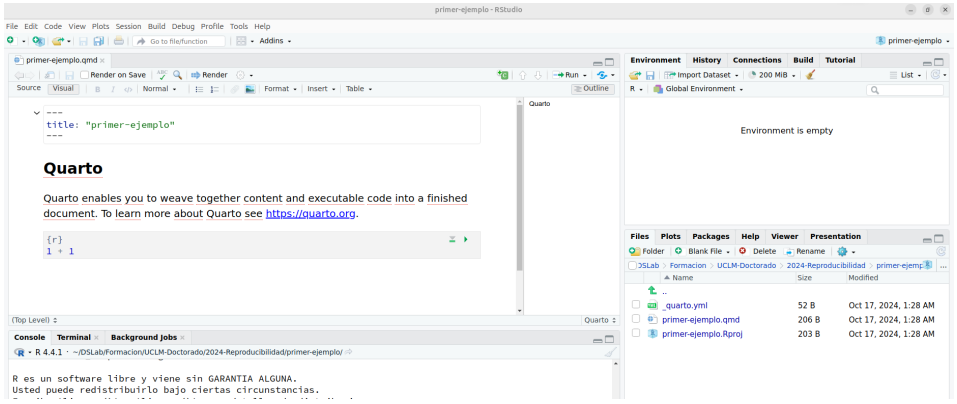


Figura 4.2: First example

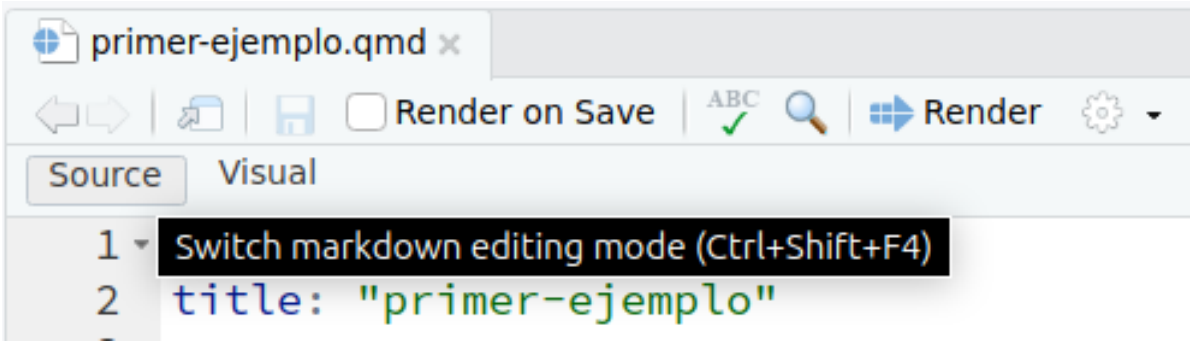


Figura 4.3: Source editor

4.2 Estructura del documento

La estructura de un documento individual en Quarto es esta.

```
---
title: "Mi primer documento"
author: John Doe
date: 2024-10-24
---

Aquí tenemos algo de contenido en formato **Markdown**.

```{r}
#| label: Etiqueta

1 + 1
```

Contenido *adicional* en Markdown.
```

El contenido del archivo consta de dos partes:

- **Preámbulo**: está delimitado por dos marcas `---`. Dentro de esta área podemos asignar valores a opciones de configuración para maquetar y crear el documento, tales como el título, autor/es, fecha, etc. También podemos configurar diversas opciones relacionadas con el formato de salida de los documentos.
- **Cuerpo del documento**: se compone de párrafos de texto formateados utilizando la sintaxis de marcado Markdown, que veremos después. Además, también se pueden intercalar en el texto fragmentos de código ejecutable o *chunks*, que se marcan siguiendo una sintaxis especial (como vemos en el ejemplo anterior).

Cada *chunk* de código ejecutable está delimitado de la siguiente manera

```
```{r}
# Código en R
```
```

Soporte para otros lenguajes de programación

Aunque en este taller nos centramos en el lenguaje R, debemos sabes que Quarto también soporta otros lenguajes de programación como Python, Julia u Observable.
Podemos cambiar el lenguaje de programación de cada *chunk* indicando su nombre al comienzo, como por ejemplo:

```
```{pyton}
# Código en Python
```
```

Sin embargo, para que pueda funcionar necesitaremos realizar algunas tareas adicionales de configuración.

4.2.1 El preámbulo

4.2.2 Listado de opciones

Existe un extenso listado de opciones de configuración que podemos incluir en nuestros documentos.

- *Opciones para salida HTML*: permiten configurar diversos aspectos básicos del documento, tales como el título y subtítulo, fecha, autor (o lista de autores), resumen o DOI; opciones de formato como el tema o estilos avanzados para contenido HTML con CSS; numeración y tabla de contenidos, etc.
  - Opciones básicas para HTML con Quarto.
  - Lista completa de opciones HTML con Quarto.
- *Opciones para salida PDF*: ofrecen la posibilidad de configurar múltiples parámetros para la creación del documento en este formato, muchas de ellas similares a las de la salida en HTML. Una opción particularmente relevante es elegir el formato de documento LaTeX (opción `documentclass`), que define el aspecto general de la maquetación que se va a emplear. Por defecto, se emplean clases del metapaquete KOMA Script, como `scrartcl` o `scrbook`. También es importante indicar la opción `papersize`, en nuestro caso para

garantizar que se usa un formato estándar como el A4. El formato de las citas también es relevante, pudiendo elegir, por ejemplo, el motor BibLaTeX que es más potente, con soporte multilenguaje y para codificación de caracteres UTF-8 nativa. Por último, también es importante indicar el motor de compilación. Al contrario que en HTML, cuando generamos documentos en PDF es **imprescidible** que tengamos **instalada una distribución TeX/LaTeX** previamente en nuestro sistema, para compilar y generar los documentos. Si no disponemos de ninguna todavía, se puede instalar TinyTeX, una distribución ligera de TeX Live que tiene mucho menor tamaño (~100 MB frente a los más de 4 GB de TeX Live completa). Si queremos una flexibilidad total en la maquetación del documento, se recomienda encarecidamente usar el motor XeLaTeX (opción `pdf-engine: xelatex`), que es el *valor por defecto* que utiliza Quarto.

  - Opciones básicas para PDF con Quarto.
  - Lista completa de opciones disponibles en PDF con Quarto.

4.2.3 Sintaxis Markdown básica

En el siguiente enlace puedes encontrar un rápido tutorial básico que muestra las opciones básicas de la sintaxis Markdown aceptada en documentos Quarto para formatear el contenido textual.

- Guía básica de sintaxis markdown.

4.3 Creación del documento (*output*)

Por defecto, si no indicamos nada Quarto generará un solo formato de salida del documento en HTML. Sin embargo, es posible definir más de un formato de salida incluyendo más opciones de configuración. Por supuesto, se pueden indicar diferentes opciones para generar varios formatos de salida simultáneamente, o bien para elegir el formato de salida que queremos producir en función de nuestros intereses, seleccionando el formato que necesitamos al previsualizar o al generar el documento definitivo.

4.3.1 Previsualización

Para previsualizar el documento tenemos que pulsar el botón *Render* en el menú de herramientas de la interfaz de RStudio, tal y como se muestra en la Figura 4.4.

Por defecto, lo normal es que se abra nuestro navegador web principal o un panel en la interfaz de RStudio mostrando la página HTML con el documento ya generado. Pulsando sobre el icono con un engranaje junto al botón *Render* se puede seleccionar, entre otros aspectos, el tipo de previsualización que queremos que se lance tras completar la creación del documento

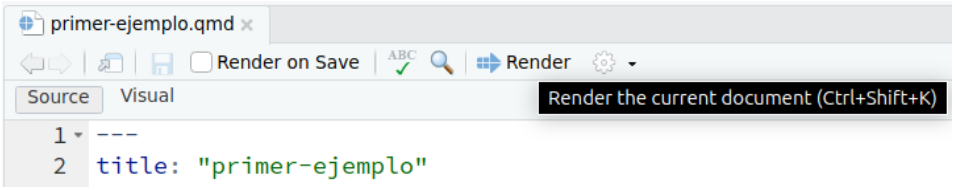


Figura 4.4: Botón *Render* para previsualizar el documento generado.

o desactivar por completo dicha previsualización. La opciones disponibles se muestran en la Figura 4.5

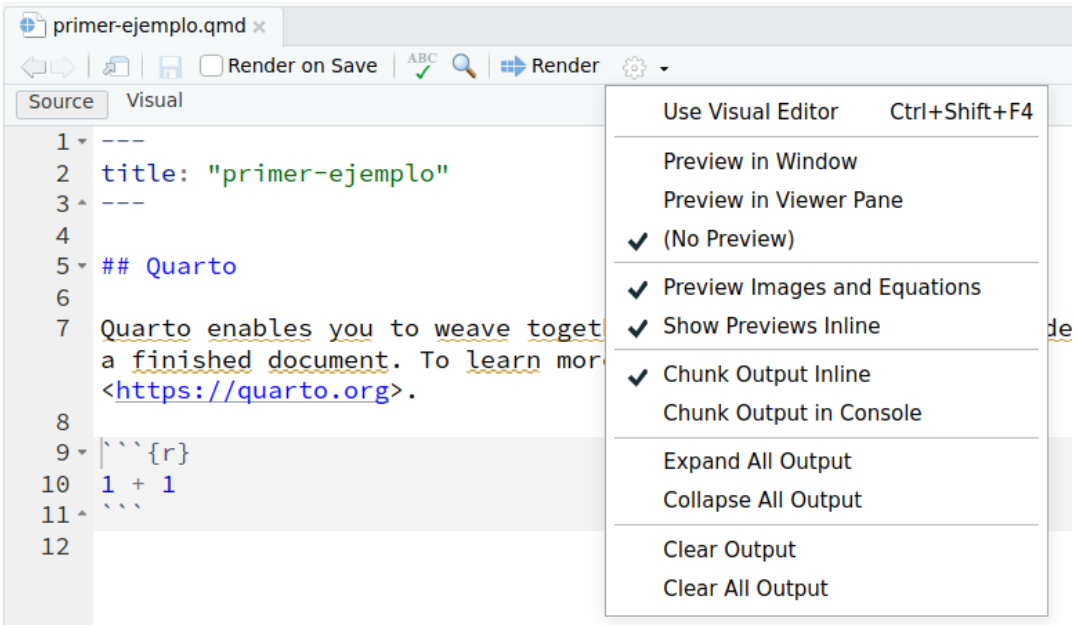


Figura 4.5: Opciones de previsualización de la operación *Render*.

4.3.2 Seleccionar el tipo de documento

Cuando tenemos varias opciones de formato de salida configuradas en nuestro documento, podemos elegir en tiempo de previsualización cuál de los formatos se elige para generar el documento. En la Figura 4.6 se puede observar un ejemplo de documento que incluye configuración para dos formatos de salida (HTML y PDF) y el cambio en el botón *Render*, en el que ahora aparece una pequeña flecha negra justo a la derecha del icono del botón para desplegar las dos opciones de salida disponibles.

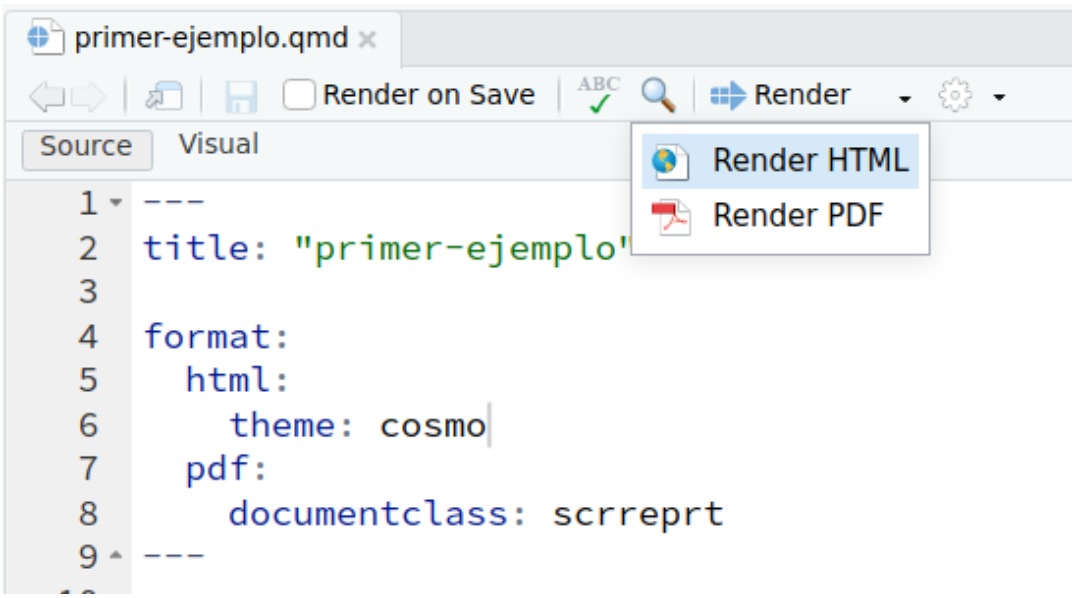


Figura 4.6: Selección de formatos de salida con la operación *Render*.

4.3.3 Opciones básicas de configuración

A continuación, se presenta un ejemplo de algunas opciones básicas de configuración que suelen ser habituales en documentos con formato de salida HTML.

```yaml
---
title: "Mi primer documento"
author:
  - "John Doe"
  - "Mary Jane"
date: 2024-10-24

lang: es
bibliography: references.bib

format:
  html:
    theme: cosmo
    toc: true
    number-sections: true
    html-math-method: katex
    css: styles.css
  pdf:
```

```
    documentclass: scrreprt
---

RESTO DEL DOCUMENTO
```

En este ejemplo se indica, además del autor y la fecha, una lista de dos autores, el lenguaje principal del documento (español), el archivo de referencias de bibliografía (en formato `.bib`) y ya dentro de las opciones HTML, el tema de maquetación, la inclusión de una tabla de contenidos (por defecto situada en la parte superior derecha), numeración de secciones, selección del motor para renderizar ecuaciones en el documento y un archivo de estilos personalizados en formato CSS para ajustar algunas opciones finas de maquetación.

Una opción que conviene destacar es la de forzar a que todos los recursos (imágenes, información de estilos, etc.) estén integrados en el propio archivo HTML, para facilitar la compartición o publicación directa del documento sin necesidad de aportar también los archivos auxiliares necesarios para mostrarlo en el navegador. Esta opción se muestra a continuación:

```
format:
  html:
    embed-resources: true
```

4.4 *Chunks* de código ejecutable

La característica más diferencial de los documentos creados con Quarto es la posibilidad de intercalar fragmentos de código ejecutable, llamadas *chunks* en el propio documento. Esto incluye también la opción de que dicho código genere diferentes resultados (numéricos, gráficos, tablas, animaciones, etc.) que se integren directamente en el documento. De este modo, si mantenemos actualizado el código siempre se generarán las versiones correcgtas de dichos resultados.

Los fragmentos de código ejecutable tienen la siguiente estructura:

````
```{r}
#| label: id-fragmento

# Aquí va el código ejecutable
a = c(1, 2, 3, 4)
b = a^2
```
````

La tripleta de caracteres ``` se denomina *fence* y delimita el comienzo y el final del fragmento de código. Justo a continuación del delimitador de apertura se escribe entre llaves el identificador del lenguaje de programación en el que está escrito el código de ese fragmento. Esa información se usa para elegir el resaltado de sintaxis apropiado para mostrar el código de ese lenguaje y para seleccionar el intérprete que ejecuta el código y produce los resultados.

En las siguientes líneas podemos incluir una o varias **opciones de configuración** específicas para ese fragmento de código, mediante la sintaxis `#| opcion: valor`. Por ejemplo, en el fragmento anterior la opción `#| label: id-fragmento` crea una etiqueta (que debe ser unívoca) para identificar a ese fragmento de código dentro del documento.

- Lista de opciones para fragmentos de código.

Algunas opciones de uso frecuente son:

- `eval: true | false | [...]`: Indica si se debe evaluar (ejecutar) el contenido de ese fragmento. Se puede pasar una lista de números de línea positivos o negativos para seleccionar explícitamente qué líneas de código se incluyen (positivos) o excluyen (negativos) de la ejecución.
- `echo: true | false | fenced | [...]`: Indica si se debe incluir el código fuente del fragmento en el documento o no. La opción `fenced` incluye también el delimitador de celda como parte de la salida. Por último, también acepta una lista de números de línea positivos o negativos para seleccionar qué líneas de código se mostrarán o no en el fragmento.
- `output: true | false | asis`: Para decidir si el resultado de la ejecución del código se incluye o no en el documento. El valor `asis`fuerza a que el resultado se trate como contenido Markdown en crudo.
- `warning: true | false`: Indica si se deben incluir los mensajes de aviso en la salida.
- `error: true | false`: Marca si los mensajes de error generados se incluyen en la salida.
- `message: true | false`: Indica si los mensajes de información generados se incluyen en la salida.

Cuando los fragmentos generan figuras, estas se insertan dentro del propio documento. Veamos un ejemplo:

````
```{r}
#| label: fig-example-cars
#| fig-cap: "Gráfico de correlación lineal positiva entre el kilometraje en ciudad y en carre

library(ggplot2)
#| label: scatterplot
#| echo: true

ggplot(mpg, aes(x = hwy, y = cty, color = cyl)) +
````
````

```
  geom_point(alpha = 0.5, size = 2) +
  scale_color_viridis_c() +
  theme_minimal()
```

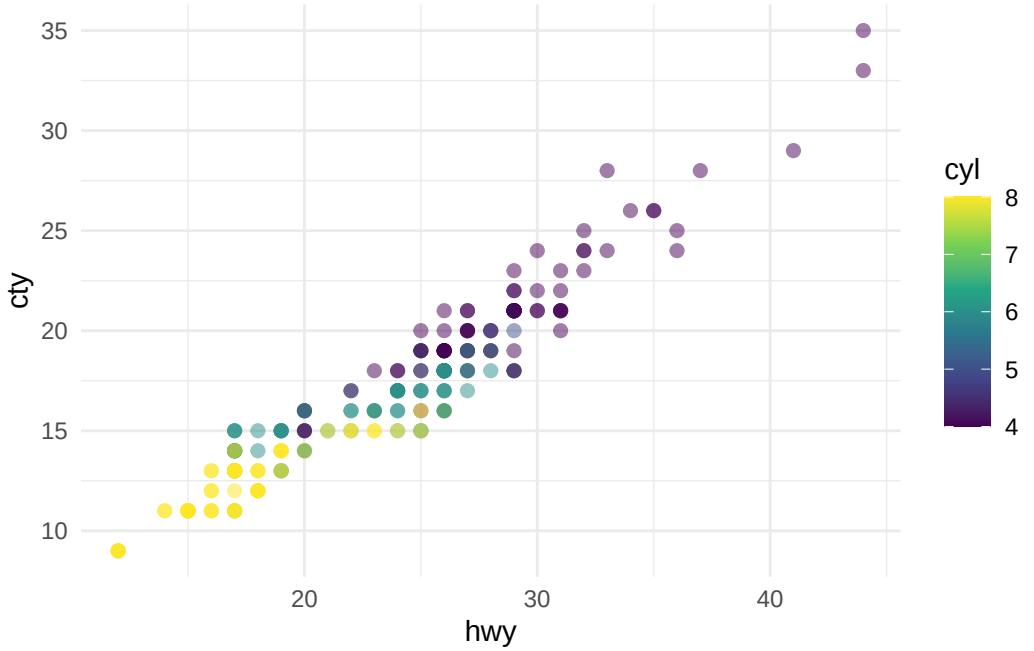


Figura 4.7: Gráfico de correlación lineal positiva entre el kilometraje en ciudad y en carretera de diferentes modelos de coches.

> **Numeración automática de figuras**
>
> Es importante que *el identificador de fragmento* que elegimos para código que genera una o varias figuras *comience por el prefijo* `fig-`. De ese modo, nos aseguramos de que Quarto le asigne automáticamente una numeración a la figura generada y que podamos crear referencias cruzadas (enlaces internos) a dicha figura en nuestro documento.
> Como veremos más adelante, otros tipos de salida como las tablas también necesitan que se les asigne un patrón concreto en su identificador de fragmento para que se numeren de forma automática y se puedan referenciar dentro del documento.

La gestión de figuras en Quarto es bastante sofisticada, hasta el punto de que se pueden organizar de forma sencilla varias subfiguras con sus respectivas descripciones individuales, como se muestra en el siguiente ejemplo usando algunas opciones adiconales.

````
```{r}
#| label: fig-mpg-subplot
#| fig-cap: "Kilometraje en ciudad y en carretera de 38 modelos populares de coches."
#|
#| fig-subcap:
#|   - "Color por núm. de cilindros."
#|   - "Color por cubicaje del motor, en litros."
#| layout-ncol: 1

ggplot(mpg, aes(x = hwy, y = cty, color = cyl)) +
  geom_point(alpha = 0.5, size = 2) +
  scale_color_viridis_c() +
  theme_minimal()

ggplot(mpg, aes(x = hwy, y = cty, color = displ)) +
  geom_point(alpha = 0.5, size = 2) +
  scale_color_viridis_c(option = "E") +
  theme_minimal()
```
````

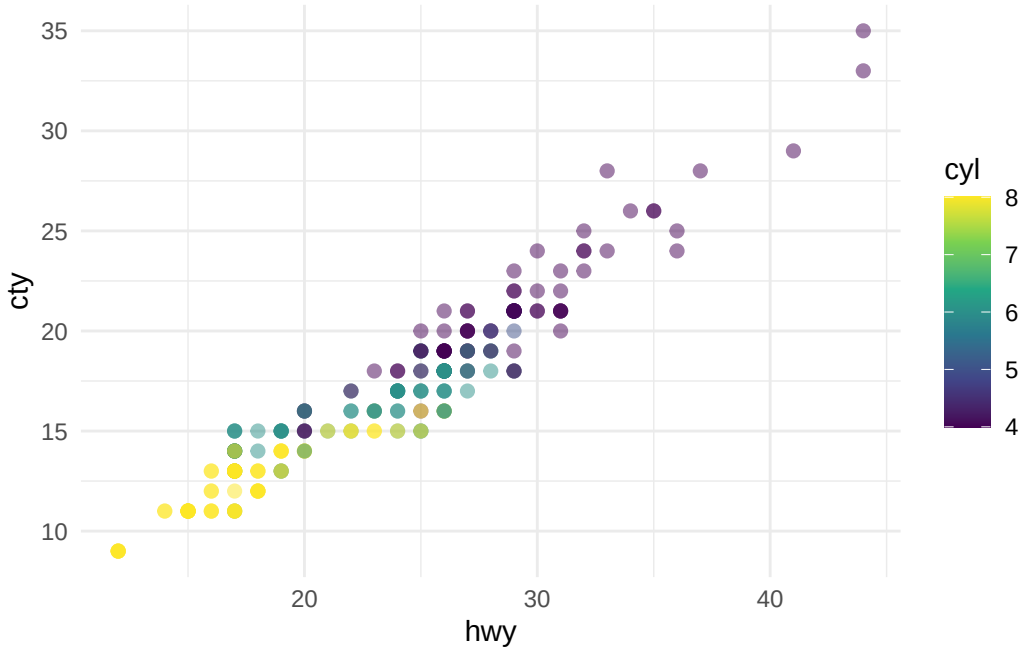
Algunas opciones frecuentes para *chunks* que generan figuras son:

- `fig-width`: Ancho de la figura.
- `fig-height`: Alto de la figura.
- `fig-cap`: String entre comillas que se insertará como descripción al pie de la figura (*caption*).
- `fig-alt`: Mensaje de texto alternativo que rellena el atributo `alt` de la imagen HTML (por ejemplo, para mejorar la accesibilidad del contenido).
- `fig-dpi`: Ajuste de la resolución de la figura (en puntos por pulgada).

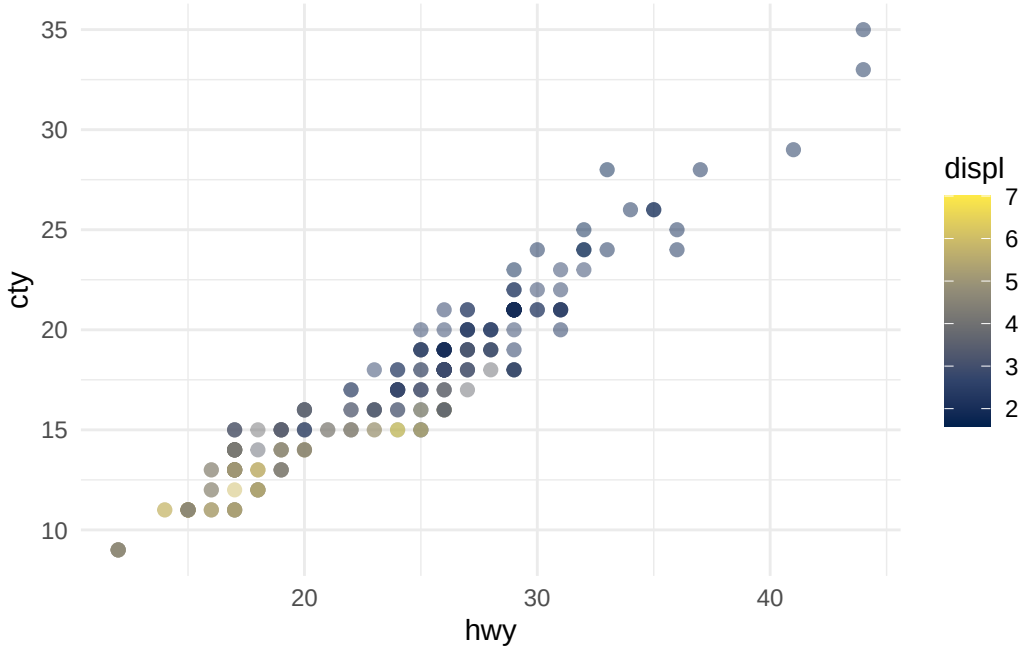
El tutorial sobre fragmentos de código ejecutables de la documentación oficial presenta más información y ejemplos sobre cómo utilizar esta potente característica de Quarto.l

4.5 Herramientas para el autor

Además de la capacidad de integrar código ejecutable y sus resultados en nuestros documentos científicos, Quarto incluye un buen número de recursos y herramientas para ofrecer una experiencia de autoría completa y eficiente.



(a) Color por núm. de cilindros.



(b) Color por cubicaje del motor, en litros.

Figura 4.8: Kilometraje en ciudad y en carretera de 38 modelos populares de coches.

4.5.1 Secciones del documento

Como ya vimos en el ejemplo de la Sección 4.3.3, existen dos opciones de configuración del documento HTML que nos permiten numerar las secciones e incorporar una tabla de contenidos generada automáticamente en la parte superior derecha de nuestro documento.

```
format:
  html:
    toc: true
    number-sections: true
```

Un funcionalidad importante para la creación de documentación científica es poder incluir **referencias cruzadas**, es decir, enlaces que nos lleven a otras secciones del documento. En Quarto esto se consigue siguiendo un sencillo procedimiento en dos pasos:

1. Añadimos una *etiqueta única* para identificar la sección con la sintaxis:

```
## Título de sección {#sec-etiqueta}
```

2. Referenciamos en otro lugar del texto la etiqueta que hemos creado para esa sección, de forma que Quarto crea automáticamente el enlace (referencia cruzada) a dicha sección:

```
En el texto añadimos una referencia a la @sec-etiqueta
```

Se puede ver un ejemplo de este tipo de referencias cruzadas creadas de forma automática al comienzo de esta misma sección. Por el contrario, si queremos que una sección del documento se excluya del esquema de numeración del resto de secciones, usamos en el título de esa sección la etiqueta especial:

```
## Sección no numerada {.unnumbered}
```

Existen varias opciones adicionales que controlan la forma y estilo con que se crean y numeran las secciones. Algunas de ellas son:

- `anchor-sections`: Hace que se muestre un enlace de anclado (para enlazar directamente esa sección en otro documento) cuando se pasa el ratón por encima del título de una sección.
- `toc-depth`: Especifica cuántos niveles de profundidad en la numeración de secciones aparecen en la tabla de contenidos. Por defecto se muestran 3 niveles.
- `toc-location: body | left | right | left-body | right-body`: Controla la ubicación en la que aparece la tabla de contenidos en el documento.
- `toc-title`: Cadena de caracteres con el título de la tabla de contenidos.

- `toc-expand`: Indica si se deben expandir todas las secciones de la tabla de contenidos o deben quedar colapsadas para que el usuario vaya pulsando en las que quiera expandir.
- `number-depth`: Determina la profundidad máxima a la que se numeran las secciones del documento (cuidado, debería estar en consonancia con el valor asignado a la opción `toc-depth`).
- `number-offset`: Permite ajustar el número por el que se empiezan a numerar las secciones. Si queremos que el documento comience a numerar la sección de más alto nivel como "4" entonces usamos `number-offset: 3`. Si queremos que el documento empiece en una sección de nivel 2 con numeración "1.5" debemos especificar `number-offset: [1,4]`. Definir un valor para esta opción implica que automáticamente `number-sections: true`.

4.5.2 Ecuaciones

4.5.3 Tablas

4.5.4 Llamadas

4.5.5 Citas bibliográficas

4.5.6 Maquetación del documento

https://quarto.org/docs/get-started/authoring/

Parte II

Libros con Quarto

5 Libros

5.1 Herramientas de redacción

5.1.1 Figuras

5.1.2 Tablas

5.1.3 Ecuaciones

5.1.4 *Callouts*

5.2 Gestión de referencias

5.2.1 Referencias cruzadas en el documento

5.2.2 Referencias bibliográficas

5.3 Plantillas y personalización

6 Taller: colección de apuntes

6.1 Plantillas y herramientas

6.2 Gestión del proyecto

6.3 Publicación

Parte III

Publicaciones

7 Artículos y publicaciones científicas

7.1 Replicabilidad en publicaciones científicas

7.2 Figuras y gráficos para publicación

7.3 EL paquete `rticles`

7.4 Ejemplos y recomendaciones

8 Principios FAIR

8.1 Visión general

8.2 Publicación del código fuente

8.3 Publicación de conjuntos de datos

8.4 Gestión de referencias

DOI

Figshare

Zenodo

arXiv

etc.

9 Recursos adicionales

Listado de recursos adicionales de interés.

See Knuth (1984) for additional discussion of literate programming.

Referencias

Knuth, D. E. (1984). Literate Programming. *Comput. J.*, *27*(2), 97-111. https://doi.org/10.1093/comjnl/27.2.97

A Comandos de utilidad

A.1 Comandos Quarto

A.2 Celdas de código en R

B Entornos de desarrollo para Quarto

B.1 R Studio

B.2 Visual Studio

C Paquetes R de interés

C.1 `rticles`

Shiny

IOSlides, otras...

Otros

D Documentos PDF con LaTeX

D.1 Salida en formato PDF

D.2 Acerca de LaTeX

D.3 Ejemplos prácticos

Referencias

Knuth, D. E. (1984). Literate Programming. *Comput. J.*, *27*(2), 97-111. https://doi.org/10.1093/comjnl/27.2.97