UCA Universidad de Cádiz

Vicerrectorado de Profesorado

Unidad de Formación e Innovación Docente

Edificio Hospital Real
Plaza Falla, 8 | 11003 Cádiz
Tel. 956 015 728
http://www.uca.es

MEMORIA FINAL[1]
Compromisos y Resultados
Proyectos de Innovación y Mejora Docente
2023/2024

Identificación del proyecto	
Código	**sol-202300256950-tra**
Título	**Aplicaciones de Maching Learning en Matlab y Python para la Ingeniería Civil e Ingeniería Industrial**
Responsable	**Ignacio José Turias Domínguez**

1. Describa los resultados obtenidos a la luz de los objetivos y compromisos que adquirió en la solicitud de su proyecto. Incluya tantas tablas como objetivos contempló.

Objetivo nº **1**	*Introducción al aprendizaje automático en los entornos de MATLAB y Phyton e identificación de casos de uso en su campo de estudio.*
Actividades que había previsto en la solicitud del proyecto:	*Introducción a la programación en MATLAB y Python: Introducción a los conceptos básicos de programación en ambos lenguajes, seguida de una presentación de las herramientas específicas de machine learning disponibles en cada uno de ellos.*
Actividades realizadas y resultados obtenidos:	• *Se ha realizado un curso completo de iniciación al ML en Matlab y Python, con numerosos ejemplos.* • *Se han realizado diversas sesiones de trabajo con estudiantes interesados después de la presentación del proyecto en las clases de las asignaturas implicadas en el proyecto.* • *Fomento de la participación de los estudiantes en competencias y desafíos de programación y ML.*
Objetivo nº 2	*Aplicar técnicas de aprendizaje automático en el aula mediante el uso de herramientas como MATLAB y Python*
Actividades que había previsto en la solicitud del proyecto:	*Implementación de algoritmos de machine learning: Ejercicios y prácticas en el aula en los que los estudiantes implementen algoritmos de aprendizaje supervisado y no supervisado en MATLAB y Python, aplicando estos conceptos a la resolución de problemas concretos en el ámbito de la ingeniería.*
Actividades realizadas y resultados obtenidos:	• *Se han desarrollado reuniones de trabajo con alumnos interesados tanto de grado como de máster y en paralelo se han desarrollado un curso OCW entre los profesores integrantes del equipo del proyecto*

[1] **Esta memoria no debe superar las 6 páginas.**

Objetivo nº 3	*Aplicación del aprendizaje automático a problemas específicos de su campo estudio.*
Actividades que había previsto en la solicitud del proyecto:	*Análisis de datos: Prácticas en las que los estudiantes trabajen con conjuntos de datos reales y aprendan a analizarlos y visualizarlos utilizando herramientas como MATLAB y Python.* *Proyecto final: Los estudiantes podrían llevar a cabo un proyecto final en el que apliquen los conocimientos adquiridos en el curso para resolver un problema concreto de ingeniería utilizando técnicas de machine learning y las herramientas proporcionadas.*
Actividades realizadas y resultados obtenidos:	• *Los estudiantes fueron capaces de programar pequeñas aplicaciones de análisis de datos: lectura de datos, prepocesamiento, uso de algún modelo de regresión o clasificación y presentación de resultados.* • *Varios estudiantes de grado van a solicitar ser alumnos colaboradores este curso* • *Una alumna ha solicitado ayuda al plan propio para la realización del TFM en esta temática.*

2. Realice una breve valoración sobre la influencia del proyecto ejecutado en la evolución de las asignaturas implicadas.

Análisis del impacto de la innovación en las asignaturas relacionadas con el proyecto
El desarrollo del proyecto ha permitido una mejora significativa en la asimilación de los conceptos en la asignatura de *Fundamentos de Informática* al menos entre los alumnos interesados. Igualmente, se ha hecho más visible que los conceptos aprendidos en *Fundamentos de Informática* pueden ser altamente transferibles e integrables en otras materias, por tanto, hacer visible que son de mucha utilidad. En conjunto con la ciencia de datos, los estudiantes son capaces de resolver problemas complejos mediante un enfoque más interdisciplinar y práctico, dotándolos de una visión más completa y coherente de los retos a los que se enfrentan en el mundo de hoy en día donde la transformación digital y la industria 4.0 son conceptos diarios. El proyecto, al incorporar dos lenguajes como MATLAB y Python, permite una aproximación tangible a la Industria 4.0, lo que enriquece las competencias de los estudiantes en ingeniería. Las herramientas que se utilizan abren nuevas perspectivas, brindando soluciones innovadoras que se alejan de los enfoques convencionales, lo cual no solo mejora la calidad de la formación, sino que también prepara a los estudiantes para afrontar desafíos reales en entornos industriales avanzados. En definitiva, esta ejecución conecta diferentes materias con las tendencias tecnológicas más actuales, haciendo que el aprendizaje sea más relevante y dinámico.

3. Incluya en la siguiente tabla el número de alumnos matriculados y el de respuestas recibidas en cada opción y realice una valoración crítica sobre la influencia que el proyecto ha ejercido en la opinión de los alumnos.

Opinión de los alumnos al inicio del proyecto
Número de alumnos matriculados: 200 (aprox.) / Encuestados: 23

*Valoración del grado de dificultad **que cree que va a tener** en la comprensión de los contenidos y/o en la adquisición de competencias asociadas a la asignatura en la que se enmarca el proyecto de innovación docente*				
Ninguna dificultad	Poca dificultad	Dificultad media	Bastante dificultad	Mucha dificultad
		x		
Opinión de los alumnos en la etapa final del proyecto				
*Valoración del grado de dificultad **que ha tenido** en la comprensión de los contenidos y/o en la adquisición de competencias asociadas a la asignatura en la que se enmarca el proyecto de innovación docente*				
Ninguna dificultad	Poca dificultad	Dificultad media	Bastante dificultad	Mucha dificultad
		x		
Los elementos de innovación y mejora docente aplicados en esta asignatura han favorecido mi comprensión de los contenidos y/o la adquisición de competencias asociadas a la asignatura				
Nada de acuerdo	Poco de acuerdo	Ni en acuerdo ni en desacuerdo	Muy de acuerdo	Completamente de acuerdo
			x	
En el caso de la participación de un profesor invitado				
La participación del profesor invitado ha supuesto un gran beneficio en mi formación				
Nada de acuerdo	Poco de acuerdo	Ni en acuerdo ni en desacuerdo	Muy de acuerdo	Completamente de acuerdo

Valoración crítica sobre la influencia que ha ejercido el proyecto en la opinión de los alumnos

El desarrollo del proyecto de innovación docente planteado ha conseguido que los alumnos afronten los conocimientos adquiridos en la asignatura de Fundamentos de Informática desde una perspectiva más abierta. Análogamente, también ha permitido un cambio de enfoque de los retos presentados en las restantes materias de ingeniería previstas en el proyecto. De hecho, uno de los aspectos más valorados ha sido el entendimiento de que muchas de las problemáticas planteadas en diferentes materias pueden ser abordadas de manera similar usando programación y análisis de datos.

La combinación de dos herramientas como MATLAB y Python que permiten explotar la ciencia de datos ha sido vista como un avance significativo por los estudiantes. La posibilidad de aplicar técnicas de análisis de datos en ambos lenguajes y en los distintos ámbitos de las materias de ingeniería ha permitido a los alumnos desarrollar una visión más completa y multidisciplinaria. Los estudiantes han expresado que el acceso a estas herramientas no solo les proporciona competencias técnicas valiosas, sino que también mejora su capacidad para tomar decisiones basadas en datos.

Los estudiantes interesados valoran positivamente el proyecto y todo lo que sea actualización en el currículo. Esta modernización ha generado un mayor interés y motivación en los alumnos, alguno de los cuales desean ser alumnos colaboradores y otros hacer el TFG y TFM con los profesores integrantes del equipo del proyecto.

4. Describa las medidas de difusión a las que se comprometió en la solicitud y las que ha llevado a cabo[2].

Descripción de las medidas comprometidas en la solicitud
Jornada workshop de presentación de los resultados del proyecto de innovación. Escuela Técnica Superior de Ingeniería de Algeciras. junio de 2024. Programa: 1. presentación del proyecto y de sus objetivos 2. presentación de casos de uso/éxito de machine learning en distintas asignaturas 3. presentación de resultados
Descripción de las medidas que se han llevado a cabo
• Jornada workshop de presentación del proyecto de innovación en asignaturas implicadas • Jornada workshop de presentación de los resultados del proyecto de innovación. Escuela Técnica Superior de Ingeniería de Algeciras. septiembre de 2024. Programa: 1. presentación del proyecto y de sus objetivos 2. presentación de casos de uso/éxito de machine learning en distintas asignaturas 3. presentación de resultados

[2] **Si en la solicitud no indicó compromiso de difusión de resultados este criterio no se tendrá en cuenta en la evaluación**

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

Curso OCW Aplicaciones de *Machine Learning* en Python y MATLAB para la Ingeniería Civil e Ingeniería Industrial

Índice

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

```
print("Los ejercicios de Python se encuentran en la carpeta Ejercicios Python. Esta carpeta ya contiene los archivos CSV o Excel necesarios para poder funcionar.")
disp("Los ejercicios de MATLAB se encuentran en la carpeta Ejercicios MATLAB. Esta carpeta ya contiene los archivos CSV o Excel necesarios para poder funcionar.");
```

NOTA: Los ejercicios **Ejercicio XX_01.** corresponden a Python y los **Ejercicio XX_02.** a MATLAB.

```
print("En Python: ")

    #  Se usa # para introducir comentarios en Python (en Spyder).
    #  Celdas de código o Markdown en Jupyter. Es recomendable Jupyter ya que permite
       trabajar con los códigos viendo directamente las salidas.
    #  Funciona con celdas, las cuales se pueden ir ejecutando individualmente o en
       secuencia

disp("En MATLAB: ");

    %  Al iniciar cualquier código en MATLAB desde cero, suelen usarse los comandos:
       clc; (limpia el command Windows), clear all; (limpia todas las variables del
       dataspace), close all; (cierra todas las figuras abiertas). No se deben borrar
       las variables de los scripts si van a seguir usándose, pues dejarán de existir
       y aparecerá un error en el command window.
    %  Se usa % para introducir comentarios en MATLAB.
    %  Si se desea que se muestre en pantalla alguna variable basta con eliminar el
       signo ; de esta variable.
    %  Se usa help en command Windows para solicitar ayuda de cualquier comando que
       se necesite.
    %  Los operadores se indican en la siguiente tabla.

disp("Operadores matemáticos: ");
```

OPERADORES	PYTHON	MATLAB
Suma	A + B	A + B
Resta	A – B	A – B
Multiplicación	A * B	A * B
División	A / B	A / B
Parte entera de la división de A entre B	A // B	fix(A/B)
A elevado a B	A ** B	A^B o power(A, B)
Operador lógico Y	A & B	A & B
Operador lógico O	A | B	A | B
Comparación A y B	A == B	A == B
A distinto de B	A != B	A ~= B
Menor o igual	A <= B	A <= B
Menor	A < B	A < B
Mayor o igual	A >= B	A >= B
Mayor	A >B	A >B

- **NOTA:** Los datos obtenidos con funciones *random* serán diferentes cada vez que se saquen los resultados en cada programa.

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

Usando MATLAB® y Python® juntos:

- **Tipos de datos en ambos lenguajes.**

PYTHON	MATLAB
float	double, single
complex	complex single, complex double
int	(u)int8, (u)int16,(u)int32,(u)int64
float(nan)	NaN
float(inf)	Inf
str	String, char
bool	Logical
dict	Structure
array.array()	Vectors
list, tuple	Cell array

- **Algunos tipos de datos en MATLAB deben ser convertidos.**

MATLAB	Conversion Function
categorical	char
string	char
table	table2struct
timetable	timetable2struct
datetime	char

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

Módulo 1. Introducción al aprendizaje automático en los entornos de MATLAB y Python

1.1. Salida por pantalla

Ejercicio 01_01. Salida por pantalla en Python.

```
print("Hola Mundo")
```

```
Hola Mundo
```

Ejercicio 01_02. Salida por pantalla en MATLAB.

```
disp("Ejemplo 001- Salida por pantalla en MATLAB");
disp("Hola Mundo");
```

```
Ejemplo 001- Salida por pantalla en MATLAB
Hola Mundo
```

Ejercicio 02_01. Salida por pantalla en Python con salto de línea \n.

```
print("salida", "de", "datos", "en", "Python", sep="\n") # si no se indica sep="\n"
se usa espacio en blanco como separador.
```

```
salida
de
datos
en
Python
```

Ejercicio 02_02. Salida por pantalla en MATLAB con salto de línea \n.

```
s1 = sprintf("Salida\nde\ndatos\nen\nMATLAB");
disp(s1);
```

```
Salida
de
datos
en
MATLAB
```

Ejercicio 03_01. Mostrar la suma de dos números en un mensaje por pantalla.

```
print("El resultado de sumar 2+2 es", 2+2)
```

```
El resultado de sumar 2+2 es 4
```

Ejercicio 03_02. Mostrar la suma de dos números en un mensaje por pantalla.

```
sprintf("El resultado de sumar 2+2 es %d",2+2);
```

```
"El resultado de sumar 2+2 es 4"
```

Ejercicio 04_01. Mostrar la multiplicación de dos números en un mensaje por pantalla.

```
print("El resultado de multiplicar 2x3 es", 2*3)
```

```
El resultado de multiplicar 2x3 es 6
```

Ejercicio 04_02. Mostrar la multiplicación de dos números en un mensaje por pantalla.

```
sprintf('El resultado de multiplicar 2x3 es %d.',2*3);
```

```
El resultado de multiplicar 2x3 es 6.
```

Ejercicio 05_01. Construir el mensaje de salida por pantalla usando valores de diferentes tipos de variables.

```
a = 1.5
print("El valor de la variable a es %f" %(a))
```

```
El valor de la variable a es 1.500000
```

Ejercicio 05_02. Construir el mensaje de salida por pantalla usando valores de diferentes tipos de variables.

```
a = 1.5 % valor float
b = 890
sprintf("El valor de la variable a es %f", a) % muestra a con todos los decimales por
defecto
sprintf("El valor de la variable a es %5.2f", a) % 5 es la parte entera y 2 los
decimales
sprintf("El valor de la variable b es %4.2f",b) % aunque b es entero, lo mostrará con
2 decimales
```

```
a = 1.5000
b = 890
ans = "El valor de la variable a es 1.500000"
ans = "El valor de la variable a es  1.50"
ans = "El valor de la variable b es 890.00"
```

Ejercicio 06_01. Construir el mensaje de salida por pantalla usando más de una variable.

```
a = 1.5
b = 2.5
print("El valor de la variable a es %d y el de b es %f" %(a,b))
```

```
El valor de la variable a es 1 y el de b es 2.500000
```

Ejercicio 06_02. Construir el mensaje de salida por pantalla usando más de una variable.

```
c = 1.5
d = 890
```

```
sprintf("El valor de la variable c es %d y de la variable d es %f", c, d)
sprintf("El valor de la variable c es %4.2d y de la variable d es %5.2f", c, d)

c = 1.5000
d = 890
ans = "El valor de la variable c es 1.500000e+00 y de la variable d es 890.000000"
ans = "El valor de la variable c es 1.50e+00 y de la variable d es 890.00"
```

1.2. Tipos de datos y asignación de variables

Ejercicio 07_01. Construir el mensaje de salida por pantalla usando una cadena o *string* en Python.

```
cadena = "hola mundo"
print(cadena) # Imprime la variable cadena
print(type(cadena)) # Nos muestra el tipo de dato

hola mundo
<class 'str'>
```

Ejercicio 07_02. Construir el mensaje de salida por pantalla usando una cadena o *string* en MATLAB.

```
cadena = 'hola mundo'
sprintf('%s', 'hola mundo')
cadena = "hola mundo"
sprintf('%s', cadena)

cadena = "hola mundo"
ans = 'hola mundo'
```

Ejercicio 08_01. Dadas unas variables indicar su tipo por un mensaje en pantalla.

```
a = 10
print("El valor del número es: %d" %(a))
print(type(a))
b = 11.5
print("El valor del número con decimales es: %f" %(b))
print(type(b))

El valor del número es: 10
<class 'int'>
El valor del número con decimales es: 11.500000
<class 'float'>
```

Ejercicio 08_02. Dadas unas variables indicar su tipo por un mensaje en pantalla.

```
a = 10
b = 11.5
c = 'hello'

sprintf("El valor del número a es: %d", a)
tipo_de_variable = class(a); % Obtener el tipo de variable de a
```

```
disp(['El tipo de variable de a es: ', tipo_de_variable]);
sprintf("El valor del número b es: %f",b)
tipo_de_variable = class(b); % Obtener el tipo de variable de b
disp(['El tipo de variable de b es: ', tipo_de_variable]);
sprintf("El valor de c es: %s",c)
tipo_de_variable = class(c); % Obtener el tipo de variable de c
disp(['El tipo de variable de c es: ', tipo_de_variable]);

a = 10
b = 11.5000
c = 'hello'
ans = "El valor del número a es: 10"
El tipo de variable de a es: double
ans = "El valor del número b es: 11.500000"
El tipo de variable de b es: double
ans = "El valor de c es: hello"
El tipo de variable de c es: char
```

Ejercicio 09_01. Reasignación de variables.

```
nuevo_numero = b
print("El valor del nuevo número es: %d" %(nuevo_numero))

El valor del nuevo número es: 11.5000
```

Ejercicio 09_02. Reasignación de variables.

```
b = 11.5
nuevo_numero = b
sprintf("El valor del número es: %5.2d", nuevo_numero)
sprintf("El valor del número es: %s", nuevo_numero)

b = 11.5000
nuevo_numero = 11.5000
ans = "El valor del número es: 1.15e+01"
ans = "El valor del número es: 1.150000e+01"
```

Ejercicio 10_01. Asignación múltiple de variables en Python.

```
a, b, c = 1, 2, 3
print(a, b, c)

1, 2, 3
```

Ejercicio 10_02. Asignación múltiple de variables en MATLAB.

```
[a, b, c] = deal(1, 2, 3);
fprintf('%d, %d, %d\n', a, b, c);

1, 2, 3
```

Ejercicio 11_01. Asignación de no valor.

```
a = None
print(a)
```

```
None
```

Ejercicio 11_02. Asignación de no valor.

```
x = NaN; % Asigna NaN a la variable x
disp(x);
```

```
NaN
```

```
% Además se puede asignar no valor a vectores:
A = [1, 2, NaN, 4]; % Crea una matriz con NaN como elemento
y = []; % Asigna un valor vacío a la variable y
B = [1, 2, [], 4]; % Crea una matriz con un valor vacío en el medio
```

```
A = 1     2   NaN     4
```

Ejercicio 12_01. Tipos de variables *booleanas*.

```
a = True # Deben empezar con mayúsculas
print(a)
b = False # Deben empezar con mayúsculas
print(b)
```

```
True
False
```

Ejercicio 12_02. Tipos de variables *booleanas*.

```
a = true % En MATLAB la palabra reservada true es en minúsculas
b = false % En MATLAB la palabra reservada false es en minúsculas
str_a = num2str(a); % Convierte el valor booleano en una cadena
str_b = num2str(b);
fprintf("El valor booleano de a es %s\n", str_a); % Muestra 1
fprintf("El valor booleano de b es %s\n", str_b);

if a
    str_a = 'True'; % Si a es verdadero, asigna 'true' a str_a
else
    str_a = 'False'; % Si a es falso, asigna 'false' a str_a
end
if b
    str_b = 'True'; % Si b es verdadero, asigna 'true' a str_b
else
    str_b = 'False'; % Si b es falso, asigna 'false' a str_b
end
fprintf("El valor booleano de a es %s\n", str_a); % Muestra la cadena 'True'
fprintf("El valor booleano de b es %s\n", str_b); % Muestra la cadena 'False'
```

```
a = logical
   1

b = logical
   0
El valor booleano de a es 1
El valor booleano de b es 0
El valor booleano de a es True
El valor booleano de b es False
```

Ejercicio 13_01. Todas las variables son objetos. Podemos comprobar el tipo de objeto de una variable.

```
a = 5
isinstance(a,int)  # Devuelve True

True
```

Ejercicio 13_02. Todas las variables son objetos. Podemos comprobar el tipo de objeto de una variable.

```
variable = 42; % Define una variable de ejemplo
tipo_de_objeto = class(variable); % Comprueba el tipo de objeto de la variable
fprintf('La variable es de tipo: %s\n', tipo_de_objeto); % Muestra el resultado

La variable es de tipo: double
```

1.3. Entrada de datos.

Ejercicio 14_01. Leer por teclado datos usando el comando input(). Este comando devuelve una cadena de caracteres.

```
# Lee un primer número entero
numero1 = input("Por favor inserte el primer número entero: ")
# Lee un segundo número con decimales
numero2 = input("Por favor inserte el segundo número con decimales: ")
#  En este punto tanto numero1 como numero2 son string
#  numero1 será entero, así que usamos int()
numero1 = int(numero1)
#  numero2 será un real, así que usamos float()
numero2 = float(numero2)
# mostramos el resultado de la suma
print("La suma de %d + %f es: %f" %(numero1,numero2, numero1 + numero2))

Por favor inserte el primer número entero: 1
Por favor inserte el segundo número con decimales: 2.4
La suma de 1 + 2.400000 es: 3.400000

# Podemos hacer la suma directamente
# Leemos un tercer número entero
numero3 = int(input("Por favor inserte el tercer número entero: "))
# Leemos un cuarto número con decimales
numero4 = float(input("Por favor inserte el cuarto número con decimales: "))
```

```
# Mostramos el resultado de la suma
print("La suma de %d + %f es: %f" %(numero3,numero4, numero3 + numero4))

Por favor inserte el tercer número entero: 1
Por favor inserte el cuarto número con decimales: 1.44
La suma de 1 + 1.400000 es: 2.440000
```

Ejercicio 14_02. Leer por teclado datos usando el comando input(). Este comando devuelve una cadena de caracteres.

```
num1 = input("Introduzca número entero: ")
num2 = input("Introduzca número decimal: ")
sprintf('El resultado de sumar %d+%f es: %f',num1+num2);
resultado = num1 + num2; % Calcular la suma
% sprintf para formatear la cadena
cadena_formateada = sprintf('El resultado de sumar %d + %f es: %f', num1, num2,
resultado);
disp(cadena_formateada);

Introduzca número entero: 4
num1 = 4
Introduzca número decimal: 2.3
num2 = 2.3000
El resultado de sumar 4 + 2.300000 es: 6.300000
```

1.4. Cadenas.

Ejercicio 15_01. Definir una cadena.

```
cad = "Hola"
cad2 = 'Hola2'
print(cad,cad2)

Hola
Hola2
```

Ejercicio 15_02. Definir una cadena.

```
cad = 'Hola';
cad2 = 'Hola2';
fprintf('%s %s\n', cad, cad2);

Hola Hola2
```

Ejercicio 16_01. Insertar valores de variables en una cadena.

```
variable = "Luis"
print("Hola %s" %variable)
# Otra forma
cad = f"Hola {variable}"
print(cad)
# Otra forma
```

```
print("Hola {}".format(variable))

Hola Luis
Hola Luis
Hola Luis
```

Ejercicio 16_02. Insertar valores de variables en una cadena.

```
variable = 'Luis';
fprintf('Hola %s\n', variable);

Hola Luis
```

Ejercicio 17_01. Concatenar cadenas con +

```
cad = "Esto"+ " " + "es" + " " + "una" + " " + "prueba"
print(cad)

# En caso de ser variables de tipo cadena se hace igual
nombre = "María"
apellidos = "García"
nombre_completo = nombre + " " + apellidos
print("El nombre completo es %s." %nombre_completo)

Esto es una prueba
El nombre completo es María García
```

Ejercicio 17_02. Insertar valores de variables en una cadena.

```
cad = 'Esto es una prueba';
disp(cad);
nombre = 'María';
apellidos = 'García';
nombre_completo = [nombre ' ' apellidos];
fprintf('El nombre completo es %s.\n', nombre_completo);

Esto es una prueba
El nombre completo es María García.
```

Ejercicio 18_01. En Python las cadenas son *arrays* de caracteres con el índice comenzando en 0.

```
# Mostramos los elementos de la cadena
palabra = "Hola"
print(palabra[0])
print(palabra[1])
print(palabra[2])
print(palabra[3])

H
o
l
a
```

Ejercicio 18_02. En MATLAB Las cadenas son *arrays* de caracteres con el índice comenzando en 1.

```
palabra = 'Hola';
fprintf('%c\n', palabra(1));
fprintf('%c\n', palabra(2));
fprintf('%c\n', palabra(3));
fprintf('%c\n', palabra(4));

H
o
l
a
```

Ejercicio 19_01. Podemos convertir otros tipos de datos a cadena.

```
a = 24
cad = "El número es " + str(a)
print(cad)

El número es 24
```

Ejercicio 19_02. Podemos convertir otros tipos de datos a cadena.

```
a = 23;
cad = sprintf('El número es %d', a);
disp(cad);

El número es 23
```

1.5. Control de flujo con *if-then-else*.

Ejercicio 20_01. Podemos convertir otros tipos de datos a cadena.

```
# Importante los dos puntos: y la identación con un tabulador en cada caso.
# Se debe mantener la identación (No hay llaves como en otros lenguajes)
# En Python no existen las sentencias tipo switch

# Leemos desde teclado, con lo que hay que convertir a entero desde string
entrada = int(input("¿Cuánto es 5 + 5? "))
if entrada == 10:
print("Respuesta correcta.")
else:
print("Error en la propuesta.")

¿Cuánto es 5 + 5? 1
Error en la respuesta
```

Ejercicio 20_02. Podemos convertir otros tipos de datos a cadena.

```
% Leemos desde el teclado, convertimos la entrada a entero desde string
```

```
entrada = input('¿Cuánto es 5 + 5?: ');
% Verificamos si la entrada es igual a 10
if entrada == 10
    disp('Respuesta correcta.');
else
    disp('Error en la propuesta.');
end

¿Cuánto es 5 + 5?: 3
Error en la propuesta.
```

Ejercicio 21_01. Podemos convertir otros tipos de datos a cadena.

```
# Obtenemos el dato por teclado con input. Se debe convertir a entero
opcion = int(input("Inserte una opción: "))
if opcion == 1:
print("Ha elegido la opción 1")
elif opcion == 2:
print("Ha elegido la opción 2")
elif opcion == 3:
print("Ha elegido la opción 3")
else:
print("Ha elegido otra opción distinta a la 1, la 2 y la 3")

Inserte una opción: 1
Ha elegido la opción 1
```

Ejercicio 21_02. Podemos convertir otros tipos de datos a cadena.

```
% Obtenemos el dato por teclado con input. Se debe convertir a entero
opcion = input('Inserte una opción: ');
% Verificamos el valor de la opción
if opcion == 1
    disp('Ha elegido la opción 1');
elseif opcion == 2
    disp('Ha elegido la opción 2');
elseif opcion == 3
    disp('Ha elegido la opción 3');
else
    disp('Ha elegido otra opción distinta a la 1, la 2 y la 3');
end

Inserte una opción: 5
Ha elegido otra opción distinta a la 1, la 2 y la 3
```

Ejercicio 22_01. Usamos condiciones más complejas.

```
# Ejemplo con or
# Obtenemos el dato por teclado con input. Se debe convertir a entero
opcion = int(input("Inserte una opción: "))
if opcion == 1 or opcion == 2:
    print("Ha elegido la opción 1 o la opción 2")
else:
```

```
    print("Ha elegido otra opción distinta a la 1 y a la 2")

Inserte una opción: 1
Ha elegido la opción 1 o la opción 2

# Ejemplo con and
# Obtenemos el dato por teclado con input. Se debe convertir a entero
opcion = int(input("Inserte una opción: "))
valor = int(input('Inserte valor: '))
if opcion == 1 and valor == 10:
    print("Ha elegido la opción 1 y el valor es 10")
else:
    print("Ha elegido otra opción distinta a la 1 o el valor no era 10")
Inserte una opción: 1
Inserte valor: 10
Ha elegido la opción 1 y el valor es 10
```

Ejercicio 22_02. Usamos condiciones más complejas.

```
% Ejemplo con or
% Obtenemos el dato por teclado con input. Se debe convertir a entero
opcion = input('Inserte una opción: ');
% Verificamos si la opción es igual a 1 o igual a 2
if opcion == 1 || opcion == 2
    disp('Ha elegido la opción 1 o la opción 2');
else
    disp('Ha elegido otra opción distinta a la 1 y a la 2');
end

Inserte una opción: 1
Ha elegido la opción 1 o la opción 2

% Ejemplo con and
% Obtenemos el dato por teclado con input. Se debe convertir a entero
opcion = input('Inserte una opción: ');
valor = input('Inserte valor: ');
% Verificamos si la opción es igual a 1 y el valor es igual a 10
if opcion == 1 && valor == 10
    disp('Ha elegido la opción 1 y valor 10');
else
    disp('Ha elegido otra opción distinta a la 1 o valor no era 10');
end

Inserte una opción: 4
Inserte valor: 10
Ha elegido otra opción distinta a la 1 o valor no era 10
```

Ejercicio 23_01. Usamos condiciones más complejas.

```
# Ejemplo con not
condicion = False
if not condicion: # es verdad si condición es falsa
    print("La condición es falsa")
```

```
else:
    print("La condición es verdadera")

La condición es falsa
```

Ejercicio 23_02. Usamos condiciones más complejas.

```
% Ejemplo con not
condicion = false;
if ~condicion  % ~ es el operador 'not' en MATLAB. Se valida si condición es falsa
    disp('La condición es falsa');
else
    disp('La condición es verdadera');
end

La condición es falsa
```

1.6. Control de flujo con *if-then-else* 2.

Ejercicio 24_01. Usamos condiciones más complejas.

```
# El operador in permite verificar si un elemento está en un objeto
# Permite verificar si un elemento está en un objeto
palabra = "hola"
if "a" in palabra:
     print("%s contiene la letra a" %palabra)
else:
    print("%s no contiene la letra a" %palabra)

hola contiene la letra a

# Python otra forma
palabra = "hola"
if "f" not in palabra:
     print("{} no contiene la letra f".format(palabra))
else:
     print("{} si contiene la letra f".format(palabra))

hola no contiene la letra f
```

Ejercicio 24_02. Usamos condiciones más complejas.

```
% Permite verificar si un elemento está en un objeto
palabra = 'hola';
if any(palabra == 'a')
    fprintf('%s contiene la letra a\n', palabra);
else
    fprintf('%s no contiene la letra a\n', palabra);
end

hola contiene la letra a
```

Ejercicio 25_01. Anidar condiciones

```
# Podemos anidar condiciones:
x,y,z, = 5,10,5
if x > y:
     print("x es mayor que y")
elif x <= y:
       if x == z:
           print("x es igual a y")
       elif x != z:
           print("x no es igual a z")

x es igual a z
```

Ejercicio 25_02. Anidar condiciones

```
x = 5;
y = 10;
z = 5;
if x > y
    disp('x es mayor que y');
elseif x <= y
    if x == z
        disp('x es igual a z');
    elseif x ~= z
        disp('x no es igual a z');
    end
end

x es igual a z
```

1.7. Bucles *for*

Son bucles muy útiles cuando se conoce el número de repeticiones que van a producirse

Python:

```
# for contador in range(valor_inicial, valor_final, tamaño_de_paso)
```

Instrucciones: Bucle for

Matlab:

Bucle for para repetir un número determinado de veces

```
% for index = values
      Statements
  end
```

Instrucciones: Bucle for Matlab

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

Ejercicio 26_01. Bucles *for* en Python

```
x = range(1,5) # range(inicio, final). El final no se incluye
print("Valores del 1 al 4: ")
for i in x:
  print(i)

Valores del 1 al 4:
1
2
3
4

print("Valores del 4 al 9, de dos en dos: ")
for j in range(4,10,2): # secuencia del 4 al 9, de 2 en 2
   print(j)

Valores del 4 al 9, de dos en dos:
4
6
8
```

Ejercicio 26_02. Bucles *for* en MATLAB

```
x = 1:4; % Crear un vector de 1 a 4 (inclusive)

fprintf('Valores del 1 al 4:\n');
for i = x
    fprintf('%d\n', i);
end

Valores del 1 al 4:
1
2
3
4

% Valores del 4 al 9, de dos en dos:
fprintf('Valores del 4 al 9, de dos en dos:\n');
for j = 4:2:9
    fprintf('%d\n', j);
end

Valores del 4 al 9, de dos en dos:
4
6
8
```

Ejercicio 27_01. Bucles *for* con *break* en Python

```
# ejemplo de uso de break
print("Secuencia del 1 al 9. Cuando llega a valer 5, salimos del bucle")
for k in range(1,10):
```

```
   if k == 5:
       print("El valor es 5. Salimos del bucle")
       break
   else:
       print(k)

Secuencia del 1 al 9. Cuando llega a valer 5, salimos del bucle:
1
2
3
4
El valor es 5. Salimos del bucle
```

Ejercicio 27_02. Bucles *for* con *break* en MATLAB

```
% Ejemplo de uso de break
fprintf('Secuencia del 1 al 9. Cuando llega a valer 5, salimos del bucle:\n');

for k = 1:9
    if k == 5
        fprintf('El valor es 5. Salimos del bucle\n');
        break;
    else
        fprintf('%d\n', k);
    end
end

Secuencia del 1 al 9. Cuando llega a valer 5, salimos del bucle:
1
2
3
4
El valor es 5. Salimos del bucle
```

Ejercicio 28_01. Bucles *for* con *continue* en Python

```
for num in range(5): # va de 0 a 4
   if num == 3:
          continue # si num vale 3, se pasa a la siguiente iteración sin ejecutar el
código siguiente que haya dentro del for
          print(num)

0
1
2
4
```

Ejercicio 28_02. Bucles *for* con *continue* en MATLAB

```
for num = 0:4
    if num == 3
```

```
        continue; % Si num es igual a 3, pasa a la siguiente iteración sin ejecutar
el código siguiente en el bucle
    end
    fprintf('%d\n', num);
end
0
1
2
4
```

Ejercicio 29_01. Bucles *for* anidados en Python

```
# bucles for anidados:
for i in range(1,5): # bucle exterior
    for j in range(1,3): # bucle interior
       print("Valor de i: %d - Valor de j: %d." %(i,j))

Valor de i: 1 - Valor de j: 1.
Valor de i: 1 - Valor de j: 2.
Valor de i: 2 - Valor de j: 1.
Valor de i: 2 - Valor de j: 2.
Valor de i: 3 - Valor de j: 1.
Valor de i: 3 - Valor de j: 2.
Valor de i: 4 - Valor de j: 1.
Valor de i: 4 - Valor de j: 2.
```

Ejercicio 29_02. Bucles *for* anidados en MATLAB

```
% Bucles for anidados:
for i = 1:4 % Bucle exterior
    for j = 1:2 % Bucle interior
        fprintf('Valor de i: %d - Valor de j: %d.\n', i, j);
    end
end

Valor de i: 1 - Valor de j: 1.
Valor de i: 1 - Valor de j: 2.
Valor de i: 2 - Valor de j: 1.
Valor de i: 2 - Valor de j: 2.
Valor de i: 3 - Valor de j: 1.
Valor de i: 3 - Valor de j: 2.
Valor de i: 4 - Valor de j: 1.
Valor de i: 4 - Valor de j: 2.
```

1.8. Bucles *while*

Los bucles *while* se suelen usar cuando no conocemos el número de iteraciones al usarse condiciones:

Python:

```
# While(condición)
```

En Python, este tipo de bucle admite *break* y *continue* (no recomendado).

Instrucciones: While Loops

Matlab:

Bucle while para repetir cuando la condición es verdadera

```
% while expression
      Statements
  end
```

Instrucciones: Bucle while Matlab

Ejercicio 30_01. Bucles *while* en Python

```
valor = int(input("Inserte valor: "))
while valor > 5:
      print("Dentro del bucle")
      valor = int(input("Inserte valor: "))
print("Fuera del bucle")
```

```
Inserte valor: 6
Dentro del bucle
Inserte valor: 3
Fuera del bucle
```

Ejercicio 30_02. Bucles *while* en MATLAB

```
valor = input('Inserte valor: ');  % Solicitar valor al usuario
while valor > 5
    disp('Dentro del bucle');
    valor = input('Inserte valor: ');  % Solicitar valor al usuario nuevamente
end
disp('Fuera del bucle');
```

```
Inserte valor: 6
Dentro del bucle
Inserte valor: 3
Fuera del bucle
```

Ejercicio 31_01. Bucles *while* en Python admiten *break* y *continue* (no recomentado)

```
# Los bucles while admiten break y continue (no recomendado)
y = 1
print("Imprimimos los valores del 1 al 10 pero salimos cuando llegamos al 5")
while y < 10:
      print(y) # Importante indentación en todo el código dentro del bucle
      if y == 5:
            print("Hemos llegado al valor 5")
      break
      y = y + 1
```

```
Imprimimos los valores del 1 al 10 pero salimos cuando llegamos al 5
1
2
3
4
5
Hemos llegado al valor 5
```

Ejercicio 31_02. Bucles *while* en MATLAB admiten *break* y *continue* (No recomendados)

```
y = 1;
fprintf('Imprimimos los valores del 1 al 10 pero salimos cuando llegamos al 5\n');
while y < 10
    fprintf('%d\n', y);
    if y == 5
        fprintf('Hemos llegado al valor 5\n');
        break;
    end
    y = y + 1;
end
```

```
Imprimimos los valores del 1 al 10 pero salimos cuando llegamos al 5
1
2
3
4
5
Hemos llegado al valor 5
```

1.9. Listas o estructuras de datos

Python. Listas (vectores):

En otros lenguajes se conocen como vectores. Las listas admiten cambios en sus elementos. También se pueden añadir nuevos elementos o eliminar elementos.

Instrucciones: Python Lists

- **En MATLAB las estructuras de datos** que se asemejan a las listas en Python son las celdas (*cells*) y las matrices (*arrays*). Sin embargo, hay algunas diferencias clave entre las listas de Python y estas estructuras de datos de MATLAB.

  Celdas (*Cells*):
  - Las celdas en MATLAB son estructuras de datos que pueden contener datos de diferentes tipos y tamaños.
  - Cada elemento de una celda puede ser una matriz, una cadena, o incluso otra celda, permitiendo una mayor flexibilidad.
  - Puedes acceder a los elementos de una celda utilizando llaves {}.

```
% Ejemplo de celdas en MATLAB
```

```
ciudades = {'Jerez', 'Puerto Real', 'Cádiz'};
contaminantes = {'CO', 'NO_{2}', 'SO_{2}', 'Ozono'};
datos = {ciudades, contaminantes};
```

Ejercicio 32_01. Declaración de una lista en Python

```
# declaración de una lista
lista1 = []
lista2 = list()
# lista con valores. El primer elemento está en la posición 0
lista3 = [1,2, True]
print("Lista 3")
print(lista3)
print("Elemento cero de la lista 3 es: %d" % lista3[0])
# otra forma de declarar una lista con valores
lista4 = list([4,9, False, 'valor 1'])
print("Los contenidos de la lista4 son: ")
print(lista4)
```

```
Lista 3
     [1, 2, True]
Elemento cero de la lista 3 es: 1
Los contenidos de la lista4 son:
    [4, 9, False, 'valor 1']
```

Ejercicio 32_02. Declaración de una lista en MATLAB

```
% Declaración de un vector vacío
lista1 = [];
% Declaración de un vector vacío de otra forma
lista2 = [];
% Declaración de un vector con valores
lista3 = [1, 2, true];
fprintf('Lista 3\n');
disp(lista3);
fprintf('Elemento cero de la lista 3 es: %d\n', lista3(1));  % En MATLAB, los índices
comienzan en 1
% Otra forma de declarar un vector con valores
% Crear una celda con diferentes tipos de datos
lista4 = {4, 9, false, 'valor 1'};
% Mostrar los contenidos de la celda
fprintf('Los contenidos de la lista4 son: \n');
disp(lista4);
```

```
Lista 3
     1     2     1
Elemento cero de la lista 3 es: 1
Los contenidos de la lista4 son:
    {[4]}    {[9]}    {[0]}    {'valor 1'}
```

Ejercicio 33_01. Modificar un elemento de una lista en Python

```
lista4[3] = "valor 2"
print(lista4)
```

```
[4, 9, False, 'valor 2']
```

Ejercicio 33_02. Modificar un elemento de una lista en MATLAB

```
% Modificar un elemento del vector
lista4{3} = 'valor 2';
fprintf('La lista4 después de la modificación es:\n');
disp(lista4);
```

```
La lista4 después de la modificación es:
    {[4]}    {[9]}    {'valor 2'}    {'valor 1'}
```

Ejercicio 34_01. Mostrar los contenidos de una lista en Python

```
# Mostrar los contenidos de una lista
print("Primera forma de mostrar el contenido de una lista: ")
for elemento in lista4:
      print(elemento)
print("Segunda forma de mostrar el contenido de una lista: ")
# Mostrar contenido de una lista forma 2 recorriendo los índices
# len nos devuelve el tamaño de la lista
for i in range(len(lista4)):
      print(lista4[i])
```

```
Primera forma de mostrar el contenido de una lista:
4
9
False
valor 1
Segunda forma de mostrar el contenido de una lista:
4
9
False
valor 1
```

Ejercicio 34_02. Mostrar los contenidos de una lista en MATLAB

```
% Mostrar los contenidos de un vector
fprintf('Primera forma de mostrar el contenido de una lista:\n');
for elemento = lista4
    disp(elemento);
end
fprintf('Segunda forma de mostrar el contenido de una lista:\n');
% Mostrar contenido de un vector forma 2 recorriendo los índices
% numel nos devuelve el tamaño del vector
for i = 1:numel(lista4) % o también for i = 1:length(lista4)
    disp(lista4{i});
end
```

```
Primera forma de mostrar el contenido de un vector:
    {[4]}
    {[9]}
    {'valor 2'}
    {'valor 1'}
Segunda forma de mostrar el contenido de un vector:
      4
      9
valor 2
valor 1
```

Ejercicio 35_01. Añadir elementos a una lista en Python

```
# Añadimos nuevo elemento en la lista4 al final
lista4.append(7)
#añadir un nuevo elemento en la lista en una posición concreta
lista4.insert(0, "nuevo") # se inserta en la posición 0
print(lista4)

['nuevo', 4, 9, False, 'valor 2', 7]
```

Ejercicio 35_02. Añadir elementos a una lista en MATLAB

```
% Añadir un nuevo elemento al final de la lista
lista4{end + 1} = 7;
% Añadir un nuevo elemento en una posición concreta (en este caso, al principio)
nuevoElemento = 'nuevo';
lista4 = [nuevoElemento, lista4];
fprintf('La lista 4 después de añadir elementos es:\n');
disp(lista4);

La lista 4 después de añadir elementos es:
    {'nuevo'}    {[4]}    {[9]}    {'valor 2'}    {'valor 1'}    {[7]}
```

Ejercicio 36_01. Eliminar elementos de una lista según su posición en Python

```
lista4.pop(1) # Eliminamos el elemento de la posición 1
print("Lista 4 con el elemento de la posición 1 eliminado: ")
print(lista4)

Lista 4 con el elemento de la posición 1 eliminado:
    ['nuevo', 9, False, 'valor 2', 7]
```

Ejercicio 36_02. Eliminar elementos de una lista según su posición en MATLAB

```
posicion = 1; % Posición del elemento a eliminar
lista4(posicion) = []; % Eliminar el elemento de la posición 1
```

```
% Mostrar la celda después de eliminar el elemento
fprintf('La lista4 con el elemento de la posición %d eliminado:\n', posicion);
disp(lista4);

La lista4 con el elemento de la posición 1 eliminado:
    {[4]}    {[9]}    {'valor 2'}    {'valor 1'}    {[7]}
```

Ejercicio 37_01. Eliminar elementos de una lista según su valor en Python

```
try:
       lista4.remove(9)
except:
   print("Ese elemento no está en la lista")
 print(lista4)

['nuevo', False, 'valor 2', 7]
```

Ejercicio 37_02. Eliminar elementos de una lista según su valor en MATLAB

```
% Valor que deseas eliminar
valor_a_eliminar = 9;
% Encontrar índices de elementos iguales al valor a eliminar
indices = find(cellfun(@(x) isequal(x, valor_a_eliminar), lista4));
% Verificar si se encontraron elementos para eliminar
if ~isempty(indices)
    % Encontrados, eliminar los elementos en los índices encontrados
    lista4(indices) = [];
else
    % No encontrados, mostrar mensaje
    disp('Ese elemento no está en la lista');
end
% Mostrar el cell array resultante
disp(lista4);

    {[4]}    {'valor 2'}    {'valor 1'}    {[7]}
```

1.10. Matrices

Python. Listas (matrices):

En este caso tenemos listas con filas y columnas.

Instrucciones: Matrices en Python

- **Matrices (*Arrays*):**

- En MATLAB, las matrices son estructuras de datos bidimensionales que pueden contener números, cadenas de texto u otros tipos de datos.
- Puedes acceder a los elementos de una matriz utilizando índices, y las operaciones entre matrices son operaciones de matriz estándar.
- La longitud de una matriz en una dimensión dada es fija (cuadrada (nxn), rectangular (nxm),… etc) y debe especificarse al crearla.

```
% Ejemplo de matriz en MATLAB
A = [1, 2, 3; 4, 5, 6; 7, 8, 9]; % Definición de matriz 3x3
```

Ambas estructuras, tanto **matrices** (*arrays*) como **celdas** (*cells*) en MATLAB pueden cumplir funciones similares a las listas de Python, dependiendo de los requisitos específicos de tu código. Las celdas son más flexibles porque pueden contener diferentes tipos de datos y tamaños, mientras que las matrices están más orientadas a operaciones numéricas y requieren que las dimensiones sean especificadas.

Ejercicio 38_01. Definir una matriz en Python

```
# Definimos una matriz de dimensión 3x4
A = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
print("Mostramos la matriz A que acabamos de definir: ")
print(A)
# Recorremos la matriz A y mostramos los elementos
filas = len(A)
col = len(A[0])
for i in range(filas):
   for j in range(col):
       print(A[i][j])
```

```
Mostramos la matriz A que acabamos de definir:
[[1,2,3,4],[5,6,7,8],[9,10,11,12]]
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12
```

Ejercicio 38_02. Definir una matriz en MATLAB

```
% Definimos una matriz de dimensiones 3x4
A = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12];
% Mostramos la matriz A que acabamos de definir
disp('Mostramos la matriz A que acabamos de definir:');
disp(A);
% Recorremos la matriz A y mostramos los elementos uno a uno
[filas, col] = size(A);
```

```
for i = 1:filas
    for j = 1:col
        disp(A(i, j));
    end
end

Mostramos la matriz A que acabamos de definir:
     1     2     3     4
     5     6     7     8
     9    10    11    12

     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
    11
    12
```

1.11. Diccionarios y *cell arrays*

Python. Diccionarios:

Los diccionarios son estructuras de datos en las que se mapean entradas con valores (claves-valor). No están ordenados. Las claves son inmutables y no pueden estar duplicadas en el diccionario. Podemos introducir listas como elementos de un diccionario.

Instrucciones: Diccionarios Python

- **MATLAB no es un lenguaje de programación que utilice diccionarios como en Python. En su lugar, puedes utilizar celdas o *cell arrays***

Ejercicio 39_01. Declaración de diccionarios en Python

```python
# En los diccionarios se mapean entradas con valores (clases-valor).
# Declaración de diccionarios:
# Declaramos un diccionario vacío:
diccionario1 = {}
# Definimos un diccionario con datos de clave - valor
miDiccionario = {"a":1,"b":2,"c":3}
```

Ejercicio 39_02. Declaración de *cell arrays* en MATLAB

```
% En MATLAB, se utilizan las estructuras para realizar tareas similares a los
diccionarios en Python.
```

```
% Declaramos una estructura vacía:
estructura1 = struct();
% Definimos una estructura con datos de campo - valor
miEstructura.a = 1;
miEstructura.b = 2;
miEstructura.c = 3;
```

Ejercicio 40_01. Acceso a los datos de un diccionario en Python

```
# Acceso a los datos de un diccionario mediante la clave:
print("Valor de a: %d" %miDiccionario["a"])

Valor de a: 1
```

Ejercicio 40_02. Acceso a los datos de un *cell array* en MATLAB

```
% Acceso a los datos de una estructura mediante el campo:
fprintf('Valor de a: %d\n', miEstructura.a);

Valor de a: 1
```

Ejercicio 41_01. Modificar un valor de un elemento del diccionario en Python

```
miDiccionario["a"] = 5
print("Valor de a tras el cambio: %d" % miDiccionario["a"])

Valor de a tras el cambio: 5
```

Ejercicio 41_02. Modificar un valor de un elemento del *cell array* en MATLAB

```
% Modificar un valor de un campo de la estructura:
miEstructura.a = 5;
fprintf('Valor de a tras el cambio: %d\n', miEstructura.a);

Valor de a tras el cambio: 5
```

Ejercicio 42_01. Acceso a un diccionario en Python

```
# Podemos acceder mediante el método get:
print(miDiccionario.get("b")) # muestra el valor para la clave b

2
```

Ejercicio 42_02. Acceso a un *cell array* en MATLAB

```
% Podemos acceder a un campo utilizando un enfoque similar a través del operador de
punto:
fprintf('%d\n', miEstructura.b);  % muestra el valor para el campo b

2
```

Ejercicio 43_01. Número de elementos del diccionario en Python

```
# Número de elementos de un diccionario:
print(len(miDiccionario))
```

```
3
```

Ejercicio 43_02. Número de elementos del *cell array* en MATLAB

```
% Número de elementos de una estructura:
fprintf('Número de elementos de una estructura: %d\n',
length(fieldnames(miEstructura)));
```

```
Número de elementos de una estructura: 3
```

Ejercicio 44_01. Mostrar contenido del diccionario en Python

```
# Mostrar el contenido de un diccionario:
# Importante %s para strings
print("Imprimimos todas las claves: %s" %miDiccionario.keys())
print("Imprimimos todas las claves: %s" %miDiccionario.values())
print("Imprimimos las parejas clave-valor del diccionario: ")
for i in miDiccionario:
   print(i, "-", miDiccionario[i])
```

```
Imprimimos todas las claves: dict_keys(['a', 'b', 'c'])
Imprimimos todos los valores: dict_values([5, 2, 3])
Imprimimos las parejas clave-valor del diccionario:
a - 5
b - 2
c - 3
```

Ejercicio 44_02. Mostrar contenido del *cell array* en MATLAB

```
% Mostrar el contenido de una estructura:
% Importante usar %s para strings
fprintf('Imprimimos todas las claves: %s\n', strjoin(fieldnames(miEstructura),', '));
values = struct2cell(miEstructura);
fprintf('Imprimimos todos los valores: %s\n', strjoin(string(values),', '));
fprintf('Imprimimos las parejas clave-valor de la estructura: \n');
fields = fieldnames(miEstructura);
for i = 1:length(fields)
    fprintf('%s - %d\n', fields{i}, miEstructura.(fields{i}));
end
```

```
Imprimimos todas las claves: a, b, c
Imprimimos todos los valores: 5, 2, 3
Imprimimos las parejas clave-valor de la estructura:
a - 5
b - 2
c - 3
```

Ejercicio 45_01. Borrar un elemento del diccionario en Python

```
# Borra un elemento por su clave:
del miDiccionario["a"]
print("Imprimimos todas las claves: %s" %miDiccionario.keys())
print("Imprimimos todos los valores: %s" %miDiccionario.values())
```

```
Imprimimos todas las claves: dict_keys(['b', 'c')]
Imprimimos todos los valores: dict_values[(2, 3)]
```

Ejercicio 45_02. Borrar un elemento del *cell array* en MATLAB

```
% Eliminar un campo por su nombre:
miEstructura = rmfield(miEstructura, 'a');
fprintf('Imprimimos todas las claves: %s\n', strjoin(fieldnames(miEstructura),', '));
values = struct2cell(miEstructura);
fprintf('Imprimimos todos los valores: %s\n', strjoin(string(values), ', '));
```

```
Imprimimos todas las claves: b, c
Imprimimos todos los valores: 2, 3
```

1.12. Diccionarios con listas como elementos

Ejercicio 46_01. Incluir listas como elementos de un diccionario en Python

```
# Definimos las listas
cuidades = ["Jerez", "Puerto Real", "Cádiz"]
conta =  ["CO", "NO2", "SO2", "Ozono"]
# Definimos el diccionario
datos = {"estaciones": ciudades, "contaminantes": conta}
print(datos["estaciones"][0]) # mostramos la primera estación
print(datos["contaminantes"][3]) # mostramos el cuarto contaminante
```

```
Jerez
Ozono
```

```
# Mostramos todos los elementos de la lista asociados a una clave:
datos["estaciones"]
```

```
['Jerez', 'Puerto Real', 'Cádiz']
```

```
datos["estaciones"][0]) = "Algeciras" # mostramos la primera estación
datos["estaciones "]
```

```
['Algeciras', 'Puerto Real', 'Cádiz']
```

Ejercicio 46_02. Incluir listas como elementos de un *cell array* en MATLAB

```
% Podemos incluir celdas como elementos de una estructura en MATLAB:
% Definimos las celdas
ciudades = {'Jerez', 'Puerto Real', 'Cádiz'};
```

```
contaminantes = {'CO', 'NO_{2}', 'SO_{2}', 'Ozono'};
% Definimos la estructura
datos = {ciudades, contaminantes}
% Mostramos la primera estación y el cuarto contaminante
disp(datos{1}{1});
disp(datos{2}{4});

Jerez
Ozono

% Mostramos todos los elementos de la lista asociados a una clave
datos{:} % muestra todos a la vez

datos{1}
  1×3 cell array
   {'Jerez'}    {'Puerto Real'}    {'Cádiz'}
datos{2}
  1×4 cell array
    {'CO'}    {'NO_{2}'}    {'SO_{2}'}    {'Ozono'}

% Mostramos la primera estación
fprintf('%s\n', datos{1}{1});
% Mostramos el cuarto contaminante
fprintf('%s\n', datos{2}{4});
% Mostramos todos los elementos de la celda asociados a una clave:
disp(datos.estaciones);
```

1.13. Tuplas o vectores

Python. Tuplas:

Las tuplas son colecciones de elementos que son de sólo lectura (listas o vectores inmutables).

Instrucciones: Tuplas Python

- **En Matlab las tuplas equivalen a vectores o matrices de una fila.**

Ejercicio 47_01. Definir una tupla en Python

```
# Las Tuplas son vectores inmutables:
a = (1,2,3)
print("El contenido de la tupla es: ")
print(a)
print("Recorremos la tupla elemento a elemento: ")
for i in range(len(a)):
  print(a[i], sep="\n")

El contenido de la tupla es:
     1     2     3
Recorremos la tupla elemento a elemento:
     1
```

```
     2
     3
```

Ejercicio 47_02. Definir un tupla como un vector en MATLAB

```
% Las tuplas no son directamente comparables a las estructuras de MATLAB.
% Sin embargo, podrían simularse con matrices de una fila, es decir, vectores.
a = [1, 2, 3];
disp('El contenido del vector es: ');
disp(a);
disp('Recorremos el vector elemento a elemento: ');
for i = 1:length(a)
    disp(a(i));
end

El contenido del vector es:
     1     2     3
Recorremos el vector elemento a elemento:
     1
     2
     3
```

1.14. Funciones

Python. Definición de funciones:

def nombreFuncion(parametros)

sentencias

return [valor1], [valor2]

Invocar funciones: *nombreFuncion(argumentos)*

Instrucciones: Funciones Python

- **En Matlab las funciones se definen:**

function nombreFunción (parámetros)

sentencias

end

Ejercicio 48_01. Funciones en Python

```
# Definición de función:
# Función suma
def sumar(x,y):
  return x + y
# Llamada a una función:
# Probamos la función suma
```

```
resultado = sumar(1,3)
print("El resultado de sumar 1 y 3 es: %d" %resultado)

El resultado de sumar 1 y 3 es: 4

# Funciones sin return:
def mensaje(valor, texto):
       if(valor == 1):
              print("Mensaje con valor 1: ", texto)
              return # Cortamos la ejecución si valor = 1
       print("Mensaje con otro valor: ", texto)

 mensaje(1, "El valor es 1.")
 mensaje(2, "El valor es 2.")

Mensaje con valor 1: El valor es 1.
Mensaje con otro valor: El valor es 2.


# Lista como parámetro de una función:
# Creamos las listas
num1 = [3, 4, 5, 4]
num2 = [1, 2, 3]
# Definimos la función
def cuadrados(x):
   for valor in x:
        print(valor**2)
# Llamamos a la función
cuadrados(num1)
cuadrados(num2)

    9
    16
    25
    16

     1
     4
     9

# Funciones con parámetros por defecto y dos parámetros de salida:
# Función que calcula el área y volumen de una esfera
# Definimos la función con el valor de pi por defecto
def calcularAreaVol(r, pi=3.14):
       a = 4*pi*r**2
       v = 4/3 * pi*r**3 # Es el operador para elevar ^
       return a, v
# Llamamos a la función pasándole sólo el radio
area, vol = calcularAreaVol(2)
print("El área es %f y el volumen %f" %(area, vol))

El área es 50.240000 y el volumen 33.493333
```

```
# Si recibimos varios parámetros de salida en una sola variable, tendremos una tupla:
c = calcularAreaVol(2)
print(c)

(50.24, 33.4933333333333)
```

Ejercicio 48_02. Funciones en MATLAB

```
% Definición de función:
% Función suma hay que crearla en otro script y guardarla exactamente con el mismo
nombre (sumar.m) en la misma carpeta donde queremos trabajar y llamarla para usarla.
function resultado = sumar(x, y)
    resultado = x + y;
end

% Llamada a una función:
% Probamos la función sumar de x = 1 e y = 3
resultado = sumar(1,3);
fprintf('El resultado de sumar 1 y 3 es: %d\n.', resultado);

El resultado de sumar 1 y 3 es: 4.


% Funciones sin return:
% Si el valor es 1, mostrar un mensaje de lo contrario, mostrar otro mensaje.
% Primero hay que crear esta función mensaje en otro script y guardarla con el mismo
nombre (mensaje.m) en la misma carpeta donde queremos llamarla para usarla.
% Puedes copiarla y pegarla en un nuevo script desde aquí
function mensaje(valor, texto)
    if(valor == 1)
        fprintf('Mensaje con valor 1: %s\n.', texto);
        return; % Cortamos la ejecución si valor = 1
    end
    fprintf('Mensaje con otro valor: %s\n.', texto);
end
% Hasta aquí y guardarla en la misma carpeta con el nombre mensaje.m

mensaje(1, 'El valor es 1');
mensaje(2, 'El valor es 2');

Mensaje con valor 1: El valor es 1.
Mensaje con otro valor: El valor es 2.


% Lista como parámetro de una función:
% Creamos las listas
num1 = [3, 4, 5, 4];
num2 = [1, 2, 3];

% Definimos la función
function cuadrados(x)
    for i = 1:length(x)
```

```
        disp(x(i)^2);
    end
end

% Llamamos a la función
cuadrados(num1);
cuadrados(num2);

    9
    16
    25
    16

     1
     4
     9

% Funciones con parámetros por defecto y dos parámetros de salida:
% Función que calcula el área y volumen de una esfera
% Definimos la función con el valor de pi por defecto
function [a, v] = calcularAreaVol(r, pi)
    if nargin < 2
        pi = 3.14;
    end
    a = 4*pi*r^2;
    v = (4/3) * pi * r^3;
end

% Llamamos a la función pasándole sólo el radio
[area, vol] = calcularAreaVol(2);
fprintf('El área es %f y el volumen %f\n', area, vol);

El área es 50.240000 y el volumen 33.493333

% Si recibimos varios parámetros de salida en una sola variable, tendremos una
matriz:
c = calcularAreaVol(2);
disp(c);

50.2400
```

1.15. Cargar librerías y comprobar su correcta instalación

- **Cargar librerías en Python:**

  SpiPy, NumPy, matplotlib, pandas, scikit-learn

  - **NumPy**
    - Es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente indicada para grandes volúmenes de datos.

- Permite la creación de *arrays* y matrices, eficientes de definir y manipular.
- Un *array*, en Python, es una estructura de datos de un mismo tipo, organizada en forma de tabla o cuadrícula de distintas dimensiones.
- Los elementos de los *arrays* deben ser del mismo tipo.
- 1D *array* (1 dimensión), 2D *array* (2 dimensiones), 3D *array* (a dimensiones).

Documentación general sobre numpy: Librería Python numpy

Documentación *arrays* y matrices con numpy: Arrays en numpy

```
# scipy
import scipy
print('Versión de scipy: %s' % scipy._version_))
# numpy
import numpy
print('Versión de numpy: %s' % numpy._version_))
# matplotlib
import matplotlib
print('Versión de matplotlib: %s' % matplotlib._version_))
# pandas
import pandas
print('Versión de pandas: %s' % pandas._version_))
# scikit-learn
import scikit-learn
print('Versión de scikit-learn: %s' % sklearn._version_))
```

```
Versión de scipy: 1.7.1
Versión de numpy: 1.20.3
Versión de matplotlib: 3.4.3
Versión de pandas: 1.3.4
Versión de scikit-learn: 0.24.2
```

- En **MATLAB** no se cargan librerías sino Toolboxes durante la instalación del programa que hacen las mismas funciones que las librerías de Python. Entre estos Toolboxes destacan:
-
    - Inteligencia artificial, data analytics y estadística
        - Deep Learning Toolbox
        - Statistics and Machine Learning Toolbox
        - Curve Fitting Toolbox
        - Text Analytics Toolbox
    - Matemáticas y Optimización
        - Optimization Toolbox
        - Global Optimization Toolbox
        - Symbolic Math Toolbox
        - Mapping Toolbox
        - Partial Differential Equation Toolbox

- Generación de informes y acceso a bases de datos
  - Database Toolbox

Documentación sobre Toolbox: MATLAB Toolbox

1.16. Aplicación de librerías en *arrays* y vectores

Ejercicio 49_01. *Arrays* con numpy en Python

```
# Creamos un array a partir de la lista o tupla:
# import mumpy as np
a1 = [1, 2, 3] # Definimos la lista o vector
a2 = np.array(a1) # Convertimos
print("Vector original: ", a1)
print("Vector convertido a numpy: ", a2)

Vector original: [1 ,2, 3]
Vector convertido a numpy: [1 2 3]

# Podemos crear arrays directamente de 1, 2 o 3 dimensiones:
a1 = np.array([1, 2, 3, 4]) # vector o array de una dimensión
a2 = np.array([1, 2, 3],[4, 5, 6], [7, 8, 9]) # Matriz
a3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]]) # Cubo 3D
print("Vector: ", a1)
print("Matriz: ", a2)
print("Cubo: ", a3)

Vector: [1 2 3 4]
Matriz: [[1 2 3] [4 5 6] [7 8 9]]
Cubo: [[[1 2] [3 4]] [[5 6] [7 8]]]
```

Ejercicio 49_02. Vectores en MATLAB

```
% Creamos un cell array a partir de un vector
a1 = [1, 2, 3]; % Definimos el vector
% Convertimos el vector a cell array
a2 = num2cell(a1);
% Mostramos el vector original y el cell array resultante
fprintf('Vector original: ');
disp(a1);
fprintf('Cell array convertido en MATLAB: ');
disp(a2);

Vector original:      1     2     3
Cell array convertido en MATLAB: {[1]}    {[2]}    {[3]}

% Crear arrays directamente de 1, 2 o 3 dimensiones
% Vector o array de una dimensión
a1 = [1, 2, 3, 4];
fprintf('Vector: ');
disp(a1);
```

```
% Matriz
a2 = [1, 2, 3; 4, 5, 6; 7, 8, 9];
fprintf('Matriz:\n');
disp(a2);
% Cubo 3D
a3 = cat(3, [1, 2; 3, 4], [5, 6; 7, 8]);
% o bien
a3(:,:,1) = [1, 2; 3, 4];
a3(:,:,2) = [5, 6; 7, 8];

fprintf('Cubo:\n');
disp(a3);

Vector:    1     2     3     4
Matriz:
     1     2     3
     4     5     6
     7     8     9
Cubo:
(:,:,1) =
     1     2
     3     4
(:,:,2) =
     5     6
     7     8
```

Ejercicio 50_01. Definir el tipo de datos en *arrays* con numpy en Python

```
a = np.array([1, 2, 3, 4, 5], dtype = int) # entero
print(a)
print(a.dtype) # Muestra el tipo
b = np.array([1.5, 2, 3.9], dtype = float) # float
print(b)
print(b.dtype) # Muestra el tipo
c = np.array([1, 0, 0], dtype = bool) # booleano
print(c)
print(c.dtype) # Muestra el tipo
d = np.array([1, 2, 3, 4, 5], dtype = np.int32) # Entero
print(d)
print(d.dtype) # Muestra el tipo

   [1 2 3 4 5]
int32
    [1.5 2.0 3.90]
float64
   [ True False False]
bool
   [1 2 3 4 5]
int32
```

Ejercicio 50_02. Definir el tipo de datos en vectores en MATLAB

```
% Entero
a = int32([1, 2, 3, 4, 5]);
disp('a:');
disp(a);
disp('Tipo de a:');
disp(class(a));
% Float
b = [1.5, 2, 3.9];
disp('b:');
disp(b);
disp('Tipo de b:');
disp(class(b));
% Booleano
c = logical([1, 0, 0]);
disp('c:');
disp(c);
disp('Tipo de c:');
disp(class(c));
% Entero de 32 bits
d = int32([1, 2, 3, 4, 5]);
disp('d:');
disp(d);
disp('Tipo de d:');
disp(class(d));
```

```
a:
   1   2   3   4   5
Tipo de a:
int32
b:
    1.5000    2.0000    3.9000
Tipo de b:
double
c:
   1   0   0
Tipo de c:
logical
d:
   1   2   3   4   5
Tipo de d:
int32
```

1.17. Aplicación de librerías en *arrays* y matrices

Ejercicio 51_01. Creación de matrices con numpy en Python

```
# Matrices con numpy
import numpy as np
A = [[1, 2, 3], [3, 4, 5]] # Definimos la matriz
print("Matriz original: ", A)
```

```
# Convertimos a numpy
B = np.array(A)
print("Matriz convertida a numpy: ", B)
%# Creación directa
C = np.array([[0, 1, 2], [-1, 2, 3]], dtype = int)
print("Matriz de forma directa: ", C)

Matriz original: [[1, 2, 3], [3, 4, 5]]
Matriz convertida a numpy: [[1 2 3] [3 4 5]]
Matriz de forma directa: [[0  1  2][-1  2  3]]
```

Ejercicio 51_02. Creación de matrices

```
% Definir una matriz
A = [1, 2, 3; 3, 4, 5];
fprintf('Matriz original:\n');
disp(A);
% Convertir a MATLAB (no se necesita una conversión explícita)
B = A;
fprintf('Matriz convertida a MATLAB:\n');
disp(B);
% Crear una matriz directamente
C = int32([0, 1, 2; -1, 2, 3]);
fprintf('Matriz de forma directa:\n');
disp(C);

Matriz original:
     1     2     3
     3     4     5
Matriz convertida a MATLAB:
     1     2     3
     3     4     5
Matriz de forma directa:
    0    1    2
   -1    2    3
```

Ejercicio 52_01. Atributos de una matriz en Python

```
# Atributos de un array
print("Matriz B: ", B)
print("Dimensiones de la matriz B: ", B.shape)
print("Número de dimensiones de la matriz B: ", B.ndim)
print("Número de elementos de la matriz B: ", B.size)

Matriz B: [[1 2 3][3 4 5]]
Dimensiones de la matriz B: (2, 3)
Número de dimensiones de la matriz B: 2
Número de elementos de la matriz B: 6
Matriz B:
     1     2     3
     3     4     5
```

Ejercicio 52_02. Atributos de una matriz en MATLAB

```
% Atributos de una matriz
fprintf('Matriz B:\n');
disp(B);
fprintf('Dimensiones de la matriz B: %s\n', mat2str(size(B)));
fprintf('Número de dimensiones de la matriz B: %d\n', ndims(B));
fprintf('Número de elementos de la matriz B: %d\n', numel(B));
```

```
Matriz B:
     1     2     3
     3     4     5

Dimensiones de la matriz B: [2 3]
Número de dimensiones de la matriz B: 2
Número de elementos de la matriz B: 6
Matriz B:
     1     2     3
     3     4     5
```

Ejercicio 53_01. Filas y columnas de una matriz en Python

```
# filas y columnas de la matriz
print("Matriz B: ", B)
print("Primera fila de la matriz B: ", B[0])
print("Primera fila de otra forma de la matriz B: ", B[0,:])
print("Última fila de la matriz B: ", B[-1])
print("Segunda columna de la matriz B: ", B[:,1])
print("Última columna de la matriz B: ", B[:,-1])
```

```
Matriz B: [[1 2 3] [3 4 5]]
Primera fila de la matriz B:
     [1  2  3]
Primera fila otra forma: [1  2  3]
Última fila de la matriz B: [3  4  5]
Segunda columna de la matriz B: [2  4]
Última columna de la matriz B: [3  5]
```

Ejercicio 53_02. Filas y columnas de una matriz en MATLAB

```
% Filas y columnas de la matriz
fprintf('Matriz B:\n');
disp(B);
fprintf('Primera fila de la matriz B:\n');
disp(B(1, :));
fprintf('Última fila de la matriz B:\n');
disp(B(end, :));
fprintf('Segunda columna de la matriz B:\n');
disp(B(:, 2));
fprintf('Última columna de la matriz B:\n');
disp(B(:, end));
```

```
Primera fila de la matriz B:
     1     2     3
Última fila de la matriz B:
     3     4     5
Segunda columna de la matriz B:
     2
     4
Última columna de la matriz B:
     3
     5
```

Ejercicio 54_01. Operaciones con matrices de numpy en Python

```
# Operaciones con matrices
C = np.array([[1, 2, 3], [1, 3, 4]])
D = np.array([[3, 3, 3], [4, 1, 2]])
print("Matriz C: ")
print(C)
print("Matriz D: ")
print(D)
print("Suma de C y D: %s" %(C+D))
print("Multiplicación de C y D: %s" %(C*D))
```

```
Matriz C:
     [[1  2  3] [1  3  4]]
Matriz D:
     [[3  3  3] [4  1  2]]
Suma de C y D:
     [[4   5   6] [5   4   6]]
Multiplicación de C y D: [[3   6   9] [4   3   8]]
```

Ejercicio 54_02. Operaciones con matrices en MATLAB

```
% Operaciones con matrices
C = [1, 2, 3; 1, 3, 4];
D = [3, 3, 3; 4, 1, 2];
fprintf('Matriz C:\n');
disp(C);
fprintf('Matriz D:\n');
disp(D);
fprintf('Suma de C y D:\n');
disp(C + D);
fprintf('Multiplicación de C y D:\n');
disp(C .* D);
```

```
Matriz C:
     1     2     3
     1     3     4
Matriz D:
     3     3     3
     4     1     2
```

```
Suma de C y D:
     4     5     6
     5     4     6
Multiplicación de C y D:
     3     6     9
     4     3     8
```

1.18. Funciones útiles

- **Funciones útiles para crear *arrays* en Python:**

Array vacío: np.empty(dimensiones)
Array relleno de ceros: np.zeros(dimensiones)
Array aleatorio: random.randint(tope, size=(dimensiones))

Documentación para: Vectores aleatorios

- **Funciones útliles para crear arrays en Matlab:**

Array vacío: v = []; % Crea un vector vacío usando corchetes vacíos
Array relleno de ceros: A = zeros(m, n); % Crea una matriz m x n de ceros
Array relleno de unos: A = ones(m, n); % Crea una matriz m x n de unos
Array aleatorio: A = randn(m, n); % Crea una matriz m x n de valores aleatorios normalmente distribuidos

Ejercicio 55_01. *Array* vacío especificando las dimensiones

```
# array vacío especificando las dimensiones 2x2:
import numpy as np
a = np.empty([2, 2])
print("Mostramos el resultado: ", a)
print("Mostramos dimensiones: ", a.shape)
# le damos valores
a[0, 0] = 1
a[0, 1] = 2
a[1, 0] = 3
a[1, 1] = 4
print("Mostramos tras dar valores: ", a)

Mostramos el resultado: [[0   0] [0   0]]
Mostramos dimensiones: (2, 2)
Mostramos tras dar valores: [[1.  2.] [3.  4.]]
```

Ejercicio 55_02. *Array* vacío especificando las dimensiones

```
% Array vacío especificando las dimensiones
a = zeros(2); % o a = []; para un array vacío
fprintf('Mostramos el resultado:\n');
disp(a);
fprintf('Mostramos dimensiones: %s\n', mat2str(size(a)));
% Le damos valores
a(1, 1) = 1;
a(1, 2) = 2;
a(2, 1) = 3;
```

```
a(2, 2) = 4;
fprintf('Mostramos tras dar valores:\n');
disp(a);

Mostramos el resultado
     0     0
     0     0
Mostramos dimensiones
     2     2
Mostramos tras dar valores:
     1     2
     3     4
```

Ejercicio 56_01. *Array* lleno de ceros

```
import numpy as np
a = np.zeros(2)
print("Vector resultado: ", a)
b = np.zeros([3, 4])
print("Matriz resultado: \n", b) #\n para salto de linea

Vector resultado: [0. 0.]
Matriz resultado:
    [[0.  0.  0.  0.]
    [0.  0.  0.  0.]
    [0.  0.  0.  0.]]
```

Ejercicio 56_02. *Array* lleno de ceros

```
% Array lleno de ceros
b = zeros(1, 2); % Vector
fprintf('Vector resultado:\n');
disp(b);
c = zeros(3, 4); % Matriz
fprintf('Matriz resultado:\n');
disp(c);

Vector resultado:
     0     0
Matriz resultado:
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Ejercicio 57_01. *Array* aleatorio Python

```
from numpy import random
# Vector de enteros aleatorio de 10 posiciones con números entre 1 y 50
v = random.randint(50, size=(10))
print("Vector: \n", v)
# Matriz de enteros aleatoria de 3x4 con números entre 1 y 10
M = random.randint(10, size=(3,4))
print("Matriz: \n", M)
```

```
Vector:
    29   4   3  27  39  47  7  29  24   1
Matriz:
     [[4   4   7   7]
      [2   6   3   8]
      [8   2   7   5]]
```

Ejercicio 57_02. *Array* aleatorio MATLAB

```
% Vector de enteros aleatorio de 10 posiciones con números entre 1 y 50
v = randi([1, 50], 1, 10);
fprintf('Vector:\n');
disp(v);
% Matriz de enteros aleatoria de 3x4 con números entre 1 y 10
M = randi([1, 10], 3, 4);
fprintf('Matriz:\n');
disp(M);
```

```
Vector:
    29     4     3    27    39    47     7    29    24     1
Matriz:
     4     4     7     7
     2     6     3     8
     8     2     7     5
```

1.19. Gráficas en Python y MATLAB

- **Librería Matplotlib para Python.** Permite la creación y manipulación de gráficos.

Documentación: Librería Matplotlib

- **En MATLAB se cargan las propias Toolboxes durante la instalación del programa.**

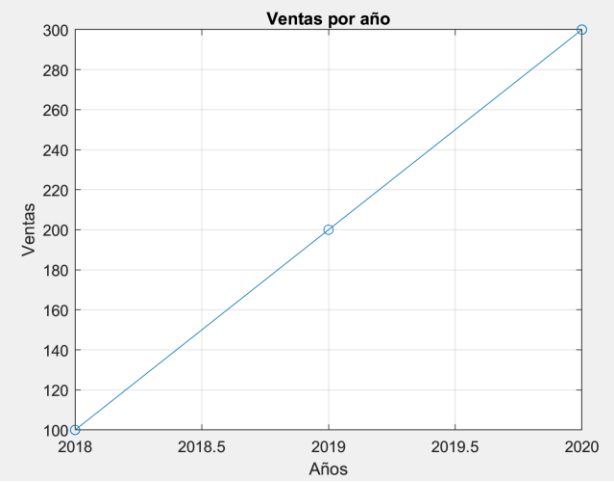
Ejercicio 58_01. Gráfico 2D con Matplotlib en Python

```
# Preparamos los datos a representar e importamos las librerías necesarias.
import matplotlib.pyplot as plt # alias para facilitarnos la escritura
import numpy as np
# Creamos arrays con los datos para cada eje
datosX = np.array(['2018', '2019', '2020'])
datosY = np.array([100, 200, 300])
# creamos una gráfica simple:
plt.plot(datosX, datosY)
# Ajustamos los ejes y título
plt.xlabel('Años')
plt.ylabel('Ventas')
plt.title('Ventas por año')
```

```
plt.grid()
plt.savefig('gráfico_simple.png') # guardamos el gráfico como png
# Mostramos la gráfica
plt.show()
```

Ejercicio 58_02. Gráfico 2D en MATLAB

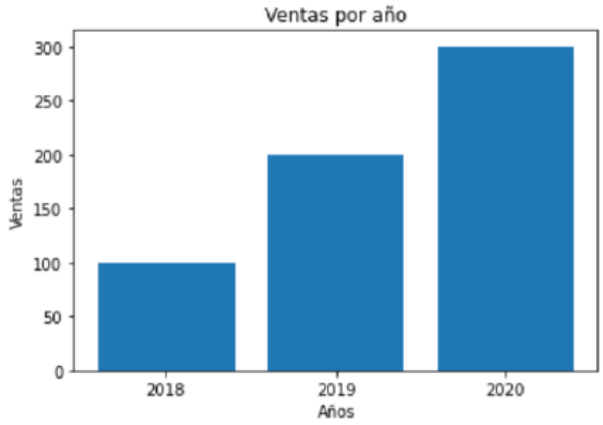
```
% Creamos arrays con los datos para cada eje
datosX = [2018, 2019, 2020];
datosY = [100, 200, 300];
% Creamos un gráfico 2D con MATLAB:
figure; % Creamos una nueva figura
plot(datosX, datosY, '-o'); % Gráfico de línea con marcadores
xlabel('Años');
ylabel('Ventas');
title('Ventas por año');
grid on;
saveas(gcf, 'grafico_simple.png'); % Guardamos el gráfico como png
```



Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

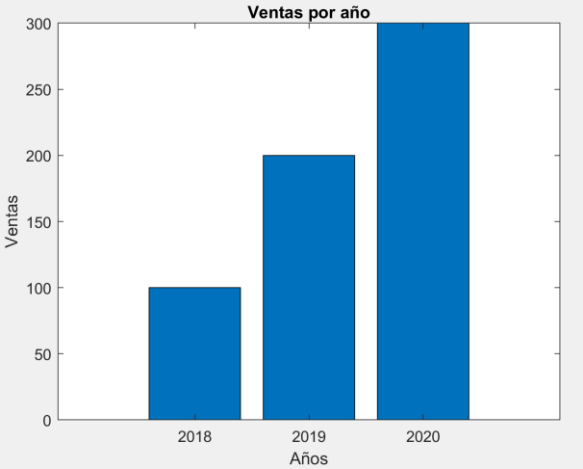
Ejercicio 59_01. Diagrama de barras en Python

```
# Creamos un diagrama de barras con Matplotlib:
plt.bar(datosX, datosY)
# Ajustamos los ejes y título
plt.xlabel('Años')
plt.ylabel('Ventas')
plt.title('Ventas por año')
plt.savefig('gráfico_barras.png') # guardamos el gráfico como png
# Mostramos la gráfica
plt.show()
```



Ejercicio 59_02. Diagrama de barras en MATLAB

```
% Creamos un diagrama de barras con MATLAB:
figure;
bar(datosX, datosY);
xlabel('Años');
ylabel('Ventas');
title('Ventas por año');
saveas(gcf, 'grafico_barras.png');
```



Ejercicio 60_01. Diagrama de dispersión en Python

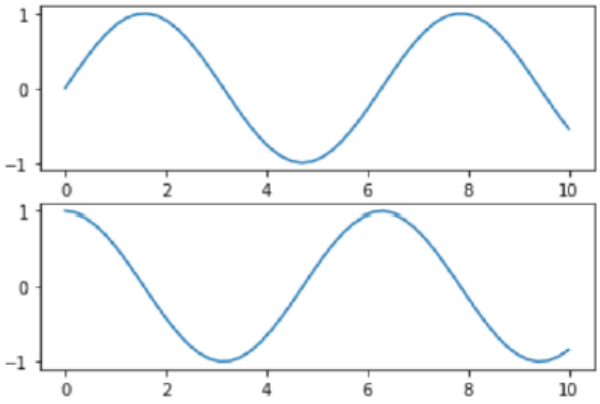
```
# Creamos un diagrama de dispersión con Matplotlib:
plt.scatter(datosX, datosY)
# Ajustamos los ejes y título
plt.xlabel('Años')
plt.ylabel('Ventas')
plt.title('Ventas por año')
plt.savefig('gráfico_dispersión.png') # guardamos el gráfico como png
# Mostramos la gráfica
plt.show()
```

Ejercicio 60_02. Diagrama de dispersión en MATLAB

```
% Creamos un diagrama de dispersión con MATLAB:
figure;
scatter(datosY, datosX, 'filled');
xlabel('Ventas');
ylabel('Años');
title('Ventas por año');
saveas(gcf, 'grafico_dispersion.png');
```

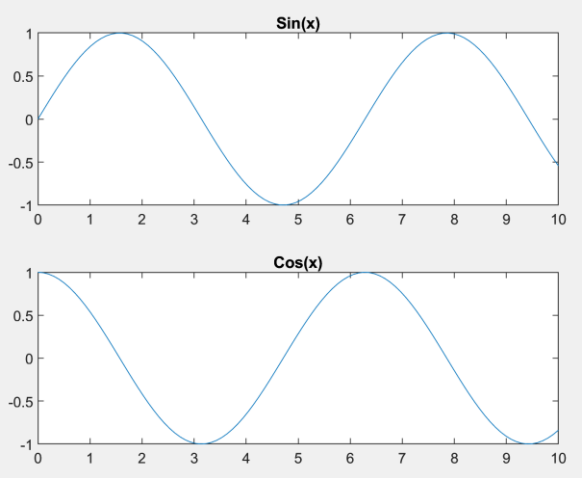
Ejercicio 61_01. Gráfica de seno y coseno en Python

```
# Creamos un subplot
plt.figure() # creamos la figura
# creamos un vecroe con los números entre 0 y 10
x = np.linspace(0, 10)
# Vamos a tener 2 subplots en la figura
# Cada subplot es una fila, por lo que tendremos 2 filas y 2 subplots o paneles
plt.subplot(2, 1, 1) # (filas, columnas, nº de panel)
plt.plot(x, np.sin(x))
plt.subplot(2, 1, 1) # (filas, columnas, nº de panel)
plt.plot(x, np.cos(x))
```



Ejercicio 61_02. Gráfica de seno y coseno en MATLAB

```
% Creamos un subplot
figure;
% Creamos un vector con los números entre 0 y 10
x = linspace(0, 10);
% Subplot 1
subplot(2, 1, 1);
plot(x, sin(x));
title('Sin(x)');
% Subplot 2
subplot(2, 1, 2);
plot(x, cos(x));
title('Cos(x)');
```



Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

1.20. Series en Python y tablas en MATLAB

- **Librería Pandas para Python.** Permite la creación y manipulación de gráficos.

Pandas. Es una librería de Python especializada en la manipulación y el análisis de datos.

Cuenta con funciones para el análisis, la limpieza, la exploración y la manipulación de datos.

Proporciona estructuras de datos de alto nivel y herramientas para tablas (DataFrame) y series (Series).

Construida sobre numpy.

Para obtener una nueva serie de booleanos se usa: **serie.isnull()**, **serie.notnull()**. Se puede usar como filtro: **serie[serie.isnull()]**, **serie[serie.notnull()]**.

Trabajaremos con series usando Pandas.

- Una serie es una *array* de una dimensión con un índice.
- Este índice puede ser una etiqueta para caracterizar mejor los elementos guardados.
- Permite almacenar información de cualquier tipo (cadenas de texto, flotantes o enteros, entre otros)

Documentación: Documentación general sobre Pandas

Series en Python con Pandas

- En **MATLAB** el equivalente a series son las tablas reducidas de dos columnas; una de etiquetas y otra de valores, que se instalan en los propios Toolboxes al instalar el propio programa, es decir, manejan estructuras de datos nativas.

Ejercicio 62_01. Creación de una serie desde numpy con pandas en Python

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
# Primero creamos un array de numpy con los datos
v = np.array([100, 200, 300],dtype = np.int32)
# Luego una lista con los datos de las etiquetas
filas = ["2018", "2019", "2020"]
# Creamos la serie
serie1 = pd.Series(v, index = filas)
# Mostramos
print(serie1)
```

```
    2018    100
    2019    200
    2020    300
dtype: int32
```

Ejercicio 62_02. Creación de una tabla en MATLAB

```
% Primero creamos un array de MATLAB con los datos
v = [100, 200, 300];
% Luego una lista con los datos de las etiquetas
filas = {'2018', '2019', '2020'};
% Creamos la tabla
tabla1 = table(v', 'RowNames', filas);
% Mostramos
disp('Tabla 1:');
disp(tabla1);
```

```
Tabla 1:
            Var1
            ____

    2018    100
    2019    200
    2020    300
```

Ejercicio 63_01. Definición de forma directa desde Numpy con Pandas en Python

```
# Los datos de una serie de Pandas son arrays de Numpy
nat = pd.series([10.7, 7.5, 7.1, 17.8, 7.9], dtype = float)
index = ["Murcia", "Cantabria", "Galicia", "Melilla", "Canarias"], name = "Tasa de
Natalidad"]
# Mostramos
print(nat)
```

```
    Murcia             10.7
    Cantabria           7.5
    Galicia             7.1
    Melilla            17.8
    Canarias            7.9
Name: Tasa de Natalidad, dtype: float64
```

Ejercicio 63_02. Definición de forma directa en Matlab

```
nat = [10.7, 7.5, 7.1, 17.8, 7.9];
index_nat = {'Murcia', 'Cantabria', 'Galicia', 'Melilla', 'Canarias'};
tabla2 = table(nat', 'RowNames', index_nat, 'VariableNames', {'Tasa_de_Natalidad'});
% Mostramos
disp('Tabla 2:');
disp(tabla2);
```

```
Tabla 2:
                Tasa_de_Natalidad
                _________________

    Murcia             10.7
    Cantabria           7.5
```

```
    Galicia             7.1
    Melilla            17.8
    Canarias            7.9
```

Ejercicio 64_01. Definición sin especificar etiquetas

```
# Creará un índice por defecto empezando en el 0
a = [4, 8, 16, 32, 64]
serie2 = pd.Series(a)
# Mostramos
print(serie2)
```

```
 0   4
 1   8
 2   16
 3   32
 4   64
```

Ejercicio 64_02. Definición sin especificar etiquetas

```
a = [4, 8, 16, 32, 64];
tabla3 = table(a', 'VariableNames', {'Columna_A'});
% Mostramos
disp('Tabla 3:');
disp(tabla3);
```

```
Tabla 3:
    Columna_A
    _________

        4
        8
       16
       32
       64
```

Ejercicio 65_01. Creación de una serie desde diccionario

```
# Las claves del diccionario pasarán a ser las etiquetas
# Definimos diccionario
emisiones_co2 = {"Enero": 12, "Febrero": 28, "Marzo": 40, "Abril": 10}
series3 = pd.Series(emisiones_co2)
print(series3)
```

```
    Enero           12
    Febrero         28
    Marzo           40
    Abril           10
```

Ejercicio 65_02. Creación de una tabla desde diccionario

```
% Definimos diccionario
emisiones_co2 = {'Enero', 12; 'Febrero', 28; 'Marzo', 40; 'Abril', 10};
tabla4 = table(emisiones_co2(:, 2), 'RowNames', emisiones_co2(:, 1), 'VariableNames',
{'Emisiones_CO_{2}'});
disp('Tabla 4:');
disp(tabla4);
```

```
Tabla 4:
               Emisiones_CO_{2}
               ________________

    Enero           {[12]}
    Febrero         {[28]}
    Marzo           {[40]}
    Abril           {[10]}
```

Ejercicio 66_01. Creación de una serie usando únicamente las claves que nos interesen

```
series4 = pd.Series(emisiones_co2, index = ["Febrero", "Abril"])
print(series4)
```

```
    Febrero       28
    Abril         10
```

Ejercicio 66_02. Creación de una tabla usando únicamente las claves que nos interesen

```
% Podemos crear una tabla usando únicamente las claves que nos interesen:
indices_interes = {'Febrero', 'Abril'};
tabla5 = table(emisiones_co2(ismember(emisiones_co2(:, 1), indices_interes), 2),
'RowNames', indices_interes, 'VariableNames', {'Emisiones_CO2'});
disp('Tabla 5:');
disp(tabla5);
```

```
Tabla 5:
               Emisiones_CO2
               _____________

    Febrero       {[28]}
    Abril         {[10]}
```

Ejercicio 67_01. Acceso a los datos de la serie

```
# Se puede acceder por el índice o por la etiqueta
print("Imprimimos primer elemento de la serie: ")
print(nat[0]) # Acceso por el índice
print("Imprimimos el valor de la serie para Cantabria")
print(nat["Cantabria"]) # Acceso por la etiqueta
```

```
Imprimimos primer elemento de la serie:
   10.7000
Imprimimos el valor de la serie para Cantabria:
    7.5000
```

Ejercicio 67_02. Acceso a los datos de la tabla

```
% Acceso a los datos de la tabla
disp('Imprimimos primer elemento de la tabla 2:');
disp(tabla2{'Murcia', 'Tasa_de_Natalidad'});
disp('Imprimimos el valor de la tabla 2 para Cantabria:');
disp(tabla2{'Cantabria', 'Tasa_de_Natalidad'});
```

```
Imprimimos primer elemento de la tabla 2:
   10.7000
Imprimimos el valor de la tabla 2 para Cantabria:
    7.5000
```

Ejercicio 68_01. Datos vacíos en una serie en Python

```
# se pueden usar serie.isna() y serie.isnull(), que son equivalentes
# se puede usar como filtro:
# serie[serie.isnull()]
# serie[serie.notnull()]
serie1 = pd.Series([1, 2, 3, np.NaN, 5])
print("Imprimimos la serie")
print(serie1)
print("Imprimimos resultado isna()")
print(serie1.isna())
print("Imprimimos resultado isnull()")
print(serie1.isnull())
print("Filtramos los no nulos")
print(serie1[serie1.notnull()])
print("Filtramos mostrando los nulos")
print(serie1[serie1.isnull()])
```

```
print(serie1)
0    1.0
1    2.0
2    3.0
3    NaN
4    5.0
dtype: float64
```

```
print(serie1.isna())
0    False
1    False
2    False
3     True
```

```
4    False
dtype: bool

print(serie1.isnull())
0    False
1    False
2    False
3     True
4    False
dtype: bool

print(serie1[serie1.isnull()])
3   NaN
dtype: float64
```

Ejercicio 68_01. Datos vacíos en una tabla en MATLAB

```
Names = {'USA'; 'España'; 'Italia'; 'Argentina'; 'Francia'};
Numbers = [1; 2; 3; NaN; 5];
T = table(Names, Numbers)

T = 5×2 table

        Names        Numbers
    _____________    _______

    {'USA'      }        1
    {'España'   }        2
    {'Italia'   }        3
    {'Argentina'}      NaN
    {'Francia'  }        5


% Filtrar nulos y no nulosT.Names(~isnan(T.Numbers))
disp('Filtramos los no nulos:');
disp(T.Names(~isnan(T.Numbers)));

disp('Filtramos mostrando los nulos:');
disp(T.Numbers(isnan(T.Numbers)));
```

1.21. *Dataframes* con Panda en Python y registros en MATLAB

Un **Dataframe** es una matriz multidimensional formada por series en Pandas

Cada columna es una serie, que tendrá su índice y su nombre.

- Podemos etiquetar los índices.

Documentación: Documentación DataFrames

- **En MATLAB el equivalente a *dataframes* son tablas de más de dos columnas, como una matriz con diferentes tipos de datos.**

```
% Crear una tabla ejemplo:
 T = table(categorical({'M';'F';'M'}),[45;32;34],logical([1;0;0]),...
   'VariableNames',{'Gender','Age','Vote'},...
   'RowNames',{'NY';'CA';'MA'})
```

```
      T=3×3 table
                  Gender    Age    Vote
                  ______    ___    _____

            NY      M       45     true
            CA      F       32     false
            MA      M       34     false
```

Documentación Command Windows: help table

Ejercicio 69_01. Definición de *Dataframe* en Python

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
df = pd.Dataframe({"ex1": [10, 3, 6, 4], "ex2": [10, 5, 8, 5], "ex3": [8, 6, 8,2]},
index = ["Alex", "Pedro", "Paula", "Miguel"])
print(df)
```

```
            ex1    ex2    ex3
    Alex     10     10      8
    Pedro     3      5      6
    Paula     6      8      8
    Miguel    4      5      2
```

Ejercicio 69_02. Definición de tabla matriz en MATLAB

```
% Creamos una Tabla en MATLAB
datos = [10, 3, 6; 10, 5, 8; 8, 6, 8; 5, 4, 2];
nombres_filas = {'Alex', 'Pedro', 'Paula', 'Miguel'};
nombres_columnas = {'ex1', 'ex2', 'ex3'};
df = array2table(datos, 'RowNames', nombres_filas, 'VariableNames',
nombres_columnas);
% Mostramos la Tabla
disp('Tabla matriz:');
disp(df);
```

```
Tabla matriz:
            ex1    ex2    ex3
            ___    ___    ___

    Alex     10      3      6
    Pedro    10      5      8
    Paula     8      6      8
```

```
    Miguel      5       4       2
```

Ejercicio 70_01. ***Definición partiendo de matriz de numpy***

```
# Matriz con numpy
matriz = np.array([[1, 2, 3], [4, 5, 6]])
# Nombramos columnas y damos etiquetas a filas
filas = ['2018', '2019']
col = ['valor1', 'valor2', 'valor3']
# Creamos dataframe
df2 = pd.Dataframe(matriz, index=filas, columns=col)
print("Imprimimos el dataframe: ")
print(df2)
```

```
Imprimimos el dataframe:
            valor1    valor2    valor3

    2018      1         2         3
    2019      4         5         6
```

Ejercicio 70_02. ***Definición partiendo de matriz***

```
matriz = [1, 2, 3; 4, 5, 6];
% Nombramos columnas y damos etiquetas a filas
filas = {'2018', '2019'};
columnas = {'valor1', 'valor2', 'valor3'};
% Creamos una tabla
df2 = array2table(matriz, 'RowNames', filas, 'VariableNames', columnas);
% Mostramos el DataFrame
disp('Matriz tabla 2:');
disp(df2);
```

```
Matriz tabla 2:
            valor1    valor2    valor3
            ______    ______    ______

    2018      1         2         3
    2019      4         5         6
```

Ejercicio 71_01. Acceso a los elementos del *dataframe* en Python

```
print("Obtenemos columna de df: ")
print(df.ex1)
print("Obtenemos fila 3 de la columna ex1: ")
print(df.ex1[2])
print("Obtenemos fila Paula de la columna ex3: ")
print(df.ex3["Paula"])
```

```
Obtenemos columna de df:
Alex     10
```

```
Pedro     3
Paula     6
Miguel    4
Name: ex1, dtype: int64
Obtenemos fila 3 de la columna ex1:
6
Obtenemos fila Paula de la columna ex3:
8
```

Ejercicio 71_02. Acceso a los elementos en MATLAB

```
% Acceso a los elementos
disp('Obtenemos columna de df:');
disp(df.ex1');
disp('Obtenemos fila 3 de la columna ex1:');
disp(df.ex1(3));
disp('Obtenemos fila Paula de la columna ex3:');
disp(df.ex3('Paula'));
```

```
Obtenemos columna de df:
    10    10     8     5
Obtenemos fila 3 de la columna ex1:
     8
Obtenemos fila Paula de la columna ex3:
     8
```

Ejercicio 72_01. Dataframes similares a diccionarios

```
# Usar columnas como índices
# Definición de dataframe similar a diccionario:
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
df = pd.DataFrame({
#Valores reales
"A": [1.0, 3.5, 3.4, 1.4],
# Fechas
"B": pd.date_range("20221129"),
# 4días
"C": pd.Series([1, 2, 3, np.NaN]),
# Valores enteros
"D": np.array([3, 4, 5, 6]),
# Valores categóricos
"E": pd.Categorical(["test", "train", "test", "train"]),
 })
# Añadimos las etiquetas de filas
filas = [["Lunes", "Martes", "Miércoles", "Jueves"]]
df.index = filas;
print(df)
print("Mostramos el tipo de dato de cada columna")
print(df.dtypes)
```

```
                A          B          C     D    E      F
```

```
    Lunes        1.0    29-Nov-2022    1.0    3    2    test
    Martes       3.5    30-Nov-2022    2.0    4    5    train
    Miércoles    3.4    01-Dec-2022    3.0    5    8    test
    Jueves       1.4    02-Dec-2022    NaN    6    1    train

Mostramos el tipo de dato de cada columna:
A      float64
B      datetime64[ns]
C      float64
D      int32
E      category
dtype: object


dfVentas = pd.DataFrame({'meses': [1, 2, 3, 4], 'años': [2018, 2019, 2020, 2021],
'ventas': [125, 130, 232, 134]})
print("Imprimimos el dataframe original dfVentas: ")
print(dfVentas)
dfVentas2 = dfVentas.set_index("años")
print("\n Mostramos el nuevo dataframe dfVentas2: ") # Usamos \n para salto de línea
print(dfVentas2)

Imprimimos el dataframe original dfVentas:
     meses    años    ventas
  0    1      2018     125
  1    2      2019     130
  2    3      2020     232
  3    4      2021     134

Mostramos el nuevo dataframe dfVentas2:
    años   meses     ventas
    2018      1        125
    2019      2        130
    2020      3        232
    2021      4        134

# Importante: al fijar el índice de esta forma, debemos observar que la columna
elimina el conjunto de características. Esto podemos controlarlo con el parámetro
“drop”.
dfVentas2 = dfVentas.set_index("años", drop = False)
print(dfVentas2)


Mostramos el nuevo dataframe dfVentas2:

    años   meses    ventas
    _____________________
    2018     1        125
    2019     2        130
    2020     3        232
    2021     4        134

# Mostramos el dataframe original
```

```
print("DataFrame dfVentas: ")
print(dfVentas)
# Cambiamos el índice con set_index haciendo esta modificiación en dfVentas
dfVentas.set_index("años", inplace = True, drop = False)
print("\n DataFrame dfVentas tras modificar con set_index: ") # usamos \n para salto de línea
print(dfVentas)
```

Ejercicio 72_02. Trabajar con registros en MATLAB

```
datos = {
    [1.0; 3.5; 3.4; 1.4],                                % Valores reales
    [datetime('2022-11-29'); datetime('2022-11-30'); …
     datetime('2022-12-01');datetime('2022-12-02')], % Fechas
    [1; 2; 3; NaN],                                      % 4 días
    [3; 4; 5; 6],
    [2; 5; 8; 1],                                         % Valores enteros
     categorical({'test', 'train', 'test', 'train'}') % Valores categóricos
 };
nombres_filas = {'Lunes', 'Martes', 'Miércoles', 'Jueves'};
nombres_columnas = {'A', 'B', 'C', 'D', 'E', 'F'};
df = table(datos{:}, 'RowNames', nombres_filas, 'VariableNames', nombres_columnas);
% Mostramos la tabla
disp('DataTable:');
disp(df);
% Mostramos el tipo de dato de cada columna
disp('El tipo de datos es:');
disp(class(df));
```

```
DataTable:
                A          B          C     D    E      F
              ___    ___________    ___    _    _    _____

    Lunes       1    29-Nov-2022      1    3    2    test
    Martes    3.5    30-Nov-2022      2    4    5    train
    Miércoles 3.4    01-Dec-2022      3    5    8    test
    Jueves    1.4    02-Dec-2022    NaN    6    1    train

El tipo de datos es:
table
```

```
% Crear un array con los datos
datos = [1, 2, 3, 4; 2018, 2019, 2020, 2021; 125, 130, 232, 134]';
% Crear un dataframe en MATLAB
dfVentas = array2table(datos, 'VariableNames', {'meses', 'anios', 'ventas'});
% Mostrar el dataframe original
disp('Imprimimos el dataframe original dfVentas:');
disp(dfVentas);
% Establecer "años" como índice
dfVentas2 = table2timetable(dfVentas, 'RowTimes', datetime(dfVentas.anios, 1, 1));
% Mostrar el nuevo dataframe con "años" como índice
disp('Mostramos el nuevo dataframe dfVentas2:');
```

```
disp(dfVentas2);

Imprimimos el dataframe original dfVentas:
    meses    anios    ventas
    _____    _____    ______

      1      2018      125
      2      2019      130
      3      2020      232
      4      2021      134

Mostramos el nuevo dataframe dfVentas2:
       Time         meses    anios    ventas
    ____________    _____    _____    ______

    01-Jan-2018       1      2018      125
    01-Jan-2019       2      2019      130
    01-Jan-2020       3      2020      232
    01-Jan-2021       4      2021      134

% Otra forma:
% Crear un array con los datos
datos = [1, 2, 3, 4; 2018, 2019, 2020, 2021; 125, 130, 232, 134]';
% Crear un dataframe en MATLAB
dfVentas = array2table(datos, 'VariableNames', {'meses', 'anios', 'ventas'});
% Mostrar el dataframe original
disp('Imprimimos el dataframe original dfVentas:');
disp(dfVentas);
% Establecer "años" como índice sin eliminar la columna
dfVentas2 = dfVentas;
dfVentas2.Properties.RowNames = cellstr(num2str(dfVentas2.anios));
dfVentas2.anios = [];
disp('Mostramos el nuevo dataframe dfVentas2:');
disp(dfVentas2);

Imprimimos el dataframe original dfVentas:
    meses    anios    ventas
    _____    _____    ______

      1      2018      125
      2      2019      130
      3      2020      232
      4      2021      134

Mostramos el nuevo dataframe dfVentas2:
            meses    ventas
            _____    ______

    2018      1       125
    2019      2       130
    2020      3       232
    2021      4       134
```

1.22. Funciones para obtener información

En **Python**, agrupar el número de apariciones de un valor de una columna:
df ['columna'].value_counts()
Número de filas del dataframe: **len(df)**
Número de valores distintos de una columna: **df['columna'].nunique()**

```
% En Matlab podríamos hacerlo así:
% Ejemplo de una matriz de datos con una columna
datos = [1; 2; 3; 2; 1; 3; 1; 4; 2; 3];
% Contar apariciones de cada valor en la columna
conteo_valores = tabulate(datos);
% Mostrar los resultados
disp(conteo_valores);
% Ejemplo de una matriz de datos aleatorios
datos = rand(100, 3); % Crear una matriz de 100 filas y 3 columnas con valores
aleatorios
% Obtener el número de filas (observaciones)
num_filas = size(datos, 1);
% Mostrar el número de filas
disp(num_filas);
% Obtener valores únicos en la columna
valores_unicos = unique(datos);
% Contar el número de valores únicos
num_valores_unicos = numel(valores_unicos);
% Mostrar el número de valores únicos
disp(num_valores_unicos);
```

Ejercicio 73_01. Funciones para obtener información en Python

```
import pandas as pd
df.pd.DataFrame({"Est": ["A", "B", "C", "D"], "Contaminante 1": [30, 50, 20, 12],
"Contaminante 2": [14, 12, 7, 4], "Tipo": ["Train", "Test", "Train", "Val"]})
print(df)
print("Filas de df: ")
print(len(df))
print("Veces que aparecen los valores de la columna tipo: ")
print(df['Tipo'].value_counts())
print('Número de tipos distintos: ')
print(df['Tipo'].value_counts())
print("Sumario general: ")
print(df.describe())
```

```
DataTable:
     Est     Contaminante1    Contaminante2    Tipo
 0   A       30                14              Train
 1   B       50                12              Test
 2   C       20                 7              Train
 3   D       12                 4              Val

Filas de df: 4
Veces que aparecen los valores de la columna tipo:
```

```
Train 2
Test  1
Val   1
Name: Tipo, dtype: int64
Número de tipos distintos:
3
Sumario general:

          Contaminante1    Contaminante2
 count        4.000000           4. 000000
mean         28.000000          18.000000
std          16.411378           9.250000
min          12.000000           4.573474
25%          18.000000           4.000000
50%          25.000000           6.250000
75%          35.000000           9.500000
Max          50.000000          14.000000
```

Ejercicio 73_02. Funciones para obtener información en MATLAB

```
% Creamos un DataTable en MATLAB

estaciones = {'A','B','C','D'};
tipos = {'Train', 'Test', 'Train', 'Val'};

cont1 = [30 50 20 12];
cont2 = [14 12 7 4];

nombres_columnas = {'Est', 'Contaminante1', 'Contaminante2', 'Tipo'};
df2 = table(estaciones',cont1',cont2',tipos', 'VariableNames', nombres_columnas);
df = table(data(:, 1), data(:, 2), data(:, 3), data(:, 4), 'VariableNames',
nombres_columnas);
% Mostramos el DataTable
disp('DataTable:');
disp(df);
% Número de filas en df
num_filas = size(df, 1);
fprintf('Filas de df: %d\n', num_filas);
% Veces que aparecen los valores de la columna 'Tipo'
conteo_tipo = countcats(categorical(df.Tipo));
fprintf('Número de tipos distintos: %d\n', numel(conteo_tipo));
% Sumario general
disp('Sumario general:');
s = summary(df);
disp(s);
disp('Sumario de un contaminante');
disp(s.Contaminante1);
```

```
Sumario general:
              Est: [1×1 struct]
    Contaminante1: [1×1 struct]
    Contaminante2: [1×1 struct]
```

```
            Tipo: [1×1 struct]

Sumario de un contaminante:

  struct with fields:

          Size: [4 1]
          Type: 'double'
   Description: ''
         Units: ''
    Continuity: []
           Min: 12
        Median: 25
           Max: 50
    NumMissing: 0


DataTable:
     Est     Contaminante1    Contaminante2      Tipo
    _____    _____________    _____________    _________

    {'A'}       {[30]}           {[14]}        {'Train'}
    {'B'}       {[50]}           {[12]}        {'Test' }
    {'C'}       {[20]}           {[ 7]}        {'Train'}
    {'D'}       {[12]}           {[ 4]}        {'Val'  }

Filas de df: 4
Número de tipos distintos: 3
Sumario general:
             Est: [1×1 struct]
    Contaminante1: [1×1 struct]
    Contaminante2: [1×1 struct]
             Tipo: [1×1 struct]
```

1.23. Tratamiento de datos perdidos

Los comandos para Python son:

- Número de filas con NaN: **df.isna().sum().sum()**
- Eliminar filas con NaN: **df.dropna()**
- Reemplazar NaN con un valor: **df.fillna ()**
- Borrar filas duplicadas: **df.drop_duplicates()**

En Matlab son:

- Número de filas con NaN: **num_rows_with_nan = sum(any(isnan(T{:,:}), 2));**
  disp(['Número de filas con NaN: ', num2str(num_rows_with_nan)]);
- Eliminar filas con NaN: **T_cleaned = rmmissing(T, 'row');**
- Reemplazar NaN con un valor: **T_filled = fillmissing(T, 'constant', 0);**
  % Reemplazar NaN con 0

- Borrar filas duplicadas: **T_unique = unique(T);**

Ejercicio 74_01. Tratamiento de datos perdidos en Python

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
df = pd.DataFrame({"Est": ["A", "B", "C", "D"], "Contaminante": [np.nan, 'SO2',
'SO2', 'SO2'], "Tipo": ["Train", "Test", "Test", "Train"]})
print('Dataset que se ha cargado: ')
print(df)
print('Número de filas con NaN en el dataset: ')
nulos =  df.isna().sum().sum()
print(nulos)
print('Eliminamos las filas con NaN: ')
df2 =  df.dropna()
print(df2)
print('Reemplazar NaN con un PM10: ')
df3 =  df.fillna('PM10')
print(df3)
print('Borrar filas duplicadas')
df4 = df.drop_duplicates()
print(df4)
```

```
Dataset que se ha cargado:
     Est       Contaminante    Tipo
0    A           NaN          Train
1    B           SO2          Test
2    C           SO2          Test
3    D           SO2          Train
Número de filas con NaN en el dataset:
1
Eliminamos las filas con NaN:
     Est       Contaminante    Tipo
1    B           SO2          Test
2    C           SO2          Test
3    D           SO2          Train
Reemplazar NaN con un PM10:
     Est       Contaminante    Tipo
0    A           PM10         Train
1    B           SO2          Test
2    C           SO2          Test
3    D           SO2          Train
Borrar filas duplicadas
     Est       Contaminante    Tipo
0    A           NaN          Train
1    B           SO2          Test
3    D           SO2          Train
```

Ejercicio 74_02. Tratamiento de datos perdidos en MATLAB

```
est = {'A';'B';'C';'D'};
```

```
contam = {''; 'SO2'; 'SO2'; 'SO2'};
tipo = {'Train';'Test';'Test';'Train'};
df = table(est, contam, tipo)
% Mostramos el DataTable original
disp('DataTable original:');
disp(df);
% Número de filas con missing values en el DataTable
nulos = ismissing(df)

DataTable original:
     est       contam        tipo
    _____    __________    _________

    {'A'}    {0×0 char}    {'Train'}
    {'B'}    {'SO2'   }    {'Test' }
    {'C'}    {'SO2'   }    {'Test' }
    {'D'}    {'SO2'   }    {'Train'}

nulos =

  4×3 logical array

   0   1   0
   0   0   0
   0   0   0
   0   0   0

% Reemplazar valores faltantes con 'PM10'
df2 = df;
nanIndices = ismissing(df2.contam);
df2.contam(nanIndices) = {'PM10'};
% Mostrar el DataFrame con valores faltantes reemplazados
disp('DataTable con valores faltantes reemplazados:');
disp(df2);

df2 = 4×3 table
     est      contam       tipo
    _____    ________    _________

    {'A'}    {'PM10'}    {'Train'}
    {'B'}    {'SO2' }    {'Test' }
    {'C'}    {'SO2' }    {'Test' }
    {'D'}    {'SO2' }    {'Train'}

% Eliminar filas duplicadas basadas en todas las columnas
df3 = unique(df2, 'rows', 'stable');
% Eliminar filas duplicadas sin tener en cuenta la columna 'estación'
colsToConsider = {'contam', 'tipo'};
df3 = unique(df2(:, colsToConsider), 'rows', 'stable');
% Mostrar el DataFrame sin filas duplicadas
disp('DataTable sin filas duplicadas:');
disp(df3);
```

```
DataTable sin filas duplicadas:
     contam       tipo
    ________    _________

    {'PM10'}    {'Train'}
    {'SO2' }    {'Test' }
    {'SO2' }    {'Train'}
```

1.24. Cargar CSV desde URL en Python y MATLAB

Características de los datos en ficheros CSV en Python:

• Pueden incluir una cabecera con los nombres de las columnas

• Pueden incluir comentarios (# al comienzo de una línea).

• Pueden usar diferentes tipos de delimitadores (el estándar es la coma, pero pueden usar punto y coma, tabulaciones, etc).

• Los valores que contiene el fichero CSV puede que tengan espacios En ese caso, deberían venir entre comillas dobles.

Podemos copiar todos los datos que aparecen en un Excel. Bien guardarlo en este caso lo guardamos como CSV, y también podemos guardarlo como libro de Excel.

Dataset para pruebas:

• Dataset "Algerian Forest Fires":

ALgerian forest dataset

• Dataset "Wine Quality":

Wine quality dataset

• Dataset "Cars" (aula en el Campus Virtual).

Cargar CSV desde URL en Python

• Dimensiones: **df.shape()**

• Primeras filas: **df.head(valor)**

• Últimas filas: **df.tail(valor)**

Características de los datos en ficheros CSV en MATLAB, lectura de datos:

Se usa readTable() o readmatrix() para datos numericos, antes se usaba csvread() y también xlsread() pero ya están obsoletas.

```
A = readmatrix(filename)
A = readmatrix(filename,opts)
A = readmatrix(___,Name,Value)
```

Description

Ejemplo:

A = readmatrix(filename) creates an array by reading column-oriented data from a file. The readmatrix function performs automatic detection of import parameters for your file.
readmatrix determines the file format from the file extension:

- .txt, .dat, or .csv for delimited text files
- .xls, .xlsb, .xlsm, .xlsx, .xltm, .xltx, or .ods for spreadsheet files

For files containing mixed numeric and text data, readmatrix imports the data as a numeric array by default.

Ejemplo:

A = readmatrix(filename,opts) additionally uses the import options opts.

Podemos copiar todos los datos que aparecen en las bases de datos en un Excel o venir directamente en formato Excel.Podemos guardarlo como CSV, o bien podemos guardarlo como libro de Excel. Una vez en Excel podemos separar esta base de datos en diferentes columnas para poder manejarlo más cómodamente en MATLAB, tratando a esta base de datos como si fuera una matriz, con sus filas y sus columnas.

Hay que tener en cuenta que, al trabajar con una base de datos en Excel, si queremos calcular con números tendremos que quitar las cabeceras de las columnas pues suelen ser letras.

En esta práctica trabajaremos con esta base de datos en CSV.

Nota: Se adjuntan estas bases de datos en el Campus Virtual.

Ejercicio 75_01. Cargar datos CSV desde URL en Python

```
import pandas as pd
# Importante: Nos permite continuar la URL en la línea de abajo
url = ("https://archive.ics.uci.edu/ml/machine-learning-
databases/00547""/Algerian_forest_fires_dataset_UPDATE.csv")
# En la primera fila aparece un título. La saltamos con skiprows = 1
# En la segunda hay un encabezado con los nombres de las columnas con lo que no
necesitamos añadirlos
datos = pd.read_csv(url, skiprows = 1)
print('Dimensiones de los datos cargados: ')
print(datos.shape)
print('\n Mostramos las 5 primeras filas: ')
print(datos.head)
print('\n Mostramos las 2 primeras filas: ')
print(datos.head(2))
print('\n Mostramos las 5 últimas filas: ')
print(datos.tail)
print('\n Mostramos las 2 últimas filas: ')
print(datos.tail(2))
```

Dimensiones de los datos cargados:

```
   (246, 14)

Mostramos las 5 primeras filas:
  day  month    year   Temperature  RH    Ws    Rain    FFMC   DMC    DC
0   1     06    2012     29         57    18     0      65.7   3.4    7.6
1   2     06    2012     29         61    13     1.3    64.4   4.1    7.6
2   3     06    2012     26         82    22    13.1    47.1   2.5    7.1
3   4     06    2012     25         89    13     2.5    28.6   1.3    6.9
4   5     06    2012     27         77    16     0      64.8   3     14.2

ISI     BUI     FWI      Classes
1.3     3.4      0.5     not fire
1       3.9      0.4     not fire
0.3     2.7      0.1     not fire
0       1.7      0       not fire
1.2     3.9      0.5     not fire

Mostramos las 2 primeras filas:
  day  month    year   Temperature  RH      Ws    Rain     FFMC    DMC      DC
0   1     06    2012     29         57      18       0     65.7    3.4      7.6
1   2     06    2012     29         61      13     1.3     64.4    4.1      7.6

ISI     BUI     FWI      Classes
1.3     3.4     0.5     not fire
1       3.9     0.4     not fire

Mostramos las 5 últimas filas:
     day  month    year   Temperature  RH     Ws    Rain     FFMC    DMC      DC
241  26      9     2012     30       65     14       0     85.4     16     44.5
242  27      9     2012     28       87     15     4.4     41.1    6.5        8
243  28      9     2012     27       87     29     0.5     45.9    3.5      7.9
244  29      9     2012     24       54     18     0.1     79.7    4.3     15.2
245  30      9     2012     24       64     15     0.2     67.3    3.8     16.5

     ISI     BUI      FWI     Classes
241  4.5     16.9     6.5     fire
242  0.1      6.2     0       not fire
243  0.4      3.4     0.2     not fire
244  1.7      5.1     0.7     not fire
245  1.2      4.8     0.5     not fire

Mostramos las 2 últimas filas:
     day  month     year   Temperature  RH     Ws    Rain     FFMC    DMC      DC
244  29      9     2012     24         54     18     0.1     79.7    4.3     15.2
245  30      9     2012     24         64     15     0.2     67.3    3.8     16.5

     ISI     BUI      FWI     Classes
244  1.7     5.1      0.7     not fire
245  1.2     4.8      0.5     not fire
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

Ejercicio 75_02. Cargar datos CSV desde URL en MATLAB

```
% URL del conjunto de datos
url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/00547/Algerian_forest_fires_dataset_UPDATE.csv';
% Cargar los datos desde la URL
datos = readtable(url, 'ReadVariableNames', true);
% Mostrar dimensiones de los datos cargados
disp('Dimensiones de los datos cargados:');
disp(size(datos));
% Mostrar las 5 primeras filas
disp('Mostramos las 5 primeras filas:');
disp(datos(1:5, :));
% Mostrar las 2 primeras filas
disp('Mostramos las 2 primeras filas:');
disp(datos(1:2, :));
% Mostrar las 5 últimas filas
disp('Mostramos las 5 últimas filas:');
disp(datos(end-4:end, :));
% Mostrar las 2 últimas filas
disp('Mostramos las 2 últimas filas:');
disp(datos(end-1:end, :));
```

```
Dimensiones de los datos cargados:
   246    14

Mostramos las 5 primeras filas:
    day    month    year    Temperature    RH    Ws    Rain    FFMC    DMC    DC
    ___    ____    ____    ____    ____    ____    ____    ____    ____    _____
    1      6       2012    29      57      18      0       65.7    3.4     7.6
    2      6       2012    29      61      13      1.3     64.4    4.1     7.6
    3      6       2012    26      82      22      13.1    47.1    2.5     7.1
    4      6       2012    25      89      13      2.5     28.6    1.3     6.9
    5      6       2012    27      77      16      0       64.8    3       14.2

    ISI    BUI      FWI       Classes
    ___    _____    _____    _____________
    1.3     3.4      0.5     {'not fire'}
    1       3.9      0.4     {'not fire'}
    0.3     2.7      0.1     {'not fire'}
    0       1.7        0     {'not fire'}
    1.2     3.9      0.5     {'not fire'}

Mostramos las 2 primeras filas:
    day    month    year    Temperature    RH    Ws    Rain    FFMC    DMC    DC
    ___    ____    ____    ____    ____    ____    ____    ____    ____    _____
    1      6       2012    29      57      18      0       65.7    3.4     7.6
    2      6       2012    29      61      13      1.3     64.4    4.1     7.6

    ISI    BUI      FWI       Classes
    ___    _____    _____    _____________
    1.3     3.4      0.5     {'not fire'}
    1       3.9      0.4     {'not fire'}
```

```
Mostramos las 5 últimas filas:
day     month     year     Temperature     RH     Ws     Rain     FFMC     DMC     DC
____    ____    ____    ____    ____    ____    ____    ____    ____    _____
26      9       2012     30      65      14       0      85.4     16     44.5
27      9       2012     28      87      15     4.4      41.1    6.5       8
28      9       2012     27      87      29     0.5      45.9    3.5     7.9
29      9       2012     24      54      18     0.1      79.7    4.3     15.2
30      9       2012     24      64      15     0.2      67.3    3.8     16.5

ISI     BUI      FWI       Classes
____    _____    _____     ____________
4.5     16.9      6.5      {'fire'    }
0.1      6.2        0      {'not fire'}
0.4      3.4      0.2      {'not fire'}
1.7      5.1      0.7      {'not fire'}
1.2      4.8      0.5      {'not fire'}

Mostramos las 2 últimas filas:
 day     month     year     Temperature     RH     Ws     Rain     FFMC     DMC     DC
____    ____    ____    ____    ____    ____    ____    ____    ____    _____
29      9       2012     24      54      18     0.1      79.7    4.3     15.2
30      9       2012     24      64      15     0.2      67.3    3.8     16.5

ISI     BUI      FWI       Classes
____    _____    _____     ____________
1.7      5.1      0.7      {'not fire'}
1.2      4.8      0.5      {'not fire'}
```

1.25. Cargar CSV desde fichero en Python y MATLAB

Ejercicio 76_01. Cargar datos CSV desde fichero en Python

```
import pandas as pd
# Cargamos un primer dataset de calidad de vinos:
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae.
# Se usa como separador, y no la coma por defecto
datosWines = pd.read_csv(nomFichero, sep=";")
print('Mostramos dimensiones: ')
print(datosWine.shape)
print('Imprimimos 3 primeras filas de los datos que hemos cargado: ')
print(datosWine.head(3))
# Cargamos otro dataset de coches:
# Una inspección del csv abriéndolo con el notepad nos indica que la separación es
por comas (La que read_csv espera por defecto), y que no hay header con los nombres
de las variables. Los pondremos en código, y son los siguientes: buying, moint,
doors, persons, Lug_boot, safety, category
nomFichero = 'cars.csv'
```

```
nombresVars = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety',
'category']
datosCars = pd.read_csv(nomFichero, names=nombresVars)
# datosCars = pd.read_csv(nomFinero)
print('Mostramos dimensiones: ')
print(datosCars.shape)
print('Imprimimos 3 primeras filas de los datos que hemos cargado: ')
print(datosCars.head(3))

Mostramos dimensiones del conjunto de datos de vinos:
        1599          12
Imprimimos 3 primeras filas del conjunto de datos de vinos:
fixedAcidity    volatileAcidity    citricAcid    residualSugar    chlorides
_____________    _______________    __________    _____________    _________
7.4              0.7                0             1.9              0.076
7.8              0.88               0             2.6              0.098
7.8              0.76               0.04          2.3              0.092

freeSulfurDioxide    totalSulfurDioxide    density     pH     sulphates    alcohol
_________________    __________________    _______    ____    _________    _______
11                    34                   0.9978     3.51      0.56       9.4
25                    67                   0.9968     3.2       0.68       9.8
15                    54                   0.997      3.26      0.65       9.8

quality
_______
5
5
5

Mostramos dimensiones del conjunto de datos de coches:
        1728           7
Imprimimos 3 primeras filas del conjunto de datos de coches:
     bying     maint    doors   persons   lug_boot    safety    category
0    vhigh       vhigh     2       2        small      low      unacc
1    vhigh       vhigh     2       2        small      med      unacc
2    vhigh       vhigh     2       2        small      high     unacc
```

Ejercicio 76_02. Cargar datos CSV desde fichero en MATLAB

```
% Cargar el primer conjunto de datos de calidad de vinos
nomFicheroWines = 'winequality-red.csv';
datosWines = readtable(nomFicheroWines, 'Delimiter', ',');
% Mostrar dimensiones
disp('Mostramos dimensiones del conjunto de datos de vinos:');
disp(size(datosWines));
% Mostrar las 3 primeras filas
disp('Imprimimos 3 primeras filas del conjunto de datos de vinos:');
disp(datosWines(1:3, :));
% Cargar el segundo conjunto de datos de coches
nomFicheroCars = 'cars.csv';
```

```
nombresVarsCars = {'buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety',
'category'};
datosCars = readtable(nomFicheroCars, 'Delimiter', ',');
% Mostrar dimensiones
disp('Mostramos dimensiones del conjunto de datos de coches:');
disp(size(datosCars));
% Mostrar las 3 primeras filas
disp('Imprimimos 3 primeras filas del conjunto de datos de coches:');
disp(datosCars(1:3, :));
```

```
Mostramos dimensiones del conjunto de datos de vinos:
        1599          12
Imprimimos 3 primeras filas del conjunto de datos de vinos:
    fixedAcidity    volatileAcidity    citricAcid    residualSugar    chlorides
    ____________    _______________    __________    _____________    _________
    7.4              0.7                0              1.9              0.076
    7.8              0.88               0              2.6              0.098
    7.8              0.76               0.04           2.3              0.092

    freeSulfurDioxide    totalSulfurDioxide    density     pH      sulphates    alcohol
    _________________    __________________    _______    ____    _________    _______
    11                    34                    0.9978     3.51       0.56      9.4
    25                    67                    0.9968     3.2        0.68      9.8
    15                    54                    0.997      3.26       0.65      9.8

    quality
    _______
    5
    5
    5

Mostramos dimensiones del conjunto de datos de coches:
        1728           7
Imprimimos 3 primeras filas del conjunto de datos de coches:
     bying        maint       doors    persons    lug_boot       safety      category
    _________    _________    _____    _______    _________    ________    _________
    {'vhigh'}    {'vhigh'}     2        2       {'small'}    {'low' }    {'unacc'}
    {'vhigh'}    {'vhigh'}     2        2       {'small'}    {'med' }    {'unacc'}
    {'vhigh'}    {'vhigh'}     2        2       {'small'}    {'high'}    {'unacc'}
```

1.26. Guardar datos en CSV

Ejercicio 77_01. Guardar datos en CSV en Python

```
import pandas as pd
df = pd.DataFrame({'Est': ['1', '2'], 'Contaminante': ['NO2', 'SO2'], 'Tipo':
['Train', 'Test']})
print(df)
# Guardamos con las cabeceras (header = True)
df.to_csv("prueba2.csv", header = True, index = False)
# Lo cargamos y comprobamos
```

```
nomFichero = 'prueba2.csv'
datos = pd.read_csv(nomFichero)
print(datos)

     Est   Contaminante  Tipo
 0   1     NO2           Train
 1   2     SO2           Test

    Est   Contaminante   Tipo
0   1     NO2            Train
1   2     SO2            Test
```

Ejercicio 77_02. Guardar datos en CSV en MATLAB

```
% Crear un DataTable similar a un DataFrame en pandas
data = { '1', 'NO2', 'Train';
    '2', 'SO2', 'Test'};
nombres_columnas = {'Est', 'Contaminante', 'Tipo'};
df = table(data(:, 1), data(:, 2), data(:, 3), 'VariableNames', nombres_columnas);
% Mostrar el DataTable original
disp('DataTable original:');
disp(df);
% Guardar el DataTable en un archivo CSV con cabeceras
writetable(df, 'prueba2.csv', 'WriteVariableNames', true);
% Cargar el archivo CSV y comprobar
datos = readtable('prueba2.csv');
disp('Datos cargados desde prueba2.csv:');
disp(datos);

DataTable original:
    Est     Contaminante      Tipo
    _____    ____________    _________

    {'1'}      {'NO2'}       {'Train'}
    {'2'}      {'SO2'}       {'Test' }

Datos cargados desde prueba2.csv:
    Est    Contaminante      Tipo
    ___    ____________    _________

     1       {'NO2'}       {'Train'}
     2       {'SO2'}       {'Test' }
```

1.27. Aplicar funciones a dataframes y tablas

Funciones aplicables: **sum()**, **min()**, **max()**, **mean()**, **var()**, **std()**, **count()**, **size()**, **median()**, **apply(function)**.

Pueden aplicarse a series, agrupaciones (ejemplo ejercicio 83_01. Agrupamientos en dataframes en Python) y columnas.
Otras funciones útiles:

- Seleccionar aleatoriamente una fracción de filas: **df.sample(frac=0.5)**
- Seleccionar n filas aleatoriamente: **df.sample(n=10)**
- Seleccionar y ordenar las n entradas más grandes: **df.nlargest(n, 'columna')**
- Seleccionar y ordenar las n entradas más pequeñas: **df.nsmallest(n, 'columna')**
- Ordenar valores: **df.sort_values('name', ascending = true)**
- Ordenar índice: **df.sort_index()**
- Renombrar columnas: **df.rename(columns={'n':'nombre'})**
- Borrar columnas: **df.drop(columns=['nombre','edad'])**

En Matlab: **sum(), min(), max(), mean(), var(), std(), count(), size(), median(), arrayfun(@(x) x^2, A);** % Aplicar función x^2 a cada elemento de A.

Ejercicio 78_01. Aplicar funciones a dataframes en Python

```
# Importamos las librerías necesarias
import pandas as pd
import numpy as np

# Creamos un dataframe con datos aleatorios (5 filas y 4 columnas)
# Para los nombres de filas se usan las fechas, para las columnas la lista
df = pd.DataFrame(np.random.randn(5, 4))
print("Dataframe generado: ")
print(df)
df2 = df.apply(np.cumsum)
print("\n Aplicamos función que calcula la suma acumulativa por columna: \n")
print(df2)
df3 = df.mean(axis=0)
print("\n Aplicamos función que calcula la media por columna: \n")
print(df3)
df4 = df.mean(axis=1)
print("\n Aplicamos función que calcula la media por fila: \n")
print(df4)
# Lambda son pequeñas funciones anónimas
df5 = df.apply(lambda x: x.max() - x.min())
print("\n Aplicamos función que calcula máximo menos mínimo por columna: \n")
print(df5)
```

```
Dataframe generado:
0 1 2 3
0 -1.856295 0.132211 0.107456 1.877389 1 -1.857194 -1.419337 -0.493022 -1.434923 2 -
1.782115 -0.329186 -1.112902 -1.141936 3 1.083036 0.184746 -0.088146 -0.034818 4
0.482112 -0.621373 1.612355 1.199380

Aplicamos función que calcula la suma acumulativa por columna:

0 1 2 3
0 -1.856295 0.132211 0.107456 1.877389 1 -3.713490 -1.287127 -0.385566 0.442466 2 -
5.495604 -1.616312 -1.498468 -0.699470 3 -4.412568 -1.431566 -1.586613 -0.734288 4 -
3.930456 -2.052939 0.025742 0.465091
```

```
Aplicamos función que calcula la media por columna:
0 -0.786091
1 -0.410588
2 0.005148
3 0.093018
dtype: float64

 Aplicamos función que calcula la media por fila: 0 0.065190
1 -1.301119
2 -1.091534
3 0.286205
4 0.668118
dtype: float64

Aplicamos función que calcula máximo menos mínimo por columna:
0 2.940230
1 1.604084
2 2.725257
3 3.312312
dtype: float64
```

Ejercicio 78_02. Aplicar funciones a tablas en MATLAB

```
% Creamos un DataTable con datos aleatorios (5 filas y 4 columnas)
x = randn(5, 4);
df = table(x(:,1),x(:,2),x(:,3),x(:,4),...
          'VariableNames',{'A','B','C','D'},...
  'RowNames',{'2022-01-01'; '2022-01-02'; '2022-01-03'; '2022-01-04'; '2022-01-05'})
% Mostramos el DataTable generado
disp('DataTable generado:');
disp(df);
% Aplicamos la función sum() por columna
df_sum = varfun(@sum, df);
disp('Suma por columna:');
disp(df_sum);
% Aplicamos la función cumsum() por columna
df_cumsum = varfun(@cumsum, df);
disp('Suma acumulativa por columna:');
disp(df_cumsum);
% Aplicamos la función mean() por columna
df_mean_col = varfun(@mean, df);
disp('Media por columna:');
disp(df_mean_col);
% Creamos la función anónima para calcular max() - min() por columna
df_max_min_diff = varfun(@(x) max(x) - min(x), df);
disp('Máximo menos mínimo por columna:');
disp(df_max_min_diff);
% Aplicamos la función mean() por fila
df_mean_row = mean(df{:,:}, 2); % dimensión 2 indica que haga la media por filas de
las columnas
disp('Media por fila:');
disp(df_mean_row);
```

```
x =

    2.4245    1.8779   -0.5583   -0.2099
    0.9594    0.9407   -0.3114   -1.6989
   -0.3158    0.7873   -0.5700    0.6076
    0.4286   -0.8759   -1.0257   -0.1178
   -1.0360    0.3199   -0.9087    0.6992

df = 5×4 table
                    A           B           C           D
                _________    _________    _________    _______

    2022-01-01      2.4245      1.8779    -0.55829    -0.2099
    2022-01-02      0.9594      0.9407    -0.31143    -1.6989
    2022-01-03    -0.31577     0.78735    -0.57001     0.6076
    2022-01-04     0.42862    -0.87587     -1.0257    -0.1178
    2022-01-05      -1.036     0.31995    -0.90875    0.69916

DataTable generado:
                    A           B           C           D
                _________    _________    _________    _______

    2022-01-01      2.4245      1.8779    -0.55829    -0.2099
    2022-01-02      0.9594      0.9407    -0.31143    -1.6989
    2022-01-03    -0.31577     0.78735    -0.57001     0.6076
    2022-01-04     0.42862    -0.87587     -1.0257    -0.1178
    2022-01-05      -1.036     0.31995    -0.90875    0.69916

Suma por columna:
    sum_A     sum_B     sum_C      sum_D
    ______    _____    _______    _______

    2.4607    3.05     -3.3742    -0.7198

Suma acumulativa por columna:
    cumsum_A    cumsum_B    cumsum_C    cumsum_D
    ________    ________    ________    ________

     2.4245      1.8779     -0.55829    -0.2099
     3.3839      2.8186     -0.86972    -1.9088
     3.0681      3.6059      -1.4397    -1.3012
     3.4967        2.73      -2.4655     -1.419
     2.4607        3.05      -3.3742    -0.7198

Media por columna:
    mean_A     mean_B     mean_C      mean_D
    _______    ______    ________    ________

    0.49215     0.61     -0.67484    -0.14396

Máximo menos mínimo por columna:
    Fun_A     Fun_B     Fun_C     Fun_D
    ______    ______    ______    ______
```

```
    1.3543    0.6783    2.3467    2.6604

Media por fila:
   -0.4249
   -0.6928
   -0.3411
   -0.8122
    0.0846
```

1.28. Concatenar dataframes y tablas

Ejercicio 79_01. Concatenar dataframes en Python

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
df1 = pd.DataFrame(np.random.randn(3, 3))
df2 = pd.DataFrame(np.random.randn(3, 3))
print('Dataframe original 1: ')
print(df1)
print('\n Dataframe original 2: ')
print(df2)
df3 = pd.concat([df1, df2])
print('\n Dataframe concatenando los anteriores en filas: ')
print(df3)
df4 = pd.concat([df1, df2], axis=1)
print('\n Dataframe concatenando los anteriores en columnas: ')
print(df4)
```

```
Dataframe original 1:
0 1 2
0 -0.551075 0.765915 -0.499833
1 0.817082 -0.350316 1.319587
2 -0.225708 0.170537 0.351697
Dataframe original 2:
0 1 2
0 0.364047 -0.387928 -2.339652
1 0.595656 0.830094 1.349182 2 -0.410960 -0.373968 0.428950
Dataframe concatenando los anteriores en filas: 0 1 2
0 -0.551075 0.765915 -0.499833
1 0.817082 -0.350316 1.319587
2 -0.225708 0.170537 0.351697
0 0.364047 -0.387928 -2.339652
1 0.595656 0.830094 1.349182 2 -0.410960 -0.373968 0.428950
Dataframe concatenando los anteriores en columnas:
0 1 2 0 1 2
0 -0.551075 0.765915 -0.499833 0.364047 -0.387928 -2.339652
1 0.817082 -0.350316 1.319587 0.595656 0.830094 1.349182 2 -0.225708 0.170537
0.351697 -0.410960 -0.373968 0.428950
```

Ejercicio 79_02. Concatenar dataframes en MATLAB

```
% Creamos un DataTable con datos aleatorios (5 filas y 4 columnas)
x1 = randn(3, 3);
x2 = randn(3, 3);
x3 = randn(3, 3);
df1 = table(x1(:,1),x1(:,2),x1(:,3),...
          'VariableNames',{'A','B','C'});
df2 = table(x2(:,1),x2(:,2),x2(:,3),...
          'VariableNames',{'D','E','F'});
df3 = table(x3(:,1),x3(:,2),x3(:,3),...
          'VariableNames',{'A','B','C'});
% Mostramos las tablas generadas
disp('Tabla 1 generado:');
disp(df1);
disp('Tabla 2 generado:');
disp(df2);
disp('Tabla 3 generado:');
disp(df3);
% Concatenar los DataTables en columnas
df4 = [df1, df2];
disp('DataTable concatenando los anteriores en columnas:');
disp(df4);
% Concatenar los DataTables en filas
df5 = [df1; df3];
disp('DataTable concatenando los anteriores en filas:');
disp(df5);
```

```
Tabla 1 generado:
        A           B           C
    ________    ________    ________

      -1.8288     0.44892      -3.073
       1.3845    -0.36326     0.62628
    -0.062727     -1.0206    -0.28668

Tabla 2 generado:
       D           E           F
    ________    ________    ________

    -0.19734    -0.72945     -1.2813
     0.40561      1.1473     -2.2033
     -1.4193     0.59786    -0.57125

Tabla 3 generado:
       A           B           C
    ________    _______    ________

       0.214    -1.1223    -0.96097
     0.94238    0.30616    -0.65374
    0.093725    -1.1723     -1.2294
```

```
DataTable concatenando los anteriores en columnas:
        A           B           C           D           E           F
    ________    ________    ________    ________    ________    ________

      -1.8288     0.44892      -3.073    -0.19734    -0.72945     -1.2813
       1.3845    -0.36326     0.62628     0.40561      1.1473     -2.2033
    -0.062727     -1.0206    -0.28668     -1.4193     0.59786    -0.57125

DataTable concatenando los anteriores en filas:
        A           B           C
    ________    ________    ________

      -1.8288     0.44892      -3.073
       1.3845    -0.36326     0.62628
    -0.062727     -1.0206    -0.28668
        0.214     -1.1223    -0.96097
      0.94238     0.30616    -0.65374
     0.093725     -1.1723     -1.2294
```

1.29. Selección subconjunto de filas loc-iloc

- **NOTA:** loc-iloc indica selección por localización e índice de localización.

Ejercicio 80_01. Seleccionar subconjunto de filas loc-iloc en Python

```
# df.loc podemos seleccionar filas según su etiqueta
# df.iloc seleccionamos según la posición
%% Importamos las librerías necesarias
import numpy as np
import pandas as pd
df = pd.DataFrame({'est': ['A', 'B', 'C', 'D'], 'contaminante': [np.nan, 'SO2',
'NO2', 'SO2'], 'tipo': ['Train', 'Test', 'Train', 'Test'], 'valor': [10, 22, 30, 2]})
df.index = ['Lunes', 'Martes', 'Miércoles','Jueves']
print(df)

                est  contaminante    tipo    valor
    Lunes        A      NaN         Train     10
    Martes       B      SO2         Test      22
    Miércoles    C      NO2         Train     30
    Jueves       D      SO2         Test      2


# selección por una sola fila
print(df.loc['Lunes'])

est                A
contaminante       NaN
tipo               Train
```

```
valor               10
Name: Lunes, dtype: object

# selección por un rango de filas

print(df.loc['Lunes':'Miércoles'])

                est     contaminante   tipo     valor

    Lunes        A      NaN            Train     10
    Martes       B      SO2            Test      22
    Miércoles    C      NO2            Train     30

# selección de filas y columnas distintas

print(df.loc['Lunes':'Miércoles'][['contaminante', 'tipo']])

                contaminante   tipo
    Lunes          NaN        Train
    Martes         SO2        Test
    Miércoles      NO2        Train

# selección por posición iloc
print(df.iloc[0:2, 2])

Lunes         Train
Martes        Test
Name: tipo, dtype: object

print(df.iloc[0, 2])

Train

# selección según criterio lógico
df[df.valor > 10]

                est     contaminante   tipo     valor
    Martes       B      SO2            Test      22
    Miércoles    C      NO2            Train     30
```

Ejercicio 80_02. Seleccionar subconjunto de columnas en MATLAB

```
% Creamos un DataTable similar al DataFrame en pandas
data = {
    'A', NaN, 'Train', 10;
    'B', 'SO2', 'Test', 22;
    'C', 'NO2', 'Train', 30;
    'D', 'SO2', 'Test', 2;
};
filas = {'Lunes', 'Martes', 'Miércoles', 'Jueves'};
columnas = {'est', 'contaminante', 'tipo', 'valor'};
```

```
df = table(data(:, 1), data(:, 2), data(:, 3), cell2mat(data(:, 4)), 'RowNames',
filas, 'VariableNames', columnas);
% Mostramos el DataTable generado
disp('DataTable generado:');
disp(df);

DataTable generado:
                 est     contaminante      tipo       valor
                _____    ____________    _________    _____

    Lunes       {'A'}      {[NaN]}       {'Train'}     10
    Martes      {'B'}      {'SO2'}       {'Test' }     22
    Miércoles   {'C'}      {'NO2'}       {'Train'}     30
    Jueves      {'D'}      {'SO2'}       {'Test' }      2

% Selección por una sola fila
disp('Selección por una sola fila:');
disp(df('Lunes', :));

Selección por una sola fila:
             est     contaminante      tipo       valor
            _____    ____________    _________    _____

    Lunes   {'A'}      {[NaN]}       {'Train'}     10

% Selección por rango de filas
disp('Selección por rango de filas:');
% disp(df('Lunes':'Miércoles', :)); % No sirve con string
disp(df(1:3,:));

Selección por rango de filas:
                 est     contaminante      tipo       valor
                _____    ____________    _________    _____

    Lunes       {'A'}      {[NaN]}       {'Train'}     10
    Martes      {'B'}      {'SO2'}       {'Test' }     22
    Miércoles   {'C'}      {'NO2'}       {'Train'}     30

% Selección de columnas específicas por rango de filas
disp('Selección de columnas específicas por rango de filas:');
% disp(df('Lunes':'Miércoles', {'contaminante', 'tipo'})); % No sirve con string
disp(df(1:3, 2:3));

Selección de columnas específicas por rango de filas:
                contaminante      tipo
                ____________    _________

    Lunes         {[NaN]}       {'Train'}
    Martes        {'SO2'}       {'Test' }
    Miércoles     {'NO2'}       {'Train'}

% Selección por posición iloc
disp('Selección por posición iloc:');
```

```
disp(df{1:2, 3});

Selección por posición iloc:
    {'Train'}
    {'Test' }

% Selección de un elemento por posición iloc
disp('Selección de un elemento por posición iloc:');
disp(df{'Lunes', 'tipo'});

Selección de un elemento por posición iloc:
    {'Train'}

% Selección según criterio lógico
disp('Selección según criterio lógico:');
disp(df(df.valor > 10, :));

Selección según criterio lógico:
                 est     contaminante      tipo       valor
                _____    ____________    _________    _____
    Martes      {'B'}      {'SO2'}       {'Test' }     22
    Miércoles   {'C'}      {'NO2'}       {'Train'}     30
```

1.30. Selección subconjunto de columnas

Ejercicio 81_01. Seleccionar subconjunto de columnas en Python

```
# df.loc podemos seleccionar filas según su etiqueta
# df.iloc seleccionamos según la posición
# Importamos las librerías necesarias
import numpy as np
import pandas as pd

df = pd.DataFrame({'Est': ['A', 'B', 'C', 'D'], 'contaminante': [np.nan, 'SO2',
'NO2', 'SO2'], 'Tipo': ['Train', 'Test', 'Train', 'Test']})
df.index = ['Lunes', 'Martes', 'Miércoles', 'Jueves']
print(df)
print('\n Puede seleccionarse una columna específica: \n')
print(df[['Est']])
print('\n Seleccionar rango de columnas por nombre: \n')
print(df.loc[:,['contaminante','tipo']])
print('\n Seleccionar rango de columnas por posición: \n')
df.iloc[:,[0,1]])

                Est   contaminante   Tipo
    Lunes       A      NaN           Train
    Martes      B      SO2           Test
    Miércoles   C      NO2           Train
    Jueves      D      SO2           Test
```

```
Puede seleccionarse una columna específica:
 Lunes      A
 Martes     B
 Miércoles  C
 Jueves     D

Seleccionar rango de columnas por nombre:
                contaminante  Tipo
    Lunes            NaN     Train
    Martes           SO2     Test
    Miércoles        NO2     Train
    Jueves           SO2     Test

Seleccionar rango de columnas por posición:
                Est   contaminante
    Lunes        A      NaN
    Martes       B      SO2
    Miércoles    C      NO2
    Jueves       D      SO2
```

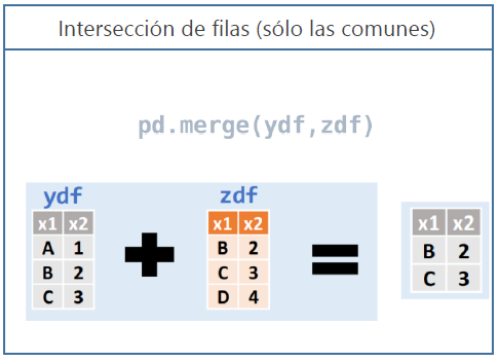
Ejercicio 81_02. Seleccionar subconjunto de columnas en MATLAB

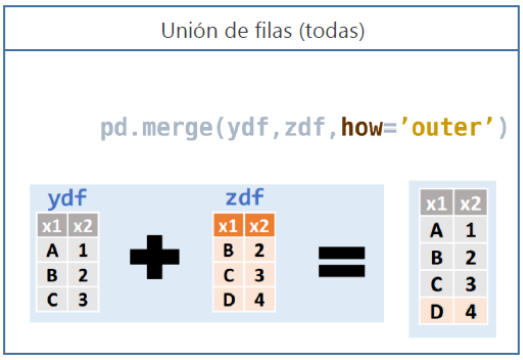
```
% Creamos un DataTable similar al DataFrame en pandas
data = {
    'A', NaN, 'Train';
    'B', 'SO2', 'Test';
    'C', 'NO2', 'Train';
    'D', 'SO2', 'Test';
};
filas = {'Lunes', 'Martes', 'Miércoles', 'Jueves'};
columnas = {'Est', 'contaminante', 'Tipo'};
df = table(data(:, 1), data(:, 2), data(:, 3), 'RowNames', filas, 'VariableNames',
columnas);
% Mostramos el DataTable generado
disp('DataTable generado:');
disp(df);
% Puede seleccionarse una columna específica
disp('Puede seleccionarse una columna específica:');
disp(df.Est);
% Seleccionar rango de columnas por nombre
disp('Seleccionar rango de columnas por nombre:');
disp(df(:, {'contaminante', 'Tipo'}));
% Seleccionar rango de columnas por posición
disp('Seleccionar rango de columnas por posición:');
disp(df(:, 1:2));

DataTable generado:
                  Est     contaminante      Tipo
                 _____    ____________    _________
    Lunes        {'A'}      {[NaN]}       {'Train'}
    Martes       {'B'}      {'SO2'}       {'Test' }
    Miércoles    {'C'}      {'NO2'}       {'Train'}
```

```
    Jueves       {'D'}      {'SO2'}       {'Test' }

Puede seleccionarse una columna específica:
    {'A'}
    {'B'}
    {'C'}
    {'D'}

Seleccionar rango de columnas por nombre:
                contaminante      Tipo
                ____________    _________
    Lunes         {[NaN]}       {'Train'}
    Martes        {'SO2'}       {'Test' }
    Miércoles     {'NO2'}       {'Train'}
    Jueves        {'SO2'}       {'Test' }

Seleccionar rango de columnas por posición:
                 Est     contaminante
                _____    ____________
    Lunes       {'A'}      {[NaN]}
    Martes      {'B'}      {'SO2'}
    Miércoles   {'C'}      {'NO2'}
    Jueves      {'D'}      {'SO2'}
```

1.31. Intersección de datasets con filas comunes

```
# combinar datasets con filas comunes:
# pd.merge(ydf,zdf) # intersección de filas solo comunes
# pd.merge(ydf,zdf, how='outer') # Unión de filas (todas)
```





Ejercicio 82_01. Combinar datasets con merge en Python

```
import pandas as pd

df1 = pd.DataFrame({'clave': ['datos 1', 'datos 2', 'datos 3'], 'valores df1': [1, 2,
3]})
```

```
df2 = pd.DataFrame({'clave': ['datos 1', 'datos 2', 'datos 4'], 'valores df2': [4, 5,
6]})
df3 = pd.merge(df1, df2, on='clave')
print("Dataframe 1: ")
print(df1)
print("\n Dataframe 2: ")
print(df2)
print("\n Dataframe intersección de ambos: ")
print(df3)

DataTable 1:
     clave       valores df1
 0   datos 1         1
 1   datos 2         2
 2   datos 3         3

DataTable 2:
       clave       valores df2
    ___________    ___________

0    datos 1         4
1    datos 2         5
2    datos 4         6

DataTable intersección de ambos:
       clave       valores df1    valores df2
0    datos 1            1             4
1    datos 2            2             5
```

Ejercicio 82_01. Combinar datasets con merge en MATLAB

```
% Creamos dos DataTables similares a los DataFrames en pandas
data1 = {'datos 1', 1; 'datos 2', 2; 'datos 3', 3;};
data2 = {'datos 1', 4; 'datos 2', 5; 'datos 4', 6;};
columnas = {'clave', 'valores_df1'};
df1 = table(data1(:, 1), cell2mat(data1(:, 2)), 'VariableNames', columnas);
df2 = table(data2(:, 1), cell2mat(data2(:, 2)), 'VariableNames', columnas);
% Mostramos los DataTables generados
disp('DataTable 1:');
disp(df1);
disp('DataTable 2:');
disp(df2);
% Fusionamos los DataTables por la columna 'clave'
df3 = outerjoin(df1, df2, 'Keys', 'clave', 'MergeKeys', true);
% Renombramos las variables para reflejar el origen
df3.Properties.VariableNames = {'clave', 'valores_df1_df1', 'valores_df1_df2'};
% Mostramos el DataTable resultante
disp('DataTable intersección de ambos:');
disp(df3);

DataTable 1:
       clave       valores_df1
```

```
    ___________    ___________
    {'datos 1'}          1
    {'datos 2'}          2
    {'datos 3'}          3

DataTable 2:
       clave       valores_df1
    ___________    ___________
    {'datos 1'}          4
    {'datos 2'}          5
    {'datos 4'}          6

DataTable intersección de ambos:
       clave       valores_df1_df1    valores_df1_df2
    ___________    _______________    _______________
    {'datos 1'}           1                  4
    {'datos 2'}           2                  5
    {'datos 3'}           3                NaN
    {'datos 4'}         NaN                  6
```

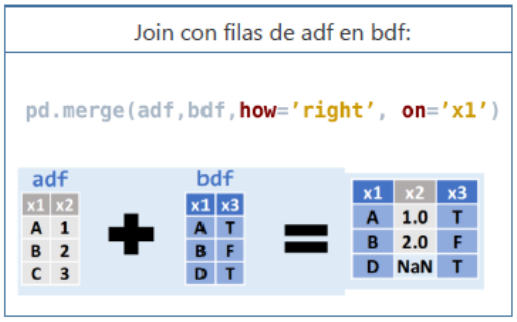
1.32. Combinaciones a nivel columna

Python

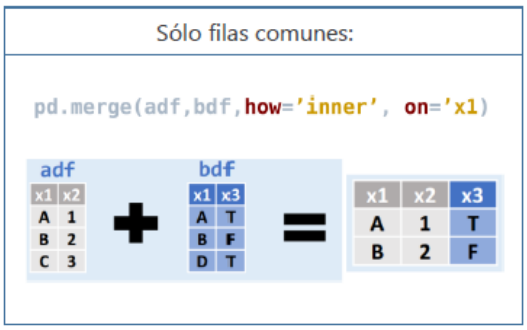
```
# join con filas de bdf en adf (ejemplo de 2 matrices 3x2)
pd.merge(adf,bdf, how='left', on='x1')
```


```
# join con filas de adf en bdf (ejemplo de 2 matrices 3x2)
pd.merge(adf,bdf, how='right', on='x1')
```

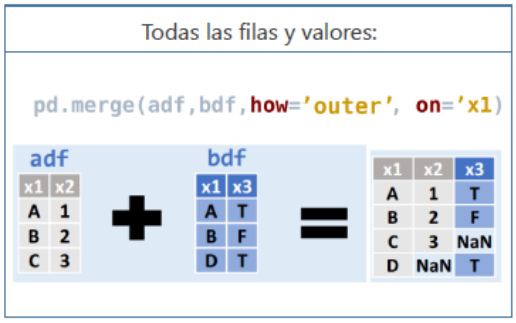
Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.




```
# join sólo filas comunes (ejemplo de 2 matrices 3x2)
pd.merge(adf,bdf, how='inner', on='x1')
```




```
%# join todas las filas y valores (ejemplo de 2 matrices 3x2)
%# pd.merge(adf,bdf, how='outer', on='x1')
```



Ejercicio 83_01. Agrupamientos en dataframes en Python

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
df = pd.DataFrame({'Estaciones': ['Est1', 'Est2', 'Est1', 'Est1', 'Est2', 'Est1',
'Est2', 'Est2'], 'contaminantes': ['CO', 'NO2', 'CO', 'PM10', 'PM10', 'CO', 'PM10',
'O3'], 'tipo': ['Train', 'Test', 'Train', 'Train','Test', 'Test', 'Test', 'Train'],
'concentraciones': np.random.rand(8), # Aleatorias})
print('Dataframe original: ')
```

```
print(df)
print('\n Sumamos agrupando estaciones: ')
df2 = df.groupby('Estaciones').sum()
print(df2)
print('\n Sumamos agrupando por estaciones y contaminantes: ')
df3 = df.groupby(['Estaciones', 'Contaminantes']).sum()
print(df3)
print('\n Sumamos agrupando por estaciones, contaminantes y tipos: ')
df4 = df.groupby(['Estaciones', 'Contaminantes', 'Tipo']).sum()
print(df4)

Dataframe original:
    Estaciones   Contaminantes    Tipo       Concentraciones
    __________   _____________   ________    _______________
 0    Est1       CO              Train       0.53063
 1    Est2       NO              Test        0.83242
 2    Est1       CO              Train       0.59749
 3    Est1       PM10            Train       0.33531
 4    Est2       PM10            Test        0.29923
 5    Est1       CO              Test        0.45259
 6    Est2       PM10            Test        0.42265
 7    Est2       O3              Train       0.35961

Sumamos agrupando estaciones:
    Estaciones   Concentraciones
    Est1               1.916
    Est2               1.9139

Sumamos agrupando por estaciones y contaminantes:
    Estaciones    Contaminantes       Concentraciones
    Est1            CO                  1.5807
                    PM10               0.33531
    Est2            NO2                 0.83242
                    PM10                0.72187
                    O3                  0.35961

Sumamos agrupando por estaciones, contaminantes y tipos:
Estaciones    Contaminantes      Tipo         Concentraciones
Est1           CO                Test          1.12810
                                 Train         0.45259
               PM10              Train         0.33531
Est2           NO2               Test          0.83242
               O3                Train         0.72187
               PM10              Test          0.35961
```

Ejercicio 83_02. Agrupaciones en tablas en MATLAB

```
% Creamos una tabla similar a un DataFrame en pandas
data = {
    'Est1', 'CO', 'Train', rand();
    'Est2', 'NO2', 'Test', rand();
    'Est1', 'CO', 'Train', rand();
```

```
    'Est1', 'PM10', 'Train', rand();
    'Est2', 'PM10', 'Test', rand();
    'Est1', 'CO', 'Test', rand();
    'Est2', 'PM10', 'Test', rand();
    'Est2', 'O3', 'Train', rand();
};
columnas = {'Estaciones', 'Contaminantes', 'Tipo', 'Concentraciones'};
df = table(data(:, 1), data(:, 2), data(:, 3), cell2mat(data(:, 4)), 'VariableNames',
columnas);
% Mostramos la tabla original
disp('Tabla original:');
disp(df);
% Sumamos agrupando por estaciones
df2 = grpstats(df, 'Estaciones', 'sum', 'DataVars', 'Concentraciones');
% Mostramos el resultado
disp('Sumamos agrupando estaciones:');
disp(df2);
% Sumamos agrupando por estaciones y contaminantes
df3 = grpstats(df, {'Estaciones', 'Contaminantes'}, 'sum', 'DataVars',
'Concentraciones');
% Mostramos el resultado
disp('Sumamos agrupando por estaciones y contaminantes:');
disp(df3);
% Sumamos agrupando por estaciones, contaminantes y tipos
df4 = grpstats(df, {'Estaciones', 'Contaminantes', 'Tipo'}, 'sum', 'DataVars',
'Concentraciones');
% Mostramos el resultado
disp('Sumamos agrupando por estaciones, contaminantes y tipos:');
disp(df4);

Tabla original:
    Estaciones    Contaminantes      Tipo       Concentraciones
    __________    _____________    _________    _______________
     {'Est1'}       {'CO'  }       {'Train'}       0.53063
     {'Est2'}       {'NO2' }       {'Test' }       0.83242
     {'Est1'}       {'CO'  }       {'Train'}       0.59749
     {'Est1'}       {'PM10'}       {'Train'}       0.33531
     {'Est2'}       {'PM10'}       {'Test' }       0.29923
     {'Est1'}       {'CO'  }       {'Test' }       0.45259
     {'Est2'}       {'PM10'}       {'Test' }       0.42265
     {'Est2'}       {'O3'  }       {'Train'}       0.35961

Sumamos agrupando estaciones:
            Estaciones    GroupCount    sum_Concentraciones
            __________    __________    ___________________
    Est1     {'Est1'}         4               1.916
    Est2     {'Est2'}         4              1.9139

Sumamos agrupando por estaciones y contaminantes:
                Estaciones    Contaminantes    GroupCount    sum_Concentraciones
                __________    _____________    __________    ___________________
    Est1_CO      {'Est1'}       {'CO'  }          3              1.5807
    Est1_PM10    {'Est1'}       {'PM10'}          1             0.33531
```

```
    Est2_NO2      {'Est2'}       {'NO2' }           1               0.83242
    Est2_PM10     {'Est2'}       {'PM10'}           2               0.72187
    Est2_O3       {'Est2'}       {'O3'  }           1               0.35961

Sumamos agrupando por estaciones, contaminantes y tipos:
                 Estaciones    Contaminantes      Tipo       GroupCount
                ____________  _______________  _________  ____________
Est1_CO_Train    {'Est1'}      {'CO'  }       {'Train'}       2
Est1_CO_Test     {'Est1'}      {'CO'  }       {'Test' }       1
Est1_PM10_Train  {'Est1'}      {'PM10'}       {'Train'}       1
Est2_NO2_Test    {'Est2'}      {'NO2' }       {'Test' }       1
Est2_PM10_Test   {'Est2'}      {'PM10'}       {'Test' }       2
Est2_O3_Train    {'Est2'}      {'O3'  }       {'Train'}       1
sum_Concentraciones
___________________
  1.12810
  0.45259
  0.33531
  0.83242
  0.72187
  0.35961
```

1.33. Tablas dinámicas

Esta funcionalidad nos permite agrupar, ordenar, calcular datos y manejar datos de una forma muy similar a la que se hace con las hojas de cálculo de Excel.

Index: Especifica el agrupamiento a nivel fila.
Column: Especifica el agrupamiento a nivel columna.
Values: Valores numéricos que queremos resumir.

Ejercicio 84_01. Tablas dinámicas con Pivot Tables en Python

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
df = pd.DataFrame({'Estaciones': ['Est1', 'Est2', 'Est1', 'Est1', 'Est2', 'Est1',
'Est2', 'Est2'], 'contaminantes': ['CO', 'NO2', 'CO', 'PM10', 'PM10', 'CO', 'PM10',
'O3'], 'tipo': ['Train', 'Test', 'Train', 'Train','Test', 'Test', 'Test', 'Train'],
'concentraciones': np.random.rand(8), # Aleatorias})
print('Dataframe original: \n')
print(df)
# Concentraciones según el tipo, agrupados por estaciones y contaminantes
df2 = pd.pivot_table(df, values='concentraciones', index=['Estaciones',
'Contaminantes'], colums=['Tipo'])
print ('\n Agrupamos datos: concentraciones según el tipo, agrupados por estaciones y
contaminantes: \n')
print(df2)
# Añadimos funciones
df3 = pd.pivot_table(df, values='concentraciones', index=['Estaciones',
'Contaminantes'], colums=['Tipo'], aggfunc=[np.size, np.sum,np.mean])
print('\n Agrupamos datos y contamos, hacemos la suma y la media: \n')
```

```
print(df3)
```

Ejercicio 84_02. Tablas dinámicas en MATLAB

```
% Creamos una tabla similar a un DataFrame en pandas
data = {
    'Est1', 'CO', 'Train', rand();
    'Est2', 'NO2', 'Test', rand();
    'Est1', 'CO', 'Train', rand();
    'Est1', 'PM10', 'Train', rand();
    'Est2', 'PM10', 'Test', rand();
    'Est1', 'CO', 'Test', rand();
    'Est2', 'PM10', 'Test', rand();
    'Est2', 'O3', 'Train', rand();
};

columnas = {'Estaciones', 'Contaminantes', 'Tipo', 'Concentraciones'};
df = table(data(:, 1), data(:, 2), data(:, 3), cell2mat(data(:, 4)), 'VariableNames',
columnas);
% Mostramos la tabla original
disp('Tabla original:');
disp(df);
% Agrupamos datos: concentraciones según el tipo, agrupados por estaciones y
contaminantes
df2 = unstack(df, 'Concentraciones', 'Tipo');
% Mostramos el resultado
disp('Agrupamos datos: concentraciones según el tipo, agrupados por estaciones y
contaminantes:');
disp(df2);
% Agrupamos datos y contamos, hacemos la suma y la media
df3 = varfun(@size, df, 'InputVariables', 'Concentraciones', 'GroupingVariables',
{'Estaciones', 'Contaminantes', 'Tipo'});
df3.Properties.VariableNames = {'Count'};
df3.Sum_Concentraciones = varfun(@sum, df, 'InputVariables', 'Concentraciones',
'GroupingVariables', {'Estaciones', 'Contaminantes', 'Tipo'});
df3.Mean_Concentraciones = varfun(@mean, df, 'InputVariables', 'Concentraciones',
'GroupingVariables', {'Estaciones', 'Contaminantes', 'Tipo'});
% Mostramos el resultado
disp('Agrupamos datos y contamos, hacemos la suma y la media:');
disp(df3);

Tabla original:
Estaciones    Contaminantes      Tipo       Concentraciones
__________    _____________    _________    _______________
{'Est1'}       {'CO'  }       {'Train'}        0.67973
{'Est2'}       {'NO2' }       {'Test' }       0.036563
{'Est1'}       {'CO'  }       {'Train'}         0.8092
{'Est1'}       {'PM10'}       {'Train'}        0.74862
{'Est2'}       {'PM10'}       {'Test' }        0.12019
{'Est1'}       {'CO'  }       {'Test' }        0.52505
{'Est2'}       {'PM10'}       {'Test' }        0.32583
{'Est2'}       {'O3'  }       {'Train'}        0.54645
```

```
Agrupamos datos: concentraciones según el tipo, agrupados por estaciones y
contaminantes:
Estaciones    Contaminantes      Test       Train
__________    _____________    ________    _______
{'Est1'}       {'CO'  }         0.52505     1.4889
{'Est2'}       {'NO2' }        0.036563        NaN
{'Est1'}       {'PM10'}             NaN    0.74862
{'Est2'}       {'PM10'}         0.44602        NaN
{'Est2'}       {'O3'  }             NaN    0.54645

Agrupamos datos y contamos, hacemos la suma y la media:
Estaciones    Contaminantes      Tipo       GroupCount    size_Concentraciones
__________    _____________    ________    __________    ____________________
{'Est1'}       {'CO'  }       {'Test' }       1              1    1
{'Est1'}       {'CO'  }       {'Test' }       1             0.52505
{'Est1'}       {'CO'  }       {'Test' }       1             0.52505
{'Est1'}       {'CO'  }       {'Train'}       2              2    1

Agrupamos datos y contamos, hacemos la suma y la media:
Estaciones    Contaminantes      Tipo       GroupCount    size_Concentraciones
__________    _____________    ________    __________    ____________________
{'Est1'}       {'CO'  }       {'Test' }       1              1    1
{'Est1'}       {'CO'  }       {'Train'}       2              2    1
{'Est1'}       {'PM10'}       {'Train'}       1              1    1
{'Est2'}       {'NO2' }       {'Test' }       1              1    1
{'Est2'}       {'O3'  }       {'Train'}       1              1    1
{'Est2'}       {'PM10'}       {'Test' }       2              2    1

Sum_Concentraciones
Estaciones    Contaminantes      Tipo       GroupCount    sum_Concentraciones
__________    _____________    ________    __________    ___________________
{'Est1'}       {'CO'  }       {'Test' }       1             0.52505
{'Est1'}       {'CO'  }       {'Train'}       2             1.4889
{'Est1'}       {'PM10'}       {'Train'}       1             0.74862
{'Est2'}       {'NO2' }       {'Test' }       1             0.036563
{'Est2'}       {'O3'  }       {'Train'}       1             0.54645
{'Est2'}       {'PM10'}       {'Test' }       2             0.44602

Mean_Concentraciones
Estaciones    Contaminantes      Tipo       GroupCount    mean_Concentraciones
__________    _____________    ________    __________    ____________________
{'Est1'}      {'CO'  }        {'Test' }       1           0.52505
{'Est1'}      {'CO'  }        {'Train'}       2           0.74447
{'Est1'}       {'PM10'}        {'Train'}      1           0.74862
{'Est2'}       {'NO2' }        {'Test' }      1           0.036563
{'Est2'}       {'O3'  }        {'Train'}      1           0.54645
{'Est2'}       {'PM10'}        {'Test' }      2           0.22301
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

El archivo usado en esta práctica no tiene exactamente las mismas dimensiones que el original, por ello salen algunos resultados diferentes.

2.1. Estadística descriptiva.

Ejercicio 85_01. Exploración de un *dataset* en Python

```
# Una vez que cargamos el dataset, una de las primeras acciones es mostrar las
primeras filas para tener una primera idea de cómo son los datos. También es
conveniente ver las dimensiones del dataset que hemos cargado, así como los tipos de
dato de cada variable.
# Importamos las librerías necesarias
import pandas as pd
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae.
# Se usa ; como separador, y no la coma por defecto.
datos = pd.read_csv(nomFichero, sep=";")
print("Mostramos las 10 primeras filas del dataset:\n")
print(datos.head(10))
print("\nDimensiones del dataset:\n")
print(datos.shape)
print("\nTipo de datos de cada variable (columna):\n")
print(datos.dtypes)
```

```
Mostramos las 10 primeras filas del dataset:
   fixed acidity      volatile acidity   citric acidity     residual sugar chlorides\

0          7.4                 0.70                0.00                1.9      0.076
1          7.8                 0.88                0.00                2.6      0.098
2          7.8                 0.76                0.04                2.3      0.092
3         11.2                 0.28                0.56                1.9      0.075
4          7.4                 0.70                0.00                1.9      0.076
5          7.4                 0.66                0.00                1.8      0.075
6          7.9                 0.60                0.06                1.6      0.069
7          7.3                 0.65                0.00                1.2      0.065
8          7.8                 0.58                0.02                2.0      0.073
9          7.5                 0.50                0.36                6.1      0.071

free sulfur dioxide        total sulfur dioxide     density         ph      sulphates
0               11.0                        34.0      0.9978       3.51       0.56
1               25.0                        67.0      0.9968       3.20       0.68
2               15.0                        54.0      0.9970       3.26       0.65
3               17.0                        60.0      0.9980       3.16       0.58
4               11.0                        34.0      0.9978       3.51       0.56
5               13.0                        40.0      0.9978       3.51       0.56
6               15.0                        59.0      0.9964       3.30       0.46
7               15.0                        21.0      0.9946       3.39       0.47
8                9.0                        18.0      0.9968       3.36       0.57
9               17.0                       102.0      0.9978       3.35       0.8
```

```
   alcohol   quality
0      9.4         5
1      9.8         5
2      9.8         5
3      9.8         6
4      9.4         5
5      9.4         5
6      9.4         5
7     10.0         7
8      9.5         7
9     10.5         5

Dimensiones del dataset:
(1599, 12)

Tipo de datos de cada variable (columna):

fixed acidity             float64
volatile acidity          float64
citric acidity            float64
residual sugar            float64
chlorides                 float64
free sulfur dioxide       float64
total sulfur dioxide      float64
density                   float64
ph                        float64
sulphates                 float64
alcohol                   float64
quality                   int64
dtype: object
```

Ejercicio 85_02. Sumario estadístico MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
% Configurar el formato de impresión
format shortG; % Mostrar tres decimales
% Mostrar el sumario estadístico
disp('Sumario estadístico:');
disp(summary(datos));
```

```
Sumario estadístico:
                  Var1: [1×1 struct]
          fixedAcidity: [1×1 struct]
       volatileAcidity: [1×1 struct]
         citricAcidity: [1×1 struct]
         residualSugar: [1×1 struct]
             chlorides: [1×1 struct]
     freeSulfurDioxide: [1×1 struct]
    totalSulfurDioxide: [1×1 struct]
               density: [1×1 struct]
```

```
                ph: [1×1 struct]
         sulphates: [1×1 struct]
           alcohol: [1×1 struct]
           quality: [1×1 struct]
```

Ejercicio 86_01. Sumario estadístico Python

```
# Mediante pd.describe() obtenemos una estadística descriptiva que resume la
tendencia central, la dispersión y la forma de la distribución de un conjunto de
datos, excluyendo los valores de NaN.
import pandas as pd
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae.
# Se usa ; como separador, y no la coma por defecto.
datos = pd.read_csv(nomFichero, sep=";")
# Mostrar sumario con tres decimales
pd.set_option('precision', 3)
# Para mostrar la información de todas las columnas
pd.set_option('max_columns', None)
print("Sumario estadístico: ")
print(datos.describe)
```

```
Sumario estadístico:
<bound method NDFrame.describe of
   fixed acidity       volatile acidity   citric acidity     residual sugar chlorides\

0          7.4                 0.700                0.00                1.9      0.076
1          7.8                 0.880                0.00                2.6      0.098
2          7.8                 0.760                0.04                2.3      0.092
3         11.2                 0.280                0.56                1.9      0.075
4          7.4                 0.700                0.00                1.9      0.076
…            …                     …                   …                  …          …

1594       6.2                 0.600                0.08                2.0      0.090
1595       5.9                 0.550                0.10                2.2      0.062
1596       6.3                 0.510                0.13                2.3      0.076
1597       5.9                 0.640                0.12                2.0      0.075
1598       6.0                 0.310                0.47                3.6      0.067

free sulfur dioxide        total sulfur dioxide     density         ph      sulphates
0                 11.0                      34.0      0.998       3.51       0.56
1                 25.0                      67.0      0.997       3.20       0.68
2                 15.0                      54.0      0.997       3.26       0.65
3                 17.0                      60.0      0.998       3.16       0.58
4                 11.0                      34.0      0.998       3.51       0.56
…                    …                         …          …          …          …
1594              32.0                      44.0      0.995       3.30       0.58
1595              39.0                      51.0      0.995       3.39       0.76
1596              29.0                      40.0      0.996       3.36       0.75
1597              32.0                      44.0      0.995       3.35       0.71
1598              18.0                      42.0      0.995       3.39       0.66
```

```
   alcohol   quality
0      9.4         5
1      9.8         5
2      9.8         5
3      9.8         6
4      9.4         5
…      …           …
1594    10.5        5
1595    11.2        6
1596    11.0        6
1597    10.2        5
1598    11.0        6

 [1599 rows x 12 columns]
```

Ejercicio 86_02. Exploración de un dataset en MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
% Mostrar las 10 primeras filas del dataset
disp('Mostramos las 10 primeras filas del dataset:');
disp(head(datos, 10));
% Mostrar las dimensiones del dataset
disp('Dimensiones del dataset:');
disp(size(datos));
% Mostrar el tipo de datos de cada variable (columna)
disp('Tipo de datos de cada variable (columna):');
disp(class(datos));
```

```
Mostramos las 10 primeras filas del dataset:

   fixedAcidity   volatileAcidity   citricAcidity   residualSugar        chlorides
   ____________   _______________   _____________   _____________        _________

          7.4             0.70          0.00             1.9             0.076
          7.8             0.88          0.00             2.6             0.098
          7.8             0.76          0.04             2.3             0.092
         11.2             0.28          0.56             1.9             0.075
          7.4             0.70          0.00             1.9             0.076
          7.4             0.66          0.00             1.8             0.075
          7.9             0.60          0.06             1.6             0.069
          7.3             0.65          0.00             1.2             0.065
          7.8             0.58          0.02             2.0             0.073
          7.5             0.50          0.36             6.1             0.071


  freeSulfurDioxide   totalSulfurDioxide       density        ph      sulphates
  _________________   __________________       _______      ____      _________

                 11                   34        0.9978      3.51       0.56
                 25                   67        0.9968      3.20       0.68
```

```
                 15                   54        0.997      3.26      0.65
                 17                   60        0.998      3.16      0.58
                 11                   34       0.9978      3.51      0.56
                 13                   40       0.9978      3.51      0.56
                 15                   59       0.9964      3.30      0.46
                 15                   21       0.9946      3.39      0.47
                  9                   18       0.9968      3.36      0.57
                 17                  102       0.9978      3.35       0.8

      alcohol    quality
      _______    _______
         9.4         5
         9.8         5
         9.8         5
         9.8         6
         9.4         5
         9.4         5
         9.4         5
        10.0         7
         9.5         7
        10.5         5

Dimensiones del dataset:
    1599     12
Tipo de datos de cada variable (columna):
table
```

Ejercicio 87_01. Distribución de clases en problemas de clasificación en Python

```
# En los problemas de clasificación es muy importante conocer cómo se distribuyen las
clases. <br>
# Si hay una gran diferencia entre los registros que corresponde a cada clase, puede
ser necesario un tratamiento posterior.
import pandas as pd
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae.
# Se usa ; como separador, y no la coma por defecto.
datos = pd.read_csv(nomFichero, sep=";")
# La variable quality es de tipo categórico, con valores de 1 a 5
conteo_por_clases = datos.grouphy('quality').size()
print("Nº de registros por cada clase en la variable quality: ")
print(conteo_por_clases)

Nº de registros por cada clase en la variable quality:
quality
3    10
4    53
5    681
6    638
7    199
8    18
dtype: int64
```

Ejercicio 87_02. Distribución de clases en problemas de clasificación en MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
% Contar el número de registros por cada clase en la variable 'quality'
conteo_por_clases = varfun(@length, datos, 'GroupingVariables', 'quality');
% Mostrar el número de registros por cada clase en la variable 'quality'
disp('Número de registros por cada clase en la variable quality:');
disp(conteo_por_clases);

Número de registros por cada clase en la variable quality:
quality    GroupCount    length_fixedAcidity    length_volatileAcidity
_______    __________    ___________________    ______________________
3           10                10                     10
4           53                53                     53
5          681               681                    681
6          638               638                    638
7          199               199                    199
8           18                18                     18
length_citricAcid    length_residualSugar    length_chlorides
_________________    ____________________    ________________
10                   10                      10
53                   53                      53
681                  681                     681
638                  638                     638
199                  199                     199
18                   18                      18
length_freeSulfurDioxide    length_totalSulfurDioxide    length_density
________________________    _________________________    ______________
10                           10                          10
53                           53                          53
681                          681                         681
638                          638                         638
199                          199                         199
18                           18                          18
length_pH     length_sulphates      length_alcohol
_________     ________________      ______________
 10              10                  10
 53              53                  53
 681             681                 681
 638             638                 638
 199             199                 199
 18              18                  18
```

2.2. Correlación entre variables.

- La correlación se refiere a la relación entre dos variables y cómo pueden o no cambiar juntas.
- El método más común para calcular la correlación es el coeficiente de correlación Pearson, que asume una distribución normal de las variables.
- Su valor está en [-1,1], siendo un valor de -1 una correlación perfectamente negativa, y un valor de +1 una correlación perfectamente positiva.
- Un valor de 0 indica que no existe correlación.
- Algunos algoritmos de aprendizaje automático pueden ver penalizado su rendimiento si hay variables altamente correlacionadas entre sus entradas.

Ejercicio 88_01. Correlaciones en Python 1

```
import pandas as pd
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae.
# Se usa ; como separador, y no la coma por defecto.
dfVinos = pd.read_csv(nomFichero, sep=";")
dfDatos = dfVinos.iloc[:,0:4];
# Mostramos las 3 primeras filas
print(dfDatos.head(3))
```

```
      fixed Acidity    volatile Acidity    citric Acidity    residual Sugar

  0          7.4              0.70                 0                1.9
  1          7.8              0.88                 0                2.6
  2          7.8              0.76              0.04                2.3
```

Ejercicio 88_02. Correlaciones en MATLAB 1

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
dfVinos = readtable(nomFichero, 'Delimiter', ',');
% Seleccionar las primeras cuatro columnas
dfDatos = dfVinos(:, [1:4]);
% Mostrar las primeras tres filas
disp('Las 3 primeras filas de la tabla:');
disp(dfDatos(1:3, :));
```

```
Las 3 primeras filas de la tabla:
    Var1    fixedAcidity    volatileAcidity    citricAcidity    residualSugar
    ____    ____________    _______________    _____________    _____________

     0          7.4              0.70                 0                1.9
     1          7.8              0.88                 0                2.6
     2          7.8              0.76              0.04                2.3
```

Ejercicio 89_01. Correlaciones en Python 2

```
# Calculamos ahora las correlaciones entre todas las parejas de variables:
print("Correlaciones entre las variables:\n")
correlaciones = dfDatos.corr(method='pearson')
print(correlaciones)
```

```
Correlaciones entre las variables:

             fixed acidity    volatile acidity    citric acidity    residual sugar

fixed acidity         1.000000        -0.67696           0.70478          -0.08590
volatile acidity     -0.67696          1.00000          -0.83131          -0.28904
citric acidity        0.70478         -0.83131           1.00000           0.59803
residual sugar       -0.085909        -0.28904           0.59803           1.00000
```

Ejercicio 89_02. Correlaciones en MATLAB 2

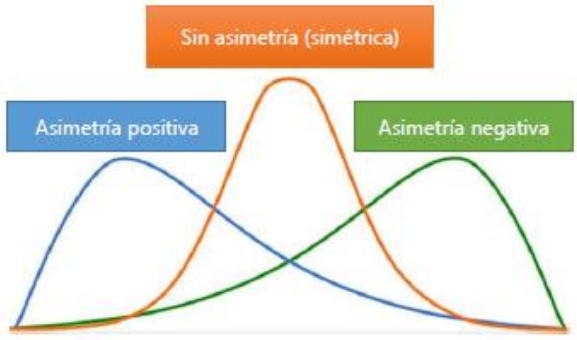
```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
dfVinos = readtable(nomFichero, 'Delimiter', ',');
% Seleccionar las primeras cuatro columnas
dfDatos = dfVinos(:, [2:5])
% dfDatos = dfVinos
% Mostrar las 3 primeras filas
disp('Las 3 primeras filas de la tabla (quitando la primera): ');
disp(dfDatos([2:4], :));
% Mostrar las 3 primeras filas (quitando la primera)
% Calcular correlaciones entre las variables
correlaciones = corrcoef(table2array(dfDatos), 'rows', 'pairwise');
% Mostrar las correlaciones en formato de tabla
nombres_variables = dfDatos.Properties.VariableNames;
correlaciones_tabla = array2table(correlaciones, 'RowNames', nombres_variables,
'VariableNames', nombres_variables);
% Mostrar las correlaciones en formato de tabla
disp('Correlaciones entre las variables:');
disp(correlaciones_tabla);
```

```
Correlaciones entre las variables:

                fixedAcidity    volatileAcidity    citricAcidity residualSugar
                ____________    _______________    _____________ _____________

fixedAcidity                1         -0.67696          0.70478         -0.08590
volatileAcidity       -0.67696                1         -0.83131         -0.28904
citricAcidity          0.70478         -0.83131                1          0.59803
residualSugar        -0.085909         -0.28904          0.59803                1
Correlaciones entre las variables:



                fixedAcidity    volatileAcidity    citricAcid    residualSugar
                ____________    _______________    __________    _____________
```

```
fixedAcidity          1.000000         -0.256131         0.671703         0.114777
volatileAcidity      -0.256131          1.000000        -0.552496         0.001918
citricAcid            0.671703         -0.552496         1.000000         0.143577
residualSugar         0.114777          0.001918         0.143577         1.000000
```

2.3. Asimetría e histogramas.

- **Asimetría (skewness)**
  - El coeficiente de asimetría es muy útil para detectar casos en los que las variables no se comportan como variables normales (distribución gaussiana).
  - La asimetría puede ser positiva o negativa. Valores mayores de +1 o menores de -1 indican una asimetría pronunciada. Valores cercanos a cero indican una distribución bastante simétrica.
  - Muchos algoritmos de aprendizaje automático asumen una distribución gaussiana.
  - Conocer la asimetría nos permite preparar los datos para poder corregir las variables necesarias, de forma que aumente la precisión de los modelos.
  - Transformaciónes comunes para reducir la asimetría son la aplicación de la raíz cuadrada, transformaciones logarítmicas o transformaciones Box-Cox.



Ejercicio 90_01. Cálculo de la asimetría en Python

```
import pandas as pd
from matplotlib import pyplot
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae.
# Se usa ; como separador, y no la coma por defecto.
dfVinos = pd.read_csv(nomFichero, sep=";")
print('Asimetría de las variables: ')
print(dfVinos.skew())
```

```
Asimetría de las variables:
    fixedAcidity           0.982751
    volatileAcidity        0.671593
    citricAcid             0.318337
    residualSugar          4.540655
    chlorides              5.680347
    freeSulfurDioxide      1.250567
    totalSulfurDioxide     1.515531
    density                0.071288
    pH                     0.193683
    sulphates              2.428672
    alcohol                0.860829
    quality                0.217802
```

Ejercicio 90_02. Cálculo de la asimetría en MATLAB

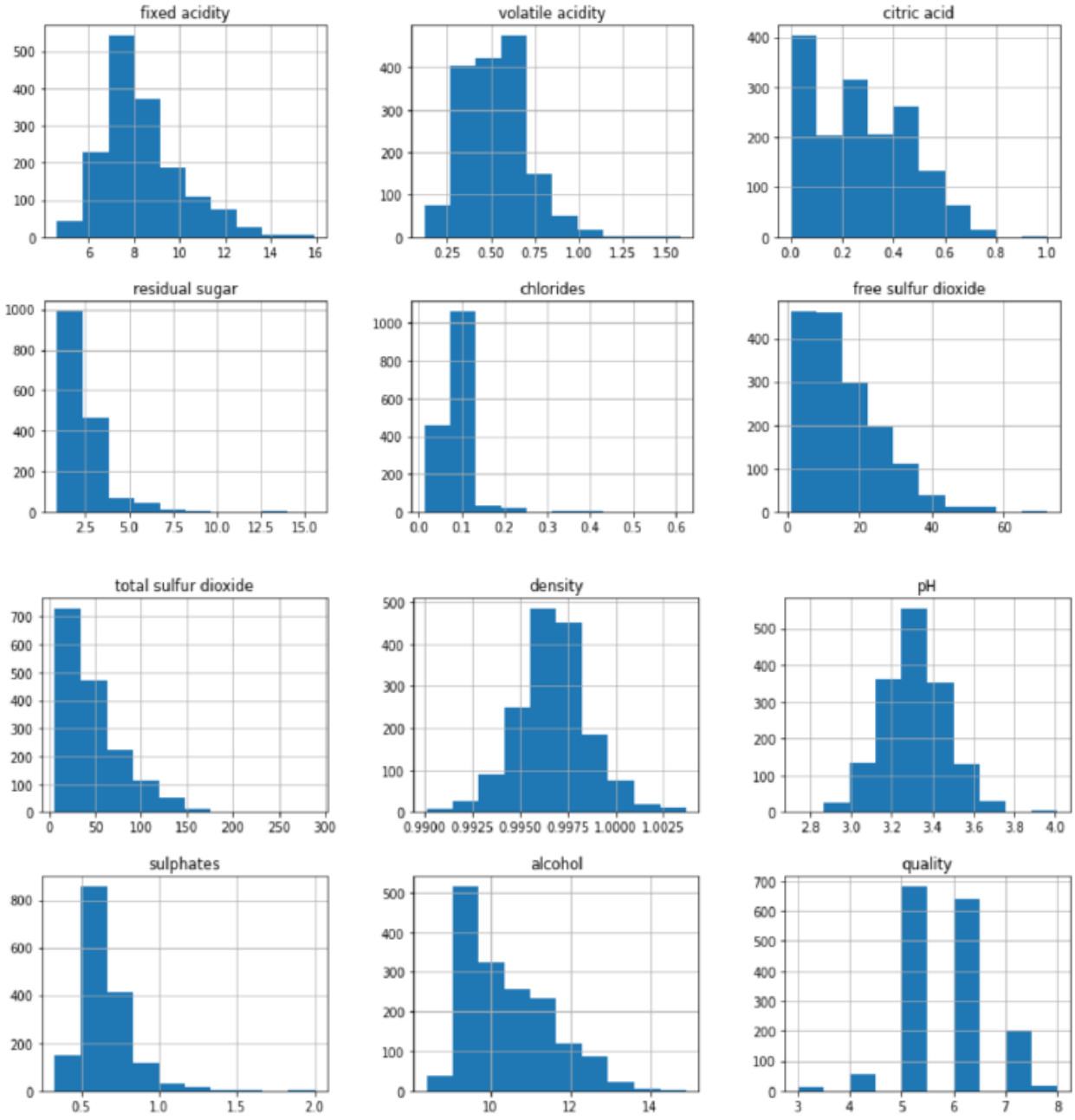
```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
dfVinos = readtable(nomFichero, 'Delimiter', ',');
% Calcular la asimetría de las variables
asimetria = skewness(table2array(dfVinos));
% Mostrar la asimetría de las variables
nombres_variables = dfVinos.Properties.VariableNames;
asimetria_tabla = table(asimetria', 'RowNames', nombres_variables', 'VariableNames',
{'Asimetría'});
% Mostrar la asimetría en formato de tabla
disp('Asimetría de las variables:');
disp(asimetria_tabla);
```

```
Asimetría de las variables:
                         Asimetria
                         _________

    fixedAcidity          0.98183
    volatileAcidity       0.67096
    citricAcid            0.31804
    residualSugar          4.5364
    chlorides               5.675
    freeSulfurDioxide      1.2494
    totalSulfurDioxide     1.5141
    density               0.071221
    pH                      0.1935
    sulphates               2.4264
    alcohol                0.86002
    quality                 0.2176
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

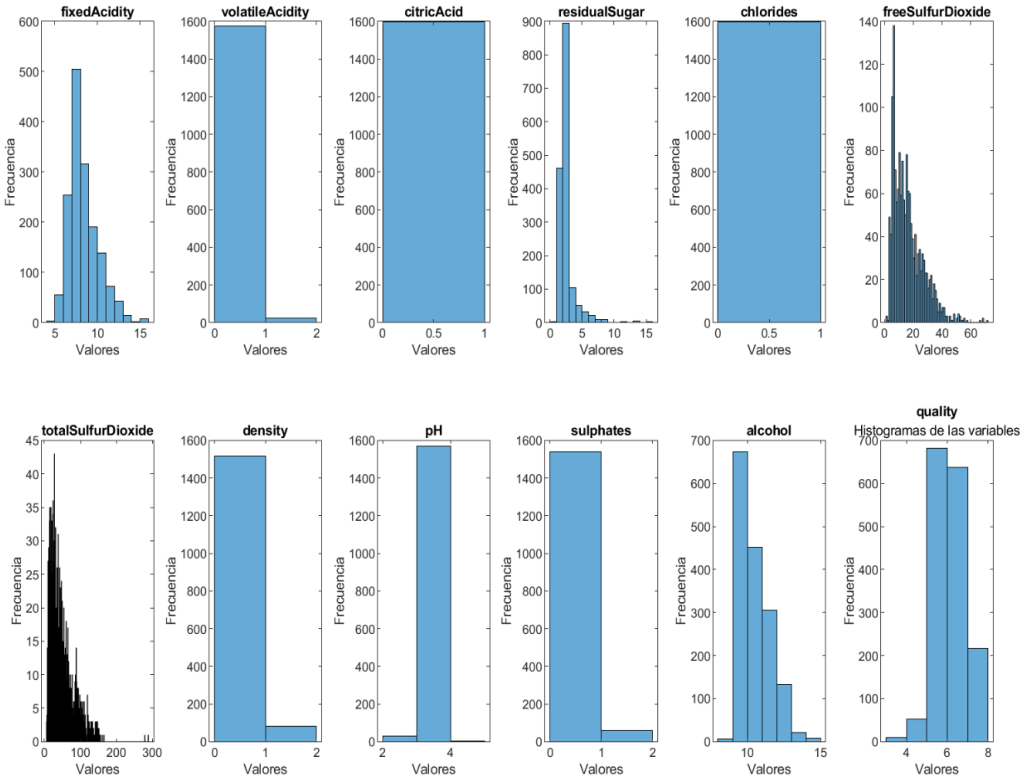
Ejercicio 91_01. Mostramos los histogramas de las variables en Python

```
# Mostramos los hostogramas de las variables
# Necesario para corregir la proporción de los histogramas
pyplot.rcParams['figure.figsize'] = (15,15)
# Histogramas de las variables
dfVinos.hist()
pyplot.show()
```



Ejercicio 91_02. Mostramos los histogramas de las variables en MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
dfVinos = readtable(nomFichero, 'Delimiter', ',');
% Configurar el tamaño del gráfico
figure('Position', [100, 100, 1200, 800]);
% Iterar a través de las columnas y mostrar histogramas
numColumnas = width(dfVinos);
for i = 1:numColumnas
    subplot(2, 6, i); % Ajusta el número de filas y columnas según sea necesario
    histogram(dfVinos.(i), 'BinWidth', 1); % Puedes ajustar el ancho de los bins
según tus preferencias
    title(dfVinos.Properties.VariableNames{i});
    xlabel('Valores');
    ylabel('Frecuencia');
end
% Mostrar el gráfico
filename = sprintf('Gráficas de asimetría en MATLAB.tif');
subtitle('Histogramas de las variables');
% savefig('Asimetría en MATLAB.tiff')
saveas(gcf, filename,'tif');
```



Ejercicio 92_01. Reducción de asimetría usando Box-Cox y Yeo-Johnson

```
import pandas as pd
import numpy as np
from matplotlib import pyplot
from sklearn.preprocessing import PowerTransformer
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae
# Se usa ; como separador, y no la coma por defecto
dfDatosVinos = pd.read_csv(nomFichero, sep=";")
# Nos quedamos con un par de variables para hacer el ejemplo:
dfDatos = dfDatosVinos.loc[:,["residual sugar", "chlorides"]]
# Mostramos las 3 primeras filas
print(dfDatos.head(3))

    residual sugar    chlorides
 0             1.9          0.076
 1             2.6          0.098
 2             2.3          0.092
```

Ejercicio 92_02. Reducción de asimetría usando Box-Cox y Yeo-Johnson en MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
dfDatosVinos = readtable(nomFichero, 'Delimiter', ',');
% Nos quedamos con un par de variables para hacer el ejemplo
columnas_seleccionadas = {'residualSugar', 'chlorides'};
dfDatos = dfDatosVinos(:, columnas_seleccionadas);
% Mostrar las 3 primeras filas
disp('Las 3 primeras filas de la tabla: ');
disp(dfDatos(1:3, :));

Las 3 primeras filas de la tabla:
    residualSugar    chlorides
    _____________    _________

         1.9           0.076
         2.6           0.098
         2.3           0.092
```

- **Reducción de asimetría mediante transformaciones Box Cox y Yeo Johnson: Realizamos la transformación Box-Cox y Yeo-Johnson:**
    - Importante: La transformación Box-Cox necesita que los valores sean estrictamente positivos, a diferencia de la transformación Yeo-Johnson, que admite valores positivos y negativos.
    - Las citadas transformaciones permiten estandarizar los valores, lo que puede ser muy aconsejable. Pero eso lo veremos en el ejemplo 45.

Ejercicio 93_01. Comprobar que no tienen valores negativos:

```
dfDatosNegativos = dfDatos[dfDatos.values < 0]
dfDatosNegativos.shape
# Tras la comprobación, realizamos las transformaciones:
# Definimos las transformaciones
```

```
power1 = PowerTransformer(method='box-cox', standardize=False)
power2 = PowerTransformer(method='yeo-johnson', standardize=False)
# Ajustamos en base a los datos
power1.fit(datos)
power2.fit(datos)
# Aplicamos las transformaciones
datosBox = power1.fit_transform(datos)
datosYeo = power2.fit_transform(datos)
```

2.4. Outliers y diagramas de cajas y bigotes.

Resumen la distribución de cada variable cuantitativa, dibujándose la mediana y una caja alrededor del primer (Q1) y tercer (Q3) cuartiles Permite visualizar cuáles son los datos centrales, los datos adyacentes y los datos atípicos, si los hubiera.

• El rango intercuartílico (RI) viene representado por la altura de la caja (Q3-Q1).

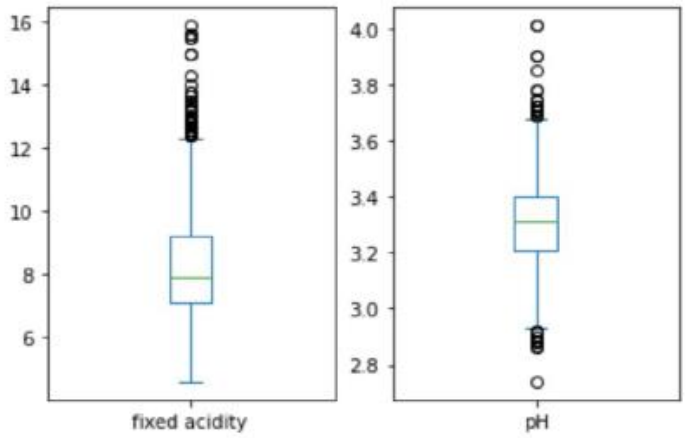
• Un punto que esté 1.5 veces el RI por encima del tercer cuartil o 1.5 veces el RI por debajo del primer cuartil, es un posible valor atípico u outlier (representados como círculos en el ejemplo 43).

• El bigote superior se representa como una línea desde Q3 hasta el máximo punto excluyendo posibles outliers De la misma forma, el bigote inferior se representa como una línea desde el Q1 hasta el mínimo punto que no cae en el rango de posibles *outliers*.

• El conocimiento del dominio del problema es fundamental a la hora de descartar posibles *outliers*.

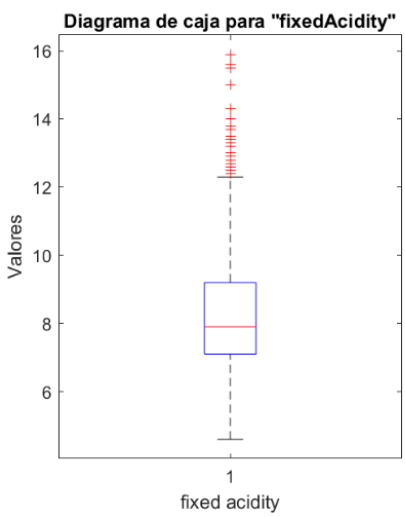
Ejercicio 94_01. Diagrama de cajas y bigotes en Python

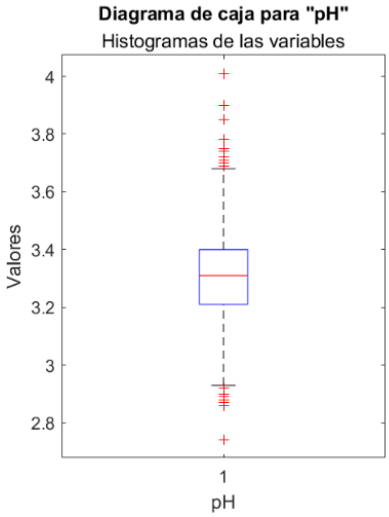
```
import pandas as pd
import matplotlib.pyplot as plt
nomFichero = "winequality-red.csv"
# No es necesario añadir nombres de columnas, ya que el csv los trae
# Se usa ; como separador, y no la coma por defecto
datos = pd.read_csv(nomFichero, sep=";")
 # Seleccionamos algunas variables y representamos sus diagramas
datos2 = datos.loc[:, ["fixed acidity", "pH"]]
#Layout, 1 fila y 2 columnas
datos2.plot(kind="box", subplots=True, layout= (1,2), sharex=False, sharey=False)
plt.show()
```



Ejercicio 94_02. Diagrama de cajas y bigotes en MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
% Seleccionar algunas variables
columnas_seleccionadas = {'fixedAcidity', 'pH'};
datos2 = datos(:, columnas_seleccionadas);
% Configurar el tamaño del gráfico
figure('Position', [100, 100, 800, 400]);
% Mostrar diagramas de caja para las variables seleccionadas
subplot(1, 2, 1);
boxplot(datos2.('fixedAcidity'));
title('Diagrama de caja para "fixedAcidity"');
xlabel('fixed acidity');
ylabel('Valores');
subplot(1, 2, 2);
boxplot(datos2.('pH'));
title('Diagrama de caja para "pH"');
xlabel('pH');
ylabel('Valores');
```





2.5. Preparación de datos para machine learning.

Esquema general:

1. Cargar los datos (desde un archivo CSV, u otra fuente).

2. Realizar las transformaciones de datos que sean más aconsejables según las características de los datos y la técnica de machine learning a aplicar.

3. En aprendizaje supervisado, primero se dividen los datos en dos conjuntos separados entrenamiento (alrededor del 70-80% de los registros) y test (los registros restantes). Las transformaciones se realizan en base a los datos de entrenamiento, para luego aplicarlas a los datos de test.

4. Estudiar de nuevo los datos tras los cambios.

Técnicas:

• Estandarización de variables numéricas: A través de este proceso, los datos tendrán media 1 y desviación típica 0.

• Normalización o reescalado de variables numéricas a un rango específico (por ejemplo, [0,1]).

• Otras, como binarización, etc.

División en subconjuntos de entrenamiento y de test en aprendizaje supervisado:

Documentación: Train and test data

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

Estandarización de variables numéricas:

• Con este proceso se consigue transformar la distribución de datos para que la media de los valores observados sea 0 y la desviación típica sea 1: $Z = \frac{X-\bar{x}}{\sigma}$

• Resulta muy útil para eliminar su dependencia respecto a las unidades de medida empleadas.

• Para poder aplicarlo, las distribuciones de datos **deben ajustarse a una distribución gaussiana (normal).**

• **Aprendizaje supervisado:**

• Generalmente, esta transformación se aplica únicamente a las variables predictoras (sobre el dataset de training y luego aplicar la misma transformación al test). De esta forma, no se hace necesario tener que deshacer la transformación para poder obtener resultados en la magnitud de la medida original.

• Sin embargo, de forma similar es posible estandarizar la variable objetivo. En este caso, tendremos que deshacer la transformación posteriormente para poder obtener los resultados en la magnitud de la medida original.

• En Python se usa StandardScaler siguiendo los siguientes pasos:

1. Ajustar StandardScaler con los datos de entrenamiento con la función fit().
2. Aplicar el scaler a los datos de enteramiento con la función transform().
3. Aplicar ese mismo scaler a los datos de test().

Documentación: Distribución normal en Python

Ejercicio 95_01. División subsets de training y test en Python

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae
# Se usa ; como separador, y no la coma por defecto
dfWines = read_csv(nomFichero, sep=";")
variables = dfWines.to_numpy()
print('Tamaño del dataset:')
print(variables.shape)

Tamaño del dataset:
        (1599, 12)

# Seleccionamos variable objetivo y variables predictoras: **<br>
x = variables[:,0:10] # Variables predictoras (0 a 10)
y = variables[:,11] # Variable obtetivo
# División en conjuntos de entenamiento y test:**<br>
Se usa un 30% para test y un 70% para entrenamiento.<br>
Con las semillas y random_state se permite la reproducibilidad.
```

```
tamTest = 0.3
semillas = 90
# Se pasa random_state para que los resultados sean reproducibles en diferentes
# llamadas train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = tamTest,
random_state = semillas)
# Mostramos resultados:
# Tamaños de las divisiones
print('Tamaño de xTrain:')
print(xTrain.shape)
print('Tamaño de xTest:')
print(xTest.shape)
print('Tamaño de yTrain:')
print(yTrain.shape)
print('Tamaño de yTest:')
print(yTest.shape)

Tamaño de xTrain:
(1119, 11)
Tamaño de xTest:
(479, 11)
Tamaño de yTrain:
(1120, 1)
Tamaño de yTest:
(479, 1)
```

Ejercicio 95_02. División subsets de training y test en MATLAB

```
% Importar datos desde un archivo CSV en MATLAB
nomFichero = 'winequality-red.csv';
dfWines = readtable(nomFichero, 'Delimiter', ',');
% Obtener la matriz de variables
variables = table2array(dfWines);
% Mostrar el tamaño del dataset
disp('Tamaño del la tabla:');
disp(size(variables));

Tamaño de la tabla:
        1599          12

% Seleccionar variable objetivo y variables predictoras
x = variables(:, 1:11); % Variables predictoras (columnas 1 a 11)
y = variables(:, 12);   % Variable objetivo (columna 12)
% División en conjuntos de entrenamiento (70%) y test (30%)
tamTest = 0.3;
semillas = 90;
% Se utiliza cvpartition para garantizar la reproducibilidad
rng(semillas);
particion = cvpartition(y, 'Holdout', tamTest);
% Conjunto de entrenamiento
xTrain = x(training(particion), :);
yTrain = y(training(particion), :);
```

```
% Conjunto de test
xTest = x(test(particion), :);
yTest = y(test(particion), :);
% Mostrar resultados
disp('Tamaño de xTrain:');
disp(size(xTrain));
disp('Tamaño de xTest:');
disp(size(xTest));
disp('Tamaño de yTrain:');
disp(size(yTrain));
disp('Tamaño de yTest:');
disp(size(yTest));

Tamaño de xTrain:
        1120          11
Tamaño de xTest:
   479    11
Tamaño de yTrain:
        1120           1
Tamaño de yTest:
   479     1
```

Ejercicio 96_01. Estandarización en Python

```
# Cargamos los datos:
# Con df.to_numpy() convertimos el dataframe a arrays de Numpy.Se establecen la
variable objetivo y las variables de entrada.
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from pandas import read_csv
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae
# Se usa ; como separador, y no la coma por defecto
datosWines = read_csv(nomFichero, sep=";")
variables  = datosWines.to_numpy()
x = variables[:,0:10] # Variables 0 a 10
y = variables[:,11] # Variable 11, que es una categoría
# Se realiza la división entrenamiento-test:
# Se usa 30% para test y 70% para entrenamiento.
tamTest = 0.3
semillas = 90
# Se pasa random_state para que los resultados sean reproducibles en diferentes
# llamadas train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=tamTest,
random_state=semillas)
# Estandarización:
# Se debe realizar sobre el conjunto de entrenamiento y, posteriormente, aplicar al
conjunto de test.
# Ajustamos el scaler sobre el conjunto de entrenamiento
scalerX   = StandardScaler().fit(xTrain)
# Estandarizamos el conjunto de entrenamiento
xTrainEst = scalerX.transform(xTrain)
```

```
# Una vez estandarizadas las xTrain, debemos estandarizar los xTest
xTestEst  = scalerX.transform(xTest)
# Deshacemos estandarización:
# Se muestra como deshacerlo por si fuera necesario
xTrain2 = scalerX.inverse_transform(xTrainEst, copy=None)
print("Datos originales xTrain:\n")
print(xTrain)
print("\nDatos obtenidos al deshacer la estandarización:\n")
print(xTrain2)
```

```
Datos originales xTrain:
[[7.4          0.37          0.43       …         3.33          0.68          9.7 ]
[10.4          0.44          0.73       …         3.17          0.85          12.0]
[ 9.6          0.33          0.52       …         3.36          0.76          12.4]
…
[7.2          0.655          0.03       …         3.34          0.39          9.5]
[8.2          0.73           0.21       …         3.20          0.52          9.5]]
[11.3          0.34           0.45       …        2.94          0.66          9.2]]

Datos obtenidos al deshacer la estandarización:
[[7.4          0.37          0.43       …         3.33          0.68          9.7 ]
[10.4          0.44          0.73       …         3.17          0.85          12.0]
[ 9.6          0.33          0.52       …         3.36          0.76          12.4]
…
[7.2          0.655          0.03       …         3.34          0.39          9.5]
[8.2          0.73           0.21       …         3.20          0.52          9.5]]
[11.3          0.34           0.45       …        2.94          0.66          9.2]]
```

Ejercicio 96_02. Estandarización en MATLAB

```
nomFichero = 'winequality-red.csv';
datosWines = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datosWines);
% Seleccionar variables predictoras y variable objetivo
% para coger todas las filas:
% x = variables(:, 1:11); % Variables 1 a 11
% y = variables(:, 12);   % Variable 12, que es una categoría
% para coger sólo las 10 primeras filas y las 7 últimas columnas
x = variables(1:10, 5:11); % Variables 5 a 11
y = variables(1:10, 12);   % Variable 12, que es una categoría
% División entrenamiento-test
tamTest = 0.3;
semillas = 90;
rng(semillas); % Establecer la semilla para reproducibilidad
particion = cvpartition(y, 'Holdout', tamTest);
% Conjunto de entrenamiento
xTrain = x(training(particion), :);
yTrain = y(training(particion), :);
% Conjunto de test
xTest = x(test(particion), :);
yTest = y(test(particion), :);
% Estandarización
```

```
xTrainEst = zscore(xTrain);
xTestEst = zscore(xTest);
% Mostrar resultados
disp('Datos originales xTrain:');
disp(xTrain);
disp('Datos obtenidos al hacer la estandarización:');
disp(xTrainEst);

Datos originales xTrain:
0.076           11           34       0.9978         3.51         0.56          9.4
0.098           25           67       0.9968          3.2         0.68          9.8
0.075           17           60        0.998         3.16         0.58          9.8
0.076           11           34       0.9978         3.51         0.56          9.4
0.075           13           40       0.9978         3.51         0.56          9.4

Datos obtenidos al hacer la estandarización:
-0.39703     -0.74587      -0.8409      0.33508      0.72807     -0.53688     -0.7303
 1.7867       1.6274       1.2937     -1.75920     -0.98179      1.76400      1.0954
-0.49629      0.27123       0.8409      0.75394     -1.20240     -0.15339      1.0954
-0.39703     -0.74587      -0.8409      0.33508      0.72807     -0.53688     -0.7303
-0.49629     -0.40684      -0.45279     0.33508      0.72807     -0.53688     -0.7303
```

- **Normalización de variables numéricas:**

Con esta transformación se realiza un cambio de escala en los datos, de forma que todos los valores estén dentro de un rango especificado, normalmente [0, 1] o también [-1, 1].

- **Aprendizaje supervisado:**
    - El reescalado se debe realizar en base a los datos de entrenamiento, para luego aplicarlas a los datos de test.
    - En Python, se usa MinMaxScaler con los siguientes pasos:

1.Ajustar MinMaxScaler con los datos de entrenamiento con la función fit

2. Aplicar el scaler a los datos de entramiento con la función transform

3. Aplicar ese mismo scaler a los datos de test

Documentación: Normalización y preprocesamiento en Python

Ejercicio 97_01. Normalización en Python

```
# Cargamos los datos:
# Con df.to_numpy() convertimos el dataframe a arrays de Numpy.
# Se establecen la variable objetivo y las variables de entrada.
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from pandas import read_csv
nomFichero = 'winequality-red.csv'
# No es necesario añadir nombres de columnas, ya que el csv los trae
```

```
# Se usa ; como separador, y no la coma por defecto
datosWines = read_csv(nomFichero, sep=";")
variables  = datosWines.to_numpy()
x = variables[:,0:11] # Variables 0 a 10
y = variables[:,11] # Variable 11, que es una categoría
# Se realiza la división entrenamiento-test:
# Se usa 30% para test y 70% para entrenamiento.
tamTest = 0.3
semillas = 90
# Se pasa random_state para que los resultados sean reproducibles en diferentes
# llamadas train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=tamTest,
random_state=semillas)
# Normalización:
# Se debe realizar sobre el conjunto de entrenamiento y, posteriormente, aplicar al
conjunto de test. Se realiza el reescalado al intervalo [0,1].
scaler    = MinMaxScaler(feature_range= (0, 1))
xTrainNor = scaler.fit_transform(xTrain)
# Una vez normalizadas las xTrain, debemos estandarizar los xTest
xTestNor  = scaler.transform(xTest)
# Deshacemos normalización:
# Se muestra como deshacerlo por si fuera necesario
xTrain2 = scaler.inverse_transform(xTrainNor)

# Mostramos resultados:
print("\nDatos originales:\n")
print(xTrain)
print("\nDatos obtenidos al deshacer la normalización:\n")
print(xTrain2)

Datos originales xTrain:
[[7.4          0.37          0.43       …        3.33         0.68          9.7 ]
[10.4          0.44          0.73       …        3.17         0.85          12.0]
[ 9.6          0.33          0.52       …        3.36         0.76          12.4]
…
[7.2          0.655          0.03       …        3.34         0.39          9.5]
[8.2          0.73           0.21       …        3.20         0.52          9.5]]
[11.3          0.34          0.45       …        2.94         0.66          9.2]]

Datos obtenidos al deshacer la normalización:
[[7.4          0.37          0.43       …        3.33         0.68          9.7 ]
[10.4          0.44          0.73       …        3.17         0.85          12.0]
[ 9.6          0.33          0.52       …        3.36         0.76          12.4]
…
[7.2          0.655          0.03       …        3.34         0.39          9.5]
[8.2          0.73           0.21       …        3.20         0.52          9.5]]
[11.3          0.34          0.45       …        2.94         0.66          9.2]]
```

Ejercicio 97_02. Normalización en Matlab

```
% Cargar los datos
```

```
nomFichero = 'winequality-red.csv';
datosWines = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datosWines);
% Seleccionar variables predictoras y variable objetivo
% x = variables(:, 1:11); % Variables 1 a 11
% y = variables(:, 12);   % Variable 12, que es una categoría
% para coger sólo las 10 primeras filas y las 7 últimas columnas
x = variables(1:10, 5:11); % Variables 5 a 11
y = variables(1:10, 12);   % Variable 12, que es una categoría
% División entrenamiento-test
tamTest = 0.3;
semillas = 90;
rng(semillas); % Establecer la semilla para reproducibilidad
particion = cvpartition(y, 'Holdout', tamTest);
% Conjunto de entrenamiento
xTrain = x(training(particion), :);
yTrain = y(training(particion), :);
% Conjunto de test
xTest = x(test(particion), :);
yTest = y(test(particion), :);
% Normalización
xTrainNor = normalize(xTrain, 'range', [0, 1]);
% Normalizar el conjunto de test
xTestNor = normalize(xTest, 'range', [0, 1]);
% % Deshacer normalización (para demostración)
% xTrain2 = xTrainNor * range(xTrain)' + min(xTrain);
% Mostrar resultados
disp('Datos originales xTrain:');
disp(xTrain);
disp('Datos obtenidos al hacer la normalización:');
disp(xTrainNor);
% disp('Datos obtenidos al deshacer la normalización:');
% disp(xTrain2);

Datos originales xTrain:
0.076           11           34       0.9978         3.51         0.56          9.4
0.098           25           67       0.9968          3.2         0.68          9.8
0.092           15           54        0.997         3.26         0.65          9.8
0.069           15           59       0.9964          3.3         0.46          9.4
0.065           15           21       0.9946         3.39         0.47           10
0.073            9           18       0.9968         3.36         0.57          9.5
0.071           17          102       0.9978         3.35          0.8         10.5

Datos obtenidos al hacer la normalización:
0.33333        0.125     0.19048        1            1          0.29412      0
1              1         0.58333        0.6875       0          0.64706      0.36364
0.81818        0.375     0.42857        0.75         0.19355    0.55882      0.36364
0.12121        0.375     0.4881         0.5625       0.32258    0            0
0              0.375     0.035714       0            0.6129     0.029412     0.54545
0.24242        0         0              0.6875       0.51613    0.32353      0.090909
0.18182        0.5       1              1            0.48387    1            1
```

Autores: María Inmaculada Rodríguez García, María Gema Carrasco García, Javier González Enrique, Juan Jesús Ruiz Aguilar, Ignacio J. Turias Domínguez. Universidad de Cádiz.

2.6. Métricas para problemas de regresión y clasificación.

Principales métricas para problemas de regresión:

• **R^2 (coeficiente de determinación):** Representa la proporción de la varianza de una variable dependiente que es posible explicar mediante una variable independiente o variables en un modelo de regresión.

Documentación: R2 Regression score function

Documentación: Regresión logística

Documentación: Regresión múltiple lineal

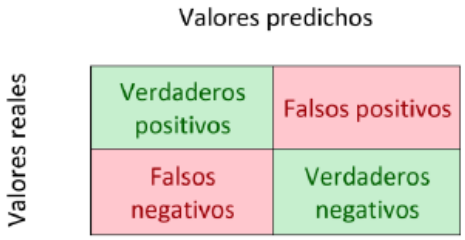
• **MSE:** El error cuadrático medio se calcula como la diferencia cuadrática media entre los valores estimados y los valores reales.

Documentación: MSE Mean squeared error regression loss

Principales métricas para problemas de clasificación:

• **Precisión de la clasificación:** Se puede definir como la ratio de predicciones correctas. Realmente sólo se emplea bien cuando hay el mismo número de observaciones en cada clase y los errores de predicción al equivocarnos al clasificar son igualmente importantes.
• **Área bajo la curva ROC:** Se emplea en problemas de clasificación binarios. Un área de 1 indica que el modelo ha podido clasificar todos los casos correctamente. Un área de 0.5 indica que el modelo es tan bueno como clasificar aleatoriamente.
• **Matriz de confusión:** Está indicado para representar la precisión de un modelo con dos o más clases.

Documentación: Confussion matrix



2.7. Validación cruzada *k-Folds* y *leave one out*.

Validación cruzada *kFolds*:

• Se divide el subconjunto de training en un número k de particiones (2, 5, 10…).

• El modelo se entrena usando k-1 particiones, también llamadas *folds*, para training y 1 para test (que algunos autores denominan validación). Esto se repite hasta que

todas las particiones han actuado como partición de test exactamente una vez (ver gráfico).

• Los indicadores de calidad del modelo se obtiene como media de esas repeticiones individuales.

• Para problemas de **clasificación** se usa una variante conocida como **validación cruzada estratificada** (por defecto en *Scikit-Learn* para problemas de clasificación) En ella, se dividen los datos de manera que las proporciones entre las clases sean las mismas en cada división que en todo el conjunto de datos original.

Documentación: Validación cruzada

Documentación: Validación cruzada estratificada

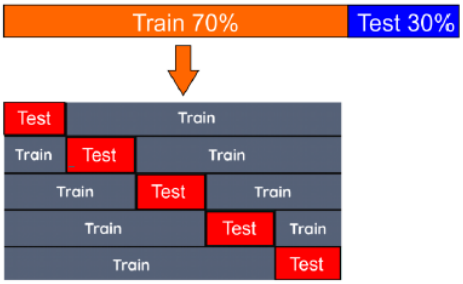
Validación cruzada *leave one out*:

• Similar a la anterior, pero en este caso hay tantas particiones como el número de observaciones en el dataset.

• Es computacionalmente mucho más costoso que la validación cruzada anteriormente expuesta.

• Se suele emplear cuando la cantidad de datos disponibles es pequeña.

Documentación: Validación cruzada *Leave one Out*



Ejercicio 98_01. Validación cruzada con regresión en Python

```
# Este es un ejemplo simple para ilustrar la validación cruzada. Se debería estudiar
si es necesario realizar un pretratamiento de los datos, etc.
from pandas import read_csv
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

```
# Cargamos los datos:
# La variable objetivo será la estación 14, mientras que el resto de variables será
nuestras variables de entrada.
nomFichero = 'no2.csv'
cols = ['Estacion 1', 'Estacion 2', 'Estacion 3', 'Estacion 4', 'Estacion 5',\
        'Estacion 6', 'Estacion 7', 'Estacion 8', 'Estacion 9', 'Estacion 10',\
        'Estacion 11', 'Estacion 12', 'Estacion 13', 'Estacion 14']
datos = read_csv(nomFichero, names = cols)
variables = datos.to_numpy()
x = variables[:,0:13]
y = variables[:,13]
tamTest = 0.30
semillas = 20
# Se pasa random_state para que los resultados sean reproducibles en diferentes
# llamadas train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = tamTest,
random_state = semillas)
# Establecemos el modelo:
# En este ejemplo usaremos regresión múltiple lineal.
modelo = LinearRegression()
# Validación cruzada:
# Usaremos 10-validacion cruzada aleatoria.<br>
# Establecemos el MSE como métrica y el número de hilos a usar para los cálculos.
particiones = 10 # Número de particiones
validación = KFold(n_splits=particiones, random_state = semillas, shuffle = True)
metrica = ('neg_mean_squared_error') # Error cuadrático medio
hilos_cores_a_usar =8; # Podemos modificarlo para usar hilos si disponibles
# Resultados en la etapa de entrenamiento:
print("Resultados de la etapa de entrenamiento:\n")
resultados = cross_val_score(modelo, xTrain, yTrain, cv = particiones,
             scoring = metrica, n_jobs = hilos_cores_a_usar)
# Hay que usar absoluto porque esta métrica en Python la da en negativo
print("MSE: %f - Desviación típica: %f" %(abs(resultados.mean()),
        resultados.std()))
metrica = 'r2'
resultados = cross_val_score(modelo, xTrain, yTrain, cv=particiones,
              scoring=metrica, n_jobs=hilos_cores_a_usar)
print("R2: %f - Desviación típica: %f" % (resultados.mean(),
        resultados.std()))
% # Resultados de la etapa de test:
modelo.fit(xTrain, yTrain)
yPredichos = modelo.predict(xTest)
print("Resultados de la etapa de test:\n")
print("R2:")
print(r2_score(yTest, yPredichos))
print("MSE:")
print(mean_squared_error(yTest, yPredichos))
```

Ejercicio 98_02. Validación cruzada con regresión en MATLAB

```
% Este es un ejemplo simple para ilustrar la validación cruzada.
% Se debería estudiar si es necesario realizar un pretratamiento de los datos, etc.
```

```
% Cargamos los datos:
% La variable objetivo será la estación 14, mientras que el resto de variables será
nuestras variables de entrada.
nomFichero = 'no2_2.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datos);
x = variables(:, 1:13);
y = variables(:, 14);
tamTest = 0.30;
semillas = 20;
% Se pasa 'rng' para que los resultados sean reproducibles en diferentes
% llamadas a 'datasample'
rng(semillas);
% Dividimos los datos en conjunto de entrenamiento y prueba
idx = datasample(1:size(variables, 1), round((1 - tamTest) * size(variables, 1)),
'Replace', false);
xTrain = x(idx, :);
yTrain = y(idx);
xTest = x(setdiff(1:size(variables, 1), idx), :);
yTest = y(setdiff(1:size(variables, 1), idx));
% Establecemos el modelo:
% En este ejemplo usaremos regresión múltiple lineal.
modelo = fitlm(xTrain, yTrain);
% Validación cruzada:
particiones = 10; % Número de particiones
cv = cvpartition(length(yTrain), 'KFold', particiones, 'Stratify', false);
metrica = 'mse'; % Error cuadrático medio
% Resultados en la etapa de entrenamiento:
disp('Resultados de la etapa de entrenamiento:');
% crossval('mcr',X1,X2,X3,y,'Predfun',@classf,'Stratify',y)
resultados = crossval('mse', xTrain, yTrain, 'Predfun', @(xTrain, yTrain, xTest,
yTest) predict(fitlm(xTrain, yTrain), xTest), 'partition', cv);
disp(['MSE: ', num2str(resultados)]);
% Calculamos R^2 manualmente:
resultados_r2 = zeros(particiones, 1);
for i = 1:particiones
    idx_train = training(cv, i);
    idx_test = test(cv, i);
        modelo_temporal = fitlm(xTrain(idx_train, :), yTrain(idx_train));
    yPredichos_temporal = predict(modelo_temporal, xTrain(idx_test, :));
        resultados_r2(i) = 1 - sum((yTrain(idx_test) - yPredichos_temporal).^2) /
sum((yTrain(idx_test) - mean(yTrain(idx_train))).^2);
end
disp(['R^2: ', num2str(mean(resultados_r2))]);
% Resultados de la etapa de test:
yPredichos = predict(modelo, xTest);
r2_test = 1 - sum((yTest - yPredichos).^2) / sum((yTest - mean(yTrain)).^2);
disp('Resultados de la etapa de test:');
disp(['R^2: ', num2str(r2_test)]);
disp(['MSE: ', num2str(mean((yTest - yPredichos).^2))]);

Resultados de la etapa de entrenamiento:
MSE: 19.2685
```

```
R^2: 0.81102
Resultados de la etapa de test:
R^2: 0.8214
MSE: 13.0131
```

Ejercicio 99_01. Clasificación con validación *leave one out* con regresión logística en Python

```
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
# Cargamos los datos:
# La variable objetivo será la variable 8 "clase", mientras que el resto de variables
será nuestras variables de entrada.
nomFichero = 'pima-indians-diabetes.data.csv'
cols = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
datos = read_csv(nomFichero, names=cols)
variables  = datos.to_numpy()
x = variables[:,0:7] # Variables 0 a 7
y = variables[:,8] # Variable 8
# División entrenamiento-test:
# Usaremos 70% para entrenamiento y 30% para test, obtenidos aleatoriamente.
# Usamos semillas y ramdon_state para que los resultados sean reproducibles.
tamTest = 0.3
semillas = 23
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = tamTest,
random_state = semillas)
% # Establecemos el modelo:
% # Al ser un ejemplo de clasificación, usaremos regresión logística.
modelo = LogisticRegression(solver='liblinear')
% # Validación cruzada leave one out:
% # Usaremos 10 particiones.<br>
% # Establecemos el número de hilos a usar para los cálculos.
validacion = LeaveOneOut()
hilos_cores_a_usar = 8; # Podemos modificarlo para usar + hilos si disponibles
particiones = 10;
% # Resultados en la etapa de entrenamiento:
% # Como métricas, usaremos la precisión y el área bajo la curva ROC.
print("Resultados de la etapa de entrenamiento:\n")
metrica = 'accuracy'
resultados = cross_val_score(modelo, xTrain, yTrain, cv = particiones,
             scoring = metrica, n_jobs = hilos_cores_a_usar)
print("Precisión validación cruzada: %f - Desviación típica: %f" %(resultados.mean(),
        resultados.std()))
metrica = 'roc_auc'
```

```
resultados = cross_val_score(modelo, xTrain, yTrain, cv=particiones,
             scoring=metrica, n_job=hilos_cores_a_usar)
print("Área bajo la curva ROC validación cruzada: %f - Desviación típica: %f"
      %(resultados.mean(), resultados.std()))

Resultados de la etapa de entrenamiento:

Precisión validación cruzada: 0.763452 – Desviación típica: 0.050628
Área bajo la curva ROC validación cruzada: 0.824573 – Desviación típica: 0.040479

% # Resultados de la etapa de test:
% # Mostramos la precisión, el área bajo la curva ROC y la matriz de confusión.
modelo.fit(xTrain, yTrain)
yPredichos =  modelo.predict(xTest)
print("Resultados de la etapa de test:\n")
precision =  accuracy_score(yTest, yPredichos)
print("Precisión de test: %.f %%" %(precision*100.0))
aRoc = roc_auc_score(yTest, yPredichos)
print("Área bajo la curva ROC test: %f" %(aRoc))
matriz = confusion_matrix(yTest, yPredichos)
print("Matriz de confusión test:")
print(matriz)

Resultados de la etapa de test:

Precisión de test: 78%
Área bajo la curva ROC test: 0.733190
Matriz de confusión test:
[[135  13]
[37  46]]
```

Ejercicio 99_02. Clasificación con validación *leave one out* con regresión logística en MATLAB

```
% Cargamos los datos:
nomFichero = 'pima-indians-diabetes.data.csv';
datos = readtable(nomFichero, 'Delimiter', ',');
variables = table2array(datos);
X = variables(:, 1:8); % Variables 1 a 8
y = variables(:, 9);   % Variable 9
% División entrenamiento-test:
tamTest = 0.3;
semillas = 23;
rng(semillas);
[idxTrain, idxTest] = datasample(1:size(variables, 1), round((1 - tamTest) *
size(variables, 1)), 'Replace', false);
xTrain = X(idxTrain, :);
yTrain = y(idxTrain);
xTest = X(idxTest, :);
yTest = y(idxTest);
% Validación Leave-One-Out:
numMuestras = length(yTrain);
predicciones = zeros(numMuestras, 1);
```

```
% Validación Leave-One-Out
num_muestras = length(y);
for i = 1:num_muestras
    % Dejar una muestra fuera para validación
    X_validacion = X(i, :);
    y_validacion = y(i);
    X_entrenamiento = X([1:i-1, i+1:end], :);
    y_entrenamiento = y([1:i-1, i+1:end]);
    % Ajustar el modelo de regresión logística
    modelo = fitglm(X_entrenamiento, y_entrenamiento, 'Distribution', 'binomial',
'Link', 'logit');
    % Realizar predicciones para la muestra dejada fuera
    y_predicho = predict(modelo, X_validacion);
    predicciones(i) = y_predicho;
    etiquetas_verdaderas(i) = y_validacion;
end
disp(modelo);
% Calcular el área bajo la curva ROC (AUC)
[X_roc, Y_roc, ~, AUC] = perfcurve(etiquetas_verdaderas, predicciones, 1);
disp(['Área bajo la curva ROC: ', num2str(AUC)]);
% Graficar la curva ROC
figure;
plot(X_roc, Y_roc);
xlabel('Tasa de Falsos Positivos');
ylabel('Tasa de Verdaderos Positivos');
title('Curva ROC');
% Realizar predicciones en el conjunto de prueba
yPredichos = predict(modelo, xTest);
% Realizar predicciones en el conjunto de prueba
yPredichos = predict(modelo, xTest);
% Convertir las probabilidades a etiquetas binarias usando un umbral (por ejemplo,
0.5)
umbral = 0.5;
etiquetas_binarias = zeros(size(yPredichos));  % Inicializar un vector de ceros
% Aplicar el umbral y asignar etiquetas binarias
for i = 1:length(yPredichos)
    if yPredichos(i) >= umbral
        etiquetas_binarias(i) = 1;
    end
end
% Calcular la precisión en el conjunto de prueba
precision = sum(etiquetas_binarias == yTest) / length(yTest);
disp(['Precisión de test: ', num2str(precision * 100), '%']);

modelo = Generalized linear regression model:
    logit(y) ~ 1 + x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8
    Distribution = Binomial
Estimated Coefficients:
                    Estimate        SE         tStat        pValue
                   ________    _________    ________    __________

   (Intercept)        -8.3979      0.71665     -11.718    1.0286e-31
    x1                0.12302     0.032074      3.8354    0.00012534
```

```
    x2                0.035143    0.0037082      9.4771    2.6154e-21
    x3               -0.013283    0.0052325     -2.5386      0.011131
    x4              0.00066315    0.0068997    0.096112       0.92343
    x5              -0.0011963    0.0009012     -1.3274       0.18436
    x6                0.089617     0.015085      5.9409     2.835e-09
    x7                  0.9439      0.29908       3.156     0.0015992
    x8                0.014843    0.0093334      1.5903       0.11177
767 observations, 758 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 269, p-value = 1.37e-53

Área bajo la curva ROC: 0.82982
Precisión de test: 76.3941%
```