# Introdución a R

MNPyR

## Índice


**1. Lenguaje R, primeros pasos** **2**

1.1. ¿Qué es R y RStudio? . . . . . 2

1.2. Ventajas de usar R . . . . . 2

1.3. Instalación de R . . . . . 3

1.4. Instalación de RStudio . . . . . 4

1.5. Interfaz de Rstudio . . . . . 4

**2. Aritmética básica** **5**

**3. Tipos de datos** **5**

**4. Objetos en R** **6**

4.1. Vectores . . . . . 6

4.2. Matrices . . . . . 8

4.3. DataFrames . . . . . 14

**5. Cursos en DataCamp** **23**

# 1. Lenguaje R, primeros pasos

## 1.1. ¿Qué es R y RStudio?

El idioma R fue creado a principios de la década de 1990 por Ross Ihaka y Robert Gentleman, ambos trabajando en la Universidad de Auckland. Se basa en el lenguaje S que se desarrolló en los Laboratorios Bell en la década de 1970.

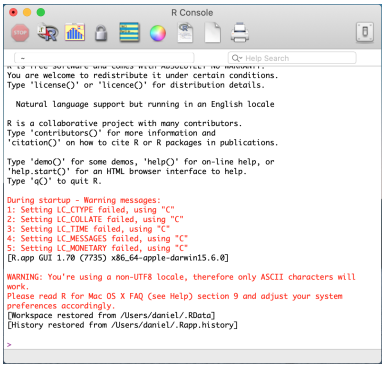
El software R es un proyecto de GNU, lo que refleja su estatus como importante software libre y de código abierto.

(a) Robert Gentleman

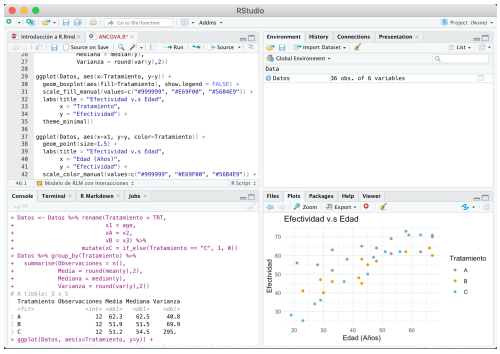
(b) Ross Ihaka

Figura 1: Creadores del lenguaje R

RStudio es un entorno de desarrollo integrado (IDE) gratuito y de código abierto para R y es una de las formas más populares de ejecutar R. A diferencia de la interfaz gráfica de usuario (GUI) de R estándar, RStudio muestra ventanas en mosaico en la pantalla y coloca diferentes ventanas en diferentes pestañas.




(a) R estándar (GUI)




(b) RStudio (IDE)

Figura 2: R estándar v.s RStudio

## 1.2. Ventajas de usar R

- ¡R es gratis! Si eres profesor o estudiante, los beneficios son obvios.
- R contiene funciones estadísticas avanzadas.
- R se ejecuta en una amplia gama de plataformas, incluidas Windows, Unix y Mac OS X. Es probable que se ejecute en cualquier computadora que tenga.
- R tiene capacidades gráficas avanzadas. Si desea visualizar datos complejos, R tiene un conjunto de funciones potentes disponibles.

- R es popular. Hay una gran diversidad de foros para expresar dudas en R (Twitter, Stack Overflow, etc).


Figura 3: Comunidad de R en Twitter, ilustración de Allison Horst

## 1.3. Instalación de R

Los usuarios de Windows y Mac OS X pueden descargar R directamente del CRAN (Comprehensive R Archive Network). Los usuarios de Linux y Unix pueden instalar paquetes R usando su herramienta de administración de paquetes.

### 1.3.1. Windows

1. Abrir https://cran.itam.mx/ en su navegador.
2. Hacer click en "Download R for Windows".
3. Hacer click en "base".
4. Hacer clic en el enlace para descargar la última versión de R (archivo .exe).
5. Cuando se complete la descarga, hacer doble clic en el archivo .exe y responder las preguntas habituales (dar siguiente a todo).

### 1.3.2. Mac OS

1. Abrir https://cran.itam.mx/ en su navegador.
2. Hacer click en "Download R for macOS".
3. Hacer clic en el enlace para descargar la última versión de R (archivo .pkg) dependiendo de la versión del sistema operativo.

   Para ver cuál versión de Mac Os tiene su computadora, realizar:

   3.1 Hacer click en el logotipo de Apple en la parte superior izquierda de la pantalla.

   3.2 Hacer click en "Acerca de esta Mac".

3.3 Verificar la versión de Mac Os de su computadora.

4. Cuando se complete la descarga, hacer doble clic en el archivo .pkg y responder las preguntas habituales (dar siguiente a todo).

## 1.4. Instalación de RStudio

1. Abrir https://posit.co/download/rstudio-desktop/ en su navegador.
2. Hacer click en "Download RStudio Desktop".
3. Cuando se complete la descarga, hacer doble clic en el archivo .exe/.dmg y responder las preguntas habituales (dar siguiente a todo).

## 1.5. Interfaz de Rstudio

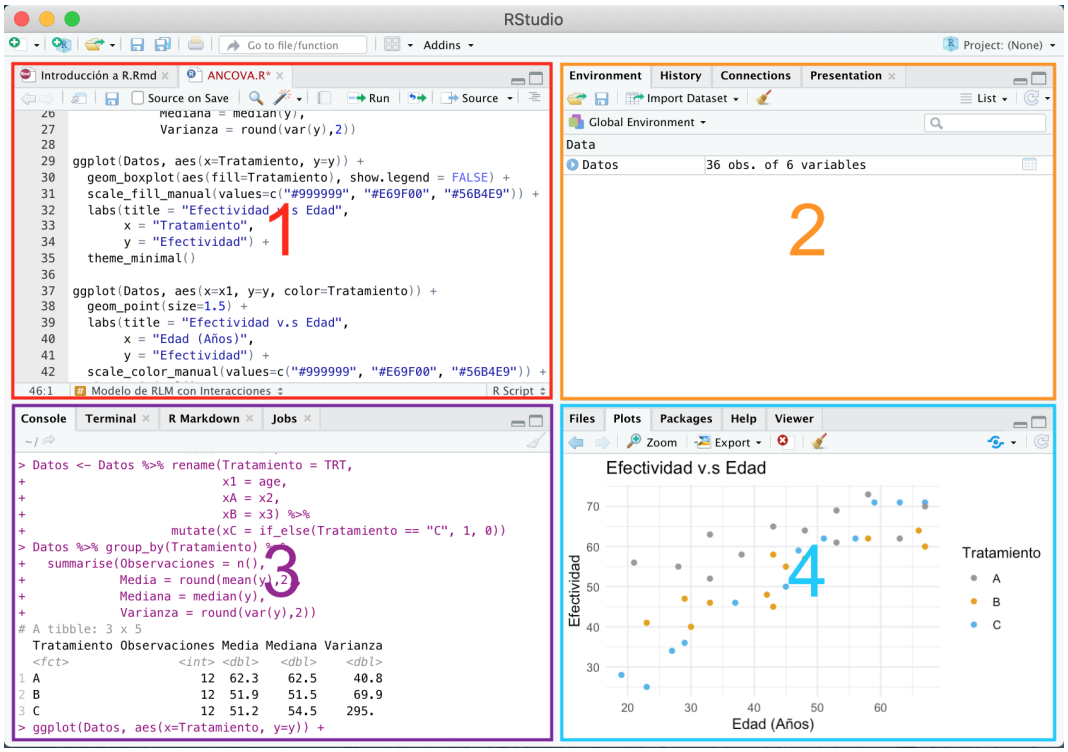



Figura 4: Interfaz de Rstudio

La interfaz principal de RStudio esta conformada por 4 paneles (Figura 4), descritos a continuación:

- *Panel 1*: Scripts y archivos.
- *Panel 2*: Objetos, historial y environment (entorno).
- *Panel 3*: Consola de R.
- *Panel 4*: Árbol de carpetas, gráficas, librerías y ventana de ayuda.

## 2. Aritmética básica

R puede usarse como una calculadora simple. Los operadores aritméticos básicos son:

```
# 1. Aritmética básica -------------------------------------------------------

## Suma
1+1
1+2+3.5

## Resta
6-9
5-4-2.3

## Multiplicación
5*9
(3+1)*5

## Dividisón
24/6
24/6+1
24/(6+1)

## Exponencial
3^3
2^3+1
2^(3+1)

## Asignación de variables
a <- 4 # Forma 1
b = 6  # Forma 2
```

## 3. Tipos de datos

R cuenta con diversos tipos de datos. Los más comunes y básicos para ir empezando son: *Numéricos*, *Enteros*, *lógicos (booleanos)* y *Caracteres (String)*. Se puede utilizar la función `class()` para determinar la clase de un objeto.

```
# 2. Tipo de datos -----------------------------------------------------------

# Asignamos variables de diferentes tipos
n <- 42
c <- "¡Hola mundo!"
l <- FALSE

## Función class()
class(n)
```

```
## [1] "numeric"
```

```r
class(c)
```

```
## [1] "character"
```

```r
class(l)
```

```
## [1] "logical"
```

# 4. Objetos en R

En R existen varios tipos de objectos que permiten almacenar la información para realizar procedimientos estadísticos y gráficos. Los principales objetos en R son Vectores, Matrices, DataFrames y Listas.

## 4.1. Vectores

Un vector es una matriz unidimensional que puede contener datos numéricos, datos tipo caracter o datos lógicos. La función `c()` se usa para formar un vector. A continuación se muestran ejemplos de cada tipo de vector:

```r
# 3. Vectores ------------------------------------------------------------

## Vector tipo número
a <- c(1, 2, 3, -19, 0)

## Vector tipo caracter
b <- c("pera", "manzana", "piña")

## Vector lógico
c <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

Algunas propiedades de los vectores en R son:

- *Homogeneidad*: Todos los elementos de un vector deben ser del mismo tipo.
- *Indexación por posición*: Todos los elementos de un vector tienen una ”posición” asociada y es posible acceder a ellos utilizando su ”posición”.
- *Indexación por múltiples posiciones*: Es posible acceder a múltiples elementos de un vector por medio de sus ”posiciones”.
- *Los elementos de un vector pueden tener nombres*: Es posible acceder a los elementos de un vector por medio de sus ”nombres”. La función `names()` permite obtener o establecer los nombres de un vector.

```r
# 3.1 Propiedades de los vectores ----------------------------------------

## Homogeneidad
a <- c("lunes", 1, "martes", TRUE, FALSE)
a
```

```
## [1] "lunes"  "1"      "martes" "TRUE"   "FALSE"
```

```r
## Indexación por posición
b <- c("Prometeo", "Biblioteca", "El pulpo")
b[2]
```

```
## [1] "Biblioteca"
```

```r
b[1]
```

```
## [1] "Prometeo"
```

```r
## Indexación por múltiples posiciones
b[c(1,2)]
```

```
## [1] "Prometeo"   "Biblioteca"
```

```r
## Indexación por nombres

### Vector con los gastos de la semana
gastos_semana <- c(100, 200, 50, 100, 400, 300, 300)

### Vector con los días de la semana
dias_semana <- c("Lunes", "martes", "miercoles", "jueves",
                 "viernes", "sabado","domingo")

### Asignamos el vector "dias_semana" a los nombres del vector "gastos_semana"
names(gastos_semana) <- dias_semana

gastos_semana
```

```
##     Lunes    martes miercoles    jueves   viernes    sabado   domingo
##       100       200        50       100       400       300       300
```

```r
gastos_semana["sabado"]
```

```
## sabado
##    300
```

Las operaciones básicas entre vectores son:

- *Multiplicación por escalar*:
  Si `e` es un valor numérico y `v = c(a, b, c)` es un vector, entonces `e*v = c(e*a, e*b, e*c)`.
- *Suma entre vectores*:
  Si `v = c(a, b, c)` y `w = c(d, e, f)` son vectores, entonces `v+w = c(a+d, b+e, c+f)`.
- *Multiplicación entre vectores*:
  Si `v = c(a, b, c)` y `w = c(d, e, f)` son vectores, entonces `v*w = c(a*d, b*e, c*f)`.
- *Producto punto*:
  Si `v = c(a, b, c)` y `w = c(d, e, f)` son vectores, entonces `v%*%w = a*d+b*e+c*f`.

```r
# 3.2 Operaciones entre vectores ----------------------------------------

## Multiplicación por escalar
c <- 5
v <- c(1, 2, 3)

c*v
```

```
## [1]  5 10 15
```

```r
## Suma de vectores
a <- c(5, 10, 15)
b <- c(2, 4, 6)

a+b
```

```
## [1]  7 14 21
```

```r
## Multiplicación de vectores
a*b
```

```
## [1] 10 40 90
```

```r
## Producto punto
a%*%b
```

```
##      [,1]
## [1,]  140
```

## 4.2. Matrices

Una matriz es un arreglo bi-dimensional que puede contener datos numéricos, datos tipo caracter o datos tipo lógico.

La función `matrix()` se usa para formar una matriz y sus argumentos son:

- `data`: Vector con los datos de cada entrada.
- `nrow`: Número deseado de filas.
- `ncol`: Número deseado de columnas.
- `byrow`: Valor lógico. Si el valor es `FALSE` (valor predeterminado) la matriz se llena por columnas, de lo contrario, la matriz se llena por filas.

A continuación se muestran ejemplos de cada tipo de matriz:

```r
# 4. Matrices ----------------------------------------------------------------

## Matriz tipo númerica
a <- matrix(c(1, 2, 3,
              4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE)
a
```

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

```
## Matriz tipo caracter
b <- matrix(c("pera", "manzana", "piña",
              "sandia", "melón", "uva"), nrow = 2, ncol = 3, byrow = TRUE)
b
```

```
##      [,1]     [,2]      [,3]
## [1,] "pera"   "manzana" "piña"
## [2,] "sandia" "melón"   "uva"
```

```
## Matriz tipo lógica
c <- matrix(c(TRUE, FALSE, FALSE,
              FALSE, FALSE, TRUE), nrow = 2, ncol = 3, byrow = TRUE)
c
```

```
##       [,1]  [,2]  [,3]
## [1,]  TRUE FALSE FALSE
## [2,] FALSE FALSE  TRUE
```

Algunas propiedades de las matrices en R son:

- *Homogeneidad*: Todos los elementos de una matriz deben ser del mismo tipo.
- *Indexación por coordenada*: Todos los elementos de una matriz tienen una "coordenada" asociada y es posible acceder a ellos utilizando su "coordenada".
- *Indexación por múltiples posiciones*: Es posible acceder a múltiples elementos de una matriz por medio de sus "coordenadas".
- *Los elementos de una matriz pueden tener nombres*: Es posible acceder a los elementos de una matriz por medio de sus "nombres". La función `colnames()` y `rownames()` permite obtener o establecer los nombres de las columnas y renglones respectivamente, de una matriz.

```
# 4.1 Propiedades de las matrices ----------------------------------------

## Homogeneidad
a <- matrix(c("lunes", 2, 3,
              4, "martes", FALSE), nrow = 2, ncol = 3, byrow = TRUE)
a
```

```
##      [,1]    [,2]     [,3]
## [1,] "lunes" "2"      "3"
## [2,] "4"     "martes" "FALSE"
```

```
## Indexación por posición
b <- matrix(c(1, 2, 3,
              4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE)
b[1,2]
```

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

```r
b[1,3]
```

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

```r
b[2,1]
```

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

```r
## Indexación por múltiples posiciones
b[1,]
```

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

```r
b[,2]
```

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

```r
## Indexación por nombres

### Vector con los gastos de la semana 1 y 2
gastos_semana_1 <- c(100, 200, 50, 100, 400, 300, 300)
gastos_semana_2 <- c(50, 100, 200, 100, 500, 200, 200)

### Matriz con los gastos de la semana 1 y 2
gastos <- matrix(c(gastos_semana_1,
                   gastos_semana_2), nrow = 2, byrow = TRUE)
gastos
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]  100  200   50  100  400  300  300
## [2,]   50  100  200  100  500  200  200
```

```r
### Vector con los días de la semana
dias_semana <- c("Lunes", "martes", "miercoles", "jueves", "viernes", "sabado","domingo")

### Vector con las semana
semana <- c("Semana 1", "Semana 2")

### Asignamos el vector "dias_semana" a los nombres de las columnas de la matriz "gastos"
colnames(gastos) <- dias_semana

### Asignamos el vector "semana" a los nombres de las filas de la matriz "gastos"
rownames(gastos) <- semana
gastos
```

```
##          Lunes martes miercoles jueves viernes sabado domingo
## Semana 1   100    200        50    100     400    300     300
## Semana 2    50    100       200    100     500    200     200
```

```
gastos["Semana 1", "sabado"]
```

```
## [1] 300
```

Las operaciones básicas entre matrices son:

- *Multiplicación por escalar*: Si

  `c` es una constante y

  `M = matrix(c(a1, a2, a3, a4, a5, a6, a7, a8, a9), ... )` es una matriz, entonces

  `c*M = matrix(c(c*a1, c*a2, c*a3, c*a4, c*a5, c*a6, c*a7, c*a8, c*a9), ...)`

- *Suma entre matrices*: Si

  `M1 = matrix(c(a1, a2, a3, a4, a5, a6, a7, a8, a9), nrow = n, ncol = m, ... )` es una matriz de $n \times m$ y

  `M2 = matrix(c(b1, b2, b3, b4, b5, b6, b7, b8, b9), nrow = n, ncol = m, ... )` es una matriz de $n \times m$, entonces

  `M1+M2 = matrix(c(a1+b1, a2+b2, a3+b3, a4+b4, a5+b5, a6+b6, a7+b7, a8+b8, a9+b9), ... )` es una matriz de $n \times m$.

- ***Multiplicación entre matrices***: Si

  `M1 = matrix(c(a1, a2, a3 a4, a5, a6), nrow = n, ncol = m, ...)` es una matriz de $n \times m$ y

  `M2 = matrix(c(b1, b2, b3, b4, b5, b6), nrow = m, ncol = l, ...)` es una matriz de $m \times l$, entonces

  `M1%*%M2 = matrix(c(a1*b1+a2*b3+a3*b5, a1*b2+a2*b4+a3*b6, a4*b1+a5*b3+a6*b5, a4*b2+a5*b4+a6*b6), nrow = n, ncol = l, ...)`
  es una matriz de $n \times l$.

- ***Inversa de una matriz***: Si

  `M = matrix(c(a1, a2, a3, a4, a5, a6, a7, a8, a9), nrow = n, ncol = n, ... )` es una matriz de $n \times n$, entonces la matriz inversa:

  `W = matrix(c(b1, b2, b3, b4, b5, b6, b7, b8, b9), nrow = n, ncol = n, ... )` es una matriz de $n \times n$ tal que:

  `M%*%W = matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1), nrow = n, ncol = n, ... )` es la matriz identidad de dimensión $n$.

  la función `solve()` permite obtener la inversa de una matriz.

- ***Traza de una matriz***: Si

  `M = matrix(c(a1, a2, a3, a4, a5, a6, a7, a8, a9), nrow = n, ncol = n, ... )` es una matriz de $n \times n$, entonces su traza es:

  `sum(diag(M)) = a1+a5+a9`

  donde la función `diag()` regresa un vector con los elementos de la diagonal de M y la función `sum()` suma los elementos de dicho vector.

- *Matriz transpuestas*: Si

  `M = matrix(c(a1, a2, a3 a4, a5, a6), nrow = n, ncol = m, ...)` es una matriz de $n \times m$, entonces la matriz:

  `Mt = matrix(c(a1, a4, a2, a5, a3, a6), nrow = m, ncol = n, ...)` es la matriz transpuesta de $m \times n$.

  la función `t()` permite obtener la transpuesta de una matriz.

```
# 4.2 Operaciones entre matrices ------------------------------------------

## Multiplicación por un escalar
c <- 4
M <- matrix(c(1, 2, 3,
              4, 5, 6,
              7, 8, 9), nrow = 3, ncol = 3, byrow = TRUE)
c*M
```

```
##      [,1] [,2] [,3]
## [1,]    4    8   12
## [2,]   16   20   24
## [3,]   28   32   36
```

```
## Suma entre matrices
M1 <- matrix(c(1, 1, 1,
               1, 1, 1,
               1, 1, 1), nrow = 3, ncol = 3, byrow = TRUE)

M2 <-matrix(c(1, 1, 1,
              2, 2, 2,
              3, 3, 3), nrow = 3, ncol = 3, byrow = TRUE)
M1+M2
```

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

```
# Multiplicación entre matrices
M1 <- matrix(c(1, 1, 1, 1,
               2, 2, 2, 2,
               3, 3, 3, 3), nrow = 3, ncol = 4, byrow = TRUE)
```

```r
M2 <-matrix(c(1, 1, 1,
              2, 2, 2,
              3, 3, 3,
              4, 4, 4), nrow = 4, ncol = 3, byrow = TRUE)

M1%*%M2
```

```
##      [,1] [,2] [,3]
## [1,]   10   10   10
## [2,]   20   20   20
## [3,]   30   30   30
```

```r
## Inversa de una matriz
M3 <- matrix(c(2, 1,
               7, 4), nrow = 2, ncol = 2, byrow = TRUE)
solve(M3)
```

```
##      [,1] [,2]
## [1,]    4   -1
## [2,]   -7    2
```

```r
M3Inv <- solve(M3)
M3%*%M3Inv
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```r
## Traza de una matriz
M4 <- matrix(c(1, 5, 4,
               9, 2, 8,
               4, 8, 3), nrow = 3, ncol = 3, byrow = TRUE)
diag(M4)
```

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

```r
sum(diag(M4))
```

```
## [1] 6
```

```r
## Matriz transpuesta
t(M4)
```

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

```
t(M2)
```

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

## 4.3. DataFrames

Un DataFrame es más general que una matriz en el sentido de que diferentes columnas pueden contener diferentes tipos de datos (numéricos, caracteres, etc.). Los DataFrames son la estructura de datos más común con la que se trabajará en el curso.

Un DataFrame puede ser entendido como una tabla de datos. Cada columna puede ser de un tipo diferente, pero cada fila del DataFrame debe tener la misma longitud.

A menudo a las columnas de un DataFrame se les llama **variables** y a las filas **observaciones** [Wickham, 2014] [1].

La función `data.frame()` se usa para formar un DataFrame a partir de vectores, donde cada vector es una variable observada.

Supongamos que tenemos información clínica sobre cuatro pacientes representada en el siguiente Cuadro.

| ID | Edad | Tipo de diabetes | Estatus |
|---|---|---|---|
| 1 | 25 | 1 | Mala |
| 2 | 34 | 2 | Regular |
| 3 | 28 | 1 | Excelente |
| 4 | 52 | 1 | Mala |

Cuadro 1: Ejemplo de "Datos ordenados": Cada renglon representa una observación (un individuo) y cada columna representa una variable.

El siguiente código muestra cómo se crea un DataFrame con la información del Cuadro 1.

```
# 5. DataFrames -------------------------------------------------------------

# Ejemplo de la construcción de un DataFrame
## Vector con la variable "ID del paciente"
ID <- seq(1:4)

## Vector con la variable "Edad
edad <- c(25, 34, 28, 52)

## Vector con la variable "Tipo de diabetes"
diabetes <- c(1, 2, 1, 1)

## Vector con la variable "Estatus"
estatus <- c("Mala", "Regular", "Excelente", "Mala")

## Creamos el DataFrame "datos_pacientes" con la función data.frame()
```

[1] A este tipo de estructura de los datos se le conoce como *"Datos ordenados"* o *"Tidy data"*.

```
datos_pacientes <- data.frame(ID, edad, diabetes, estatus)
datos_pacientes
```

```
##   ID edad diabetes   estatus
## 1  1   25        1      Mala
## 2  2   34        2   Regular
## 3  3   28        1 Excelente
## 4  4   52        1      Mala
```

Al igual que los vectores y las matrices, es posible seleccionar elementos de un DataFrame con la ayuda de los paréntesis cuadrados `[,]`. Al usar una coma, puede indicar qué filas y qué columnas se seleccionan respectivamente.

En ocasiones es necesario seleccionar todos los elementos de una fila. Por ejemplo,

```
datos_pacientes[1,]
```

selecciona todos los elementos de la primera fila del DataFrame `datos_pacientes`.

Para seleccionar todos los elementos de una columna existen dos formas:

- *Forma 1*: Utilizando los paréntesis cuadrados `[,]`. Por ejemplo,

  ```
  datos_pacientes[,2]
  ```

  selecciona todos los elementos de la segunda columna (variable `edad`) del DataFrame `datos_pacientes`.

- *Forma 2*: Utilizando el signo `$`. Por ejemplo,

  ```
  datos_pacientes$edad
  ```

  selecciona todos los elementos de la variable `edad` (segunda columna) del DataFrame `datos_pacientes`.

```
# 5.1 Manipulación ------------------------------------------------------

# Manipulación del DataFrame
## DataFrame "datos_pacientes"
datos_pacientes
```

```
##   ID edad diabetes   estatus
## 1  1   25        1      Mala
## 2  2   34        2   Regular
## 3  3   28        1 Excelente
## 4  4   52        1      Mala
```

```
## Seleccionar todos los elementos de la primera fila
datos_pacientes[1,]
```

```
##   ID edad diabetes estatus
## 1  1   25        1    Mala
```

```r
## Selecciona todos los elementos de la segunda columna
datos_pacientes[,2]
```

```
## [1] 25 34 28 52
```

```r
## Selecciona todos los elementos de la variable edad
datos_pacientes$edad
```

```
## [1] 25 34 28 52
```

Existen dos funciones importantes que se utilizan con frecuencia al trabajar con DataFrames:

- `cbind()`: Agrega columnas (variables) a un DataFrame.
- `rbind()`: Agrega filas (observaciones) a un DataFrame.

Supongamos que nos dan información de un nuevo paciente: ID = 5, edad = 40, tipo de diabetes = 2 y estatus = Excelente.

Nos interesa agregar esta nueva observación (fila) al DataFrame original (Cuadro 1).

| ID | Edad | Tipo de diabetes | Estatus |
|---|---|---|---|
| 1 | 25 | 1 | Mala |
| 2 | 34 | 2 | Regular |
| 3 | 28 | 1 | Excelente |
| 4 | 52 | 1 | Mala |
| 5 | 40 | 2 | Excelente |

Cuadro 2: Resultado de agregar una nueva observación con la función `rbind()` al DataFrame del Cuadro 1.

```r
# Ejemplo de la función rbind()
## DataFrame "datos_pacientes" original
datos_pacientes
```

```
##   ID edad diabetes   estatus
## 1  1   25        1      Mala
## 2  2   34        2   Regular
## 3  3   28        1 Excelente
## 4  4   52        1      Mala
```

```r
## Creamos el DataFrame "nueva_observacion"
nueva_observacion <- data.frame(ID = 5,
                                edad = 40,
                                diabetes = 2,
                                estatus = "Excelente")

## Agregamos la "nueva_observacion" al DataFrame original "datos_pacientes"
datos_pacientes <- rbind(datos_pacientes, nueva_observacion)
datos_pacientes
```

```
##   ID edad diabetes   estatus
## 1  1   25        1      Mala
## 2  2   34        2   Regular
## 3  3   28        1 Excelente
## 4  4   52        1      Mala
## 5  5   40        2 Excelente
```

Supongamos ahora que nos dan información de dos nuevas variables: `peso` (en kg) y `estatura` (en m).

Los datos del peso son: 75, 70, 81, 70 y 75.

Los datos de la estatura son: 1.75, 1.65, 1.85, 1.69 y 1.73.

Nos interesa agregar estas nuevas variables (columnas) al DataFrame del Cuadro 2.

| ID | Edad | Tipo de diabetes | Estatus | peso | estatura |
|---|---|---|---|---|---|
| 1 | 25 | 1 | Mala | 75 | 1.75 |
| 2 | 34 | 2 | Regular | 70 | 1.65 |
| 3 | 28 | 1 | Excelente | 81 | 1.85 |
| 4 | 52 | 1 | Mala | 70 | 1.69 |
| 5 | 40 | 2 | Excelente | 75 | 1.73 |

Cuadro 3: Resultado de agregar las variables `peso` y `estatura` con la función `cbind()` al DataFrame del Cuadro 2.

```
# Ejemplo de la función cbind()
## DataFrame "datos_pacientes" original
datos_pacientes
```

```
##   ID edad diabetes   estatus
## 1  1   25        1      Mala
## 2  2   34        2   Regular
## 3  3   28        1 Excelente
## 4  4   52        1      Mala
## 5  5   40        2 Excelente
```

```
## Creamos el vector "estatura"
estatura <- c(1.75, 1.65, 1.85, 1.69, 1.73)

## Creamos el vector "peso"
peso <- c(75, 70, 81, 70, 75)

## Agregamos las variables "estatura" y "peso" al DataFrame original "datos_pacientes"
datos_pacientes <- cbind(datos_pacientes, estatura, peso)
datos_pacientes
```

```
##   ID edad diabetes   estatus estatura peso
## 1  1   25        1      Mala     1.75   75
## 2  2   34        2   Regular     1.65   70
## 3  3   28        1 Excelente     1.85   81
## 4  4   52        1      Mala     1.69   70
## 5  5   40        2 Excelente     1.73   75
```

En ocasiones será necesario definir nuevas variables a partir de las variables existentes de un DataFrame, para realizar este tipo de manipulación en el DataFrame, trabajar con el signo `$` será muy útil.

Supongamos que es de interés calcular el indice de masa corporal (`IMC`) de los pacientes del Cuadro 3. Podemos calcular la información a mano o construirla de una manera mas fácil y directa utilizando la información de las variables `peso` y `estatura` de la siguiente manera:

$$\texttt{IMC} = \frac{\texttt{peso}}{\texttt{estatura}^2} \tag{1}$$

| ID | Edad | Tipo de diabetes | Estatus | peso | estatura | IMC |
|---|---|---|---|---|---|---|
| 1 | 25 | 1 | Mala | 75 | 1.75 | 24.48 |
| 2 | 34 | 2 | Regular | 70 | 1.65 | 25.71 |
| 3 | 28 | 1 | Excelente | 81 | 1.85 | 23.66 |
| 4 | 52 | 1 | Mala | 70 | 1.69 | 24.50 |
| 5 | 40 | 2 | Excelente | 75 | 1.73 | 25.05 |

Cuadro 4: Resultado de aplicar la Formula 1 con las variables `peso` y `estatura` para construir la variable `IMC` en el DataFrame del Cuadro 3.

```
# Definir variables a partir de variables ya existentes
## DataFrame "datos_pacientes" original
datos_pacientes
```

```
##   ID edad diabetes   estatus estatura peso
## 1  1   25        1      Mala     1.75   75
## 2  2   34        2   Regular     1.65   70
## 3  3   28        1 Excelente     1.85   81
## 4  4   52        1      Mala     1.69   70
## 5  5   40        2 Excelente     1.73   75
```

```
## Utilizamos el símbolo $ para definir la variable IMC a partir de las
## variables peso y estatura
datos_pacientes$IMC <- datos_pacientes$peso/(datos_pacientes$estatura)^2

datos_pacientes
```

```
##   ID edad diabetes   estatus estatura peso      IMC
## 1  1   25        1      Mala     1.75   75 24.48980
## 2  2   34        2   Regular     1.65   70 25.71166
## 3  3   28        1 Excelente     1.85   81 23.66691
## 4  4   52        1      Mala     1.69   70 24.50895
## 5  5   40        2 Excelente     1.73   75 25.05931
```

Otro tipo de manipulación que podemos realizar es el filtrado y ordenamiento de un DataFrame a partir de los valores de una columna (variable). Para continuar con el ejemplo es importante mencionar un tipo de dato especial con el que se trabajara con mucha frecuencia.

### 4.3.1. Factores

Las variables categóricas (nominales) y categóricas ordenadas (ordinales) se denominan factores en R. Las variables de tipo factor son de gran importancia en R porque determinan cómo se analizan y representan visualmente los datos.

La función `factor()` guarda los valores categóricos (niveles) como un vector de enteros en el rango $[1, ..., k]$, (donde $k$ es el número de valores **únicos** de la variable categórica) y un vector "interno" (metadato) de cadena de caracteres (valores originales) que son asignados a los valores enteros.

```
# 5.2 Factores ------------------------------------------------------------

# Ejemplo de varialbe tipo factor y la función factor()
## Vector con la variable "Tipo de diabetes"
diabetes <- c(1, 2, 1, 1)
class(diabetes)
```

```
## [1] "numeric"
```

```
## Vector con la variable "Estatus"
estatus <- c("Mala", "Regular", "Excelente", "Mala")
class(estatus)
```

```
## [1] "character"
```

```
## Convertimos el objeto diabetes de tipo "numeric" a tipo "factor"
diabetes <- factor(diabetes)
class(diabetes)
```

```
## [1] "factor"
```

```
diabetes
```

```
## [1] 1 2 1 1
## Levels: 1 2
```

```
## Convertimos el objeto estatus de tipo "character" a tipo "factor"
estatus <- factor(estatus)
class(estatus)
```

```
## [1] "factor"
```

```
estatus
```

```
## [1] Mala      Regular   Excelente Mala
## Levels: Excelente Mala Regular
```

En el código anterior la instrucción:

```
estatus <- factor(estatus)
```

guardó el vector `c("Mala", "Regular", "Excelente", "Mala")` como `(2, 3, 1, 2)` y asocia estos valores internamente como: `1 = Excelente`, `2 = Mala`, `3 = Regular` (es decir, la asignación se realiza en orden alfabetico).

Por último, para filtrar u ordenar un DataFrame a partir de los valores de una columna (variable) es necesario verificar que la "estructura" del DataFrame sea la indicada. Una función de gran utilidad que nos va a permitir verificar la estructura del DataFrame es la función `str()`.

```r
# Función str()
## Aplicamos la función str() al DataFrame "datos_pacientes"
str(datos_pacientes)
```

```
## 'data.frame':    5 obs. of  7 variables:
##  $ ID      : num  1 2 3 4 5
##  $ edad    : num  25 34 28 52 40
##  $ diabetes: num  1 2 1 1 2
##  $ estatus : chr  "Mala" "Regular" "Excelente" "Mala" ...
##  $ estatura: num  1.75 1.65 1.85 1.69 1.73
##  $ peso    : num  75 70 81 70 75
##  $ IMC     : num  24.5 25.7 23.7 24.5 25.1
```

Podemos notar que las variables `diabetes`, `estatus` del DataFrame son de tipo *numeric* y *character* respectivamente, sin embargo, para evitar complicaciones al momento de realizar analisis y/o representar visualmente los datos será necesario convertir estas variables a tipo factor. Para hacer esto el signo `$` y la función `factor()` serán de gran utilidad.

```r
# Convertir variables entre factor, numeric y character
## Signo $ y función factor()

## Accedemos a la variable "diabetes" y la cambiamos de tipo numeric a tipo factor
datos_pacientes$diabetes <- factor(datos_pacientes$diabetes)

## Accedemos a la variable "estatus" y la cambiamos de tipo character a tipo factor
datos_pacientes$estatus <- factor(datos_pacientes$estatus)

## Revisamos la estructura de "datos_pacientes"
str(datos_pacientes)
```

```
## 'data.frame':    5 obs. of  7 variables:
##  $ ID      : num  1 2 3 4 5
##  $ edad    : num  25 34 28 52 40
##  $ diabetes: Factor w/ 2 levels "1","2": 1 2 1 1 2
##  $ estatus : Factor w/ 3 levels "Excelente","Mala",..: 2 3 1 2 1
##  $ estatura: num  1.75 1.65 1.85 1.69 1.73
##  $ peso    : num  75 70 81 70 75
##  $ IMC     : num  24.5 25.7 23.7 24.5 25.1
```

### 4.3.2. Filtrar y ordenar un DataFrame

Los operadores de comparación (`==`, `!=`, `<`, `>`, `<=`, `>=`) pueden realizar una comparación elemento por elemento de dos vectores. También pueden comparar el elemento de un vector con un escalar. El resultado es un vector de valores lógicos en el que cada valor es el resultado de una comparación de elementos.

```r
# 5.3 Filtrar y Ordenar ----------------------------------------------------

## Valores de "diabetes"
datos_pacientes$diabetes
```

```
## [1] 1 2 1 1 2
## Levels: 1 2
```

```r
## Comparación lógica:
datos_pacientes$diabetes == "1"
```

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

```r
## Valores de "estatus"
datos_pacientes$estatus
```

```
## [1] Mala      Regular   Excelente Mala      Excelente
## Levels: Excelente Mala Regular
```

```r
## Comparación lógica:
datos_pacientes$estatus == "Mala"
```

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

```r
## Filtramos las observaciones donde la variable "diabetes" == "1"
datos_pacientes[datos_pacientes$diabetes == "1", ]
```

```
##   ID edad diabetes   estatus estatura peso      IMC
## 1  1   25        1      Mala     1.75   75 24.48980
## 3  3   28        1 Excelente     1.85   81 23.66691
## 4  4   52        1      Mala     1.69   70 24.50895
```

```r
## Filtramos las observaciones donde la variable "estatus" == "Mala"
datos_pacientes[datos_pacientes$estatus == "Mala", ]
```

```
##   ID edad diabetes estatus estatura peso      IMC
## 1  1   25        1    Mala     1.75   75 24.48980
## 4  4   52        1    Mala     1.69   70 24.50895
```

```r
## Valores de "edad"
datos_pacientes$edad
```

```
## [1] 25 34 28 52 40
```

```r
## Comparación lógica:
datos_pacientes$edad >= 30
```

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

```r
## Filtramos las observaciones donde la variable "edad" >= 30
datos_pacientes[datos_pacientes$edad >= 30, ]
```

```
##   ID edad diabetes   estatus estatura peso      IMC
## 2  2   34        2   Regular     1.65   70 25.71166
## 4  4   52        1      Mala     1.69   70 24.50895
## 5  5   40        2 Excelente     1.73   75 25.05931
```

```r
## Valores de "edad"
datos_pacientes$edad
```

```
## [1] 25 34 28 52 40
```

```r
## Valores de "estatus"
datos_pacientes$estatus
```

```
## [1] Mala      Regular   Excelente Mala      Excelente
## Levels: Excelente Mala Regular
```

```r
## Comparación lógica doble:
datos_pacientes$edad >= 30 & datos_pacientes$estatus == "Mala"
```

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

```r
## Filtramos las observaciones donde la variable "edad" >= 30 y "estatus" == "Mala"
datos_pacientes[datos_pacientes$edad >= 30 & datos_pacientes$estatus == "Mala", ]
```

```
##   ID edad diabetes estatus estatura peso      IMC
## 4  4   52        1    Mala     1.69   70 24.50895
```

Para ordenar un DataFrame podemos utilizar la función `order()`.

```r
# Ejemplo de la función order()
## Ordenamos las observaciones (filas) utilizando la variable (columna) "edad"
datos_pacientes[order(datos_pacientes$edad),]
```

```
##   ID edad diabetes   estatus estatura peso      IMC
## 1  1   25        1      Mala     1.75   75 24.48980
## 3  3   28        1 Excelente     1.85   81 23.66691
## 2  2   34        2   Regular     1.65   70 25.71166
## 5  5   40        2 Excelente     1.73   75 25.05931
## 4  4   52        1      Mala     1.69   70 24.50895
```

```r
## Ordenamos las observaciones (filas) utilizando la variable (columna) "peso"
datos_pacientes[order(datos_pacientes$peso),]
```

```
##   ID edad diabetes   estatus estatura peso      IMC
## 2  2   34        2   Regular     1.65   70 25.71166
## 4  4   52        1      Mala     1.69   70 24.50895
## 1  1   25        1      Mala     1.75   75 24.48980
## 5  5   40        2 Excelente     1.73   75 25.05931
## 3  3   28        1 Excelente     1.85   81 23.66691
```

#### 4.3.3. Exportar un DataFrame en formato csv y en formato Excel

```
# 5.4 Exportar DataFrame ----------------------------------------------------

# Exportar un DataFrame
## Fijamos carpeta de trabajo
setwd("~/Desktop")

## Exportar un DataFrame en formato csv
write.csv(x = datos_pacientes,
          file = "datos.csv",
          row.names = FALSE)

## Exportar un DataFrame en formato Excel
### install.packages("writexl")
library("writexl")
write_xlsx(x = datos_pacientes,
           path = "datos.xlsx")
```

#### 4.3.4. Importar un DataFrame en formato csv y en formato Excel

```
# 5.5 Importar DataFrame ----------------------------------------------------

# Importar un DataFrame
## Importar un DataFrame en formato csv
datos_csv <- read.csv(file = "datos.csv",
                      header = TRUE)

## Importar un DataFrame en formato Excel
### install.packages("readxl")
library(readxl)
datos_xlsx <- read_excel(path = "datos.xlsx")
```

## 5. Cursos en DataCamp

- Introduction to R
- Intermediate R
- Introduction to the Tidyverse

## Referencias

Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. doi: 10.18637/jss.v059.i10. URL https://www.jstatsoft.org/index.php/jss/article/view/v059i10.