

Práctica 2. Introducción y manipulación de datos en R (1)

Jesús Martín Fernández

Contenidos

1. Tipos de objetos en R.	1
2. Operadores básicos en R	3
3. Vectores	4
3.1 Creación de vectores	5
3.2 Operaciones con vectores	8
4. Matrices	13
4.1 Creación de matrices	13
4.2 Operaciones con matrices	14
5. Ejercicios	19
5.1 Ejercicios con vectores	19
5.2 Ejercicios con matrices	22

1. Tipos de objetos en R.

R es capaz de gestionar diversos tipos de datos, que se organizan en distintas estructuras de datos. A continuación se presenta una tabla que describe los tipos de datos básicos en R:

Tipo de dato	Descripción	Ejemplo
Numeric	Números decimales	<code>numero <- 1.0</code>
Integer	Números enteros	<code>int <- 1</code>
Character	Cadenas de texto	<code>str <- "un texto"</code>
Complex	Números complejos	<code>comp <- 3+2i</code>
Logical	Valores de verdad: TRUE o FALSE, a menudo generados como resultado de operaciones lógicas.	<code>a <- 1; b <- 2; a < b</code>

Tipo de dato	Descripción	Ejemplo
Factor	Los factores son un tipo de variable, no de dato, las variables categóricas. Los vectores de caracteres suelen convertirse en factores para aprovechar las funciones que manejan datos categóricos, como en el análisis de regresión.	Convertir un vector de caracteres usando <code>as.factor()</code> .

Las estructuras de datos son objetos que contienen datos y organizados de diferentes maneras. Las estructuras vienen definidas por su número de dimensiones y por la homogeneidad o no de sus datos.

La siguiente tabla muestra las principales estructuras de datos que maneja R.

Objeto	Modos	Descripción	¿Múltiples Modos ?	Dimensiones
Vector	Numérico, Carácter, Complejo, Lógico	Secuencia de elementos del mismo tipo (numérico, carácter, complejo, lógico).	No	1D
Array	Numérico, Carácter, Complejo, Lógico	Datos en varias dimensiones con elementos del mismo tipo (numérico, carácter, complejo, lógico).	No	Multi-dimensional (2D, 3D, ...)
Matriz	Numérico, Carácter, Complejo, Lógico	Colección bidimensional de elementos del mismo tipo (numérico, carácter, complejo, lógico).	No	2D
Data Frame	Numérico, Carácter, Complejo, Lógico	Similar a una matriz pero puede contener diferentes tipos de datos (numéricos, caracteres, etc.) en sus columnas.	Sí	2D
TS	Numérico, Carácter, Complejo, Lógico	Serie temporal (time series), un conjunto de datos ordenados por tiempo, del mismo tipo (numérico, carácter, complejo, lógico).	No	1D (univariada) o 2D (multi-variada)

Objeto	Modos	Descripción	¿Múltiples Modos ?	Dimensiones
Lista	Numérico, Carácter, Complejo, Lógico, Función, Expresión, etc.	Contenedor flexible que puede almacenar elementos de diferentes tipos (numérico, carácter, complejo, lógico, función, expresión, etc.).	Sí	1D (pero puede contener estructuras de cualquier dimensión)

2. Operadores básicos en R

Vamos a hacer un resumen de los principales operadores y las funciones matemáticas más utilizadas en R

Operador	Descripción	Ejemplo
==	Igual a	x == 5
!=	Distinto de	x != 3
>	Mayor que	x > 3
<	Menor que	x < 5
>=	Mayor o igual que	x >= 5
<=	Menor o igual que	x <= 5
&	Y lógico (AND)	x>2 & x<3
	O lógico (OR)	x<2 x>3
!	Negación lógica (NOT)	!TRUE
all()	Devuelve TRUE si todos los elementos son TRUE	all(c(TRUE, TRUE))
any()	Devuelve TRUE si al menos un elemento es TRUE	any(c(FALSE, TRUE))

Ahora vamos a recordar las funciones matemáticas básicas más utilizadas

Función	Descripción	Ejemplo	Resultado
+	Suma dos números	5 + 3	8
-	Resta un número a otro	5 - 3	2
*	Multiplica dos números	5 * 3	15
/	Divide un número por otro	6 / 3	2
^	Exponenciación	2^3	8
%%	Resto de la división (módulo)	5 %% 2	1

Función	Descripción	Ejemplo	Resultado
%/%	División entera	5 %/% 2	2
round()	Redondea a un número especificado de decimales	round(3.14159, 2)	3.14
ceiling()	Redondea al entero superior más cercano	ceiling(2.3)	3
floor()	Redondea al entero inferior más cercano	floor(2.7)	2
trunc()	Trunca la parte decimal, dejando solo la parte entera	trunc(2.7)	2
sqrt()	Calcula la raíz cuadrada	sqrt(16)	4
exp()	Calcula el exponencial de un número (e^x)	exp(1)	2.71882
log()	Calcula el logaritmo natural (base e)	log(10)	2.302585
log10()	Calcula el logaritmo en base 10	log10(100)	2
log2()	Calcula el logaritmo en base 2	log2(8)	3
abs()	Calcula el valor absoluto	abs(-5)	5
sum()	Suma todos los elementos de un vector	sum(c(1, 2, 3))	6
prod()	Calcula el producto de todos los elementos de un vector	prod(c(1, 2, 3))	6
mean()	Calcula el promedio de un conjunto de números	mean(c(1, 2, 3))	2
min()	Encuentra el mínimo valor en un conjunto de números	min(c(1, 2, 3))	1
max()	Encuentra el máximo valor en un conjunto de números	max(c(1, 2, 3))	3

3. Vectores

Un **vector** es una secuencia ordenada de datos del mismo tipo (siendo éstos numéricos, cadenas, fechas, etc). Es la estructura de datos más sencilla. Un simple valor (por ejemplo el número 5) constituye un vector numérico. Los escalares son vectores de longitud 1

```
x<- 5
x
```

```
[1] 5
```

Pero un vector puede contener otro tipo de datos como cadenas de texto o datos lógicos

```
x<- as.vector (TRUE)
x
```

```
[1] TRUE
```

```
y<-as.vector(c("Andrés", "Luis"))
y
```

```
[1] "Andrés" "Luis"
```

3.1 Creación de vectores

El vector es el elemento más básico en R, y contiene elementos de la misma clase.

Se crea con la función `c()`, que significa ‘concatenar’ o ‘combinar’ y el paréntesis debe incluir los datos que deseamos integrar en ese vector, separados por comas (y en caso de datos de tipo `character` flanqueados por comillas).

Vector formado por números enteros:

Con la función concatenar

```
x <- c(1,2,3,4,5,6,7,8,9,10)
x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

Hay otras formas de generar una secuencia de números

Con expresiones específicas

```
x <- 1:10
# Crea un vector con elementos del 1 al 10
x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

El operador `:` tiene prioridad

```
x <- 1:10-2 #Primero se genera el vector 1:10 (de 1 a 10) y luego se resta -2 a cada elemento
x
```

```
[1] -1  0  1  2  3  4  5  6  7  8
```

Se pueden cambiar las prioridades usando paréntesis

```
x <- 1:(10-2) # Genera una serie de 1 a 8
x
```

```
[1] 1 2 3 4 5 6 7 8
```

Se puede crear secuencias de números enteros con **sequence**

```
x<-sequence(4:5)
#Genera una secuencia hasta 4 y otra hasta 5
x
```

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

La función **seq** es una función más general y flexible para generar secuencias numéricas. **seq**(from, to, by, length.out, along.with), Sus parámetros son from: El valor inicial de la secuencia. to: El valor final de la secuencia. by: El incremento (o decremento si es negativo) entre elementos de la secuencia. length.out: El número total de elementos en la secuencia. Si se especifica, el valor de by se ajusta automáticamente.

```
x<- seq(1, 2, by= 0.1)
#Genera una secuencia de números entre 1 y 2 con valores separados por 0,1
x
```

```
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

```
x<- seq(length=9, from=1, to=5)
#La función seq crea una secuencia de 9 elementos, con valor inicial 1 y final 5.
x
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Otra forma de crear un vector con elementos repetidos es con **rep**

```
x <-rep(1, 10) #Crea un vector en el que aparece 10 veces el numero 1
x
```

```
[1] 1 1 1 1 1 1 1 1 1 1
```

Se pueden generar secuencias con distribuciones fijas, por ejemplo la normal con la función `rnorm`

```
x<-rnorm(10, 50, 5) # 10 valores de media 50 y desviación típica 5
x
```

```
[1] 46.55738 55.78267 50.23971 59.75445 47.47144 50.00747 47.98614 43.63511
[9] 51.08935 43.78145
```

Se puede crear un vector con elementos repetidos en forma de series con la función `gl` (generate levels)

```
x<-gl(3, 4) #gl (generate levels) genera un factor con 3 niveles (1, 2, 3),\
#repitiendo cada nivel 4 veces.
x
```

```
[1] 1 1 1 1 2 2 2 2 3 3 3 3
Levels: 1 2 3
```

Vector formado por caracteres:

Cada uno de los elementos de un vector de caracteres debe escribirse entre comillas. En caso contrario, la máquina devuelve un mensaje de error.

```
ciudades<- c("Jaen", "Córdoba", "Sevilla")
ciudades
```

```
[1] "Jaen"      "Córdoba" "Sevilla"
```

Vector lógico:

Solo requiere definir los valores verdaderos (TRUE) y falsos (FALSE):

```
vector_logico <- c(TRUE, FALSE, TRUE, FALSE)
print(vector_logico)
```

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

Los vectores lógicos se pueden crear al poner una condición a otro vector:

```
# Crear un vector numérico
numeros <- c(1, -5, 7, -2, 8, 9, 2)

# Crear un vector lógico que indica si cada elemento es positivo o no
positivo <- numeros>0
print(positivo)
```

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

3.2 Operaciones con vectores

La primera operación que describiremos será comprobar si un objeto es un vector

```
v0 <- 10
v0
```

```
[1] 10
```

```
is.vector(v0) # Verificar si el objeto o variable 'v0' es un vector.
```

```
[1] TRUE
```

Una vez creados los vectores, pueden ser modificados posteriormente. Por ejemplo, si deseamos agregar un elemento a un vector ya existente, lo añadimos asignándolo a nuestro vector original.

Vamos a operar con el vector `v1(2,5,10)`

```
v1<-c(2,5,10)
v1
```

```
[1] 2 5 10
```


Se pueden añadir elementos a un vector

```
v2<-c (v1,8)
v2
```

```
[1]  2  5 10  8
```

Ahora tenemos dos vectores $v1(2,5,10)$, $v2(2,5,10,8)$

Se pueden realizar operaciones aritméticas con vectores. Se pueden sumar $v1$ y $v2$

```
v1+v2
```

Warning in $v1 + v2$: longitud de objeto mayor no es múltiplo de la longitud de uno menor

```
[1]  4 10 20 10
```

Aviso en $v1 + v2$: longitud de objeto mayor no es múltiplo de la longitud de uno menor. Al último valor de $v2$, le suma el primero de $V1$ (“reciclaje”)

No ocurriría si $v1$ y $v2$ tuviesen igual tamaño.

También pueden multiplicarse $v1$ y $v2$

```
v1*v2
```

Warning in $v1 * v2$: longitud de objeto mayor no es múltiplo de la longitud de uno menor

```
[1]  4  25 100  16
```

Aviso en $v1*v2$: longitud de objeto mayor no es múltiplo de la longitud de uno menor. Multiplica último valor de $v2$, por el primero de $V1$ (“reciclaje”). #No ocurriría si $v1$ y $v2$ tuviesen igual tamaño.

Se pueden elevar al cuadrado los términos de un vector

```
v1^2
```

```
[1]  4  25 100
```

Le podemos pedir a R que nos calcule la raíz cuadrada de los elementos de un vector con la función `sqrt`,

```
sqrt (v2) #v2(2,5,10,8)
```

```
[1] 1.414214 2.236068 3.162278 2.828427
```

Se pueden sumar los términos de un vector, con la función `'sum'`

```
sum (v2) #v2(2,5,10,8)
```

```
[1] 25
```

Se puede calcular la media de los términos de un vector con la función `'mean'`

```
mean (v2) #v2(2,5,10,8)
```

```
[1] 6.25
```

Atención, si un vector tiene valores ausentes `NA` hay que hacérselo notar a la función. Imaginemos `v3(0,10,8,NA,6)`

```
v3<-c(0,10,8,NA,6)  
mean (v3)
```

```
[1] NA
```

la forma correcta de calcular la media sería

```
mean(v3, na.rm = TRUE)
```

```
[1] 6
```

Existen otras funciones que operan con los valores de un vector

```
v3<-c(0,10,8,NA,6)  
length(v3) #número de valores de v3
```

```
[1] 5
```

```
max(v3, na.rm = TRUE) #Valor máximo de v3, sabiendo que hay valores NA
```

```
[1] 10
```

```
min(v3, na.rm = TRUE) #Valor mínimo de v3, sabiendo que hay valores NA
```

```
[1] 0
```

Se puede operar con un solo vector

```
v2*2+10
```

```
[1] 14 20 30 26
```

Cuando le indicamos a R que calcule $v2 * 2 + 10$, lo que realmente está calculando es: $v2 * c(2, 2, 2, 2) + c(10, 10, 10, 10)$

Algunas operaciones se pueden escribir de forma simplificada

```
v2 + c(2, 10)
```

```
[1] 4 15 12 18
```

Esta operación suma 2 a los valores impares y 10 a los pares. Recuerde que $v2(2,5,10,8)$,

Los elementos de un vector se pueden recuperar, por medio de corchetes. Siendo $v1(2,5,10)$ $v2(2,5,10,8)$

```
v1[2]
```

```
[1] 5
```

```
v2[3]
```

```
[1] 10
```

Se puede operar con valores de determinada posición de los vectores, vamos a sumar el valor de la segunda posición de $v1$ con el de la tercera posición de $v2$

```
v1[2] + v2[3]
```

```
[1] 15
```

También se pueden añadir los corchetes para añadir un valor en una posición o para cambiarlo. Veamos `v1(2,5,10)`

```
v1[4] <- 12  
v1
```

```
[1] 2 5 10 12
```

```
v1[2] <- 12  
v1
```

```
[1] 2 12 10 12
```

Los elementos de un vector se pueden recuperar en función de una orden lógica. Siendo `v1(2,12,10,12)`

```
v1 > 7
```

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

```
v1 < 7
```

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

```
v1 == 12
```

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

También puede usarse la función `which`. Tenemos `v3(0,10,8,NA,6)`

```
which(v3 > 2) # Posiciones de los elementos mayores que 2
```

```
[1] 2 3 5
```

```
which(v3 < 2 | v3 >= 8) # Posiciones de los elementos <2 o >= 8.
```

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

Atención, hay un elemento NA, aunque R en esta versión lo identifica como tal, es más recomendable

```
which(!is.na(v3) & v3 < 2 | v3 >= 8)
```

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

Posiciones de los elementos >2 y <= 8 sabiendo que existen valores ausentes.

4. Matrices

En R, una matriz es una colección bidimensional de elementos que deben ser del mismo tipo (numérico, carácter, lógico, etc.).

4.1 Creación de matrices

La función `matrix()` es la forma más común de crear matrices en R.

```
matrix(1:12, nrow = 4, ncol = 3)
```

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

Cuando no se especifica nada la matriz se cumplimenta por columnas.

Pero se puede pedir que se cumplimente por filas

```
matrix(1:12, nrow = 4, ncol = 3, byrow = TRUE)
```

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

La función `matrix()` se puede sustituir con la instrucción `dim`, que transforma un vector en una matriz

```
x<-1:15
dim(x) <- c(5, 3)
x
```

	[,1]	[,2]	[,3]
[1,]	1	6	11
[2,]	2	7	12
[3,]	3	8	13
[4,]	4	9	14
[5,]	5	10	15

4.2 Operaciones con matrices

Las funciones básicas explicadas se pueden emplear en una matriz

```
range(x) #Rango de la matriz x
```

```
[1] 1 15
```

```
length(x) #Número de elementos en la matriz x
```

```
[1] 15
```

```
mean(x) #Media de los valores de la matriz x
```

```
[1] 8
```

```
median(x) #Mediana de los valores de la matriz x
```

```
[1] 8
```

```
sum(x) #Suma de los valores de la matriz x
```

```
[1] 120
```

```
prod(x) #Producto de los valores de la matriz x
```

```
[1] 1.307674e+12
```

```
max(x) #Valor máximo de la matriz x
```

```
[1] 15
```

```
min(x) #Valor mínimo de la matriz x
```

```
[1] 1
```

```
which.max(x) # Devuelve el valor máximo de la matriz x
```

```
[1] 15
```

```
which.min(x) #Devuelve valor mínimo de la matriz x
```

```
[1] 1
```

Las matrices pueden unirse entre sí

```
m1<-matrix(1:4, nrow=2, ncol=2)
m2<-matrix(5:8, nrow=2, ncol=2)
m1
```

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

```
m2
```

	[,1]	[,2]
[1,]	5	7
[2,]	6	8

Se pueden unir por filas con la orden `rbind`

```
rbind (m1,m2)
```

	[,1]	[,2]
[1,]	1	3
[2,]	2	4
[3,]	5	7
[4,]	6	8

Y se pueden unir por columnas con la orden `cbind`

```
cbind (m1,m2)
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	3	5	7
[2,]	2	4	6	8

Las matrices también se pueden sumar o multiplicar

```
m1+m2
```

	[,1]	[,2]
[1,]	6	10
[2,]	8	12

```
m1*m2
```

	[,1]	[,2]
[1,]	5	21
[2,]	12	32

Las matrices se pueden transponer con la orden `t`


```
t (m1)
```

	[,1]	[,2]
[1,]	1	2
[2,]	3	4

```
t (m2)
```

	[,1]	[,2]
[1,]	5	6
[2,]	7	8

Las filas y las columnas de las matrices pueden etiquetarse

```
colnames(m1) <- c("Columna 1", "Columna 2")
rownames (m1) <- c("Fila 1", "Fila 2")
m1
```

	Columna 1	Columna 2
Fila 1	1	3
Fila 2	2	4

Las posiciones de los valores de una matriz pueden recuperarse, supongamos una nueva matriz m3

```
m3<- rbind(m1,m2)
rownames (m3) <- c("Fila 1", "Fila 2", "Fila 3", "Fila 4")
m3
```

	Columna 1	Columna 2
Fila 1	1	3
Fila 2	2	4
Fila 3	5	7
Fila 4	6	8

Vamos a intentar localizar algunas posiciones en la matriz

```
m3[1,] # fila 1 todas las columnas.
```

Columna 1	Columna 2
1	3

```
m3[,2] # todas filas, columna 2
```

Fila 1	Fila 2	Fila 3	Fila 4
3	4	7	8

```
m3[1:2,1] #filas 1 y 2, columna 1
```

Fila 1	Fila 2
1	2

```
m3[-(1:2),2] #excluir filas 1 a 2, incluir columna 2.
```

Fila 3	Fila 4
7	8

```
m3[,"Columna 2"] #La columna 2, usando el nombre de la columna
```

Fila 1	Fila 2	Fila 3	Fila 4
3	4	7	8

```
m3[3,c("Columna 1","Columna 2")] # La fila 3 y todas las columnas, llamadas por su nombre
```

Columna 1	Columna 2
5	7

5. Ejercicios

5.1 Ejercicios con vectores

Crea un vector `v1` en el que se repita la secuencia 1, 2, 3, 4, 5 dos veces. Luego, encuentra la suma, la media y la desviación estándar de los valores en el vector.

Este es el código que se puede usar (piensa que puede haber otras alternativas, si el resultado es el mismo, serán válidas)

```
v1 <- rep(c(1, 2, 3), times = 5)

# Mostrar el vector
print(v1)

# Calcular la suma
sum_v1
print(sum_v1)
# Imprime: 45

# Calcular la media
media_v1 <- mean(v1)
print(media_v1)
# Imprime: 2.5

# Calcular la desviación estándar
sd_v1 <- sd(sd_v1)
print(sd_v1)
# Imprime: 0.8164966
```

Encuentra el valor de la suma de los primeros 100 números naturales

```
# Crear el vector con números naturales del 1 al 100
v100 <- 1:100

# Mostrar el vector (no es necesario)
print(v100)

# Calcular la suma de los valores del vector
suma_v100 <- sum(v100)
print(suma_v100)
# Imprime: 5050
```

Crea dos vectores de la misma longitud, `vector_a` (1,2,3,4,5) y `vector_b` (5,4,3,2,1). Realiza las siguientes operaciones: Suma elemento a elemento, resta elemento a elemento, multiplicación elemento a elemento y divide elemento a elemento.

```
# Crear los vectores
vector_a <- c(1, 2, 3, 4, 5)
vector_b <- c(5, 4, 3, 2, 1)

# Suma elemento a elemento
suma_elemento <- vector_a + vector_b
print(suma_elemento) # Imprime: 6 6 6 6 6

# Resta elemento a elemento
resta_elemento <- vector_a - vector_b
print(resta_elemento) # Imprime: -4 -2 0 2 4

# Multiplicación elemento a elemento
multiplicacion_elemento <- vector_a * vector_b
print(multiplicacion_elemento) # Imprime: 5 8 9 8 5

# División elemento a elemento
division_elemento <- vector_a / vector_b
print(division_elemento) # Imprime: 0.2 0.5 1 2 5
```

Crea un vector `v3` con los números del 1 al 15. Luego extrae los elementos en las posiciones 3, 5 y 7, y después extrae todos los elementos que sean mayores a 10.

```
# Crear el vector
v3 <- 1:15

# Extraer elementos en posiciones 3, 5 y 7
el_selecc <- v3[c(3, 5, 7)]
print(el_selec) # Imprime: 3 5 7

# Extraer todos los elementos mayores a 10
mayores10 <- v3[v3 > 10]
print(mayores10) # Imprime: 11 12 13 14 15
```

Crea un vector lógico `vlog` (TRUE, FALSE, TRUE, TRUE, FALSE) . Realiza las siguientes operaciones: calcula la cantidad de TRUE en el vector, verifica si todos los valores en el vector son TRUE, verifica si al menos uno de los valores en el vector es TRUE.

```

# Crear el vector
vlog <- c(TRUE, FALSE, TRUE, TRUE, FALSE)

# Contar la cantidad de TRUE
cantidad_true <- sum(vlog)
print(cantidad_true) # Imprime: 3

# Verificar si todos los valores son TRUE
todos_true <- all(vlog)
print(todos_true) # Imprime: FALSE

# Verificar si al menos uno es TRUE
al_menos_un_true <- any(vlog)
print(al_menos_un_true) # Imprime: TRUE

```

Vamos a crear un vector de 10 números aleatorios `v_aleat` y encuentra el valor máximo y mínimo. Después ordena el vector de menor a mayor. Ten en cuenta que los resultados pueden ser diferentes cada vez que ejecutes el código

```

# Crear un vector con 10 números aleatorios
v_aleat <- sample(1:100, 10)

# Mostrar el vector
print(v_aleat)
# Ejemplo de salida: 55 7 2 80 35 19 74 6 48 63

# Encontrar el valor máximo
maximo <- max(v_aleat)
print(maximo)
# Ejemplo de salida: 80

# Encontrar el valor mínimo
minimo <- min(v_aleat)
print(minimo)
# Ejemplo de salida: 2

# Ordenar el vector de menor a mayor
v_ordenado <- sort(v_aleat)
print(v_ordenado)
# Ejemplo de salida: 2 6 7 19 35 48 55 63 74 80

```

5.2 Ejercicios con matrices

Crea una matriz `m1` de dimensiones 5x5 con los números del 1 al 25 creados por filas

```
# Crear una matriz de 5 filas y 5 columnas con números del 1 al 25 por filas
m1 <- matrix(1:25, nrow = 5, ncol = 5, byrow = TRUE)

# Mostrar la matriz
print(m1)
# Imprime:
#      [,1] [,2] [,3] [,4] [,5]
# [1,]    1    2    3    4    5
# [2,]    6    7    8    9   10
# [3,]   11   12   13   14   15
# [4,]   16   17   18   19   20
# [5,]   21   22   23   24   25
```

Usa la matriz `m1` creada anteriormente. Accede al elemento en la tercera fila y cuarta columna. También, extrae la segunda fila completa y la cuarta columna completa.

```
# Acceder al elemento en la tercera fila y cuarta columna
elemento <- m1[3, 4]
print(elemento)
# Imprime: 14

# Extraer la segunda fila completa
fila2 <- m1[2, ]
print(fila2)
# Imprime: 6 7 8 9 10

# Extraer la cuarta columna completa
columna4 <- m1[ , 4]
print(columna4)
# Imprime: 4 9 14 19 24
```

Usa la matriz `m1` y realiza las siguientes operaciones: suma todos los elementos de la matriz, encuentra la media de los elementos de la matriz, multiplica cada elemento de la matriz por 3.

```
# Sumar todos los elementos de la matriz
suma_m1 <- sum(m1)
print(suma_m1)
```

```

# Imprime: 325

# Encontrar la media de los elementos de la matriz
media_m1 <- mean(m1)
print(media_m1)
# Imprime: 13

# Multiplicar cada elemento de la matriz por 3
m1_multiplicada <- m1 * 3
print(m1_multiplicada)
# Imprime:
#      [,1] [,2] [,3] [,4] [,5]
# [1,]   3   6   9  12  15
# [2,]  18  21  24  27  30
# [3,]  33  36  39  42  45
# [4,]  48  51  54  57  60
# [5,]  63  66  69  72  75

```

Traspón la matriz m1 y después sustituye el valor en la primera fila y primera columna a 100 y el valor en la tercera fila y quinta columna a 200.

```

# Transponer la matriz
m1_transpuesta <- t(m1)

# Mostrar la matriz transpuesta
print("Matriz transpuesta:")
print(m1_transpuesta)
# Imprime:
#      [,1] [,2] [,3] [,4] [,5]
# [1,]   1   6  11  16  21
# [2,]   2   7  12  17  22
# [3,]   3   8  13  18  23
# [4,]   4   9  14  19  24
# [5,]   5  10  15  20  25

# Cambiar el valor en la primera fila y primera columna a 100
m1_transpuesta[1, 1] <- 100

# Cambiar el valor en la tercera fila y quinta columna a 200
m1_transpuesta[3, 5] <- 200

# Mostrar la matriz transpuesta modificada

```

```

print("Matriz transpuesta modificada:")
print(m1_transpuesta)
# Imprime:
#      [,1] [,2] [,3] [,4] [,5]
# [1,] 100   6  11  16  21
# [2,]   2   7  12  17  22
# [3,]   3   8  13  18 200
# [4,]   4   9  14  19  24
# [5,]   5  10  15  20  25

```

Añade a m1 una fila con los valores 26, 27, 28 , 29 y 30.

```

fila6 <- c(26, 27, 28, 29, 30)
m1_mod <- rbind(m1, fila6)

# Mostrar la matriz con la nueva fila añadida

print(m1_mod)
# Imprime:
#      [,1] [,2] [,3] [,4] [,5]
# [1,]   1   2   3   4   5
# [2,]   6   7   8   9  10
# [3,]  11  12  13  14  15
# [4,]  16  17  18  19  20
# [5,]  21  22  23  24  25
# [6,]  26  27  28  29  30

```