UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Posgrado en Ingeniería

Departamento en Ingeniería de Sistemas

Curso de introducción para programar en lenguaje R

Elaborado por:

Adam Romero
adam_romero@unam.edu

Introducción

Presentación del programa

R es un lenguaje de programación orientado a objetos de código abierto, al cual actualmente le brinda soporte The R Foundation. Tiene su origen en 1992 pero su versión final estable salió en el 2000 y, a día de hoy, su versión más reciente es la 4.5.2

R está basado en el lenguaje de programación S, desarrollado y utilizado por Bell Laboratories (anteriormente AT&T y ahora propiedad de Nokia); los encargados de esta reimplementación fueron Robert Gentleman y Ross Ihaka del Departamento de Estadística de la Universidad de Auckland (Nueva Zelanda).

Es importante destacar que R tiene un enfoque de análisis estadístico, lo cual lo vuelve una herramienta propicia para la investigación científica y el manejo de grandes bases de datos, por ello mismo ha sido utilizado como principal herramienta en proyectos importantes para Google (análisis de efectividad de anuncios, estudios estadísticos internos), Facebook (análisis de comportamiento de usuarios, modelos estadísticos para mejorar el *News Feed*), el Banco Mundial (indicadores globales, dashboards a través de Shiny) y Airbnb (optimización de precios *smart pricing*, modelos de oferta y demanda).

R y RStudio (descarga e instalación)

Para tener una mejor gestión de la programación y del espacio de trabajo se hace uso de un entorno de desarrollo integrado, en este caso se sugiere el uso de RStudio, por lo cual se sugiere descargar e instalar en conjunto tanto R como RStudio en la siguiente liga: https://posit.co/download/rstudio-desktop/

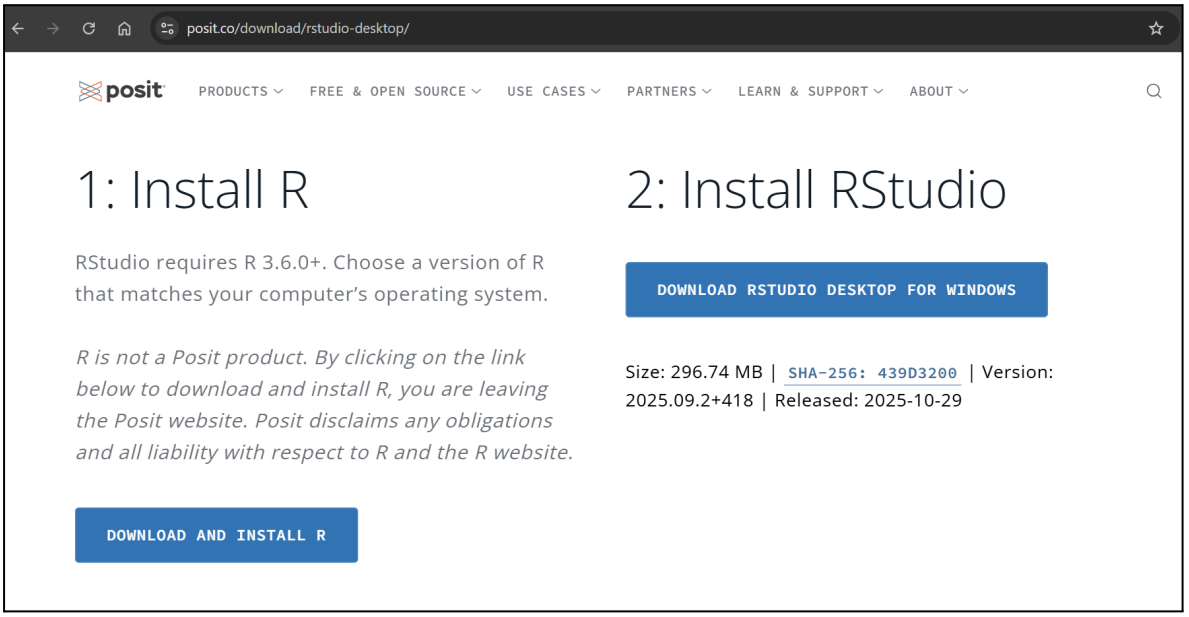


Figura 1. Ventana de instalación para R y RStudio.

Apariencia

La apariencia de RStudio se compone de cuatro ventanas: el script, la consola, el entorno y las gráficas.

[1] En la ventana del script es donde se escribe el código para su posterior ejecución, de esta manera se puede editar, guardar, e incluso ejecutar por secciones.

[2] La ventana de consola nos muestra el código ejecutado y también ahí se puede ejecutar directamente, con el inconveniente de tener una edición menos cómoda y no poder guardar el código.

[3] En la ventana de entorno se observan los objetos que se van declarando; desde esta ventana se puede también importar información de bases de datos y explorar a detalle los objetos declarados.

[4] Finalmente, en la ventana de gráficas es donde se proyectarán todas las gráficas programadas, además de contar también con una pestaña muy útil, la de ayuda, en donde nos ofrecerá información sobre las funciones alojadas en las distintas paqueterías.

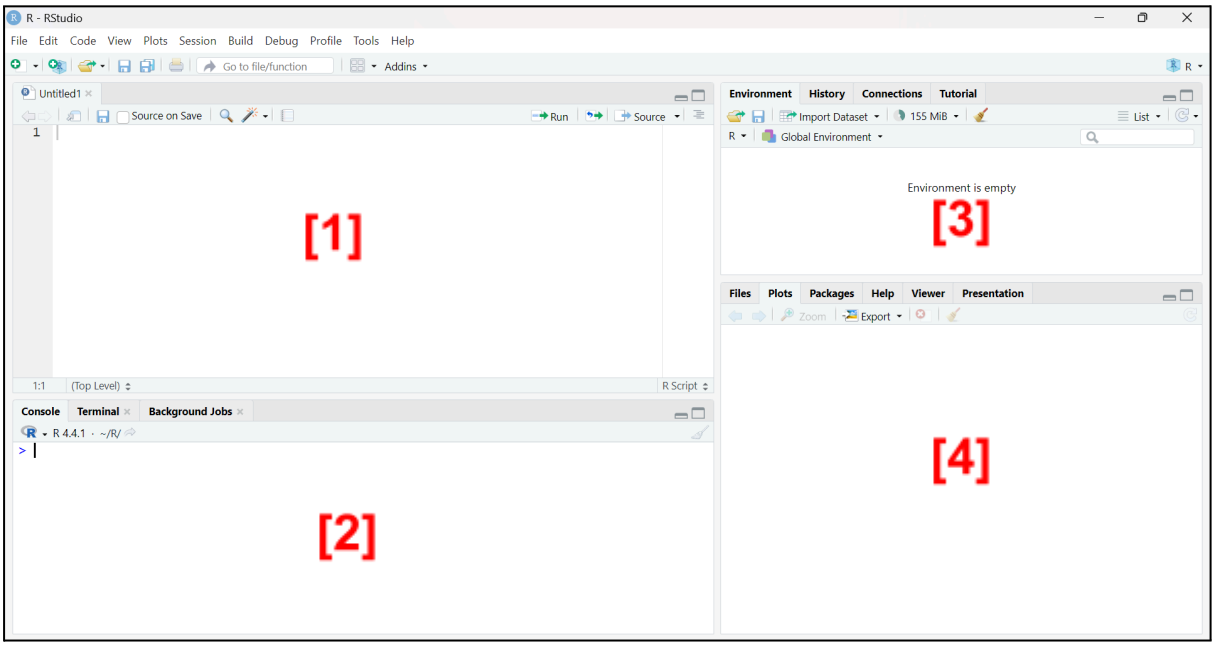


Figura 2. Entorno de RStudio.

Tipos de objetos

En R existen distintos tipos de objetos, los cuales tienen nombre y contenido, y se utilizan para almacenar información. Estos objetos cuentan con dos atributos principales: tipo y longitud. Los distintos tipos de información son: números decimales, números enteros, cadenas de caracteres, valores booleanos y números complejos. La longitud es el número de elementos que aloja el objeto.

Variables

Las variables dentro de R son el objeto más sencillo y guarda un valor para utilizarlo en algún procedimiento posterior.

```
a <- 3
a
## [1] 3
```

Podemos comprobar el tipo y la longitud de la variable con las funciones *class()* y *length()*.

```
class(a)
## [1] "numeric"
length(a)
## [1] 1
```

Ahora creamos una variable llamada *nombre* para alojar una cadena de caracteres y exploramos sus atributos.

```
nombre <- "Brenda"
nombre
## [1] "Brenda"
class(nombre)
##[1] "character"
length(nombre)
##[1] 1
```

Otro tipo de dato que vale la pena señalar son las fechas, ya que R permite alojarlas pero necesitan un tratamiento previo. Primero intentaremos declarar la fecha directo a una variable con el formato “YYYY-MM-DD”.

```
fecha <- “1998-12-05”
fecha
## [1] “1998-12-05”
class(fecha)
## [1] “character”
```

Ahora indicamos que se trata de una fecha al momento de declarar la variable usando la función *as.Date()* e intentamos hacer operaciones con esa variable.

```
fecha <- as.Date(“1998-12-05”)
fecha
## [1] “1998-12-05”
class(fecha)
## [1] “Date”
fecha - 1365
## [1] “1995-03-11”
```

Después de ejecutar todo lo anterior, las ventanas de RStudio deberían de verse de la siguiente manera.

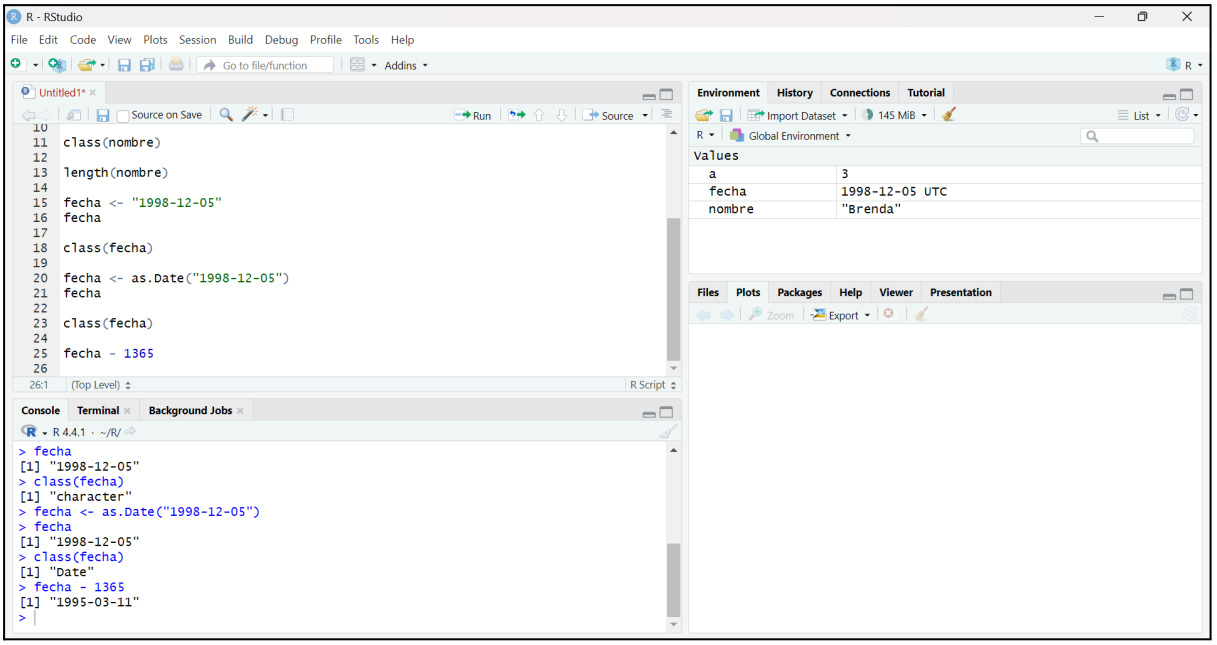


Figura 3. Ventanas de RStudio después de ejecutar el código.

Es importante observar cómo se alojan las variables que se han ido declarando en la ventana de entorno.

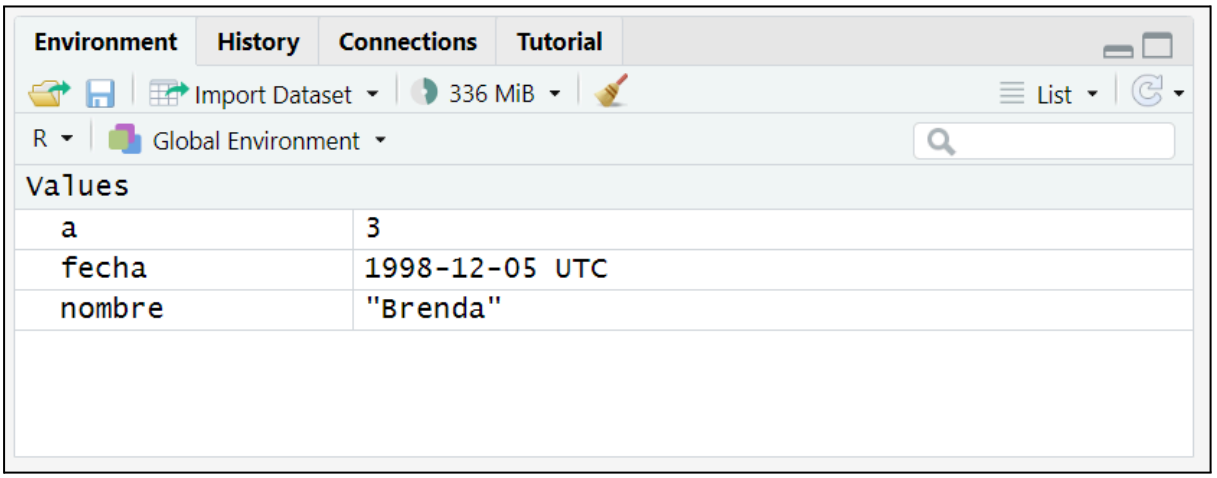


Figura 4. Variables declaradas en el entorno de RStudio.

Vectores

Los vectores son objetos que almacenan a un grupo de datos ordenados del mismo tipo (numérico, caracter, lógico, etc.); en R la función para crear un vector es *c()* (concatenar). Veamos ahora un ejemplo de cómo asignar el valor de un vector a una variable.

```
vector <- c(1, 2, 3)
vector
## [1] 1 2 3
```

A continuación creamos tres vectores para alojar información de un grupo de personas.

```
edad <- c(15, 20, 23, 17, 25)
deporte <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
nombre <- c("Carlos", "Roberto", "Laura", "Paola", "Luis")
```

El vector *edad* es de tipo numérico y aloja las edades de las cinco personas; por su parte, el vector *deporte* contiene variables booleanas y aloja las respuestas a la pregunta de si la persona practica deporte; por último, el vector *nombre* es de tipo caracter y aloja los nombres de las cinco personas.

Comprobamos que la información haya sido alojada de forma correcta, para esto escribiremos el nombre de cada vector y correremos esa indicación con Ctrl + Enter.

```
edad
## [1] 15 20 23 17 25
deporte
## [1] TRUE FALSE FALSE TRUE TRUE
nombre
## [1] "Carlos" "Roberto" "Laura" "Paola" "Luis"
```

Ahora veremos cómo podemos extraer información de un vector, para lo cual se utilizan corchetes [ ], por ejemplo, si queremos extraer la edad de la tercera persona, lo haríamos de la siguiente manera.

```
edad[3]
## [1] 23
```

Si queremos conocer los nombres de la segunda y la quinta persona, haremos uso de un vector dentro del corchete.

```
nombre[c(2, 5)]
## [1] "Roberto" "Luis"
```

Si nos interesa conocer todas las respuestas al deporte, excepto el de la segunda persona, se puede excluir el dato de la siguiente manera.

```
deporte[-2]
## [1] TRUE FALSE TRUE TRUE
```

Matrices

Para construir una matriz en R se utiliza la función *matrix()*. Como ejemplo vamos a crear una matriz de 3x4 con números del 1 al 12; inmediatamente después de declarar la matriz, llamamos al objeto para explorarlo.

```
matriz <- matrix(1:12, nrow = 3, ncol = 4, byrow = FALSE)
matriz
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Al cambiar el valor de *byrow* por *TRUE* los datos se ordenan por fila y no por columna, a continuación cambiamos este parámetro y exploramos la matriz.

```
matriz <- matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)
matriz
##     [,1] [,2] [,3] [,4]
## [1,] 1    2    3    4
## [2,] 5    6    7    8
## [3,] 9   10   11   12
```

Para extraer valores dentro de la matriz también se utilizan los corchetes, pero en este caso la ubicación se define por el número de fila y el número de columna separados por una como [,].

```
matriz[3, 2]
## [1] 10
```

Si se busca extraer un renglón completo, se deja en blanco la parte de la columna.

```
matriz[2, ]
## [1] 5 6 7 8
```

En caso de querer extraer una columna completa, se deja en blanco la parte del renglón.

```
matriz[ ,3]
## [1] 3 7 11
```

Si queremos observar todos los datos de la matriz, excepto las columnas 2 y 3.

```
matriz[, -c(2, 3)]
##     [,1] [,2]
## [1,] 1    4
## [2,] 5    8
## [3,] 9   12
```

Si lo que queremos ahora es ver a la matriz sin los valores de la columna 1 y sin los valores del renglón 3.

```
matriz[-3, -1]
##     [,1] [,2] [,3]
## [1,] 2    3    4
## [2,] 6    7    8
```

Arreglos

Un arreglo es una matriz de varias dimensiones, para construirlos se usa la función *array()*, a continuación vamos a construir un arreglo de 3 dimensiones.

```
arreglo <- array(c(1:9, 11:19, 21:29), c(3, 3, 3))
arreglo
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   11   14   17
## [2,]   12   15   18
## [3,]   13   16   19
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   21   24   27
## [2,]   22   25   28
## [3,]   23   26   29
```

Si ahora queremos extraer el número ubicado en el renglón 1, la columna 2 y la dimensión 3, tendríamos que usar los corchetes de la siguiente manera.

```
arreglo[1, 2, 3]
## [1] 24
```

Si lo que queremos es extraer todos los datos de la segunda dimensión, utilizamos el siguiente código.

```
arreglo[, , 2]
##      [,1] [,2] [,3]
## [1,]   11   14   17
## [2,]   12   15   18
## [3,]   13   16   19
```

Para extraer todos los datos ubicados en las terceras columnas de las tres dimensiones, lo hacemos con el siguiente código.

```
arreglo[, 3, ]
##     [,1] [,2] [,3]
## [1,] 7   17   27
## [2,] 8   18   28
## [3,] 9   19   29
```

Data frames

Los data frames son objetos muy versátiles, ya que estos sí permiten alojar información de distintos tipos en los vectores que lo componen, la única restricción es que todos los vectores deben de ser de la misma longitud. Para crear un data frame se utiliza la función *data.frame()*.

```
data <- data.frame(nombre, edad, deporte)
data
##   nombre edad deporte
## 1 Carlos  15   TRUE
## 2 Roberto 20   FALSE
## 3 Laura   23   FALSE
## 4 Paola   17   TRUE
## 5 Luis    25   TRUE
```

Existen tres opciones para extraer datos de un data frame, con los corchete, con corchetes dobles o con el operador $.

```
data$edad
## [1] 15 20 23 17 25

data[, 1]
## [1] "Carlos" "Roberto" "Laura" "Paola" "Luis"

data[["deporte"]]
## [1] TRUE FALSE FALSE TRUE TRUE
```

Para extraer algún subconjunto específico se utiliza la función *subset()*.

```
subset(data, edad >= 18)
##   nombre edad deporte
## 2 Roberto 20   FALSE
## 3 Laura   23   FALSE
## 5 Luis    25   TRUE

subset(data, deporte == TRUE)
##   nombre edad deporte
## 1 Carlos  15   TRUE
## 4 Paola   17   TRUE
## 5 Luis    25   TRUE

subset(data, deporte == TRUE & edad >= 18)
##   nombre edad deporte
## 5 Luis    25   TRUE
```

Listas

Las listas en R son objetos que sirven para almacenar a otros objetos dentro de sí, que pueden o no ser del mismo tipo. Para crear una lista se utiliza la función *list()*. A continuación vamos a crear una lista con algunos de los objetos que ya tenemos declarados.

```
lista <- list(E1 = matriz, E2 = data, E3 = arreglo)
lista
## $E1
##    [,1] [,2] [,3] [,4]
## [1,] 1   4    7    10
## [2,] 2   5    8    11
## [3,] 3   6    9    12
##
## $E2
##   nombre edad deporte
## 1 Carlos  15   TRUE
## 2 Roberto 20   FALSE
## 3 Laura   23   FALSE
## 4 Paola   17   TRUE
## 5 Luis    25   TRUE
##
```

```
## $E3
## , , 1
##
##      [,1] [,2] [,3]
## [1,] 1    4    7
## [2,] 2    5    8
## [3,] 3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,] 11   14   17
## [2,] 12   15   18
## [3,] 13   16   19
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,] 21   24   27
## [2,] 22   25   28
## [3,] 23   26   29
```

Para extraer información de una lista también se pueden utilizar los corchetes, corchetes dobles y el operador $.

```
lista$E2
##    nombre edad deporte
## 1 Carlos   15    TRUE
## 2 Roberto 20    FALSE
## 3 Laura    23    FALSE
## 4 Paola    17    TRUE
## 5 Luis     25    TRUE

lista[[1]]
##      [,1] [,2] [,3] [,4]
## [1,] 1    4    7    10
## [2,] 2    5    8    11
## [3,] 3    6    9    12
```

Funciones básicas

De manera general una función cuenta con una entrada de datos para su procesamiento y arroja datos de salida. En R estas funciones se caracterizan por tener un nombre corto que nos da idea de lo que hacen, se siguen de un paréntesis que engloba a los datos y los parámetros que utiliza, viéndose algo así: *función(datos, parámetro 1, parámetro 2, parámetro 3)*. Es importante señalar que una función puede dar más de un resultado, y estos pueden ser numéricos y/o gráficos.

Operadores binarios

A continuación se listan las funciones binarias en R:

- + sumar
- - restar
- * multiplicar
- / dividir
- ^ potencia
- %/% cociente de una división
- %% residuo de una división

Pruebas lógicas

Dentro de las pruebas lógicas que se pueden aplicar en R están:

- < menor que
- > mayor que
- == igual a
- <= menor o igual que
- >= mayor o igual que
- != distinto a

Operadores lógicos

Otras herramientas muy útiles son los operadores lógicos, los cuales se listan a continuación:

- Negación: !x
- Conjunción: x & y
- Disyunción: x | y
- Disyunción exclusiva: xor(x, y)

A continuación se presentan algunos ejemplos de su uso.

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

!v < 4
## [1] FALSE FALSE TRUE TRUE TRUE

v > 2 & v < 6
## [1] FALSE TRUE TRUE FALSE FALSE

v > 4 | v > 6
## [1] FALSE FALSE TRUE TRUE TRUE

xor(v > 4, v > 6)
## [1] FALSE FALSE TRUE FALSE FALSE
```

Funciones sobre vectores

En R existen diversas funciones que pueden aplicarse a los vectores, a continuación se muestran las más básicas.

- *min()*: arroja el valor mínimo dentro del vector.
- *max()*: arroja el valor máximo dentro del vector.
- *length()*: nos da la longitud o número de elementos del vector.
- *range()*: nos da los valores mínimo y máximo.
- *sum()*: suma todos los elementos dentro del vector.
- *prod()*: multiplica todos los elementos dentro del vector.
- *which.min()*: arroja la posición del valor mínimo dentro del vector.
- *which.max()*: arroja la posición del valor máximo dentro del vector.
- *rev()*: invierte los valores del vector.

Funciones sobre matrices

Para el caso de las matrices también existen algunas funciones básicas para su procesamiento y manipulación dentro de R.

- *nrow()*: devuelve el número de renglones de una matriz.
- *ncol()*:devuelve el número de columnas de una matriz.
- *dim()*: arroja el número de renglones y de columnas.
- *t()*: arroja la transpuesta de la matriz.
- %*%: se utiliza para obtener el producto matricial (a diferencia de * que multiplica elemento por elemento).
- *solve()*: nos da la matriz inversa.
- *cbind()*: combina matrices (o vectores) por columnas.
- *rbind()*: combina matrices (o vectores) por renglones.

```
M1 <- matrix(1:9, nrow = 3, ncol = 3)
M2 <- matrix(c(2, 5, 1, 7, 9, 2, 3, 6, 1), nrow = 3, ncol = 3))

M1 * M2
##    [,1] [,2] [,3]
## [1,]  2  28   21
## [2,] 10  45   48
## [3,]  3  12    9

M1 %*% M2
##    [,1] [,2] [,3]
## [1,] 29  57   34
## [2,] 37  75   44
## [3,] 45  93   54

t(M1)
##    [,1] [,2] [,3]
## [1,]  1   2    3
## [2,]  4   5    6
## [3,]  7   8    9

cbind(M1, v[1:3])
##    [,1] [,2] [,3] [,4]
## [1,]  1   4    7    1
## [2,]  2   5    8    3
## [3,]  3   6    9    5
```

Funciones estadísticas

Dentro de las funciones estadísticas básicas tenemos:

- *mean()*: sirve para obtener la media o promedio.
- *median()*: nos da la mediana de los datos.
- *var()*: arroja la varianza de los datos.
- *sd()*: nos da la desviación estándar.
- *cov()*: se usa para calcular la covarianza entre datos.
- *cor()*: nos arroja la correlación entre datos.
- *rnorm()*: genera números aleatorios que siguen una distribución normal, dada una media y una desviación estándar.
- *dnorm()*: calcula la densidad de probabilidad de un valor específico, con una media y una desviación estándar definidas.
- *pnorm()*: nos da la probabilidad acumulada de un valor específico, dada una media y una desviación estándar.
- *qnorm()*: nos da el valor correspondiente a una probabilidad, con una media y una desviación estándar definidas.

A continuación vamos a utilizar algunas de las funciones estadísticas con los datos precargados de *mtcars*, en específico con la información de millas por galón.

```
data <- mtcars$mpg

media <- mean(data)
media
## [1] 20.09062

desv <- sd(data)
desv
## [1] 6.026948

pnorm(max(data), media, desv)
## [1] 0.9890261

qnorm(c(0.68, 0.95, 0.99), media, desv)
## [1] 22.90942 30.00407 34.11140
```

Otras funciones

Existen otras funciones muy útiles dentro de R, algunas de las cuales son:

- *seq()*: genera una secuencia de números dando un inicio, un final y un paso.
- *rep()*: repite un valor un número determinado de veces.
- *head()*: muestra los primeros datos de una secuencia.
- *tail()*: muestra los últimos datos de una secuencia.
- *round()*: redondea un valor con un número indicado de decimales.
- *floor()*: redondea hacia abajo.
- *ceiling()*: redondea hacia arriba.
- *trunc()*: devuelve la parte entera de un valor.
- *sort()*: ordena los valores dentro del objeto.
- *rank()*: indica las posiciones que ocuparía cada valor si se ordena.

```
seq(1, 3, 0.5)
## [1] 1.0 1.5 2.0 2.5 3.0

rep(3, 9)
## [1] 3 3 3 3 3 3 3 3 3

head(data, 3)
## [1] 21.0 21.0 22.8

tail(data, 1)
## [1] 21.4

round(sqrt(2), 2)
## [1] 1.41

ceiling(sqrt(2))
## [1] 2

x <- c(5, 2, 8, 4, 6)

sort(x, decreasing = FALSE)
## [1] 2 4 5 6 8

rank(x)
## [1] 3 1 5 2 4
```

Instrucciones de control

Estas funciones son muy utilizadas en programación, ya que se utilizan para ejecutar una o varias operaciones si se cumple cierta condición.

if

Es la más sencilla de todas, verifica que se cumpla cierta condición y, de ser verdadero, se ejecutan una serie de operaciones, de lo contrario no se activa. Su estructura es la siguiente:

```
if (condición) {
  operación 1
  operación 2
  ...
  operación final
}
```

Ahora un ejemplo sencillo de su uso.

```
calificacion <- 8

if (calificación >= 6) {
  print("El alumno aprobó")
}
## [1] "El alumno aprobó"
```

if else

Esta función se utiliza cuando se requieren realizar operaciones si la condición definida por el *if* no se cumple; su estructura es la siguiente.

```
if (condición) {
  operación 1
  operación 2
  ...
  operación final
} else {
  operación 1
  operación 2
  ...
  operación final
}
```

Un ejemplo sencillo, basado en el anterior, sería:

```
calificacion <- 5

if (calificación >= 6) {
  print("El alumno aprobó")
} else {
  print("El alumno reprobó")
}
## [1] "El alumno reprobó"
```

ifelse

Esta función se utiliza cuando sólo existe una instrucción para el caso de *if* y uno sola instrucción para el *else*; su estructura es la siguiente:

```
ifelse(condición, operación si TRUE, operación si FALSE)
```

El ejemplo anterior, utilizando esta función, quedaría así:

```
calificacion <- 8

ifelse(calificacion >=6, print("El alumno aprobó"),
print("El alumno reprobó"))
## [1] "El alumno aprobó"
```

for

La instrucción *for* es útil para repetir un procedimiento cierto número de veces; su estructura es la siguiente:

```
for (i in secuencia) {
  operación 1
  operación 2
  ...
  operación final
}
```

Un ejemplo sencillo para mostrar su funcionamiento sería el siguiente:

```
cuadrado <- rep(NA, 10)

for (i in 1:10) {
  cuadrado[i] <- i^2
}

cuadrado
## [1] 1 2 9 16 25 36 49 64 81 100
```

while

Esta instrucción nos ayuda a repetir operaciones mientras se cumpla cierta condición; su estructura es la siguiente:

```
while (condición) {
  operación 1
  operación 2
  ...
  operación final
}
```

Un ejemplo sencillo de su aplicación se presenta a continuación:

```
monto <- 100
objetivo <- 150

while (monto < objetivo) {
  monto <- monto + 10
  print(monto)
}
## [1] 110
## [1] 120
## [1] 130
## [1] 140
## [1] 150
```

repeat

Esta instrucción repite ciertas operaciones hasta que se cumple una condición y se rompe el ciclo, se diferencia del *while* porque la condición se evalúa al final y no al principio, por lo que un *repeat* se tiene que correr al menos una vez y un *while* puede no correrse; su estructura es la siguiente:

```
repeat {
  operación 1
  operación 2
  ...
  operación final
  if (condición) break
}
```

Un ejemplo similar al anterior para entender sus diferencias, sería el siguiente:

```
monto <- 100
objetivo <- 150

repeat {
  monto <- monto +10
  print(monto)
  if (monto == objetivo) break
}
## [1] 110
## [1] 120
## [1] 130
## [1] 140
## [1] 150
```

Creación de funciones en R

R permite crear funciones personalizadas, lo cual es una herramienta bastante útil ya que nos permite automatizar procedimientos muy largos y facilitar cálculos repetitivos, además que esto se puede escalar colocando funciones dentro de otras funciones e incluso creando nuestras propias librerías. La estructura general de una función es la siguiente:

```
función <- function (parámetro 1, parámetro 2, ...) {
  cuerpo
  cuerpo
  cuerpo
  ...
  return(resultado)
}
```

Es importante señalar que aunque la función termine con un solo resultado en *repeat()*, éste nos sirve para indicar el fin de la función, pero esta nos puede arrojar más de un resultado de ser necesario (gráfica, data frame, vector, variable, etc.).

Como ejemplo, vamos a crear una función con la fórmula general para ecuaciones de segundo grado.

```
chicharronera <- function (a, b, c) {
  x1 <- (-b + sqrt(b^2 - (4 * a * c))) / (2 * a)
  x2 <- (-b - sqrt(b^2 - (4 * a * c))) / (2 * a)
  print(x1)
  return(x2)
}
```

Una vez definida nuestra función, la veremos declarada en nuestro entorno.

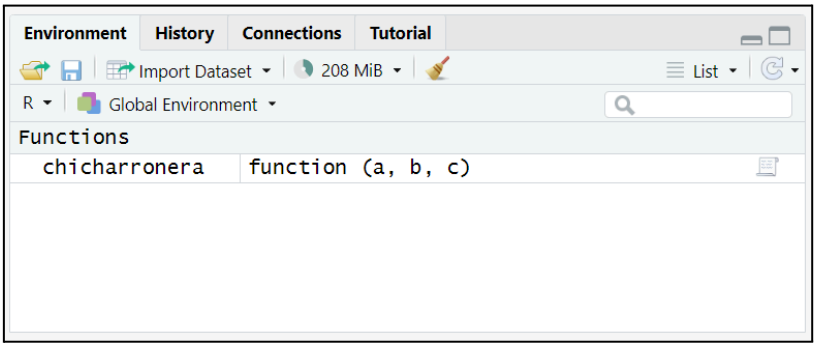


Figura 5. Vemos a nuestra función declarada en el entorno.

Ahora hacemos una prueba y observamos los resultados.

```
chicharronera(-1, 7, -10)
## [1] 2
## [1] 5
```

Para mostrar algo más del alcance que se puede obtener al definir una función en R, realizaremos una función más elaborada con el mismo fin que la anterior.

```
chicharronera <- function(a, b, c) {
  disc <- b^2 - 4 * a * c

  if (disc < 0) {
    stop("La ecuación no tiene soluciones reales")
  }

  x1 <- (-b + sqrt(disc)) / (2 * a)
  x2 <- (-b - sqrt(disc)) / (2 * a)

  f <- function(x) a * x^2 + b * x + c

  x_vals <- seq(min(x1, x2) - 2, max(x1, x2) + 2, length.out = 200)
  y_vals <- f(x_vals)

  plot(x_vals, y_vals, type = "l", main = "Parábola y soluciones",
       xlab = "x", ylab = "y", lwd = 2)
  abline(h = 0, lty = 2, col = "blue")
  points(c(x1, x2), c(0, 0), pch = 19, col = "red")
  text(x1, 0, labels = "x1", pos = 3)
  text(x2, 0, labels = "x2", pos = 3)

  cat("x1 =", round(x1, 2), "\n")
  return(cat("x2 =", round(x2, 2), "\n"))
}
```

Ahora al evaluar nuestra función con los mismos parámetros que utilizamos anteriormente, obtenemos como resultado, además de los valores de x1 y x2, una gráfica ilustrativa de la parábola y sus soluciones.

```
chicharronera(-1, 7, -10)
## x1 = 2
## x2 = 5
```

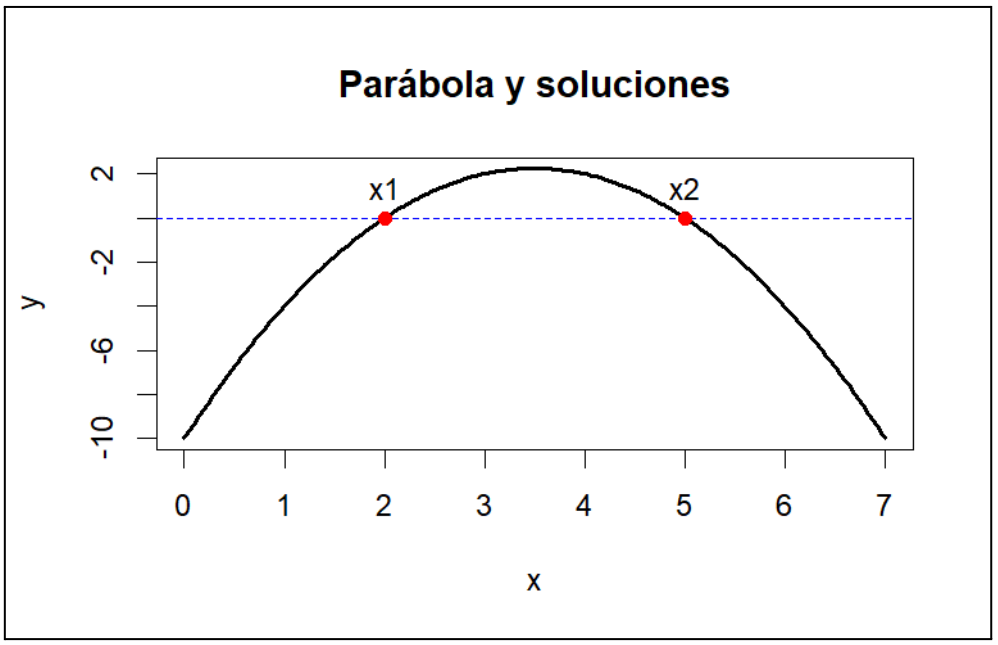


Figura 6. Gráfica arrojada por nuestra función.

Graficar en R

Para la presentación gráfica de datos R cuenta con varias funciones base y con librerías especializadas en la elaboración de gráficos, todas las opciones con múltiples parámetros que nos permiten ajustar los gráficos según nuestras necesidades, además de hacerlos llamativos y fáciles de entender.

Cargar datos

Es muy común tener datos almacenados en .xlsx, .csv, .txt, o incluso buscar tomarlos de alguna base de datos de internet, para eso R nos ofrece múltiples formas de importación de datos.

La manera más sencilla de importar datos desde un archivo es utilizando la opción de *Import Dataset* de la ventana del entorno.

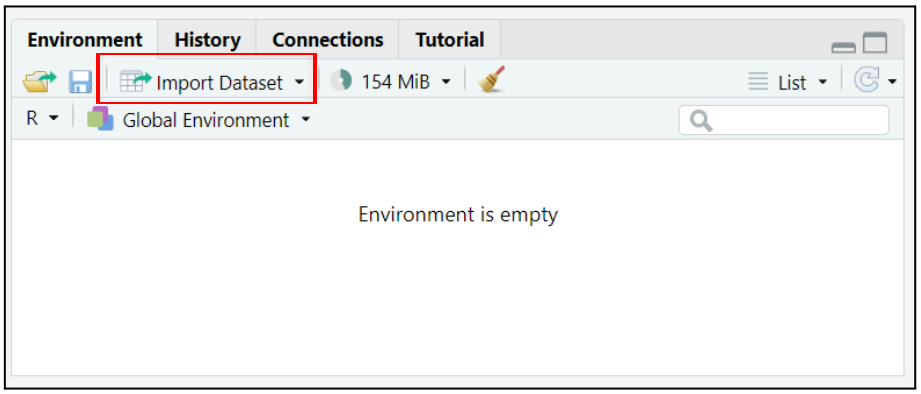


Figura 7. Importación de datos desde la ventana de entorno.

Como ejemplo, vamos a cargar unos datos alojados en un archivo .xlsx, dicho archivo es *Walmex.xlsx*.

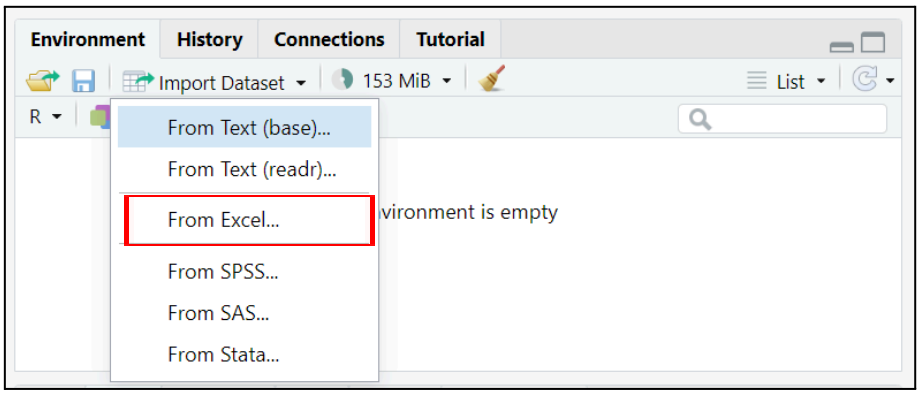


Figura 8. Importando datos desde un archivo .xlsx.

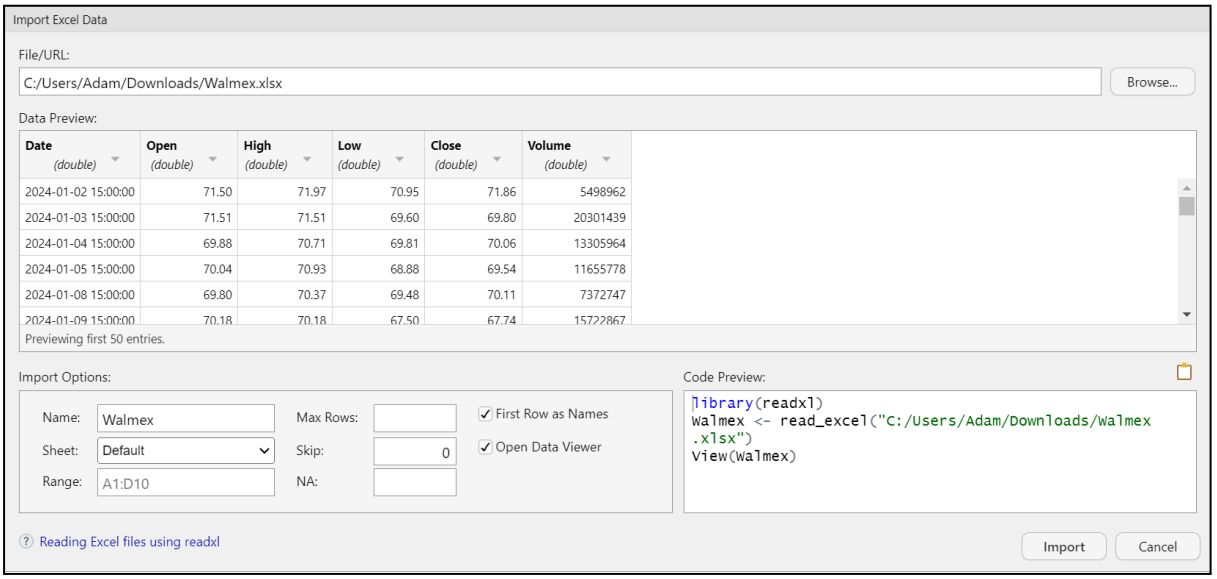


Figura 9. Ventana de exportación de datos.

Después de buscar el archivo en su ubicación, usando el botón *Browse...*, verificamos que la información para la extracción de los datos sea la correcta; también podemos observar que en la ventana inferior derecha, de *Code Preview*, tenemos el código utilizado para importar la información, el cual se puede copiar y agregar directamente en el script. Una vez que esté todo en orden podemos dar click en el botón *Import*.

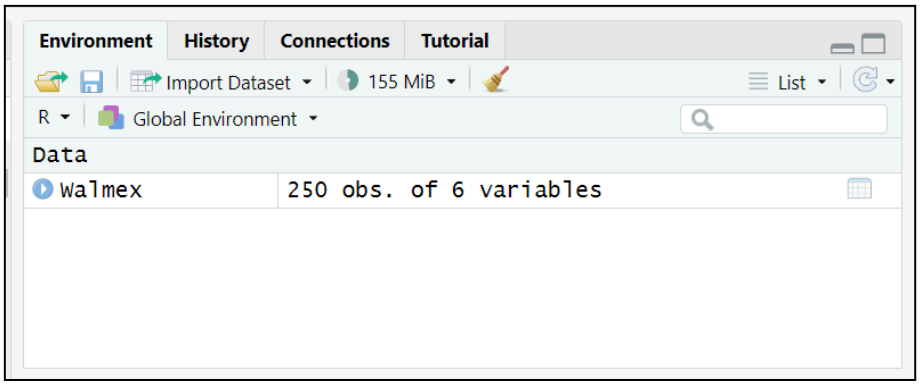


Figura 10. Datos cargados.

Una vez importados los datos, automáticamente se abrirán en una ventana para su vista previa y podremos ver el data frame como un objeto declarado.

plot

Los datos cargados anteriormente son los precios y volúmenes de cotización de la acción de Walmart de México y Centroamérica durante 2024; ahora haremos una gráfica con los precios de cierre utilizando la función base *plot()*.

```
plot(Walmex$Close)
```

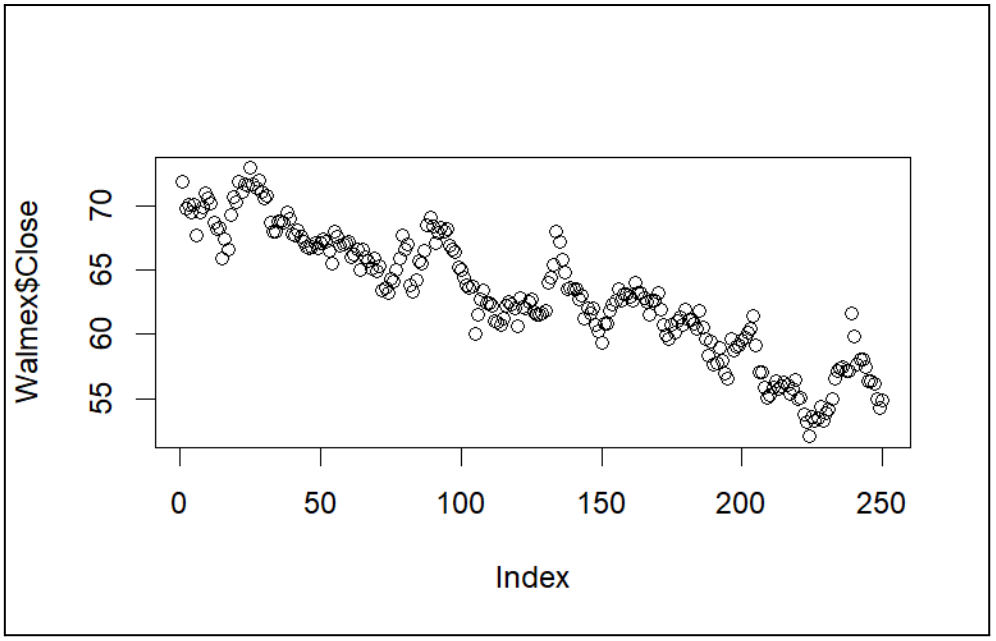


Figura 11. Gráfica obtenida con la función *plot()*.

Se pueden observar varias deficiencias en la gráfica obtenida, por lo cual ahora buscaremos mejorarla agregando un título, un subtítulo, cambiando la gráfica por una de líneas, agregando las fechas en el eje horizontal, ajustando el tamaño de los textos y añadiendo colores referentes a la marca; para todo lo anterior generamos el siguiente código.

```
par(bg = "lightsteelblue2")

plot(Walmex$Date, Walmex$Close, type = "l", lwd = 2,
     col = "blue3", main = "Precios de Walmex", cex.main = 2,
     col.main = "yellow", cex.lab = 0.8, ylab = "Precio",
     xlab = "Fecha", col.lab = "steelblue4", cex.axis = 0.8,
     col.axis = "steelblue", cex.sub = 0.7,
     sub = "Precios del 1 de enero al 31 de diciembre de 2024")

abline(h = seq(50, 75, 5), lty = 2, col = "steelblue2")

abline(v = seq(min(Walmex$Date), max(Walmex$Date),
     length.out = 6), lty = 2, col = "steelblue2")
```

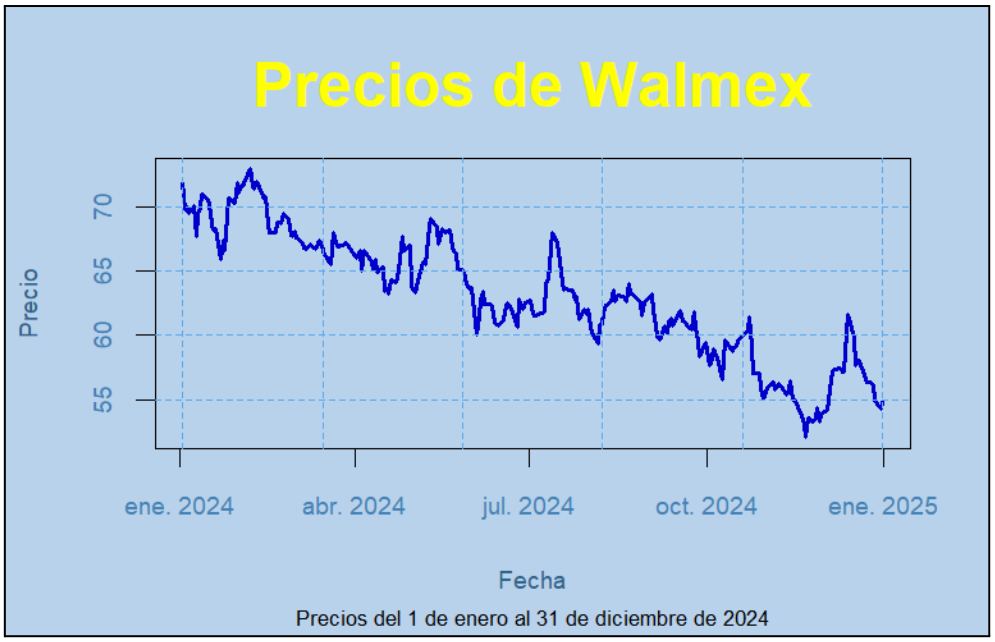


Figura 12. Gráfica mejorada.

Con la gráfica mejorada se puede observar un poco del alcance que puede tener R para la creación de gráficos, y existen aún más parámetros que se pueden ajustar para obtener gráficas robustas, e incluso crear dashboards.

ggplot

Una librería especializada para la creación de gráficas es *ggplot2*, esta ofrece un manejo más cómodo de la información y ofrece más parámetros de personalización, a continuación vemos como ejemplo la gráfica anterior con *ggplot2*.

```
library(ggplot2)

ggplot(Walmex, aes(x = Date, y = Close)) +
  geom_line(color = "blue3", linewidth = 1.2) +
  geom_hline(yintercept = seq(50, 75, 5),linetype = "dashed",
             color = "steelblue2") +
  geom_vline(xintercept = as.numeric(seq(min(Walmex$Date),
                           max(Walmex$Date), length.out = 6)),
    linetype = "dashed", color = "steelblue2") +
  labs(title = "Precios de Walmex",
     subtitle = "Precios del 1 de enero al 31 de diciembre de 2024",
       x = "Fecha", y = "Precio") +
  theme(plot.background = element_rect(fill = "lightsteelblue2",
        color = NA),
        panel.background = element_rect(fill = "white", color = NA),
        plot.title = element_text(size = 20, color = "yellow",
                                  face = "bold", hjust = 0.5),
        plot.subtitle = element_text(size = 10, hjust = 0.5),
        axis.title = element_text(size = 9, color = "steelblue4"),
        axis.text = element_text(size = 8, color = "steelblue"))
```

Figura 13. Gráfica con *ggplot2*.

hist

Para analizar frecuencias y distribuciones es muy útil un histograma, para lo cual usaremos la función *hist()*, pero antes debemos hacer un tratamiento en los datos para obtener los rendimientos y haremos uso de la librería *quantmod*.

```
library(quantmod)

walmex <- xts(as.numeric(Walmex$Close),
              order.by = as.Date(Walmex$Date))
rends <- dailyReturn(walmex)
plot(rends)
```

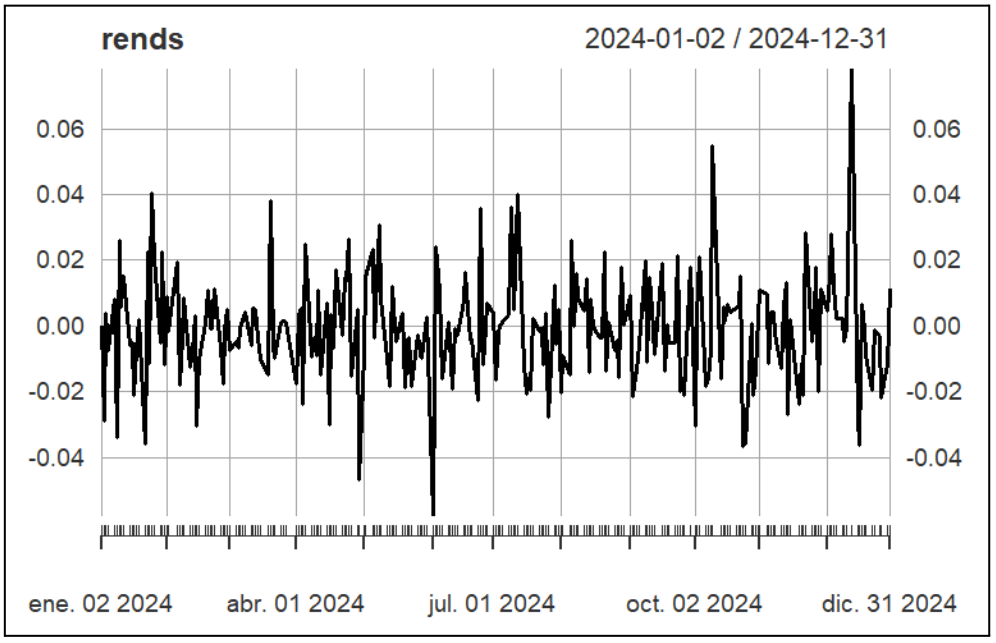


Figura 14. Gráfica de rendimientos.

Con la gráfica de rendimientos podemos verificar que realizamos correctamente el cálculo, ahora metemos el objeto *rends* dentro de la función *hist()* para obtener un histograma con los parámetros predeterminados.

```
hist(rends)
```

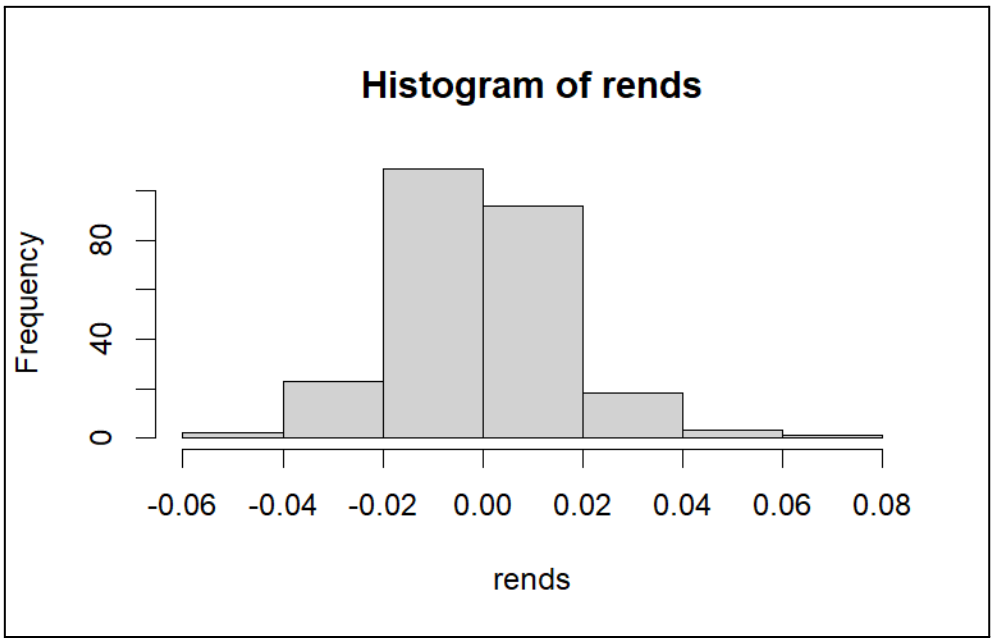


Figura 15. Histograma de los rendimientos.

Al no definir ningún parámetro podemos observar varios elementos a mejorar dentro de nuestro histograma, como puede ser el número de barras, cambiar el título, los nombres de los ejes, además podemos agregar como referencia una curva de densidad de distribución normal y la ubicación de la media, y ahora que tendremos más de un elemento en el gráfico, será necesario también incluir leyendas para identificar cada elemento.

```
hist(rends, breaks = 20, border = "white",
     main = "Histograma de rendimientos", xlab = "Rendimiento",
     ylab = "Frecuencia", cex.lab = 0.8, cex.axis = 0.7)
curve(dnorm(x, mean = mean(rends), sd = sd(rends)), add = T,
      lwd = 2, col = "firebrick", lty = 3)
abline(v = mean(rends), lty = 2, lwd = 2)
legend("topleft", c("Rendimientos", "Distribución normal",
                    paste0("Promedio = ", round(mean(rends), 3))),
       fill = c("grey", "firebrick", "black"), bg = NA,
       border.col = "NA", cex = 0.8)
```

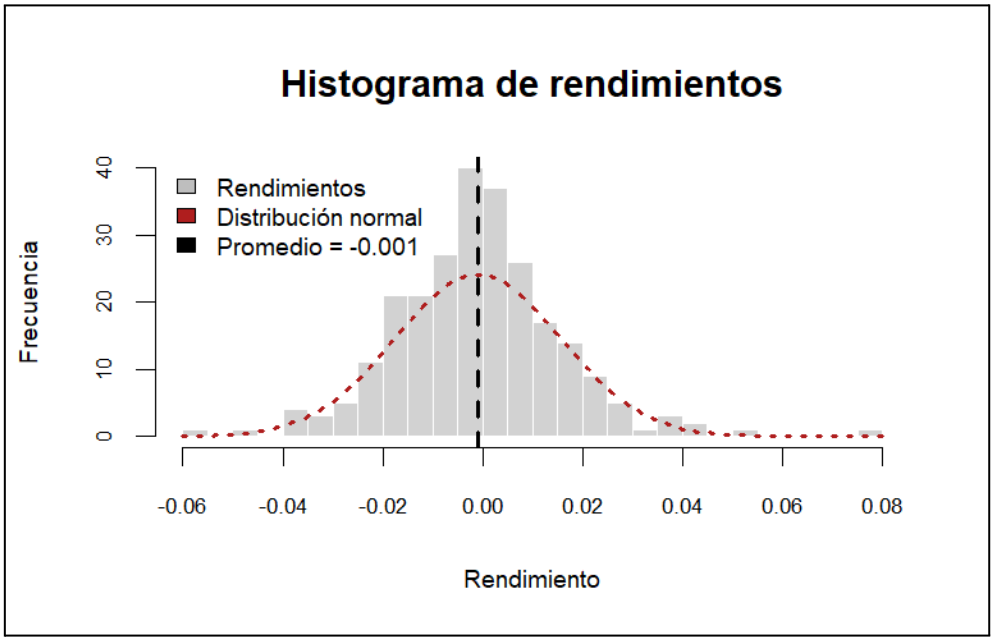


Figura 16. Histograma de rendimientos mejorado.

Con este último histograma podemos observar nuevamente los alcances que tiene R para la presentación gráfica de datos, y aún quedaron parámetros sin personalizar pero que nos pueden ayudar a señalar información importante o a darle un carácter más académico o corporativo a nuestras gráficas.

Datos útiles adicionales

En este apartado se presenta información útil para continuar el camino desarrollándose como programadores en R y para poder encauzar el conocimiento hacia su área de interés, con herramientas que ayudan a explorar y explotar el potencial que tiene R.

CRAN (manejo de librerías)

Una parte medular para el uso de R son las librerías, muchas veces puede que ya exista una con las funciones que necesitamos para desarrollar nuestro código, pero desconocemos dicha librería o bien no sabemos usar las funciones que aloja, para ello es recomendable visitar el CRAN (Comprehesive R Archive Network).

https://cran.r-project.org/

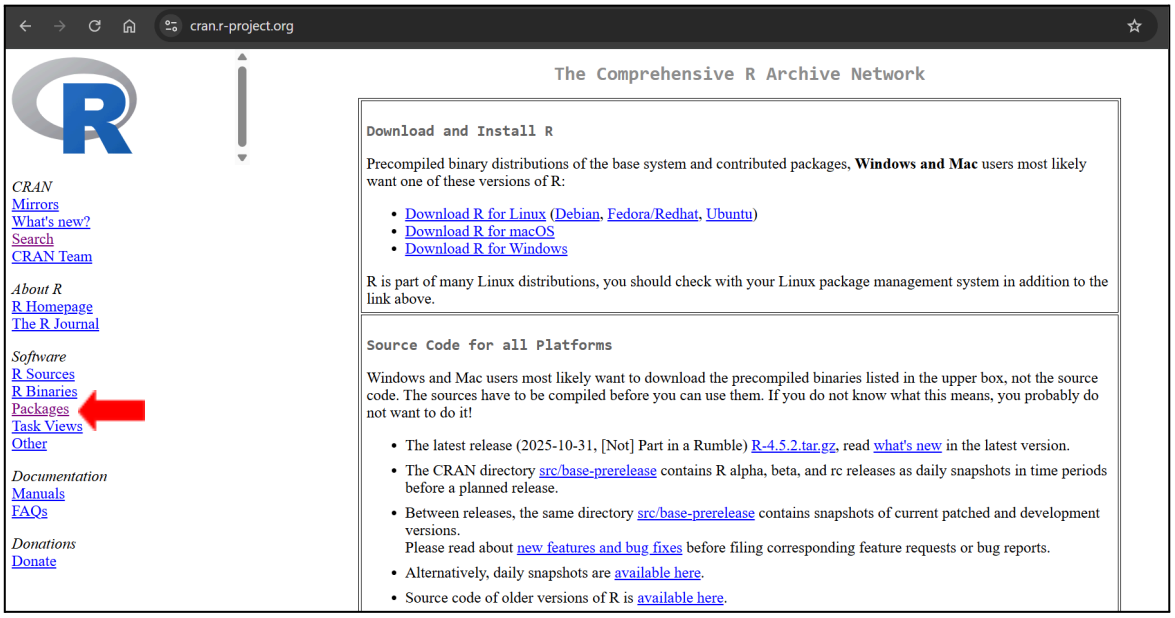


Figura 17. Página principal del CRAN.

Dentro de la liga se puede encontrar diverso contenido, pero para explorar las librerías hay que hacer click en *Packages*.

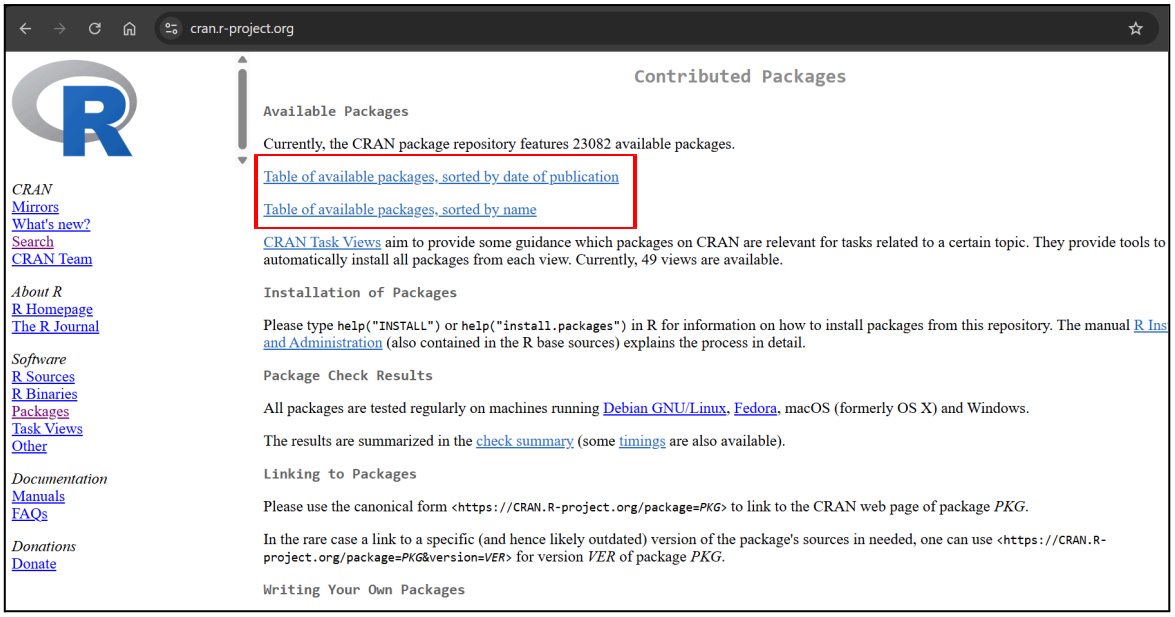
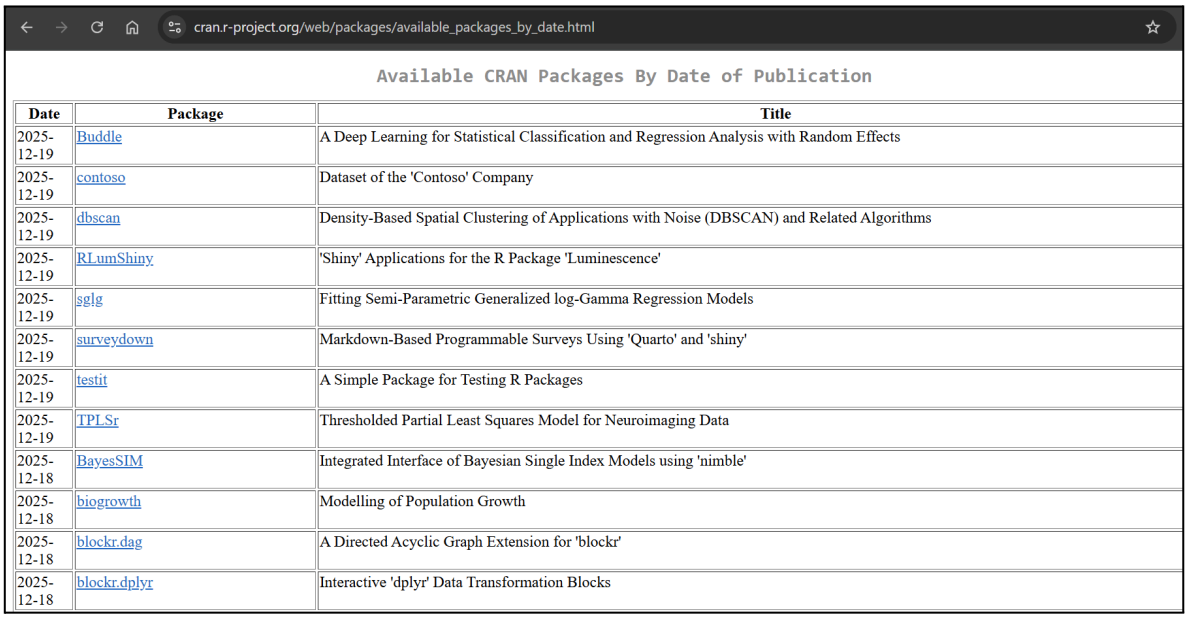


Figura 18. Paqueterías en CRAN.

Una vez dentro de esa sección encontramos dos opciones: las librerías ordenadas por fecha de publicación y ordenadas por nombre; exploraremos la primera opción.



Date	Package	Title
2025-12-19	Buddle	A Deep Learning for Statistical Classification and Regression Analysis with Random Effects
2025-12-19	contoso	Dataset of the 'Contoso' Company
2025-12-19	dbscan	Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Related Algorithms
2025-12-19	RLumShiny	'Shiny' Applications for the R Package 'Luminescence'
2025-12-19	sglg	Fitting Semi-Parametric Generalized log-Gamma Regression Models
2025-12-19	surveydown	Markdown-Based Programmable Surveys Using 'Quarto' and 'shiny'
2025-12-19	testit	A Simple Package for Testing R Packages
2025-12-19	TPLSr	Thresholded Partial Least Squares Model for Neuroimaging Data
2025-12-18	BayesSIM	Integrated Interface of Bayesian Single Index Models using 'nimble'
2025-12-18	biogrowth	Modelling of Population Growth
2025-12-18	blockr.dag	A Directed Acyclic Graph Extension for 'blockr'
2025-12-18	blockr.dplyr	Interactive 'dplyr' Data Transformation Blocks

Figura 19. Exploración de paqueterías en CRAN.

Aquí encontramos todas las librerías existentes y públicas en R, haciendo click en el hipervínculo del nombre de alguna podemos explorar más datos técnicos y su manual de usuario, lo cual nos ayudará a saber utilizar sus funciones.

Foros

Otra opción muy útil para resolver dudas son los foros, ya que R, al ser de código abierto, no cuenta con soporte técnico, pero las comunidades virtuales ofrecen bastante ayuda en temas comunes e incluso especializados. Dentro de los principales foros están los siguientes:

https://rpubs.com/

https://www.rdocumentation.org/

https://r-charts.com/

https://www.reddit.com/r/RStudio/

Uso de IA

Finalmente y como herramienta más reciente está la IA, la cual es una buena plataforma de inicio pero es necesario tener ciertos cuidados para evitar errores:

- Ofrecer el mayor contexto posible, explicando cuáles son los objetos utilizados, cuál es el objetivo, restricciones y qué librerías se utilizarán.
- Avanzar de manera modular y no solicitar códigos largos.
- Verificar la lógica del código y que no existan errores.
- Si existe un error proporcionarle la línea con el error completa y el error que arroja; muchas veces la IA puede entrar en un loop de errores y es recomendable revisar el error en foros o replantear el problema para que la IA busque otra solución.
- No utilizar la IA como un experto en programación, sino como una plataforma para aprender a programar y complementar con otras herramientas como los foros o videos.
- Es mejor hacer preguntas técnicas o proporcionarle líneas de código ya hechas para resolver los problemas; ver a la IA como una especie de Google avanzado que nos ofrece una respuesta más refinada pero nada distinto de lo que se podría encontrar en internet.