CAPÍTULO 1

El entorno R

Contenidos

1.1 ¿Qué es R?

`R` es un lenguaje de programación y entorno orientado al análisis de datos. Es una herramienta muy flexible, ya que se puede utilizar a nivel de usuario "no experto", utilizando métodos ya implementados sin demasiado esfuerzo, así como a nivel de usuario "avanzado", implementando métodos personalizados.

Algunas ventajas destacables de `R` son:

- Es gratuito. El código es libre y lo puede utilizar cualquiera.
- Existen entornos de desarrollo integrado (IDE) para `R`, que facilitan el desarrollo de nuestros programas, como `RStudio`.
- Se pueden realizar análisis estadísticos muy sofisticados.

- Los propios usuarios de R pueden crear paquetes y compartirlos con la comunidad.
- Existe una gran colección de herramientas. Todo está documentado. Sin embargo, no todos los paquetes disfrutan de buena documentación, ya que esto depende del autor del paquete.

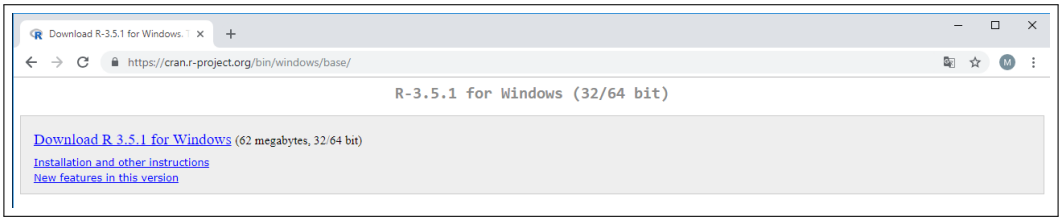
Este manual cubre los conceptos más elementales de R, por lo que se recomienda al lector que desee ampliar su conocimiento sobre este lenguaje de programación la consulta de [1] y [2].

1.2 Instalación de R y RStudio

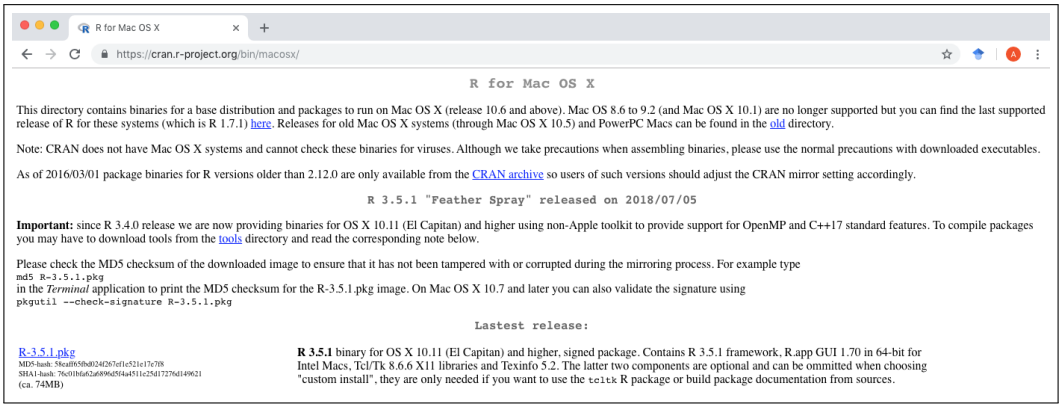
Para instalar R, podemos ir a la siguiente dirección:

- https://cran.r-project.org/bin/windows/base/
- https://cran.r-project.org/bin/macosx/

para Windows y Mac, respectivamente (si tuviéramos que establecer un *mirror*, nos conectaríamos al más cercano a nosotros). Nos descargamos el archivo ejecutable correspondiente (`exe` o `pkg`) e instalamos el programa.



(a) Windows



(b) Mac

Figura 1.1 Webs de descarga de R.

En el caso de `RStudio`, podemos obtenerlo accediendo a la dirección https://www.rstudio.com/products/rstudio/download/, donde descargaremos el instalador apropiado de acuerdo a nuestro sistema operativo. Por comodidad, en este manual se utilizará `RStudio`.

1.3 Interfaz básica de RStudio

Cuando abrimos `RStudio` por primera vez, obtenemos una interfaz similar a la mostrada en la Figura 1.2.

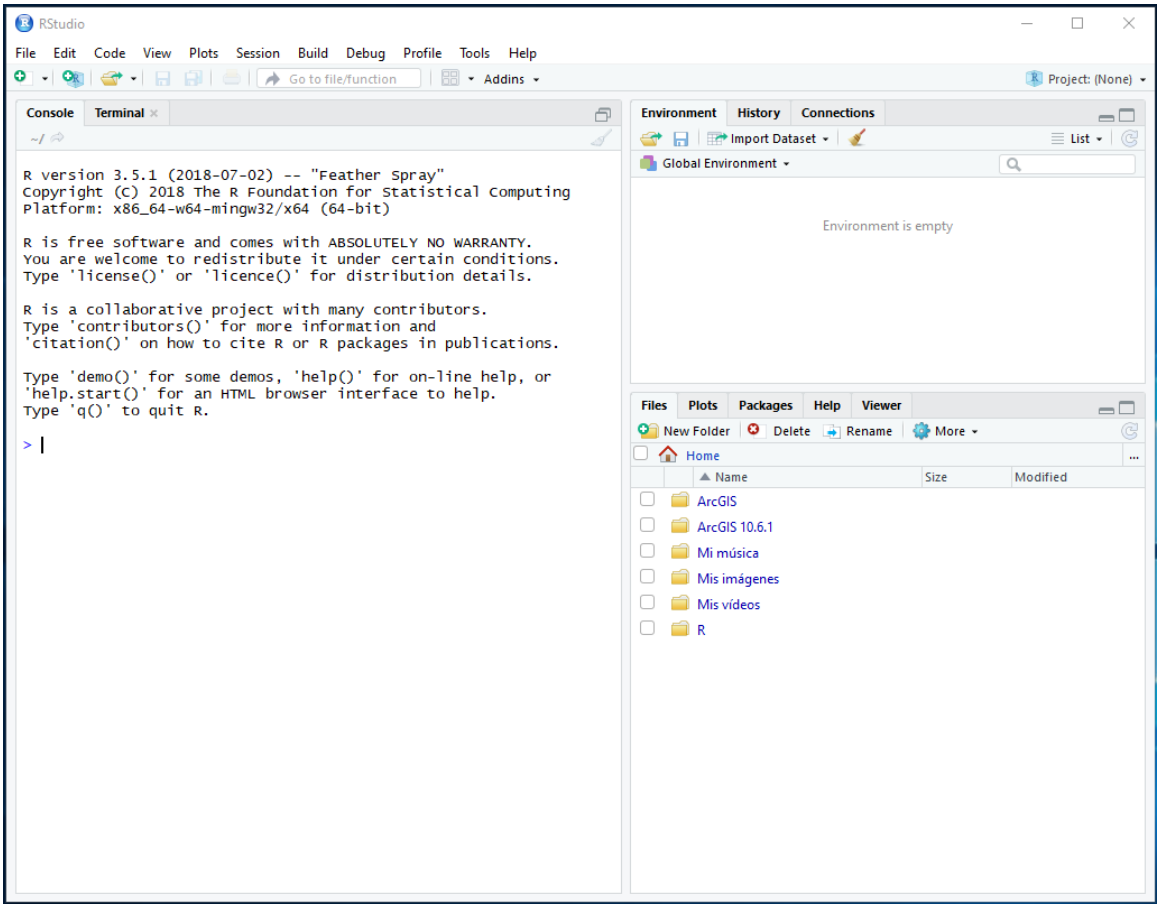


Figura 1.2 Aspecto de `RStudio` al iniciarlo por primera vez.

Para empezar a trabajar, debemos abrir un `R Script` nuevo. Para ello, pulsamos sobre el icono  , situado en la esquina superior izquierda. Aparecerá un nuevo panel y el aspecto de `RStudio` será como el mostrado en la Figura 1.3.

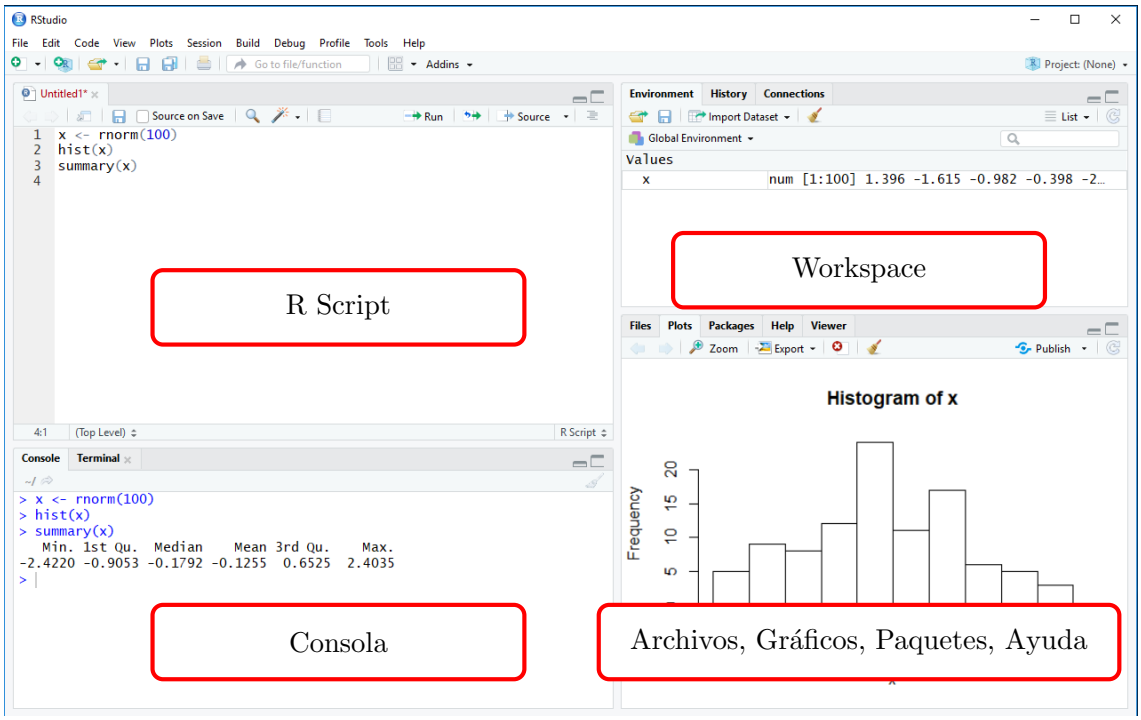


Figura 1.3 Aspecto general de `RStudio`.

- R Script. Es el lugar donde se escribe el código. Se puede guardar como archivo con extensión `.R`. El código se puede ejecutar:
  - línea a línea, mediante ctrl+↵ (o ⌘+↵ en el caso de Mac), o pulsando el icono Run
  - el script completo, secuencialmente desde arriba hasta abajo, seleccionando todo el código+ctrl+↵ (o ⌘+↵), o pulsando el icono Source
- Consola. Es el lugar donde aparecen los resultados al ejecutar un proceso desde R Script o la propia consola.
- Workspace. Es el lugar donde se ven todos los *objetos* definidos por el usuario.
- Archivos, gráficos, paquetes, ayuda. Este panel presenta por defecto estas cuatro pestañas:
  - Archivos. Es un explorador donde podemos buscar los archivos alojados en nuestro ordenador.
  - Gráficos. Los gráficos que creemos aparecerán aquí.
  - Paquetes. Podemos instalar y cargar paquetes desde esta pestaña. También se puede hacer por línea de comandos. Por ejemplo:

```r
# Instalar paquete
install.packages("forecast")


# Cargar paquete
library("forecast")
```

¡Importante! Los paquetes se instalan solo 1 vez. En cambio, cada vez que iniciemos sesión, tendremos que cargar el paquete (utilizando `library()`) para poder utilizarlo.

– Ayuda. Podemos consultar la ayuda de paquetes y funciones desde esta pestaña. También se puede hacer por línea de comandos. Por ejemplo:

```r
# Consultar ayuda con 'help'
help("forecast")


# Consultar ayuda con ?
?forecast
```

CAPÍTULO 2

Objetos de R

Contenidos

De manera general, un *objeto* es cualquier elemento capaz de ser manejado por R. Existen multitud de clases: un valor numérico, un vector, una matriz, un data frame, una lista, una función, un gráfico, etc. Para crear un objeto se utiliza el operador de asignación `<-` o `=`. Por ejemplo:

```
# Crear el objeto x
x <- 10
```

2.1 Tipos de datos

Podemos distinguir entre los siguientes tipos fundamentales de datos:

- Numeric (ej., 1, 3.5). Se utiliza el punto (.) como separador de decimales.
- Character (ej., "10", 'hola'). Representa texto, siempre escrito entre comillas, simples o dobles.
- Integer (ej., 1L, 4L). Si se quiere un número explícitamente entero, hay que añadir el sufijo `L`.
- Logical (`TRUE`, `FALSE`). Indican si una condición se cumple (`T`) o no (`F`).
- Complex (ej., 1i, 2+3i). Se definen mediante la notación `i`.

Existen valores numéricos especiales, como `Inf` (infinito) o `NaN`(valor indefinido), así como la constante lógica `NA` que indica un valor faltante. Para saber qué tipo de dato estamos manejando, podemos utilizar la función `class()`:

```
x <- 10
class(x)
## [1] "numeric"
x <- "diez"
class(x)
## [1] "character"
x <- "10"
class(x)
## [1] "character"
```

```
x <- 10L
class(x)
## [1] "integer"
x <- T
class(x)
## [1] "logical"
x <- 1i
class(x)
## [1] "complex"
```

El lenguaje R tiene algunas palabras reservadas, como `function`, `for`, `NULL`, etc., que no deben utilizarse como nombres de objetos. Puede consultarse la lista de palabras reservadas en la ayuda de R, ejecutando la orden `?reserved`.

2.1.1 Coerción

Se puede forzar la conversión de una clase a otra explícitamente mediante las funciones `as.character()`, `as.numeric()`, etc. La forma general de estas funciones es `as.class()`. Sin embargo, no todos los tipos de datos pueden ser transformados a los demás. Generalmente, la coerción se puede realizar en el siguiente orden:

logical → integer → numeric → complex → character

Si hacemos la transformación en el sentido inverso, podemos perder información u obtener `NA` como resultado de la coerción fallida. Ejemplos:

```
z <- c(T , F)
class(z)
## [1] "logical"
as.integer(z)
## [1] 1 0
as.numeric(z)
## [1] 1 0
as.complex(z)
## [1] 1+0i 0+0i
as.character(z)
## [1] "TRUE"  "FALSE"
```

```
y <- "diez"
as.complex(y)
## Warning:  NAs introduced by coercion
## [1] NA
w = 2+3i
as.numeric(w)
## Warning:  imaginary parts discarded in
coercion
## [1] 2
```

Si coercionamos un dato de tipo lógico a tipo numérico, `T` devolverá 1 y `F` 0. Si realizamos la operación inversa, cualquier número distinto de 0 dará como resultado `T`, mientras que el 0 devolverá `F`. Si intentamos convertir un número entrecomillado a tipo numérico, la coerción ocurrirá con éxito.

```
z <- c(0, 1, 2)
as.logical(z)
## [1] FALSE  TRUE  TRUE
```

```
z <- c(0, 1, 2)
as.character(z)
## [1] "0" "1" "2"
```

```
y <- "10"
as.numeric(y)
## [1] 10
```

2.2 Estructuras de datos

Podemos diferenciar entre vectores, listas, factores, matrices y data frames.

2.2.1 Vectores

Un *vector* es una colección ordenada de elementos de la misma clase. Para crear vectores se utiliza la función de concatenación `c()`, utilizando la coma (,) como separador de elementos. Además, se pueden crear vectores de secuencias numéricas, con incremento o decremento de 1, mediante el operador `:`. Ejemplo:

```r
# numeric
x <- c(0.5, 0.6)
# logical
x <- c(TRUE , FALSE)
# logical
x <- c(T , F)
```

```r
# character
x <- c("a", "b", "c")
# integer
x <- 9:29
# complex
x <- c(1+0i, 2+4i)
```

El vector será del mismo tipo que los datos que contiene. Al ser un objeto unidimensional, podemos averiguar el número de elementos que contiene mediante la función `length()`. Además, podemos asignarle atributos, esto es, información adicional que describe características de los datos, mediante la función `attributes()`. Esta función toma como único argumento el objeto, en este caso un vector, y se le asigna una *lista nombrada* mediante el operador de asignación `=` o `<-`. Ejemplo:

```r
# vector de tipo integer
z <- 0:5
# asignar atributos dim y mycomment
attributes(z) = list(dim = length(z), mycomment = "integer vector")
# obtener los atributos del vector
attr(z, "dim")

## [1] 6

attr(z, "mycomment")

## [1] "integer vector"
```

Por otro lado, se pueden asignar nombres a cada elemento de un vector. Ejemplo:

```r
v = c(0.2, 0.3, 0.5)
names(v) = c("low", "moderate", "high")
v

##      low moderate     high
##      0.2      0.3      0.5
```

2.2.2 Factores

Una estructura de datos tipo *factor* es un vector de tipo integer donde cada valor numérico tiene asignada una etiqueta predefinida (`levels`). Los factores se utilizan para representar datos cualitativos y son muy útiles cuando se conocen los posibles valores que puede tomar una variable. Para crear esta clase de objeto, utilizamos la función `factor()`, que toma como primer argumento un vector de datos, normalmente con pocos valores distintos. Ejemplo:

```r
x <- factor(c("si", "no", "no", "no")); x

## [1] si no no no
## Levels: no si

typeof(x)

## [1] "integer"

attributes(x)

## $levels
## [1] "no" "si"
##
## $class
## [1] "factor"
```

El argumento `levels` de la función `factor()` permite especificar los posibles valores que puede tomar la variable, aunque estos no aparezcan en el vector de datos. Ejemplo:

```r
x <- factor(c("no", "no", "no", "no"), levels = c("si", "no")); x

## [1] no no no no
## Levels: si no
```

2.2.3 Matrices

Una estructura de datos tipo `matrix` es un objeto organizado en dos dimensiones (filas y columnas), siendo todos los elementos que lo componen del mismo tipo. Podemos crear matrices utilizando la función `matrix()`, que toma como primer argumento un vector de datos. Opcionalmente, se puede especificar el número de filas y el de columnas con los argumentos `nrow` y `ncol`, respectivamente. Ejemplo:

```
# Crear un vector x con seis elementos
x <- c(1, 2, 3, 4, 5, 6)
# Organizar esos seis elementos en dos columnas
matrix(x, ncol = 2)

##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Por defecto, los datos se ordenan por columnas, de arriba a abajo y de izquierda a derecha. El argumento `byrow` es de tipo lógico y permite ordenar los datos por filas cuando toma el valor `TRUE`. Ejemplo:

```
matrix(x, ncol = 2, byrow = TRUE)

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

El argumento opcional `dimnames` permite asignar nombres a las filas y columnas de la matriz y se construye como una lista compuesta de dos vectores: `dimnames = list(nombres_filas, nombres_columnas)`. Ejemplo:

```
nombres = list(c("F1", "F2", "F3"), c("C1", "C2"))
matrix(x, ncol = 2, byrow = T, dimnames = nombres)

##    C1 C2
## F1  1  2
## F2  3  4
## F3  5  6
```

También podemos crear matrices mediante las funciones `cbind()` y `rbind()`, que unen