Práctica 1. Instalación de R y Rstudio. Primeros pasos.

04/02/2025

Objetivos

Los objetivos que se persiguen en esta práctica son:

1. Aprender a instalar y configurar R y RStudio
2. Conocer el entorno de trabajo en RStudio
3. Entender conceptos básicos de Rstudio (script, consola, proyectos)
4. Aprender a instalar librerías o paquetes
5. Conocer el sistema de ayudas
6. Conocer el funcionamiento básico de R

Instalación

R

Para instalar R accedemos a la web de R y seleccionamos nuestro sistema.

En el caso de disponer de un sistema operativo Windows, podemos pinchar el siguiente enlace **descargar**

Rstudio

Para instalar Rstudio accedemos a la web de Rstudio y seleccionamos nuestro sistema.

En el caso de disponer de un sistema operativo Windows, podemos pinchar el siguiente enlace **descargar**

Entorno Rstudio

Una vez instalados R y Rstudio, al acceder a Rstudio se obtiene el siguiente entorno:

RStudio es un entorno de desarrollo integrado (IDE) para R, que proporciona una interfaz amigable y potentes herramientas para trabajar con el lenguaje de programación R. A continuación se muestra un desglose de su estructura principal:

1. Ventana de Script (Editor de Código):
    - Ubicación: Generalmente en la parte superior izquierda.
    - Función: Aquí es donde puedes escribir, editar y ejecutar scripts R. Ofrece resaltado de sintaxis, autocompletado y herramientas de depuración.
2. Consola de R:
    - Ubicación: Generalmente en la parte inferior izquierda.
    - Función: Aquí es donde se ejecutan los comandos de R en tiempo real. Puedes escribir comandos directamente en la consola o ejecutar scripts desde el editor de código.
3. Ventana de Entorno/Historial:
    - Ubicación: Generalmente en la parte superior derecha.
    - Entorno: Muestra todas las variables y datos que has cargado en la sesión actual. Puedes inspeccionarlas y gestionarlas fácilmente.

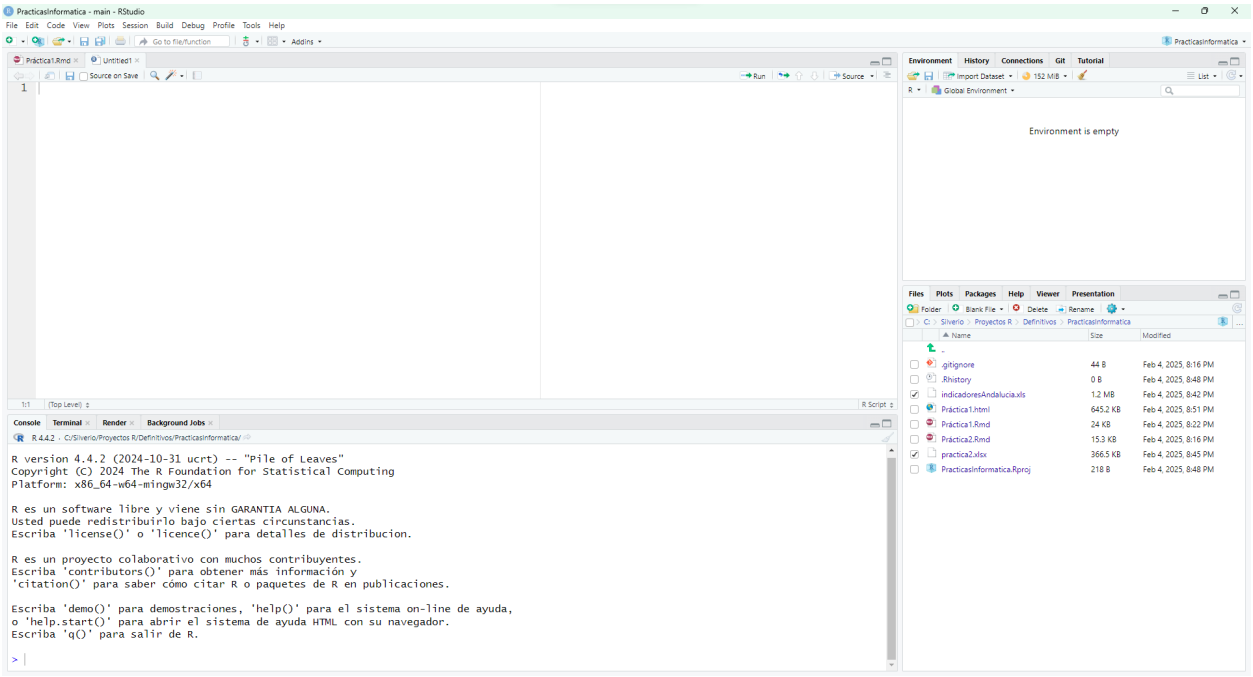

Figure 1: Captura pantalla Rstudio

- Historial: Guarda un registro de todos los comandos que has ejecutado durante la sesión. Puedes buscar y reutilizar comandos anteriores.

4. Ventana de Archivos/Gráficos/Paquetes/Ayuda/Visor:
   - Ubicación: Generalmente en la parte inferior derecha.
   - Archivos: Navega por el sistema de archivos para abrir scripts y otros archivos.
   - Gráficos: Muestra gráficos generados durante la sesión. Puedes ver, guardar y gestionar tus gráficos aquí.
   - Paquetes: Muestra los paquetes de R instalados. Puedes cargar/desinstalar paquetes y actualizar los que están desactualizados.
   - Ayuda: Accede a la documentación de R. Puedes buscar ayuda sobre funciones y paquetes.
   - Visor: Se utiliza para ver documentos HTML, páginas web y otros contenidos multimedia.
5. Barra de Herramientas: *Función: Proporciona accesos directos a acciones comunes, como guardar archivos, ejecutar código, crear gráficos y gestionar proyectos.
6. Pestañas y Paneles Personalizables:
   - Función: Puedes mover y ajustar los paneles y pestañas según tus preferencias. RStudio es altamente personalizable para adaptarse a tu flujo de trabajo.

RStudio te ofrece todas las herramientas necesarias para realizar análisis de datos, modelado y visualización de manera eficiente.

Entender los conceptos básicos de R y Rstudio

La primera tarea que vamos a realizar es comprobar que **R** funciona correctamente. Para ello, vamos a ejecutar nuestra primera instrucción en **R**: accedemos a la consola, escribimos `sessionInfo()` y pulsamos ENTER.

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 8.1 x64 (build 9600)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=Spanish_Spain.1252  LC_CTYPE=Spanish_Spain.1252
## [3] LC_MONETARY=Spanish_Spain.1252 LC_NUMERIC=C
## [5] LC_TIME=Spanish_Spain.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.2.1  magrittr_2.0.3  fastmap_1.1.0   cli_3.3.0
##  [5] tools_4.2.1     htmltools_0.5.5 rstudioapi_0.13 yaml_2.3.5
##  [9] stringi_1.7.6   rmarkdown_2.14  knitr_1.39      stringr_1.4.0
## [13] xfun_0.31       digest_0.6.29   rlang_1.1.0     evaluate_0.15
```

Como se puede observar, se genera un listado con la información de la versión de R y el sistema.

Si lo has ejecutado y has obtenido algo similar a lo anterior significa que ya tienes el entorno de trabajo configurado y listo para empezar a *aprender*.

Consola

La consola de RStudio es una herramienta fundamental para trabajar con el lenguaje de programación **R** de manera más eficiente y organizada. La consola funciona como un intérprete de comandos, puedes escribir tus instrucciones y pulsar **ENTER**, lo que ejecutará la instrucción introducida.

```
a <- 10
b <- 3
c <- a*b
print(c)
```

```
## [1] 30
```

En este fragmento de código, se crean variables a,b,c y se le asignan valores, para finalizar, se imprime en consola el valor de c.

Tal y como observamos en el ejemplo de muestra anterior, en **R**, se ejecutan conjuntos de instrucciones, que normalmente desearemos almacenar, para poder ejecutar de forma conjunta, cuando así se requiera.

Scripts

Un **script** es un archivo de texto, con extensión `.R`, que contiene una serie de comandos y código escrito en lenguaje de programación R. Los scripts se utilizan para automatizar y organizar procesos de análisis de datos, visualización y modelado estadístico.

Proyecto

Para una adecuada organización de la tarea de desarrollo en *R*, repartiremos el código de los proyectos en varios scripts y estos serán almacenados en un **proyecto**.

Paquetes

R dispone de una serie de comandos y funciones disponibles por defecto en lo que se conoce como `**R** base`, sin necesidad de instalar ningún paquete. No obstante, existe una ingente cantidad de paquetes disponibles en R para su uso.

Para instalar un paquete ulitizamos la pestaña **Packages** o en su defecto ejecutamos `install.packages("nombre_paquete")`. Para desinstalar un paquete escribimos `remove.packages("nombre_paquete")`.

Una vez instalado el paquete en el sistema, debemos cargarlo para poder usarlo. Para ello, ejecutamos la función `library()`. Para descargar un paquete se usa la función `detach`.

Por ejemplo, para cargar el paquete **MASS**:

```
library(MASS)
```

Y para borrarlo:

```
detach(package:MASS)
```

Sistema de ayuda

La forma más rápida de utilizar la ayuda, si se conoce el comando en Rstudio, consiste en escribir el comando en la consola y pulsar **F1**.

Otras formas de solicitar ayuda son:

```
?mean
```

```
## starting httpd help server ... done
```

```
help("median")
```

Si no conocemos el comando exacto, pero tenemos una idea, se puede usar `apropos` que genera un listado con todos las funciones relacionadas.

```
apropos("mean")
```

```
##  [1] ".colMeans"     ".rowMeans"     "colMeans"      "kmeans"
##  [5] "mean"          "mean.Date"     "mean.default"  "mean.difftime"
##  [9] "mean.POSIXct"  "mean.POSIXlt"  "rowMeans"      "weighted.mean"
```

Por último, y para concluir con el bloque de ayuda, R proporciona una interfaz para buscar en la web de R-PROJECT. Esto, permitirá buscar funciones que en su ayuda contengan la palabra pasada como parámetro.

```
#Quitar # para utilizar
#RSiteSearch("football")
```

Primeros pasos con R

Estructuras de datos

Lo primero que debemos plantearnos es qué estructuras posee **R** para el almacenamiento de los datos. Las principales estructuras de datos que utilizaremos en este curso son:

1. Variables
2. Vectores
3. Matrices
4. Factores
5. Listas
6. Hojas de datos (dataFrame)

Veamos un ejemplo de cada una de ellas:

```
#Las variables almacenan información que puede ser usada y manipulada. Para asignar un
#valor a una variable, utiliza <- o =:
# Asignación con <-
precio <- 42

# Asignación con =
precio = 42
```

```
#Los vectores son una colección de elementos del mismo tipo. Puedes crear vectores
#usando la función c():

# Crear un vector de números
mi_vector <- c(1, 2, 3, 4, 5)

# Vector de caracteres
mi_vector_caracteres <- c("a", "b", "c")

print(mi_vector)
```

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

```
print(mi_vector_caracteres)
```

```
## [1] "a" "b" "c"
```

```
# Las matrices son arreglos bidimensionales que contienen elementos del mismo tipo. Puedes
#crearlas usando matrix():

# Crear una matriz de 3x3
mi_matriz <- matrix(1:9, nrow = 3, ncol = 3)
print(mi_matriz)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
# Los factores son útiles para categorizar datos y almacenar niveles categóricos como
#variables de factor:

# Crear un factor
mi_factor <- factor(c("bajo", "medio", "alto", "medio"))
print(mi_factor)
```

```
## [1] bajo  medio alto  medio
## Levels: alto bajo medio
```

```
# Listas. Las listas en R pueden contener elementos de diferentes
#tipos (números, cadenas, vectores, etc.):

# Crear una lista
mi_lista <- list(numeros = 1:5, palabras = c("Hola", "Mundo"), logica = TRUE)
print(mi_lista)
```

```
## $numeros
## [1] 1 2 3 4 5
```

```
##
## $palabras
## [1] "Hola"  "Mundo"
##
## $logica
## [1] TRUE
```

```r
print(mi_lista$numeros)
```

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

```r
# Hojas de datos (dataFrame) son tablas bidimensionales que pueden contener diferentes
#tipos de datos:

# Crear un dataframe
mi_dataFrame <- data.frame(
  nombres = c("Ana", "Juan", "María", "Carmen"),
  edades = c(23, 35, 28, 19),
  alturas = c(1.65, 1.80, 1.70, 1.68)
)
print(mi_dataFrame)
```

```
##   nombres edades alturas
## 1     Ana     23    1.65
## 2    Juan     35    1.80
## 3   María     28    1.70
## 4  Carmen     19    1.68
```

Resumen:

Variables guardan valores.

Vectores contienen una colección de elementos del mismo tipo.

Matrices son arreglos bidimensionales de datos del mismo tipo.

Factores representan datos categóricos o cualitativos.

Listas pueden almacenar elementos de diferentes tipos.

DataFrames son tablas con datos de diferentes tipos en columnas.

Operadores en R

1. **Operadores Aritméticos**

   Suma (+): `a + b`

   Resta (-): `a - b`

   Multiplicación (*) : `a * b`

   División (/): `a / b`

   Exponenciación (^): `a ^ b`

   Módulo (%%): `a %% b` (devuelve el resto de la división de a por b)

   División entera (%/%): `a %/% b` (devuelve la parte entera de la división de a por b)

2. **Operadores Relacionales**

   Igual a (==): `a == b`

Diferente de (!=): `a != b`

Mayor que (>): `a > b`

Menor que (<): `a < b`

Mayor o igual que (>=): `a >= b`

Menor o igual que (<=): `a <= b`

3. **Operadores Lógicos**

   Y lógico (&): `a & b` (elemento por elemento)

   O lógico (|): `a | b` (elemento por elemento)

   Y lógico (&&): `a && b` (primer elemento)

   O lógico (||): `a || b` (primer elemento)

   Negación lógica (!): `!a`

4. **Operadores de Asignación**

   Asignación (<-): `a <- b`

   Asignación (->): `b -> a`

   Igual (=): `a = b`

5. **Operadores Especiales**

   Secuencia (:): `a:b` (crea una secuencia desde a hasta b)

   Operador de pertenencia (%in%): `a %in% b` (indica si los elementos de a están en b)

   Operadores de subíndice ([ ]): `a[x]` (accede a los elementos del índice x en a)

Ejemplos

```r
# Operadores Aritméticos
a <- 10
b <- 3
suma <- a + b            # 13
resta <- a - b           # 7
multiplicacion <- a * b  # 30
division <- a / b        # 3.333333
modulo <- a %% b         # 1
division_entera <- a %/% b # 3

# Operadores Relacionales
igual <- (a == b)         # FALSE
diferente <- (a != b)     # TRUE
mayor <- (a > b)          # TRUE
menor <- (a < b)          # FALSE
mayor_o_igual <- (a >= b) # TRUE
menor_o_igual <- (a <= b) # FALSE

# Operadores Lógicos
y_logico <- (a > 5 & b < 5) # TRUE
o_logico <- (a > 5 | b > 5) # TRUE
negacion <- !(a > 5)       # FALSE
```

```r
# Secuencia
secuencia <- 1:5          # 1 2 3 4 5
```

Estructuras de control

Las estructuras de control en **R*, son fundamentales para la programación y permiten tomar decisiones, repetir tareas y controlar el flujo del programa. Aquí tienes algunas de las estructuras de control más comunes:

1. **Condicionales (if, else if, else)**: Se utilizan para ejecutar código basado en condiciones.

```r
x <- 5

if (x > 0) {
  print("x es positivo")
} else if (x < 0) {
  print("x es negativo")
} else {
  print("x es cero")
}
```

```
## [1] "x es positivo"
```

2. **Bucles (for, while)**: Permiten repetir un conjunto de instrucciones varias veces.

```r
#Bucle for
for (i in 1:5) {
  print(i)
}
```

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

```r
#Bucle while
y <- 1

while (y <= 5) {
  print(y)
  y <- y + 1
}
```

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

3. **Declaración `repeat`**: Se utiliza para crear bucles que se ejecutan indefinidamente hasta que se usa break para salir del bucle.

```r
z <- 1

repeat {
  print(z)
  z <- z + 1
  if (z > 5) {
```

```
    break
  }
}
```

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

4. **Declaración `break`**: Se utiliza para salir de un bucle prematuramente.

```
for (i in 1:10) {
  if (i == 6) {
    break
  }
  print(i)
}
```

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

5. **Función `switch`**: Permite seleccionar una opción de múltiples alternativas basándose en el valor de una variable. “‘ option <- “B”

result <- switch(option, “A” = “Elegiste A”, “B” = “Elegiste B”, “C” = “Elegiste C”, “Otra opción”)

print(result) “‘

Funciones base en R

Las funciones base en R son las que vienen integradas por defecto, sin necesidad de instalar paquetes adicionales. Aquí tienes una lista de algunas de las funciones base más útiles y utilizadas en R:

1. **Funciones de Estructura de Datos**

   `length(x)`: Devuelve la longitud de un objeto.

   `dim(x)`: Devuelve las dimensiones de un objeto (como una matriz o dataframe).

   `class(x)`: Muestra la clase de un objeto.

   `str(x)`: Muestra la estructura de un objeto.

   `names(x)`: Devuelve o establece los nombres de un objeto.

   `summary(x)`: Da un resumen estadístico de un objeto.

2. **Funciones de Generación de Datos**

   `seq(from, to, by)`: Crea una secuencia de números.

   `rep(x, times)`: Repite un objeto un número específico de veces.

   `sample(x, size)`: Toma una muestra aleatoria de un objeto.

3. **Funciones Matemáticas**

   `sum(x)`: Suma los elementos de un vector.

   `cumsum(x)`: Devuelve un vector con la suma acumulada

`prod(x)`: Devuelve el producto de los elementos del vector.

`cumprod(x)`: Equivalente a `cumsum`pero con productos.

`rowSum(x) y colSum(x)`: suman, por filas y pos columnas respectivamente los valores de x.

`min(x)`: Devuelve el valor mínimo.

`max(x)`: Devuelve el valor máximo.

`round(x, digits)`: Redondea un número al número de dígitos especificado.

`abs(x)`: Devuelve los valores absolutos

`sqrt(x)`: Devuelve las raices cuadradas

4. **Funciones de Manipulación de Datos**

   `cbind(...)`: Combina vectores columna a columna.

   `rbind(...)`: Combina vectores fila a fila.

   `subset(x, ...)`: Extrae subconjuntos de objetos.

   `merge(x, y, ...)`: Fusiona dos dataframes por columnas comunes.

5. **Funciones Lógicas y Condicionales**

   `all(x)`: Verifica si todas las condiciones son verdaderas.

   `any(x)`: Verifica si alguna condición es verdadera.

   `which(x)`: Devuelve los índices de los elementos verdaderos.

   `ifelse(test, yes, no)`: Vectoriza sentencias condicionales.

6. **Funciones de Aplicación**

   `apply(X, MARGIN, FUN)`: Aplica una función a los márgenes de un array o matriz.

   `lapply(X, FUN)`: Aplica una función a cada elemento de una lista y devuelve una lista.

   `sapply(X, FUN)`: Similar a lapply, pero intenta simplificar el resultado.

7. **Funciones básicas**

   `print(x)`: Imprime el valor de un objeto en la consola.

   `cat(...)`: Concatenar y mostrar.

   `head(x)`: Muestra las primeras filas de un objeto.

   `tail(x)`: Muestra las últimas filas de un objeto.

   `unique(x)`: Devuelve los valores únicos de un objeto.

   `sort(x)`: Ordena un vector.

   `order(x)`: Devuelve los índices de ordenamiento, permitiendo reordenar otros vectores o data frames basados en esos índices.

Ejemplos

```r
# Ejemplo de length
vect <- c(10, 20, 30, 40, 50)
result <- length(vect)
print(result)
```

```
## [1] 5
```

```r
# Ejemplo de dim
matrix <- matrix(1:12, nrow = 3, ncol = 4)
result <- dim(matrix)
print(result)
```

```
## [1] 3 4
```

```r
# Ejemplo de class
data <- data.frame(
  ID = 1:3,
  Name = c("Alice", "Bob", "Charlie")
)
result <- class(data)
print(result)
```

```
## [1] "data.frame"
```

```r
# Ejemplo de str
data <- data.frame(
  ID = 1:3,
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(23, 35, 29)
)
result <- str(data)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ ID  : int  1 2 3
##  $ Name: chr  "Alice" "Bob" "Charlie"
##  $ Age : num  23 35 29
```

```r
# Ejemplo de names
data <- data.frame(
  ID = 1:3,
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(23, 35, 29)
)
colnames <- names(data)
print(colnames)
```

```
## [1] "ID"   "Name" "Age"
```

```r
names(data)<-c("ident","Nombre","Edad")
print(names(data))
```

```
## [1] "ident"  "Nombre" "Edad"
```

```r
# Ejemplo de seq
# Crea una secuencia de números desde 1 hasta 10 con incrementos de 2
sequence <- seq(from = 1, to = 10, by = 2)
print(sequence)
```

```
## [1] 1 3 5 7 9
```

```r
# Ejemplo de rep
# Repite el vector c(4, 5, 6) dos veces
repeated <- rep(c(4, 5, 6), times = 2)
print(repeated)
```

```
## [1] 4 5 6 4 5 6
```

```r
# Ejemplo de sample
# Toma una muestra aleatoria de 3 elementos del vector c(1, 2, 3, 4, 5)
sampled <- sample(c(1, 2, 3, 4, 5), size = 3)
print(sampled)
```

```
## [1] 2 5 4
```

```r
# Ejemplo de sum
vect <- c(1, 2, 3, 4, 5)
result <- sum(vect)
print(result)
```

```
## [1] 15
```

```r
# Ejemplo de cumsum
vect <- c(1, 2, 3, 4, 5)
result <- cumsum(vect)
print(result)
```

```
## [1]  1  3  6 10 15
```

```r
# Ejemplo de prod
vect <- c(2, 3, 4)
result <- prod(vect)
print(result)
```

```
## [1] 24
```

```r
# Ejemplo de cumprod
vect <- c(2, 3, 4)
result <- cumprod(vect)
print(result)
```

```
## [1]  2  6 24
```

```r
# Ejemplo de rowSums y colSums
matrix <- matrix(1:9, nrow = 3, ncol = 3)

# Suma por filas
row_sums <- rowSums(matrix)
print(row_sums)
```

```
## [1] 12 15 18
```

```r
# Suma por columnas
col_sums <- colSums(matrix)
print(col_sums)
```

```
## [1]  6 15 24
```

```r
# Ejemplo de min
vect <- c(4, 2, 8, 6, 3)
result <- min(vect)
print(result)
```

```
## [1] 2
```

```r
# Ejemplo de max
vect <- c(4, 2, 8, 6, 3)
```

```r
result <- max(vect)
print(result)
```

```
## [1] 8
```

```r
number <- 3.14159
result <- round(number, digits = 2)
print(result)
```

```
## [1] 3.14
```

```r
# Ejemplo de abs
vect <- c(-3, -5, 2, -8, 6)
result <- abs(vect)
print(result)
```

```
## [1] 3 5 2 8 6
```

```r
# Ejemplo de sqrt
vect <- c(4, 9, 16)
result <- sqrt(vect)
print(result)
```

```
## [1] 2 3 4
```

```r
# Ejemplo de cbind
vect1 <- c(1, 2, 3)
vect2 <- c(4, 5, 6)
result <- cbind(vect1, vect2)
print(result)
```

```
##      vect1 vect2
## [1,]     1     4
## [2,]     2     5
## [3,]     3     6
```

```r
# Ejemplo de rbind
vect1 <- c(1, 2, 3)
vect2 <- c(4, 5, 6)
result <- rbind(vect1, vect2)
print(result)
```

```
##       [,1] [,2] [,3]
## vect1    1    2    3
## vect2    4    5    6
```

```r
# Ejemplo de subset
data <- data.frame(
  ID = 1:5,
  Score = c(50, 75, 42, 83, 66)
)
subset_data <- subset(data, Score > 60)
print(subset_data)
```

```
##   ID Score
## 2  2    75
## 4  4    83
## 5  5    66
```

```r
# Ejemplo de merge
data1 <- data.frame(
  ID = c(1, 2, 3),
  Name = c("Alice", "Bob", "Charlie")
)
data2 <- data.frame(
  ID = c(1, 2, 4),
  Age = c(23, 35, 29)
)
merged_data <- merge(data1, data2, by="ID")
print(merged_data)
```

```
##   ID  Name Age
## 1  1 Alice  23
## 2  2   Bob  35
```

```r
# Ejemplo de all
vect <- c(TRUE, TRUE, FALSE)
result <- all(vect)
print(result)
```

```
## [1] FALSE
```

```r
# Ejemplo de any
vect <- c(TRUE, FALSE, FALSE)
result <- any(vect)
print(result)
```

```
## [1] TRUE
```

```r
# Ejemplo de which
vect <- c(5, 8, 12, 7, 3)
result <- which(vect > 6)
print(result)
```

```
## [1] 2 3 4
```

```r
# Ejemplo de ifelse
vect <- c(5, 8, 12, 7, 3)
result <- ifelse(vect > 6, "Alta", "Baja")
print(result)
```

```
## [1] "Baja" "Alta" "Alta" "Alta" "Baja"
```

Funciones definidas por el usuario

En **R**, puedes definir tus propias funciones utilizando la sintaxis `function`.

Ejemplos

```r
#Función simple
#Esta función toma un número y devuelve "Hola" seguido del número.

# Definir la función
hola <- function(num) {
  mensaje <- paste("Hola", num)
  return(mensaje)
}
```

```r
# Usar la función
resultado <- hola(5)
# Resultado: "Hola 5"
print(resultado)
```

```
## [1] "Hola 5"
```

```r
#Función con más de un argumento
#Esta función toma dos números y devuelve su suma.


# Definir la función
sumar <- function(a, b) {
  resultado <- a + b
  return(resultado)
}

# Usar la función
resultado <- sumar(10, 15)
# Resultado: 25
print(resultado)
```

```
## [1] 25
```

```r
#Función con una condición
#Esta función verifica si un número es par o impar.
# Definir la función
par_impar <- function(num) {
  if (num %% 2 == 0) {
    return("Par")
  } else {
    return("Impar")
  }
}

# Usar la función
resultado <- par_impar(7)
# Resultado: "Impar"
print(resultado)
```

```
## [1] "Impar"
```

```r
#Función con un bucle
#Esta función calcula la suma de los primeros n números naturales.

# Definir la función
suma_numeros <- function(n) {
  suma <- 0
  for (i in 1:n) {
    suma <- suma + i
  }
  return(suma)
}

# Usar la función
```

```r
resultado <- suma_numeros(5)
print(resultado)
```

```
## [1] 15
```

```r
# Función con argumentos opcionales
#Esta función calcula el área de un rectángulo, con valores por defecto para el largo y el
#ancho.

# Definir la función
area_rectangulo <- function(largo = 5, ancho = 3) {
  area <- largo * ancho
  return(area)
}

# Usar la función
resultado1 <- area_rectangulo()
resultado2 <- area_rectangulo(7, 4)
# Resultado: 15
print(resultado1)
```

```
## [1] 15
```

```r
# Resultado: 28
print(resultado2)
```

```
## [1] 28
```

```r
#Función que devuelve múltiples valores
#Esta función calcula el cociente y el residuo de una división.

# Definir la función
division <- function(a, b) {
  cociente <- a %/% b
  residuo <- a %% b
  return(list(cociente = cociente, residuo = residuo))
}

# Usar la función
resultado <- division(10, 3)
# Resultado: lista con cociente = 3 y residuo = 1
print(resultado)
```

```
## $cociente
## [1] 3
##
## $residuo
## [1] 1
```

EJERCICIOS

Todos los ejercicios que se proponen a continuación deberan estar almacenados en uno o varios scripts dentro de un proyecto llamado **practica**.

1. Almacene en una lista los siguientes datos:

Antiguedad	Categoria	Errores_Cometidos	Salario_Mensual
15	Mid	33	4794
19	Junior	28	4503
14	Senior	4	5925
3	Senior	7	3182
10	Junior	11	2751
18	Senior	12	5280
11	Mid	17	2373
5	Junior	32	3841
20	Senior	26	4081
14	Junior	24	6611

2. Genere un dataFrame a partir de los datos anteriores.
3. ¿Cuántas variables e individuos hay en los datos? Almacénelos en variables e imprima los valores con sus respectivos textos.
4. Suponiendo que son todos los trabajadores de una empresa. ¿Cuánto paga mensualmente la empresa a sus trabajadores?
5. Cree una función llamada media que acepte como parámetro un vector y devuelva la media de los valores del vector. Úsala para conocer el salario medio de los trabajadores de la empresa.
6. Genere una muestra de 5 programadores. ¿Cuál es su salario medio? ¿Y el salario máximo? ¿Y el mínimo?
7. Almacene en un vector los números divisibles por 5 desde 5 a 100. ¿Cuántos elementos tiene el vector?
8. Genere el siguiente vector (-3,-2.9,...0,0.1,0.2,....3) ¿Cúal es su media?
9. Genere el siguiente vector (1, 2, 3, . . . , 19, 20, 19, 18, . . . , 2, 1). Obtener un vector con los valores acumulados hasta ese elemento incluido el mismo.
10. Genere una función llamada `momento_k`que acepte como parámetros un vector (x) y un entero positivo (k). La función debe devolver
$$\frac{1}{n}\sum_{i=1}^{n} x_i^k$$
11. Programe una función llamada `coef.asimetria` que acepte un vector `x` y devuelva el coeficiente de asimetría de Fisher, esto es
$$\text{coef.asimetria} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{nS_n^3}$$
donde
$$S_n^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$
12. Programe una función llamada `coef.curtosis` que acepte un vector `x` y devuelva el coeficiente de curtosis de Fisher, esto es
$$\text{coef.curtosis} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{nS_n^4} - 3$$