Ayudantía 1

Benjamín Puran

2024-08-13

RMarkdown y Quarto

RMarkdown

Está específicamente diseñado para usuarios de R. Permite crear documentos que combinan código (principalmente R, aunque también soporta otros lenguajes como Python) y texto. Se utiliza principalmente para crear informes, presentaciones y dashboards dentro de la comunidad de R.

Quarto

`Quarto` es una herramienta más general y flexible, diseñada como una alternativa de próxima generación a RMarkdown. Soporta múltiples lenguajes de programación (R, Python, Julia, Observable, etc.) y no está limitado a un lenguaje específico. `Quarto` está diseñado para ser una herramienta integral que permite crear no solo informes, sino también blogs, sitios web, presentaciones y artículos científicos.

`Rmarkdown` soporta una variedad de formato de salida como HTLM, PDF o Word. `Quarto`, por otra parte, una gama más amplia de formatos de salida y permite una mayor personalización y configuración a través de su sistema de mandatos YAML.

Código ("chunks")

`RMarkdown` permite introducir código de `R` en el documento de texto, evaluar tal código y mostrar los resultados directamente en el informe. A modo de ejemplo, comenzaremos mostrando un `summary` de la base de datos `iris`, que viene incluida en `R`.

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
```

```
##
##
```

El trozo de arriba es un chunk de código `R`. Al compilar el documento, (click en el botón `knitr`, en el panel) el código se ejecutará y mostrarán los resultados en el documento final. Los chunks pueden tienen diversas opciones que permiten una mayor flexibilidad en como se muestra el código y los resultados. Las opciones más usadas son:

- echo
- eval

Por ejemplo, el chunk abajo mostrará el código (`echo = TRUE`), lo evaluará y mostrará los resultados en el documento final (`eval = TRUE`). Así se ve:

```
a <- summary(iris)
print(a)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Si sólo queremos mostrar el código (`echo = TRUE`) pero no evaluarlo (`eval = FALSE`), escribimos lo siguiente:

```
a <- summary(iris)
print(a)
```

Por el contrario, si queremos evaluar el código, mostrar sus resultados, pero no mostrar el código mismo, escribimos:

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Por último, si queremos NO mostrar el código (`echo = FALSE`), SI evaluarlo (`eval = FALSE`), PERO NO se mostrar los resultados (`results = "hide"`), escribimos:

Que el código haya sido evaluado significa que el objeto "a" existirá en la memoria y podrá ser usado para posterior análisis.

Escribir install.packages("tinytex") en la consola para instalar "tinytex" Carga "tinytex" para compilar PDF

Tidyverse

Es un conjunto de paquetes de R diseñado para hacer que la ciencia de datos sea más fácil, consistente y eficiente.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

##readr: Cargar archivos .cvs

```
library("readr")
```

##readr: Cargar archivos .dta (Stata)

```
library("haven")
```

##Crear un tibble

```
library("tibble")
```

mi_tibble <- tibble(columna1 = c(1, 2, 3), columna2 = c("A", "B", "C"))

print(mi_tibble)

```
df <- data.frame(
  x = 1:3,
  y = c("A", "B", "C")
)

print(df)
```

```
##   x y
## 1 1 A
## 2 2 B
## 3 3 C
```

```r
library(tibble)

tib <- tibble(
  x = 1:3,
  y = c("A", "B", "C")
)

print(tib)
```

```
## # A tibble: 3 x 2
##       x y
##   <int> <chr>
## 1     1 A
## 2     2 B
## 3     3 C
```

Comandos básicos

Operadores básicos

- `2 + 2`: suma
- `2/2`: división
- `2*2`: multiplicación
- `2^2`: exponente
- `sqrt(2)`: raíz cuadrada
- `log(2)`: logaritmo natural
- `exp(2)`: exponencial
- `2 == 2`: evaluación lógica
- `42 >= 2`: evaluación lógica
- `2 <= 42`: evaluación lógica
- `2 != 42`: evaluación lógica
- `23 %/% 2`: división por entero → 11

Símbolos básicos

- `x <- 7`: operador de asignación
- `x = 7`: igual, operador de asignación
- `x == 7`: evaluación lógica

Vectores

Son la unidad básica de almacenamiento de datos en R. Se construye con el comando c(), por concatenate.

- `x <- c(1, 2, 3, 4, 5)`: crea un vector `x` con los elementos 1, 2, 3, 4, 5
- `y <- c(6:10)`: crea un vector `y` con los elementos del 6 al 10
- `z <- c(x, y); z`: combina los vectores `x` e `y` en un nuevo vector `z`, y muestra el contenido de `z`

Operaciones con vectores

- `x <- c(1, 5, 2)`: crea un vector `x` con los elementos 1, 5, 2
- `y <- c(1, 2, 3)`: crea un vector `y` con los elementos 1, 2, 3
- `x + y`: suma los elementos de los vectores `x` e `y` elemento por elemento
- `y / 2`: divide cada elemento del vector `y` por 2
- `x * y`: multiplica los elementos de los vectores `x` e `y` elemento por elemento

Construcción de vectores

- `rep(3, 5)`: repite el valor 3 cinco veces, creando un vector
- `seq(4, 12, by = 2)`: genera una secuencia de números desde 4 hasta 12 con un incremento de 2

Trabajo en clases

Con RMarkdown o Quarter realiza los siguientes ejercicios:

1. Resuelve por x: $3^x = 27$
2. Simplifica: $\frac{e^{10x}}{e^{8x}}$
3. Calcula la derivada de: $f(x) = 3x^2 - 9x - 2$