
PROGRAMACIÓN R

REPOSITORIO

- https://github.com/jorloicono/AF-INTRODUCCION-R

INTRODUCCIÓN


RStudio® Cloud

INSTALACIÓN

- R: https://cran.r-project.org/bin/windows/base/
- Rstudio: https://www.rstudio.com/products/rstudio/download/

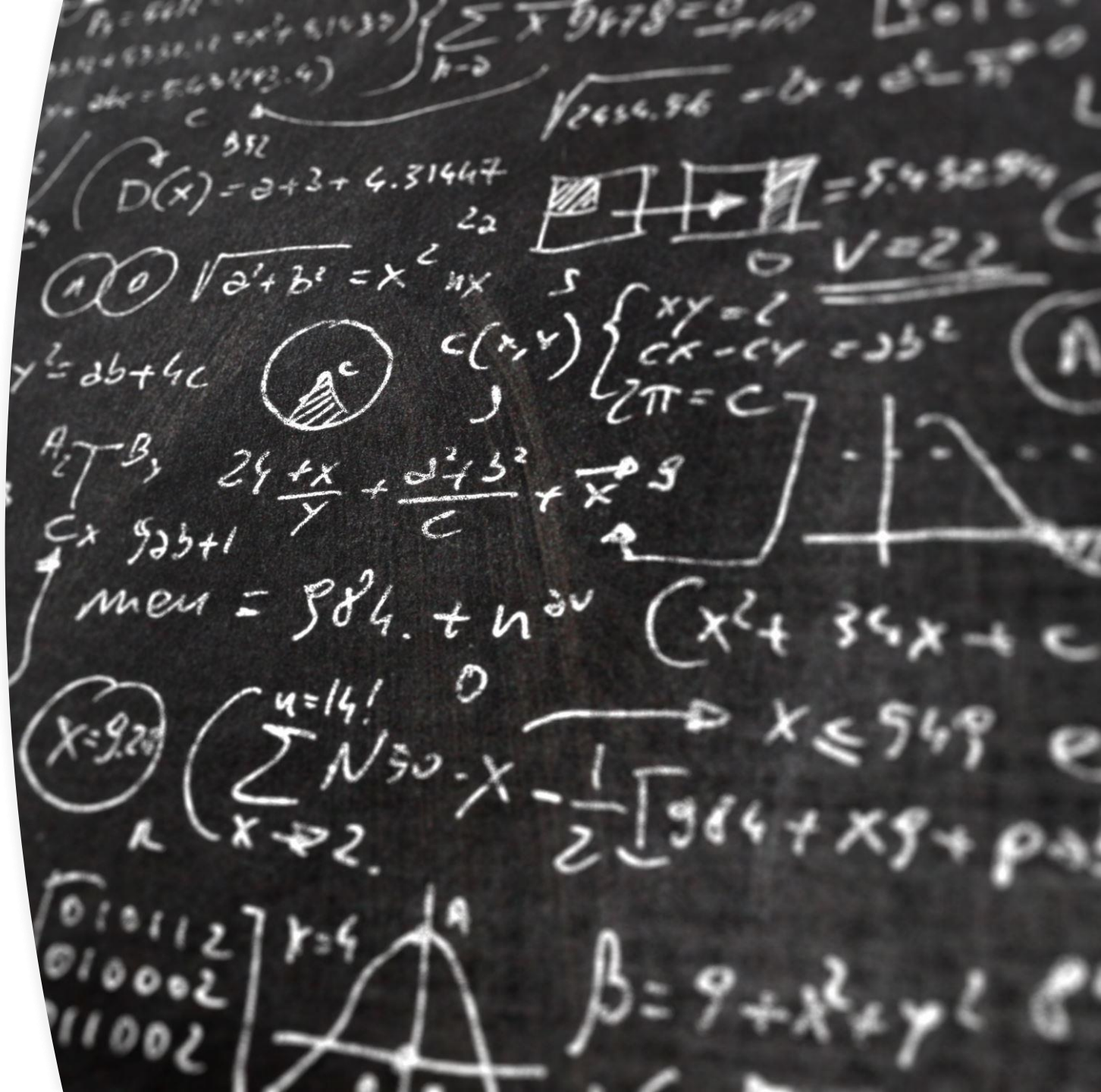
TIPOS DE DATOS

Tipo	Ejemplo	Nombre en inglés
Entero	1	integer
Numérico	1.3	numeric
Cadena de texto	“uno”	character
Factor	uno	factor
Lógico	TRUE	logical
Perdido	`NA`	NA
Vacio	`NULL`	null

OPERADORES

Los tipos de operadores principales son los siguientes:

- Aritméticos
- Relacionales
- Lógicos
- De asignación



ARITMÉTICOS

Operador	Operación	Ejemplo	Resultado
`+`	Suma	`5 + 3`	8
`-`	Resta	`5 - 3`	2
`*`	Multiplicación	`5 * 3`	18
`/`	División	`5 /3`	1.666667
`^`	Potencia	`5 ^ 3`	125
`%%`	División entera	`5 %% 3`	2

RELACIONALES

Operador	Comparación	Ejemplo	Resultado
`<`	Menor que	`5 < 3`	`FALSE`
`<=`	Menor o igual que	`5 <= 3`	`FALSE`
`>`	Mayor que	`5 > 3`	`TRUE`
`>=`	Mayor o igual que	`5 >= 3`	`TRUE`
`==`	Exactamente igual que	`5 == 3`	`FALSE`
`!=`	No es igual que	`5 != 3`	`TRUE`

LÓGICOS

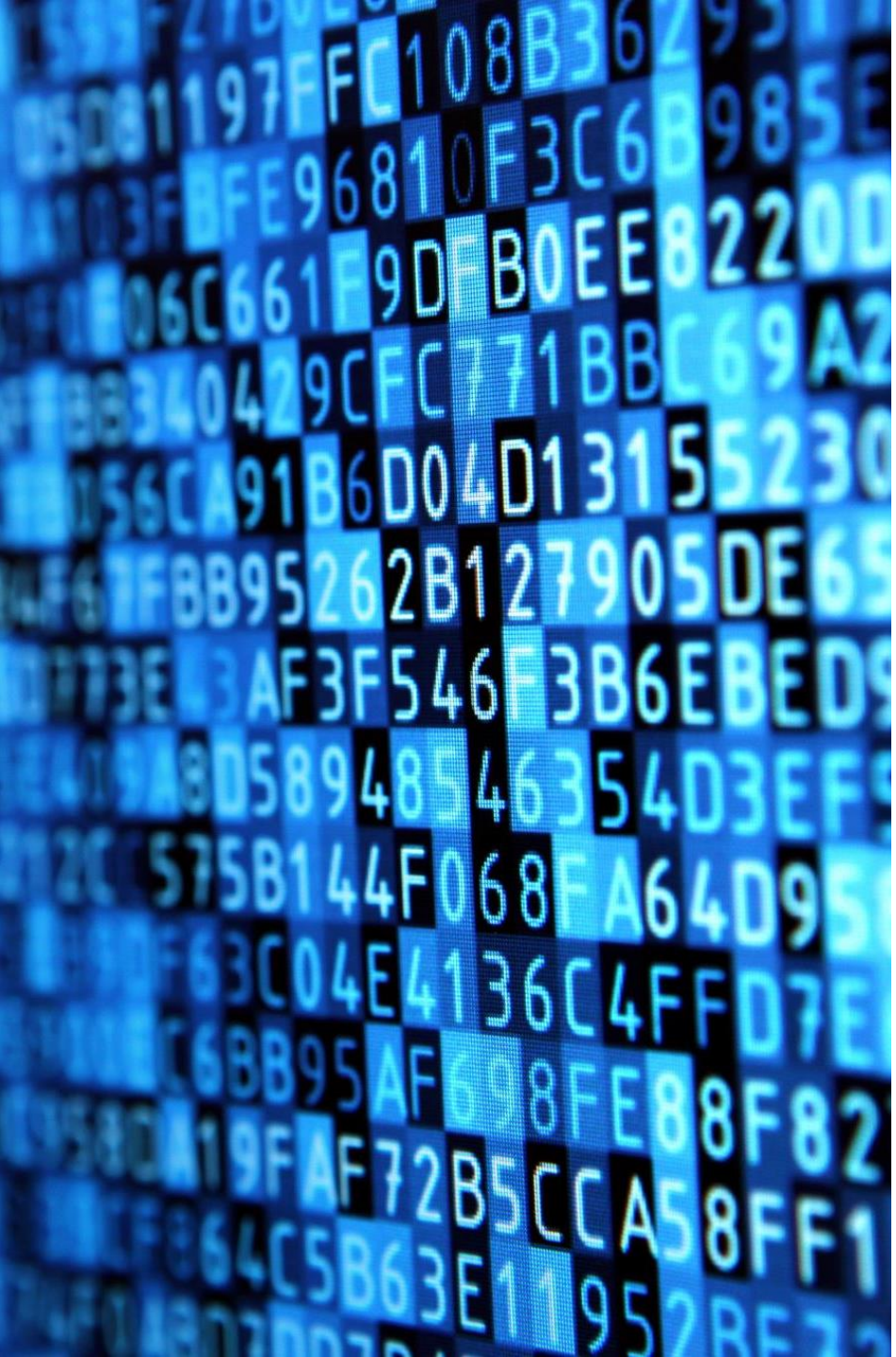
Operador	Comparación	Ejemplo	Resultado
`x | y`	x Ó y es verdadero	`TRUE | FALSE`	`TRUE`
`x & y`	x Y y son verdaderos	`TRUE & FALSE`	`FALSE`
`!x`	x no es verdadero (negación)	`!TRUE`	`FALSE`
`isTRUE(x)`	x es verdadero (afirmación)	`isTRUE(TRUE)`	`TRUE`

ASIGNACIÓN

Operador	Operación
`<-`	Asigna un valor a una variable
`=`	Asigna un valor a una variable

ESTRUCTURAS DE DATOS

Dimensiones	Homogeneas	Heterogeneas
1	Vector	Lista
2	Matriz	Data frame
n	Array	



VECTORES

Todos los vectores tienen tres propiedades:

- **Tipo**. Un vector tiene el mismo tipo que los datos que contiene. Si tenemos un vector que contiene datos de tipo numérico, el vector será también de tipo numérico.
- **Largo**. Es el número de elementos que contiene un vector.
- **Atributos**. Los vectores pueden tener metadatos de muchos tipos, los cuales describen características de los datos que contienen.

MATRICES

- Las matrices y arrays pueden ser descritas como **vectores multidimensionales**. Al igual que un vector, únicamente pueden contener datos de un sólo tipo, pero además de largo, tienen más dimensiones.

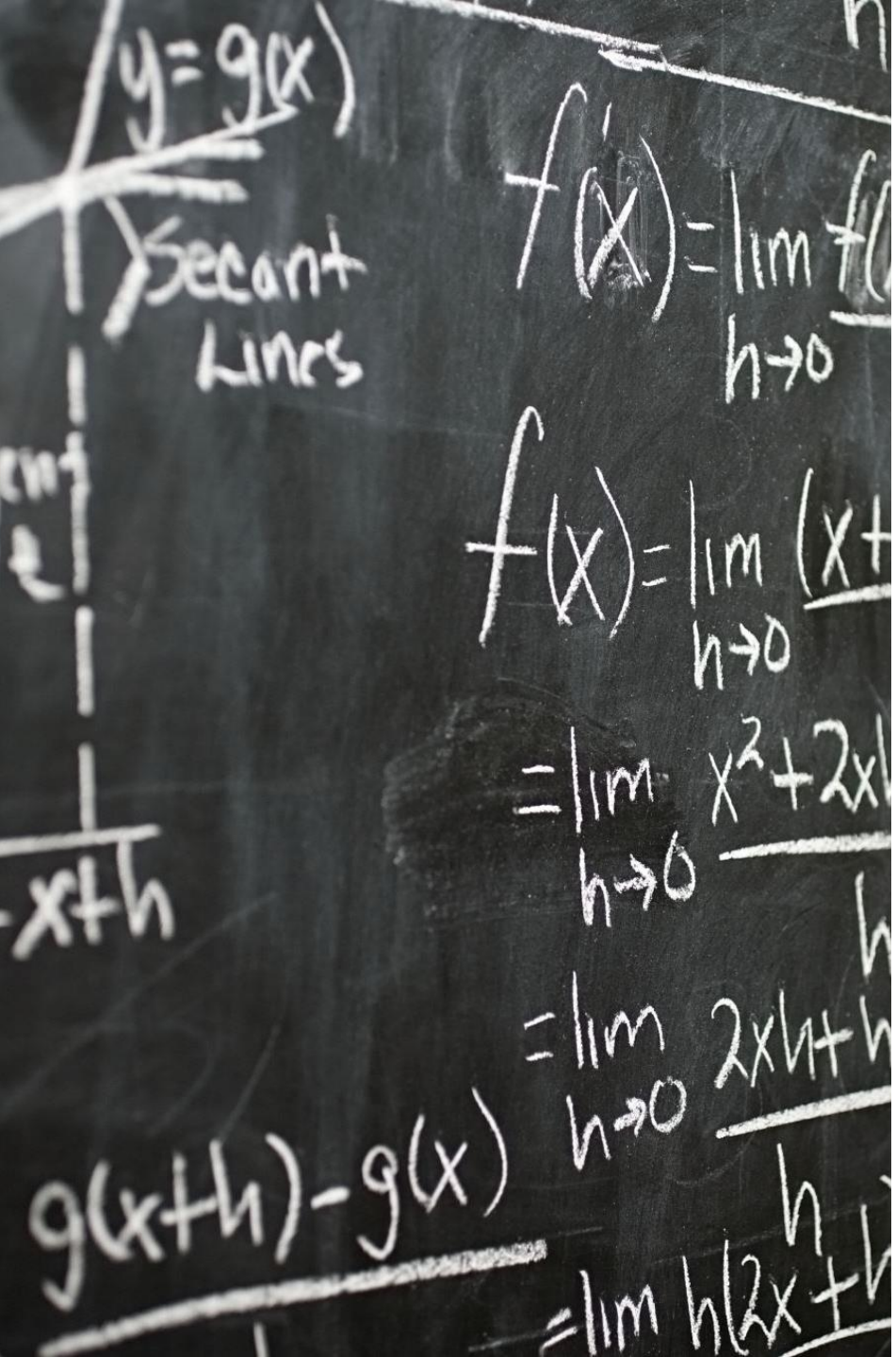
EJERCICIO

Se te pide escribir un programa en R que realice las siguientes operaciones con matrices:

1. Definir dos matrices A y B, cada una de tamaño 3x3, con los siguientes valores:
   - Matriz A: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
   - Matriz B: [[9, 8, 7], [6, 5, 4], [3, 2, 1]]
2. Calcular la suma de las matrices A y B, y almacenar el resultado en una nueva matriz C.
3. Calcular el producto de las matrices A y B, y almacenar el resultado en una nueva matriz D.
4. Imprimir las matrices C y D resultantes.

DATAFRAMES

- Los data frames son estructuras de datos de dos dimensiones (rectangulares) que pueden contener datos de diferentes tipos, por lo tanto, son heterogéneas. Esta estructura de datos es la más usada para realizar análisis de datos y seguro te resultará familiar si has trabajado con otros paquetes estadísticos.



LISTAS

- Las listas, al igual que los vectores, son estructuras de datos unidimensionales, sólo tienen largo, pero a diferencia de los vectores cada uno de sus elementos puede ser de diferente tipo o incluso de diferente clase, por lo que son estructuras heterogéneas.

ÍNDICES

- Usar índices para obtener subconjuntos es el procedimiento más universal en R, pues funciona para todas las estructuras de datos.
- Un segundo método para extrar subconjuntos es usar los nombres de los elementos en una estructura de datos. Este forma de obtener subconjuntos es usada principalmente con data frames y listas.
- Las condicionales nos permiten obtener subconjuntos que para los que una o más condiciones son verdaderas (TRUE).

EJERCICIO

Se te proporciona un conjunto de datos que contiene información sobre el rendimiento académico de los estudiantes en diferentes asignaturas. El conjunto de datos consta de las siguientes columnas:

'Nombre': el nombre del estudiante. 'Edad': la edad del estudiante. 'Asignatura': el nombre de la asignatura. 'Calificación': la calificación obtenida por el estudiante en la asignatura.

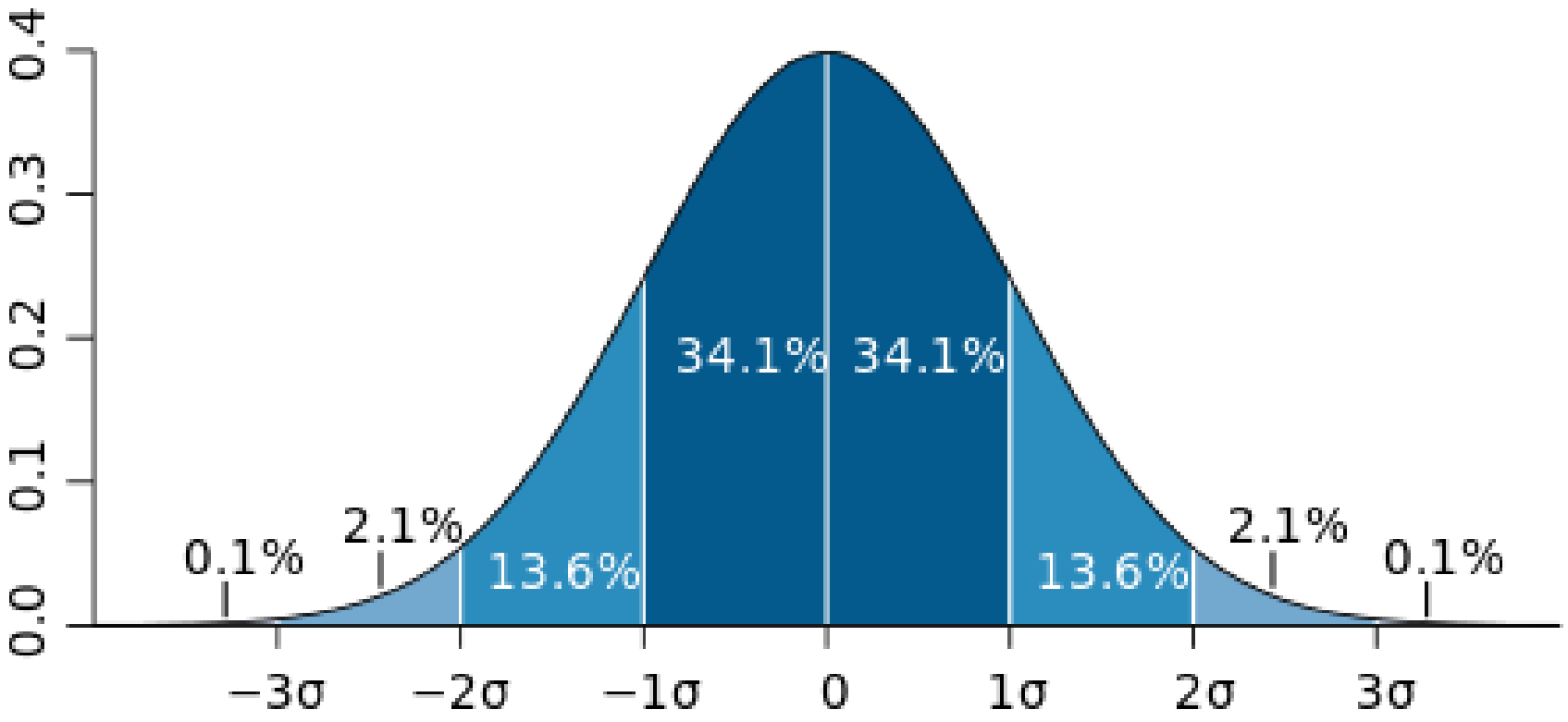
- Tu objetivo es escribir un programa en R que realice las siguientes tareas:

1. Crear un data frame llamado 'datos_estudiantes' con los siguientes datos:
    1. Nombre: 'Juan', 'María', 'Pedro', 'Laura'
    2. Edad: 20, 21, 19, 22
    3. Asignatura: 'Matemáticas', 'Historia', 'Ciencias', 'Inglés'
    4. Calificación: 85, 76, 92, 88
2. Imprimir el data frame 'datos_estudiantes' completo.
3. Calcular el promedio de calificaciones de los estudiantes.
4. Filtrar el data frame para mostrar únicamente los estudiantes mayores de 20 años.
5. Imprimir el data frame resultante después de aplicar los filtros y la ordenación.

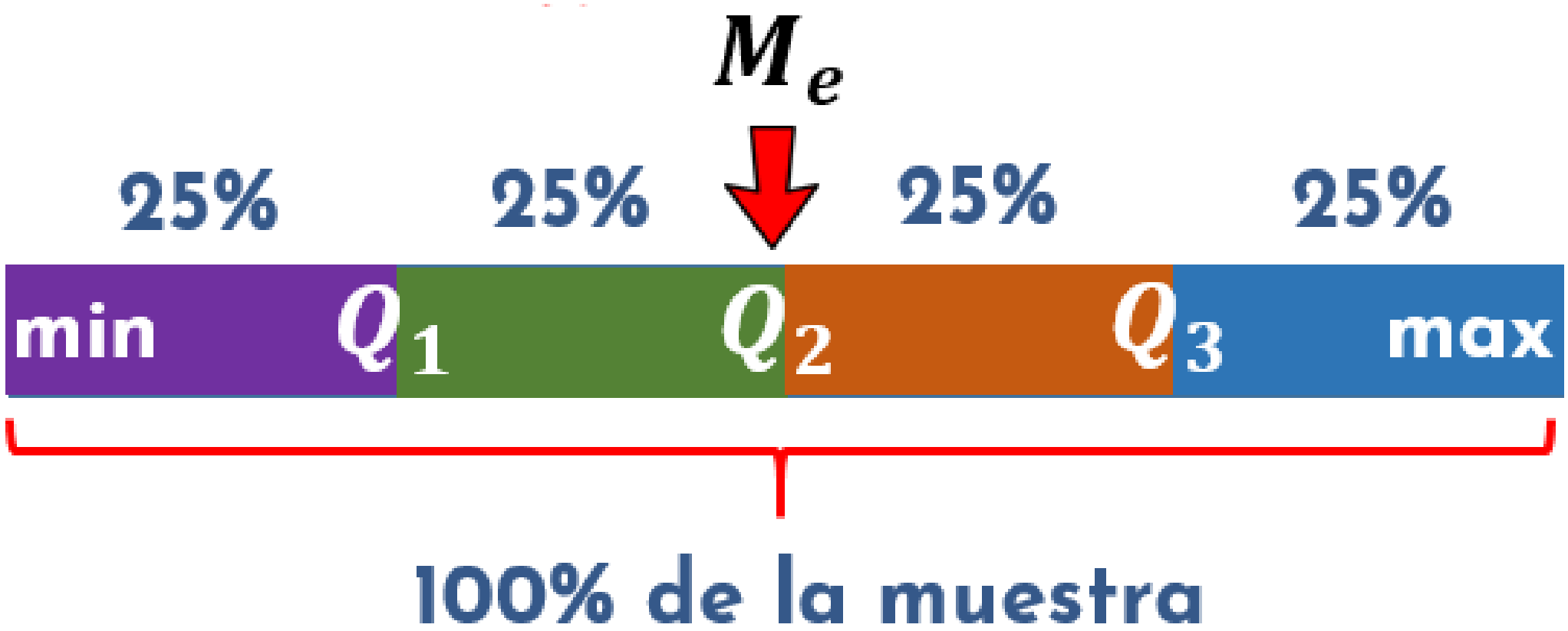
MEDIA


8 14 9 12 14 12 15 11 12 8 10 14 12 9 12
172 / 15 = 11,5
Suma total de las edades
Cantidad de datos
Media

DESVIACIÓN TÍPICA



CUARTILES



FUNCIONES

Una función tiene un nombre, argumentos y un cuerpo. Las funciones definidas por el usuario son creadas usando la sigiente estructura.

```
nombre <-
function(argumentos) {
        operaciones
}
```

EJERCICIO

Se te pide escribir una función en R que tome como argumento una lista de números y devuelva la suma de los números pares de la lista.

Escribe la función para calcular la suma de los números pares y luego utilízala para calcular la suma de los números pares de una lista ingresada por el usuario y mostrar el resultado.

CONDICIONAL IF

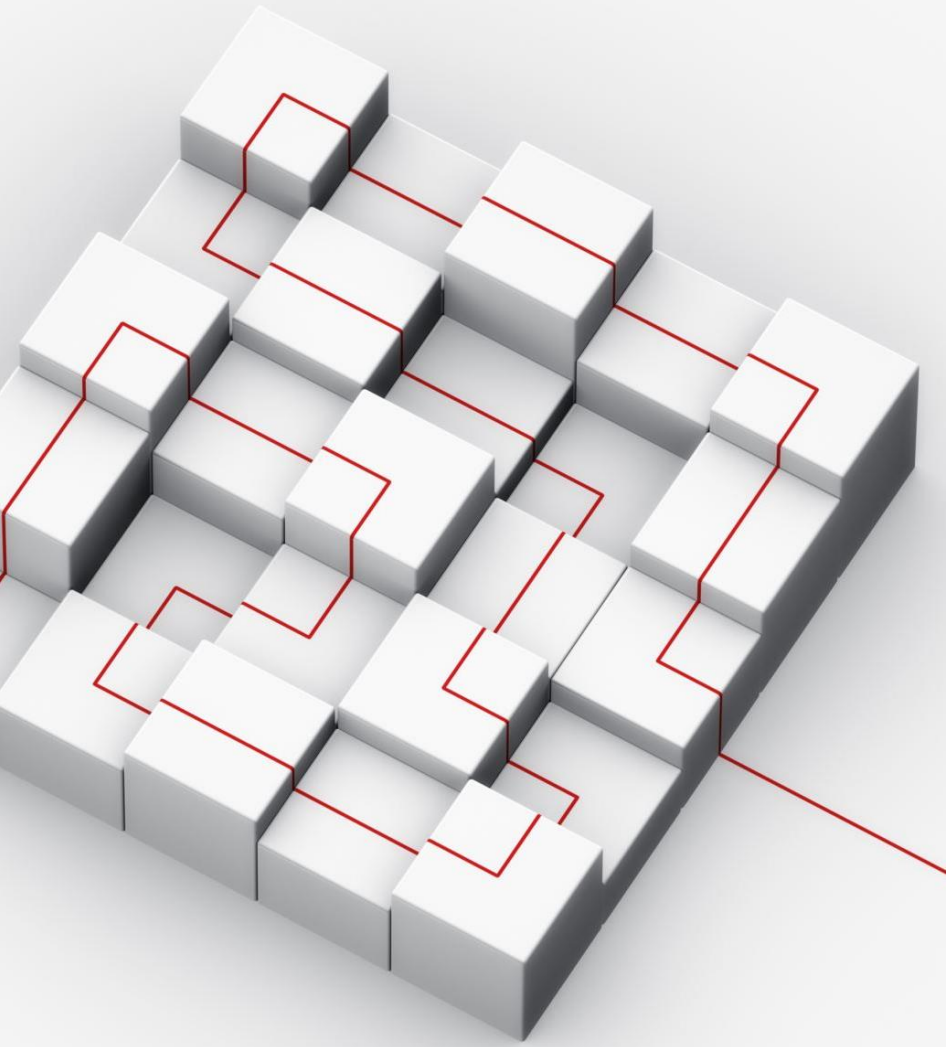
- if (si) es usado cuando deseamos que una operación se ejecute únicamente cuando una condición se cumple.

- else (de otro modo) es usado para indicarle a R qué hacer en caso de la condición de un if no se cumpla.

EJERCICIO

Se te pide escribir una función en R que tome como argumento un número entero y calcule su factorial. El factorial de un número entero positivo n se define como el producto de todos los números enteros desde 1 hasta n.

Tu función debe devolver el factorial del número ingresado. Luego, debes utilizar esta función para calcular el factorial de un número ingresado por el usuario y mostrar el resultado.

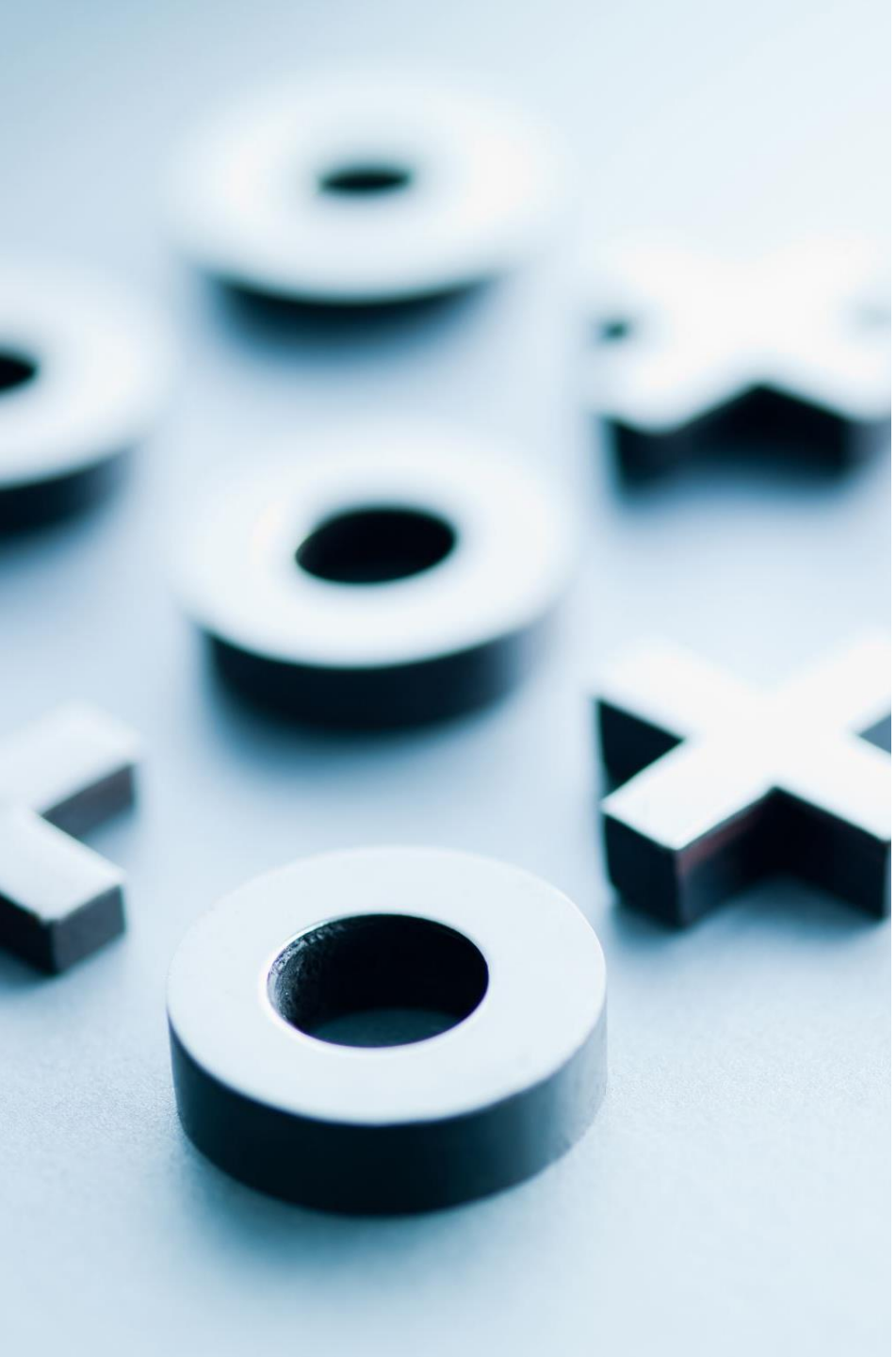


FOR

La estructura for nos permite ejecutar un bucle (loop), realizando una operación para cada elemento de un conjunto de datos.

Su estructura es la siguiente:

```
for(elemento in objeto){
  operacion_con_elemento
}
```



EJERCICIO

Se te pide escribir un programa en R que utilice un bucle for para imprimir los números del 1 al 10 junto con sus cuadrados. El programa debe mostrar los resultados de la siguiente manera:

1 - 1

2 - 4

3 - 9

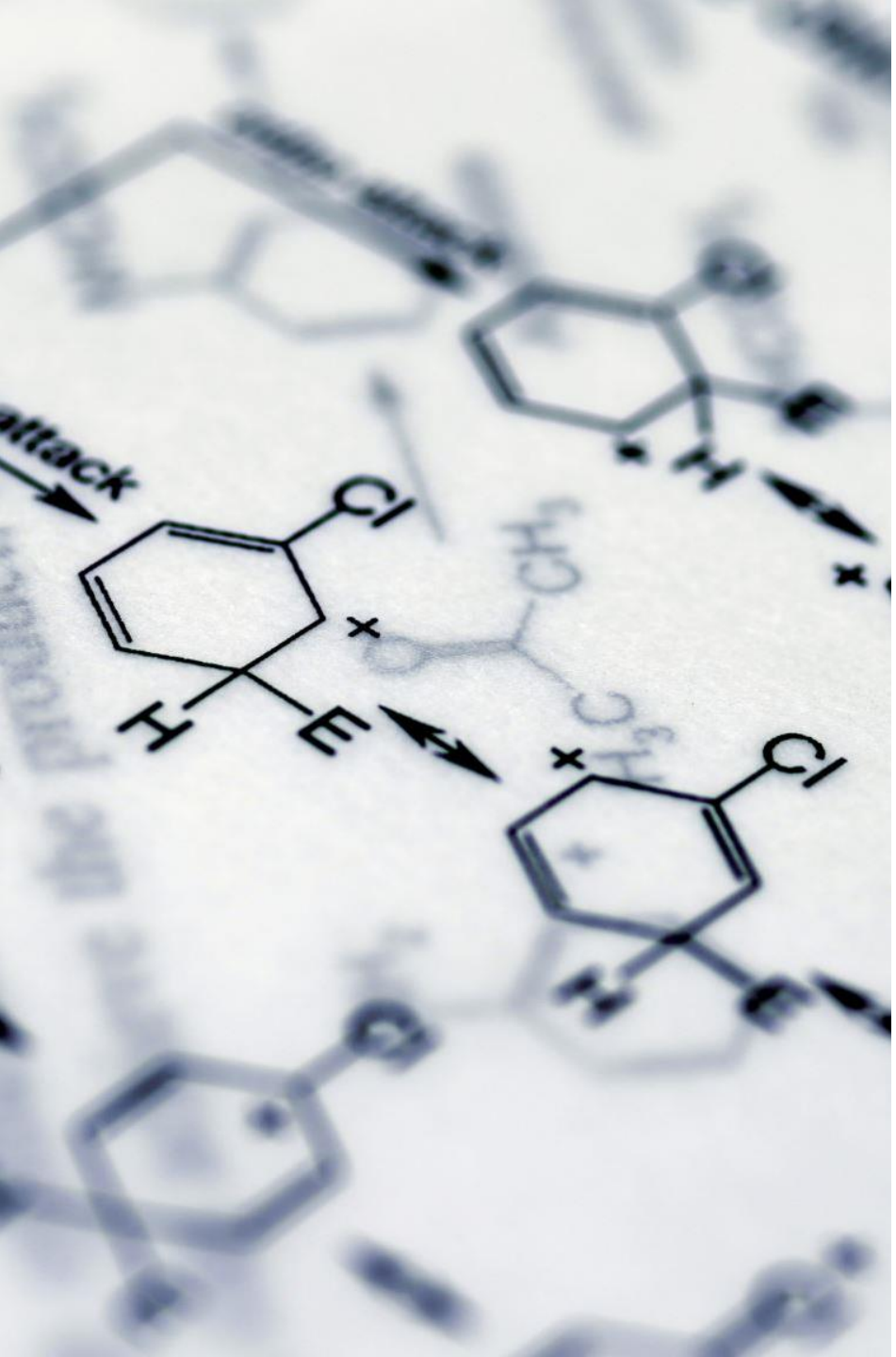
...

10 - 100

EJERCICIO

- Se te pide escribir un programa en R que calcule la suma de los elementos de una matriz numérica de tamaño 3x3. El programa debe utilizar bucles for anidados para recorrer la matriz y sumar todos los elementos.

- Escribe el programa que realice esta tarea y muestre el resultado de la suma.



WHILE

Este es un tipo de bucle que ocurre mientras una condición es verdadera (TRUE). La operación se realiza hasta que se se llega a cumplir un criterio previamente establecido.

El modelo de while es:

```
while(condicion){
  operaciones
}
```

EJERCICIO

Se te pide escribir un programa en R que utilice un bucle while para calcular la suma de los números pares del 1 al 100. El programa debe seguir sumando los números pares hasta que la suma alcance un valor mayor o igual a 1000. Luego, debe mostrar la suma obtenida.

Escribe el programa que realice esta tarea y muestre el resultado de la suma.

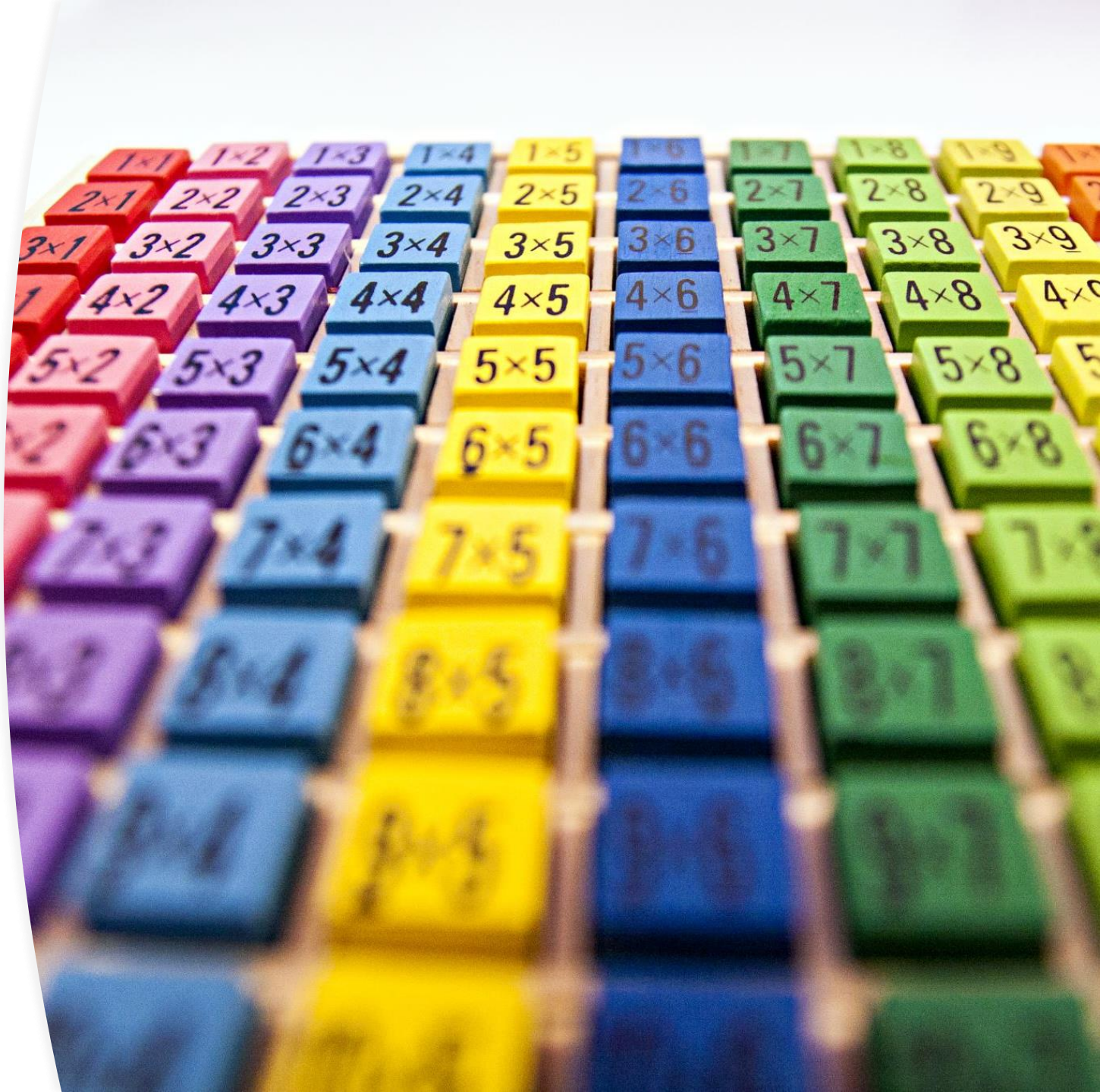
BREAK Y NEXT

- break y next son palabras reservadas en R, no podemos asignarles nuevos valores y realizan una operación específica cuando aparecen en nuestro código.

- break nos permite interrumpir un bucle, mientras que next nos deja avanzar a la siguiente iteración del bucle, “saltándose” la actual. Ambas funcionan para for y while.

APPLY

La familia de funciones apply es usada para aplicar una función a cada elemento de una estructura de datos. En particular, es usada para aplicar funciones en matrices, data frames, arrays y listas.Todas las funciones de esta familia tienen una característica en común: reciben como argumentos a un objeto y al menos una función.



EJERCICIO

- Se te pide escribir un programa en R que utilice la función apply() para calcular el promedio de cada columna de una matriz numérica de tamaño 4x3. El programa debe mostrar los promedios de cada columna.

- Escribe el programa que realice esta tarea y muestre los promedios obtenidos.

DESCARGAR

La función download.file() nos pide como argumento url, la dirección de internet del archivo que queremos descargar y destfile el nbombre que tendrá el archivo en nuestra computadora. Ambos argumentos como cadenas de texto, es decir, entre comillas.

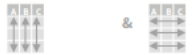
CSV Y TABLAS

- Un caso particular de las tablas, son los archivos separados por comas, con extensión **.csv**, por *Comma Separated Values*, sus siglas en inglés. Este es un tipo de archivo comunmente usado para compartir datos, pues es compatible con una amplia variedad de sistemas diferentes además de que ocupa relativamente poco espacio de almacenamiento.

DPLYR

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:



Each **variable** is in its own **column**

&

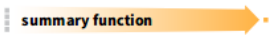
Each **observation**, or **case**, is in its own **row**

pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).



summarise(.data, …)
Compute table of summaries. Also **summarise_**().
summarise(mtcars, avg = mean(mpg))

count(x, …, wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

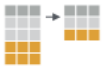
ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

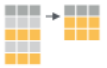
EXTRACT CASES

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

filter(.data, …) Extract rows that meet logical criteria. Also **filter_()**. *filter(iris, Sepal.Length > 7)*

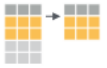


distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. Also **distinct_()**. *distinct(iris, Species)*



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*



slice(.data, …) Select rows by position. Also **slice_()**. *slice(iris, 10:15)*

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

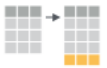
<	<=	is.na()	%in%	| xor()
>	>=	!is.na()	!	&

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

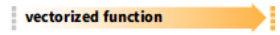
select(.data, …)
Extract columns by name. Also **select_if()**
select(iris, Sepal.Length, Species)

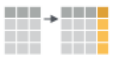
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match)	**num_range**(prefix, range)	**:**, e.g. mpg:cyl
ends_with(match)	**one_of**(…)	**-**, e.g, -Species
matches(match)	**starts_with**(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).





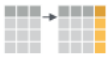
mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**.
mutate_all(faithful, funs(log(.), log2(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

mutate_if(.tbl, .predicate, .funs, …)
Apply funs to all columns of one type.
Use with **funs()**.
mutate_if(iris, is.numeric, funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s).
add_column(mtcars, new = 1:32)

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

EJERCICIO

Se te proporciona un conjunto de datos que contiene información sobre las ventas diarias de productos en diferentes sucursales de una tienda.

```
ventas <- data.frame(

  Sucursal = c("Sucursal A", "Sucursal B", "Sucursal A", "Sucursal B", "Sucursal A"),

  Fecha = c("2023-05-01", "2023-05-01", "2023-05-02", "2023-05-02", "2023-05-03"),

  Ventas = c(100, 150, 200, 120, 180)

)
```

Tu tarea es utilizar el paquete dplyr para realizar las siguientes acciones:

Cargar el conjunto de datos y examinar su estructura. Filtrar el conjunto de datos para incluir solo las ventas de la sucursal "Sucursal A". Agrupar los datos por fecha. Calcular el total de ventas para cada fecha. Calcular el promedio de ventas diarias. Ordenar los resultados por fecha en orden ascendente. Mostrar el resultado obtenido.

EJERCICIO

Se te proporciona un conjunto de datos que contiene información sobre los alumnos de una escuela. El conjunto de datos incluye las siguientes columnas: 'Nombre', 'Edad', 'Género' y 'Promedio'.

```
alumnos <- data.frame(
  Nombre = c("María", "Ana", "Laura", "Sofía", "Lucía"),
  Edad = c(18, 17, 19, 18, 17),
  Género = c("Femenino", "Femenino", "Femenino", "Femenino", "Femenino"),
  Promedio = c(8.5, 9.2, 7.8, 8.9, 9.1)
)
```

Tu tarea es utilizar el paquete dplyr para realizar las siguientes acciones:

Cargar el conjunto de datos y examinar su estructura. Filtrar el conjunto de datos para incluir solo a los alumnos de género femenino. Calcular la edad promedio de las alumnas. Calcular el promedio de notas de las alumnas. Ordenar los resultados por edad en orden descendente. Mostrar el resultado obtenido.