Chapter-wise Lab Codes
on
Machine Learning Concept 1(CSE 3967)

Centre for Data Science
Faculty of Engineering and Technology (ITER)
Siksha 'O' Anusandhan (Deemed to be University)

August 2025

CHAPTER 1

LAB-1:BASIC PYTHON

install Python Anaconda distribution from https://anaconda.org/anaconda/python. After that we will use Jupyter as a web-based interactive notebook environment for writing and running code.

In [ ]:
```
pip install numpy
```

In [ ]:
```
pip install pandas
```

In [ ]:
```
!pip install matplotlib
```

Basic data types in Python

Python has five standard data types −

1. Number
2. String
3. List
4. Tuple
5. Dictionary

Boolean data type (bool) is a subtype of integer. It is a unique data type, consisting of two constants, True and False.

In [1]:
```
num=input("enter a numner")
print(type (num))
```

```
enter a numner3
<class 'str'>
```

In [2]:
```
num=5
type(num)
```

Out[2]:
```
int
```

In [3]:
```
num=7.9
type(num)
```

Out[3]:
```
float
```

In [4]:
```
num = -3+7.2j
```

```python
type(num)
```

Out[4]: complex

In [5]:
```python
var1 = True
type(var1)
```

Out[5]: bool

In [6]:
```python
# Arithmetic Operations
a = 10
b = 3

print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
print("Floor Division:", a // b)
print("Modulus:", a % b)
print("Exponentiation:", a ** b)
```

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.3333333333333335
Floor Division: 3
Modulus: 1
Exponentiation: 1000
```

In [7]:
```python
# Comparison Operations
x = 5
y = 8

print("Equal:", x == y)
print("Not Equal:", x != y)
print("Greater than:", x > y)
print("Less than:", x < y)
print("Greater than or equal:", x >= y)
print("Less than or equal:", x <= y)
```

```
Equal: False
Not Equal: True
Greater than: False
Less than: True
Greater than or equal: False
Less than or equal: True
```

In [8]:
```python
# Logical Operations
p = True
q = False

print("AND:", p and q)
print("OR:", p or q)
print("NOT p:", not p)
```

```
print("NOT q:", not q)
```

```
AND: False
OR: True
NOT p: False
NOT q: True
```

In [9]:
```
# String Operations
str1 = "Hello"
str2 = "World"

print("Concatenation:", str1 + " " + str2)
print("Repetition:", str1 * 3)
print("Length:", len(str1))
print("Uppercase:", str1.upper())
print("Lowercase:", str1.lower())
print("Contains 'ell':", "ell" in str1)
```

```
Concatenation: Hello World
Repetition: HelloHelloHello
Length: 5
Uppercase: HELLO
Lowercase: hello
Contains 'ell': True
```

String is a group of characters. These characters may be alphabets, digits or special characters including spaces.

In [ ]:
```
```

In [10]:
```
str1 = 'Hello Friend'
str2 = "452"
print(type(str1))
print(type(str2))
```

```
<class 'str'>
<class 'str'>
```

List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

In [11]:
```
list1 = [5, 3.4, "New Delhi", "20C", 45] #print the elements of the list list1
print(list1)
type(list1)
```

```
[5, 3.4, 'New Delhi', '20C', 45]
```

Out[11]:
```
list
```

In [12]:
```
# List Slicing and Methods
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print("Original:", numbers)
print("First 3:", numbers[:3])
```

```
print("Last 3:", numbers[-3:])
print("Middle:", numbers[3:7])
print("Every second:", numbers[::2])
print("Reverse:", numbers[::-1])

print("Length:", len(numbers))
print("Sum:", sum(numbers))
print("Max:", max(numbers))
print("Min:", min(numbers))
```

```
Original: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
First 3: [0, 1, 2]
Last 3: [7, 8, 9]
Middle: [3, 4, 5, 6]
Every second: [0, 2, 4, 6, 8]
Reverse: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Length: 10
Sum: 45
Max: 9
Min: 0
```

Tuple

Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ). This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change the tuple.

In [13]:
```
tuple1 = (10, 20, "Apple", 3.4, 'a') #print the elements of the tuple tuple1
print(tuple1)
type(tuple1)
```

```
(10, 20, 'Apple', 3.4, 'a')
```

Out[13]:
```
tuple
```

Dictionary

Dictionary in Python holds data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key : value pairs of a dictionary can be accessed using the key. The keys are usually strings and their values can be any data type. In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

In [14]:
```
dict1 = {'Fruit':'Apple',
'Climate':'Cold', 'Price(kg)':120}
print(dict1)
print(dict1['Price(kg)'])
```

```
{'Fruit': 'Apple', 'Climate': 'Cold', 'Price(kg)': 120}
120
```

In [15]:
```
dict2= {1:'Akash',
2:'Amit', 3:'Rohit'}
print(dict2)
print(dict2[3])
```

```
{1: 'Akash', 2: 'Amit', 3: 'Rohit'}
Rohit
```

for–while Loops & if–else Statement

In [16]:
```
for i in range(1,4):
 print(i*i)
```

```
1
4
9
```

In [17]:
```
print("For loop - list items:")
fruits = ["apple", "banana", "cherry"]
for i in fruits:
    print(i)
```

```
For loop - list items:
apple
banana
cherry
```

In [18]:
```
for i in range(0,4,2):
 print(i)
```

```
0
2
```

In [19]:
```
#Printing squares of all integers from 0 to 4.
i = 0
while i < 5:
 print(i*i)
 i = i + 1
```

```
0
1
4
9
16
```

if-else statement

In [20]:
```
i = -5
if i < 0:
 print(i*i)
else:
```

```
print(i)
```

25

Writing functions

Writing a function (in a script): Syntax: def functionname(parametername,...): (function_body)

In [21]:
```
#Function to calculate factorial of an input number n.
def factorial(n):
 fact = 1
 for i in range(1, n+1):
  fact = fact *i
 return(fact)
factorial(6)
```

Out[21]: 720

Numpy

Numpy is a library for scientific computing. It is useful for working with arrays and matrices. Numpy is used in many scientific computing applications, including machine learning and deep learning. Numpy is imported using the import keyword. Numpy is usually imported using the alias np .

In [22]:
```
import numpy as np
```

Numpy arrays

Numpy arrays are used to store multiple items in a single variable. They can be created using the np.array function. Numpy arrays are similar to lists, but they are faster and more efficient. Numpy arrays can be created from lists, tuples, and other arrays.

In [23]:
```
x = np.array([1, 2, 3])
print(x)
type(x)
```

[1 2 3]

Out[23]: numpy.ndarray

In [24]:
```
list=[[[1,'2',3], [4,5,6], [5,6,7]]]
list
```

Out[24]: [[[1, '2', 3], [4, 5, 6], [5, 6, 7]]]

In [26]:
```
arr = np.array([10, 20, 30, 40])
print(np.mean(arr))
```

```
25.0
```

In [27]:
```
arr = np.array([10, 20, 30, 40])
print(np.median(arr))
```

```
25.0
```

In [28]:
```
arr = np.array([10, 20, 30, 40])
print(np.std(arr))
```

```
11.180339887498949
```

Creating arrays in numpy

An ndarray is a generic multidimensional container for homogeneous data; that is, all of the elements must be the same type

The easiest way to create an ndarray is to use the array function in numpy module.

Nested sequences, like a list of equal length lists, will be converted into a multidimensional array

In [29]:
```
data=np.array([1.9, 4.9, -4])
print(data, 'and its type is', type(data))
```

```
[ 1.9  4.9 -4. ] and its type is <class 'numpy.ndarray'>
```

In [30]:
```
# Different data type converted into the same
data1=np.array(['Hello', 1])
print(data)
```

```
[ 1.9  4.9 -4. ]
```

Shape and dimension of ndarray

arr.ndim: function return the number of dimensions of an array.

arr.shape: shape of an array is the number of elements in each dimension.

arr.size: Try this and see what you are getting.

In [31]:
```
data=np.array([[1,4.7,3],['Hello', 5, 'ITER']])
print(data)
print(data.ndim)
print(data.shape)
```

```
print(data.size)
```

```
[['1' '4.7' '3']
 ['Hello' '5' 'ITER']]
2
(2, 3)
6
```

#Other functions for creating new arrays

numpy.zeros and numpy.ones create arrays of 0s or 1s, respectively, with a given length or shape.

numpy.empty creates an array without initializing its values to any particular value.

To create a higher-dimensional array with these methods, pass a tuple for the shape.

In [32]:
```
x=np.zeros(10)
x
```

Out[32]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [33]:
```
y=np.ones(10)
y
```

Out[33]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [34]:
```
x=np.zeros((2,3))
x
```

Out[34]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [35]:
```
x=np.empty(2)
x
```

Out[35]:
```
array([4.24399158e-313, 8.48798317e-313])
```

arange Like the built-in range but returns an ndarray instead of a list syntax: numpy.arange(start = 0, stop, step = 1, dtype = None) linspace() function is used to create an array of evenly spaced numbers within a specified range

In [36]:
```
np.arange(6)
```

Out[36]:
```
array([0, 1, 2, 3, 4, 5])
```

In [37]:
```
np.arange(10,20,5)
```
Out[37]:
```
array([10, 15])
```
In [38]:
```
np.linspace(10,20,5)
```
Out[38]:
```
array([10. , 12.5, 15. , 17.5, 20. ])
```

Creating new arrays

Produce an array of the given shape and data type with all values set to the indicated ”fill value” numpy.full(shape, fill value) eye/identity Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere) Return a new array of given shape and type, filled with fill value.¶

In [39]:
```
np.full(10, 5)
```
Out[39]:
```
array([5, 5, 5, 5, 5, 5, 5, 5, 5, 5])
```
In [40]:
```
np.full((2,3), 5)
```
Out[40]:
```
array([[5, 5, 5],
       [5, 5, 5]])
```
In [41]:
```
np.eye(4)
```
Out[41]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```
In [42]:
```
np.identity(4)
```
Out[42]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```
In [43]:
```
np.eye(4,k=-1)
```
Out[43]:
```
array([[0., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
```

Data types for ndarrays

The data type or dtype is a special object containing the information about data.

numpy tries to infer a good data type for the array that it creates. You can explicitly convert or cast an array from one data type to another using ndarray‟s astype method. A string which cannot be converted to float64, if we use astype method, a Value Error will be raised

In [44]:
```
arr1=np.array([1,2,3], dtype=np.float64)
arr2=np.array([1,2,3], dtype=np.int32)
arr3=np.array([1,2,3])
arr4=np.array(['1','2','3'])
arr5=np.array([['1111','2222','3333'],['1','2','3']])
print( arr1, arr2)
print(type(arr1))
print(arr1.dtype)
print(arr2.dtype)
print(arr3.dtype)
print(arr4.dtype) #Unicode string.
print(arr5.dtype)
```

```
[1. 2. 3.] [1 2 3]
<class 'numpy.ndarray'>
float64
int32
int32
<U1
<U4
```

In [45]:
```
arr1=np.array([1,2,3,4,5,6])
print(arr1.dtype)
```

```
int32
```

In [46]:
```
arr1=arr1.astype(np.float64)
print(arr1, arr1.dtype)
```

```
[1. 2. 3. 4. 5. 6.] float64
```

Arithmetic With NumPy arrays

Batch operations on data can be performed in numpy without writing any for loops. This is known as vectorization. Any arithmetic operations between equal size arrays apply the operation element wise.

In [47]:
```
arr1=np.array([[1,2,3],[4,5,6]])
arr2=np.array([[5,6,7],[9,3,2]])
arr1+arr2
```

Out[47]:
```
array([[ 6,  8, 10],
       [13,  8,  8]])
```

In [48]:
```
arr1*arr2
```

Out[48]:
```
array([[ 5, 12, 21],
       [36, 15, 12]])
```

In [49]:
```
7*arr1
```

Out[49]:
```
array([[ 7, 14, 21],
       [28, 35, 42]])
```

In [50]:
```
arr1**2
```

Out[50]:
```
array([[ 1,  4,  9],
       [16, 25, 36]])
```

In [51]:
```
arr1>arr2
```

Out[51]:
```
array([[False, False, False],
       [False,  True,  True]])
```

In [52]:
```
7/arr2
```

Out[52]:
```
array([[1.4       , 1.16666667, 1.        ],
       [0.77777778, 2.33333333, 3.5       ]])
```

In [ ]:

In [ ]:

CHAPTER 2

INTRODUCTION TO PANDA

Pandas is a powerful and open-source Python library used for data manipulation and analysis.Pandas consist of data structures and functions to perform efficient operations on data.It is built on top of the NumPy library which means that a lot of the structures of NumPy are used or replicated in Pandas and the data produced by Pandas is often used as input for plotting functions in Matplotlib.

Pandas Usage

• Data set cleaning, merging, and joining.

• Easy handling of missing data (represented as NaN) in floating point as well as non-floating-point data.

• Columns can be inserted and deleted from DataFrame and higher-dimensional objects.

• Powerful group by functionality for performing split-apply-combine operations on data sets.

• Data Visualization.

• The Pandas library allows to work with tabular data with columns of different data types, such as that from Excel spreadsheets, CSV files from the internet, and SQL database tables,Time series data, either at fixed-frequency or not, other structured datasets, such as those coming from web data, like JSON files

The Panda module is generally imported as follows: import pandas as pd

Data Structures in Pandas Library

Pandas generally provide two data structures for manipulating data. They are:
• **Series**: The Pandas Series structure, is a one-dimensional homogenous array.
• **DataFrame**: The pandas DataFrame structure, is a twodimensional, mutable, and potentially heterogeneous structure.

Syntax to create a Series

pandas.Series (data, index=idx (optional))

Where data may be python sequence (Lists), ndarray, scalar value or a python dictionary

How to create Series with nd array

In [1]:
```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22])
s = pd.Series(arr)
print(s)
```

```
0    10
1    15
2    18
3    22
dtype: int64
```

How to create Series with Mutable index

In [2]:
```
import pandas as pd
import numpy as np
arr=np.array(['a','b','c','d'])
s=pd.Series(arr, index=['first','second','third','fourth'])
print(s)
```

```
first     a
second    b
third     c
fourth    d
dtype: object
```

Creating a series from Scalar value

In [3]:
```
import pandas as pd
s = pd.Series(50, index=[0, 1, 2, 3, 4])
print(s)
```

```
0    50
1    50
2    50
3    50
4    50
dtype: int64
```

Mathematical Operations in Series

In [7]:
```
import pandas as pd

s1 = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
s2 = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'])
```

```
s3 = pd.Series([5, 14, 23, 32], index=['a', 'b', 'c', 'd'])

print('To Add Series1 & Series2')
print('---------------------------------------------')
print(s1 + s2)

print('To Add Series2 & Series3')
print('---------------------------------------------')
print(s2 + s3) #While adding two series, if Non-Matching Index is found in eit

print('To Add Series2 & Series3 and Fill Non-Matching Index with 0')
print('---------------------------------------------')
print(s2.add(s3, fill_value=0))#If Non-Matching Index is found in either of th
```

```
To Add Series1 & Series2
---------------------------------------------
a    11
b    22
c    33
d    44
e    55
dtype: int64
To Add Series2 & Series3
---------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e     NaN
dtype: float64
To Add Series2 & Series3 and Fill Non-Matching Index with 0
---------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e    50.0
dtype: float64
```

Head and Tail Functions in Series

head (): It is used to access the first 5 rows of a series.

Note :To access first 3 rows we can call series_name.head(3)

In [2]:
```
import pandas as pd
import numpy as np
arr = np.array([10, 15, 18, 22, 55, 77, 42, 48, 97])
s = pd.Series(arr)
print(s.head())
print(s.head(3))
```

```
0    10
1    15
2    18
3    22
4    55
dtype: int64
0    10
1    15
2    18
dtype: int64
```

tail(): It is used to access the last 5 rows of a series.

Note :To access last 4 rows we can call series_name.tail (4)

In [9]:
```
import pandas as pd
import numpy as np
arr = np.array([10, 15, 18, 22, 55, 77, 42, 48, 97])
s = pd.Series(arr)
print(s.tail())
print(s.tail(4))
```

```
4    55
5    77
6    42
7    48
8    97
dtype: int64
5    77
6    42
7    48
8    97
dtype: int64
```

Selection in Series Series provides index label loc and iloc and [] to access rows and columns.

1. loc index label :-

Syntax:-series_name.loc[StartRange: StopRange]

2. Selection Using iloc index label :-

Syntax:-series_name.iloc[StartRange : StopRange]

3. Selection Using [] :

Syntax:-series_name[StartRange> : StopRange] or series_name[index]

In [10]:
```
import pandas as pd
import numpy as np
arr = np.array([10, 15, 18, 22, 55, 77])
s = pd.Series(arr)
print(s)
print(s.loc[:2])
print(s.loc[3:4])
print(s.loc[2:3])
```

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int64
0    10
1    15
2    18
dtype: int64
3    22
4    55
dtype: int64
2    18
3    22
dtype: int64
```

In [5]:
```
import pandas as pd
import numpy as np
#dictionary to series
S=pd.Series({'a':1,'b':2,'c':3})
print(S)
```

```
a    1
b    2
c    3
dtype: int64
```

In [6]:
```
import pandas as pd
import numpy as np
S=pd.Series((1,2,3,4),index=['a','b','c','d'])
print(S)
```

```
a    1
b    2
c    3
d    4
dtype: int64
```

Indexing in Series

Pandas provide index attribute to get or set the index of entries or values in series.

In [11]:
```
import pandas as pd
import numpy as np

# create a numpy array
arr = np.array(['a', 'b', 'c', 'd'])
s = pd.Series(arr, index=['first', 'second', 'third', 'fourth'])
print(s)
print(s.index)
```

```
first     a
second    b
third     c
fourth    d
dtype: object
Index(['first', 'second', 'third', 'fourth'], dtype='object')
```

DATAFRAME-It is a two-dimensional object that is useful in representing data in the form of rows and columns. It is similar to a spreadsheet or an SQL table. This is the most commonly used pandas object. Once we store the data into the Dataframe, we can perform various operations that are useful in analyzing and understanding the data. A Dataframe has axes (indices)-

➢ Row index (axis=0)

➢ Column index (axes=1)

2. It is similar to a spreadsheet , whose row index is called index and column index is called column name.

3. A Dataframe contains Heterogeneous data.

4. A Dataframe Size is Mutable.

5. A Dataframe Data is Mutable.

5. The data can also contain missing data, as represented by the NaN (not a number) values.

A data frame can be created using any of the following-

1. Series
2. Lists
3. Dictionary
4. A numpy 2D array

How to create Dataframe From Series

In [1]:
```python
import pandas as pd
s = pd.Series(['a','b','c','d'])
df=pd.DataFrame(s)
print(df)
```

```
   0
0  a
1  b
2  c
3  d
```

DataFrame from Dictionary of Series

In [1]:
```python
import pandas as pd
name = pd.Series(['Hardik', 'Virat'])
team = pd.Series(['MI', 'RCB'])
dic = {'Name': name, 'Team': team}
```

```
df = pd.DataFrame(dic)
print(df)
```

```
     Name Team
0  Hardik   MI
1   Virat  RCB
```

DataFrame from List of Dictionaries

In [2]:
```
import pandas as pd

dicl = [
    {'FirstName': 'Sachin', 'LastName': 'Bhardwaj'},
    {'FirstName': 'Vinod', 'LastName': 'Verma'},
    {'FirstName': 'Rajesh', 'LastName': 'Mishra'}
]
df1 = pd.DataFrame(dicl)
print(df1)
```

```
  FirstName  LastName
0    Sachin  Bhardwaj
1     Vinod     Verma
2    Rajesh    Mishra
```

Iteration on Rows and Columns If we want to access record or data from a data frame row wise or column wise then iteration is used. Pandas provide 2 functions to perform iterations-

1. iterrows (): It is used to access the data row wise.
2. items (): It is used to access data coulumn wise

In [8]:
```
import pandas as pd

dicl = [
    {'FirstName': 'Sachin', 'LastName': 'Bhardwaj'},
    {'FirstName': 'Vinod', 'LastName': 'Verma'},
    {'FirstName': 'Rajesh', 'LastName': 'Mishra'}
]
df1 = pd.DataFrame(dicl)
print(df1)
for (row_index, row_value) in df1.iterrows():
    print("\nRow index is ::", row_index)
    print("Row Value is ::")
    print(row_value)
```

```
  FirstName  LastName
0    Sachin  Bhardwaj
1     Vinod     Verma
2    Rajesh    Mishra

Row index is :: 0
Row Value is ::
FirstName      Sachin
LastName     Bhardwaj
Name: 0, dtype: object

Row index is :: 1
Row Value is ::
FirstName    Vinod
LastName     Verma
Name: 1, dtype: object

Row index is :: 2
Row Value is ::
FirstName    Rajesh
LastName     Mishra
Name: 2, dtype: object
```

In [6]:

```
import pandas as pd

dicl = [
    {'FirstName': 'Sachin', 'LastName': 'Bhardwaj'},
    {'FirstName': 'Vinod', 'LastName': 'Verma'},
    {'FirstName': 'Rajesh', 'LastName': 'Mishra'}
]
df1 = pd.DataFrame(dicl)
print(df1)
for (coulumn_name, coulumn_value) in df1.items():
    print("\n Coulumn name is ::", coulumn_name)
    print("Coulumn Value is ::")
    print(coulumn_value)
```

```
  FirstName  LastName
0    Sachin  Bhardwaj
1     Vinod     Verma
2    Rajesh    Mishra

 Coulumn name is :: FirstName
Coulumn Value is ::
0    Sachin
1     Vinod
2    Rajesh
Name: FirstName, dtype: object

 Coulumn name is :: LastName
Coulumn Value is ::
0    Bhardwaj
1       Verma
2      Mishra
Name: LastName, dtype: object
```

Select operation in data frame To access the column data ,we can mention the column name as subscript. e.g. - df[empid]. This can also be done by using df.empid. To access multiple columns we can write as df[[col1, col2,---]]

In [11]:
```
import pandas as pd

empdata = {
    'empid': [101, 102, 103, 104, 105, 106],
    'ename': ['Sachin', 'Vinod', 'Lakhbir', 'Anil', 'Devinder', 'UmaSelvi'],
    'Doj': ['12-01-2012', '15-01-2012', '05-09-2007', '17-01-2012', '05-09-200
}

df = pd.DataFrame(empdata)
print(df)
```

```
   empid     ename         Doj
0    101    Sachin  12-01-2012
1    102     Vinod  15-01-2012
2    103   Lakhbir  05-09-2007
3    104      Anil  17-01-2012
4    105  Devinder  05-09-2007
5    106  UmaSelvi  16-01-2012
```

In [12]:
```
df.empid
```

Out[12]:
```
0    101
1    102
2    103
3    104
4    105
5    106
Name: empid, dtype: int64
```

In [14]:
```
df['empid']
```

Out[14]:
```
0    101
1    102
2    103
3    104
4    105
5    106
Name: empid, dtype: int64
```

In [16]:
```
df[['empid','ename']]
```

Out[16]:

	empid	ename
0	101	Sachin
1	102	Vinod
2	103	Lakhbir
3	104	Anil
4	105	Devinder
5	106	UmaSelvi

To Add & Rename a column in data frame

In [9]:
```
import pandas as pd
s = pd.Series([10,15,18,22])
df=pd.DataFrame(s)
print(df)
df.columns=['List1'] #To Rename the default column of Data Frame as List1
df['List2']=20 #To create a new column List2 with all values as 20
df['List3']=df['List1']+df['List2'] #Add Column1 and Column2 and store in New
print(df)
```

```
    0
0  10
1  15
2  18
3  22
   List1  List2  List3
0     10     20     30
1     15     20     35
2     18     20     38
3     22     20     42
```

To Delete a Column in data frame

We can delete the column from a data frame by using any of the the following –

1. del
2. pop()
3. drop()

In [25]:
```
import pandas as pd
```

```
s = pd.Series([10,15,18,22])
df=pd.DataFrame(s)
df.columns=['List1'] #To Rename the default column of Data Frame as List1
df['List2']=20 #To create a new column List2 with all values as 20
df['List3']=df['List1']+df['List2'] #Add Column1 and Column2 and store in New
print(df)
del df['List3']
print(df)
```

```
   List1  List2  List3
0     10     20     30
1     15     20     35
2     18     20     38
3     22     20     42
   List1  List2
0     10     20
1     15     20
2     18     20
3     22     20
```

In [26]:
```
df.pop('List2')
print(df)
```

```
   List1
0     10
1     15
2     18
3     22
```

In [14]:
```
import pandas as pd
s= pd.Series([10,20,30,40])
df=pd.DataFrame(s)
df.columns=['List1']
df['List2']=40
print(df)
df1=df.drop('List2',axis=1) #(axis=1) means to delete Data column wise
df2=df.drop(index=[2,3],axis=0) #(axis=0) means to delete data row wise with g
print(" After deletion::")
print(df1)
print (" After row deletion::")
print(df2)
```

```
   List1  List2
0     10     40
1     20     40
2     30     40
3     40     40
 After deletion::
   List1
0     10
1     20
2     30
3     40
 After row deletion::
   List1  List2
0     10     40
1     20     40
```

Accessing the data frame through loc() and iloc() method or indexing using Labels

Pandas provide loc() and iloc() methods to access the subset from a data frame using row/column.

Accessing the data frame through loc()

It is used to access a group of rows and columns.

Syntax- Df.loc[StartRow : EndRow, StartColumn : EndColumn]

In [29]:
```
import pandas as pd

Runs = {
    'TCS': {'Qtr1': 2500, 'Qtr2': 2000, 'Qtr3': 3000, 'Qtr4': 2000},
    'WIPRO': {'Qtr1': 2800, 'Qtr2': 2400, 'Qtr3': 3600, 'Qtr4': 2400},
    'L&T': {'Qtr1': 2100, 'Qtr2': 5700, 'Qtr3': 35000, 'Qtr4': 2100}
}

df = pd.DataFrame(Runs)
print(df)

# Select only Qtr3 row
print(df.loc['Qtr3', :])

# Select rows from Qtr1 to Qtr3
print(df.loc['Qtr1':'Qtr3', :])
```

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
TCS       3000
WIPRO     3600
L&T      35000
Name: Qtr3, dtype: int64
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
```

In [32]:
```
print(df.loc[:,'TCS'])#To access single column
print(df.loc[:,'TCS':'WIPRO'])#To access multiple column
```

```
Qtr1    2500
Qtr2    2000
Qtr3    3000
Qtr4    2000
Name: TCS, dtype: int64
       TCS  WIPRO
Qtr1  2500   2800
Qtr2  2000   2400
Qtr3  3000   3600
Qtr4  2000   2400
```

In [33]:
```
import pandas as pd

# Data dictionary
empdata = {
    'empid': [101, 102, 103, 104, 105, 106],
    'ename': ['Sachin', 'Vinod', 'Lakhbir', 'Anil', 'Devinder', 'UmaSelvi'],
    'Doj': ['12-01-2012', '15-01-2012', '05-09-2007', '17-01-2012', '05-09-200
}
df = pd.DataFrame(empdata)
print(df)
# Access first row using .loc
print(df.loc[0])
# Access first three rows using .loc
print(df.loc[0:2])
```

```
   empid     ename         Doj
0    101    Sachin  12-01-2012
1    102     Vinod  15-01-2012
2    103   Lakhbir  05-09-2007
3    104      Anil  17-01-2012
4    105  Devinder  05-09-2007
5    106  UmaSelvi  16-01-2012
empid           101
ename        Sachin
Doj      12-01-2012
Name: 0, dtype: object
   empid    ename         Doj
0    101   Sachin  12-01-2012
1    102    Vinod  15-01-2012
2    103  Lakhbir  05-09-2007
```

Accessing the data frame through iloc()

It is used to access a group of rows and columns based on numeric index value.

Syntax- Df.loc[StartRowindexs : EndRowindex, StartColumnindex : EndColumnindex]

In [34]:
```
import pandas as pd

# Data dictionary for company earnings
Runs = {
    'TCS': {'Qtr1': '2500', 'Qtr2': '2000', 'Qtr3': '3000', 'Qtr4': '2000'},
    'WIPRO': {'Qtr1': '2800', 'Qtr2': '2400', 'Qtr3': '3600', 'Qtr4': '2400'},
    'L&T': {'Qtr1': '2100', 'Qtr2': '5700', 'Qtr3': '35000', 'Qtr4': '2100'}
}

df = pd.DataFrame(Runs)
print(df)
# Accessing specific rows and columns using iloc
print(df.iloc[0:2, 1:2])  # First 2 rows, second column
print(df.iloc[:, 0:2])    # All rows, first 2 columns
```

```
       TCS WIPRO    L&T
Qtr1  2500  2800   2100
Qtr2  2000  2400   5700
Qtr3  3000  3600  35000
Qtr4  2000  2400   2100
     WIPRO
Qtr1  2800
Qtr2  2400
       TCS WIPRO
Qtr1  2500  2800
Qtr2  2000  2400
Qtr3  3000  3600
Qtr4  2000  2400
```

head() and tail() Method

The method head() gives the first 5 rows and the method tail() returns the last 5 rows.

To display first 2 rows we can use head(2) and to returns last2 rows we can use tail(2) and to return 3rd to 4th row we can write df[2:5].

In [35]:
```
import pandas as pd
empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007', '17-01-2012','05-09-2
df=pd.DataFrame(empdata)
print(df)
print(df.head())
print(df.tail())
```

```
          Doj  empid     ename
0  12-01-2012    101    Sachin
1  15-01-2012    102     Vinod
2  05-09-2007    103   Lakhbir
3  17-01-2012    104      Anil
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
          Doj  empid     ename
0  12-01-2012    101    Sachin
1  15-01-2012    102     Vinod
2  05-09-2007    103   Lakhbir
3  17-01-2012    104      Anil
4  05-09-2007    105  Devinder
          Doj  empid     ename
1  15-01-2012    102     Vinod
2  05-09-2007    103   Lakhbir
3  17-01-2012    104      Anil
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
```

Accessing a Specific DataFrame Cell by Row and Column

In [36]:
```
import pandas as pd

# Data for students' marks in different tests
marks = pd.DataFrame({
    'Wally' : [87, 89, 93, 87, 92],
    'Eva' : [95, 99, 87, 88, 84],
    'Sam' : [88, 94, 85, 89, 95],
    'Katie' : [87, 92, 95, 84, 85],
    'Bob' : [83, 93, 86, 83, 87]
}, index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])

# Printing all marks
print('All marks:')
print(marks)

# Accessing Eva's Test01 marks using different methods
print('\nAccessing Eva\'s Test01 marks: ')
print('Method 01:', marks['Eva']['Test01'])  # Column name then Index name
print('Method 02:', marks.Eva.Test01)        # Column name then Index name
print('Method 03:', marks.at['Test01', 'Eva'])  # Row name then Column name
```

```
print('Method 04:', marks.iat[0, 1])  # Row index then Column index
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
Test04     87   88   89     84   83
Test05     92   84   95     85   87

Accessing Eva's Test01 marks:
Method 01: 95
Method 02: 95
Method 03: 95
Method 04: 95
```

Boolean Indexing

In [37]:
```
import pandas as pd

# Data for students' marks in different tests
marks = pd.DataFrame({
    'Wally' : [87, 89, 93, 87, 92],
    'Eva' : [95, 99, 87, 88, 84],
    'Sam' : [88, 94, 85, 89, 95],
    'Katie' : [87, 92, 95, 84, 85],
    'Bob' : [83, 93, 86, 83, 87]
}, index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])

# Displaying 0 grade students
print('0 grade students:')
print(marks[marks >= 90])  # Filter students with marks >= 90

# Displaying A grade students
print('\nA grade students:')
print(marks[(marks >= 80) & (marks < 90)])  # Filter students with marks betwe
```

```
0 grade students:
        Wally   Eva   Sam  Katie   Bob
Test01    NaN  95.0   NaN    NaN   NaN
Test02    NaN  99.0  94.0   92.0  93.0
Test03   93.0   NaN   NaN   95.0   NaN
Test04    NaN   NaN   NaN    NaN   NaN
Test05   92.0   NaN  95.0    NaN   NaN

A grade students:
        Wally   Eva   Sam  Katie   Bob
Test01   87.0   NaN  88.0   87.0  83.0
Test02   89.0   NaN   NaN    NaN   NaN
Test03    NaN  87.0  85.0    NaN  86.0
Test04   87.0  88.0  89.0   84.0  83.0
Test05    NaN  84.0   NaN   85.0  87.0
```

Transposing the DataFrame with the T Attribute

In [39]:

```python
import pandas as pd

# Create a DataFrame with student marks
marks = pd.DataFrame({
    'Wally': [87, 89, 93, 87, 92],
    'Eva': [95, 99, 87, 88, 84],
    'Sam': [88, 94, 85, 89, 95]
}, index=['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])

# Transpose the DataFrame
marksTransposed = marks.T

# Display results
print('All marks:')
print(marks)

print('\nAll marks Transposed:')
print(marksTransposed)
```

```
All marks:
        Wally  Eva  Sam
Test01     87   95   88
Test02     89   99   94
Test03     93   87   85
Test04     87   88   89
Test05     92   84   95

All marks Transposed:
       Test01  Test02  Test03  Test04  Test05
Wally      87      89      93      87      92
Eva        95      99      87      88      84
Sam        88      94      85      89      95
```

Sorting by Rows and Columns by Their Indices (axis=0 for rows and axis = 1 for columns)

In [41]:

```python
import pandas as pd

# Create a DataFrame with student marks
marks = pd.DataFrame({
    'Wally': [87, 89, 93],
    'Eva': [95, 99, 87],
    'Sam': [88, 94, 85],
    'Katie': [87, 92, 95],
    'Bob': [83, 93, 86]
}, index=['Test01', 'Test02', 'Test03'])

# Display the DataFrame
print('All marks:')
print(marks)

# Sort by row indices (descending order)
print('\nAll marks Sorted by Row indices:')
```

```
print(marks.sort_index(ascending=False))

# Sort by column indices (alphabetical order of names)
print('\nAll marks Sorted by Column indices:')
print(marks.sort_index(axis=1))
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86

All marks Sorted by Row indices:
        Wally  Eva  Sam  Katie  Bob
Test03     93   87   85     95   86
Test02     89   99   94     92   93
Test01     87   95   88     87   83

All marks Sorted by Column indices:
        Bob  Eva  Katie  Sam  Wally
Test01   83   95     87   88     87
Test02   93   99     92   94     89
Test03   86   87     95   85     93
```

Sorting by Column Values (axis = 1 for columns)

In [42]:

```
import pandas as pd

# Create a DataFrame with student marks
marks = pd.DataFrame({
    'Wally': [87, 89, 93],
    'Eva': [95, 99, 87],
    'Sam': [88, 94, 85],
    'Katie': [87, 92, 95],
    'Bob': [83, 93, 86]
}, index=['Test01', 'Test02', 'Test03'])

# Display original marks
print('All marks:')
print(marks)

# Sort columns by values of Test01 and Test02 (descending order)
print('\nAll marks Sorted by Column values:')
print(marks.sort_values(by='Test01', axis=1, ascending=False))
print(marks.sort_values(by='Test02', axis=1, ascending=False))

# Transpose and sort by Test01 row values (descending order)
print('\nTranspose and sort:')
print(marks.T.sort_values(by='Test01', ascending=False))

# Select Test01 row and sort values (descending order)
print('\nSelect and sort:')
print(marks.loc['Test01'].sort_values(ascending=False))
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86

All marks Sorted by Column values:
        Eva  Sam  Wally  Katie  Bob
Test01   95   88     87     87   83
Test02   99   94     89     92   93
Test03   87   85     93     95   86
        Eva  Sam  Bob  Katie  Wally
Test01   95   88   83     87     87
Test02   99   94   93     92     89
Test03   87   85   86     95     93

Transpose and sort:
       Test01  Test02  Test03
Eva        95      99      87
Sam        88      94      85
Wally      87      89      93
Katie      87      92      95
Bob        83      93      86

Select and sort:
Eva      95
Sam      88
Wally    87
Katie    87
Bob      83
Name: Test01, dtype: int64
```

Creating CSV file

In [4]:
```python
import pandas as pd

# Create a sample dictionary of data
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [24, 27, 22, 32],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}

# Convert dictionary to DataFrame
df = pd.DataFrame(data)

# Save DataFrame to CSV file
df.to_csv('Data.csv', index=False)

print("CSV file 'Data.csv' created successfully!")
```

```
CSV file 'Data.csv' created successfully!
```

In [5]:
```python
df2 = pd.read_csv('Data.csv')
```

```
print(df2)
```

```
      Name  Age         City
0    Alice   24     New York
1      Bob   27  Los Angeles
2  Charlie   22      Chicago
3    David   32      Houston
```

Downloading a csv data file directly from the web

In [10]:
```
import pandas as pd

# Define column names from UCI Auto MPG dataset
column_names = [
    "mpg", "cylinders", "displacement", "horsepower", "weight",
    "acceleration", "model_year", "origin", "car_name"
]

# URL of Auto MPG dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto

# Load dataset
df = pd.read_csv(
    url,
    sep='\s+',   # data separated by spaces
    names=column_names,      # assign column names
    na_values="?"            # missing values marked as '?'
)

# Save to local CSV file
df.to_csv("auto_mpg.csv", index=False)

print("✅ Auto MPG dataset saved as 'auto_mpg.csv'")
print(df.head())
```

```
✅ Auto MPG dataset saved as 'auto_mpg.csv'
    mpg  cylinders  displacement  horsepower  weight  acceleration  \
0  18.0          8         307.0       130.0  3504.0          12.0
1  15.0          8         350.0       165.0  3693.0          11.5
2  18.0          8         318.0       150.0  3436.0          11.0
3  16.0          8         304.0       150.0  3433.0          12.0
4  17.0          8         302.0       140.0  3449.0          10.5

   model_year  origin                   car_name
0          70       1  chevrolet chevelle malibu
1          70       1          buick skylark 320
2          70       1         plymouth satellite
3          70       1              amc rebel sst
4          70       1                ford torino
```

In [3]:
```
df.head(10)
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
0	18.0	8	307.0	130.0	3504.0	12.0	70
1	15.0	8	350.0	165.0	3693.0	11.5	70
2	18.0	8	318.0	150.0	3436.0	11.0	70
3	16.0	8	304.0	150.0	3433.0	12.0	70
4	17.0	8	302.0	140.0	3449.0	10.5	70
5	15.0	8	429.0	198.0	4341.0	10.0	70
6	14.0	8	454.0	220.0	4354.0	9.0	70
7	14.0	8	440.0	215.0	4312.0	8.5	70
8	14.0	8	455.0	225.0	4425.0	10.0	70
9	15.0	8	390.0	190.0	3850.0	8.5	70

In [10]:

```
df.tail(10)
```

Out[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_ye
388	26.0	4	156.0	92.0	2585.0	14.5	
389	22.0	6	232.0	112.0	2835.0	14.7	
390	32.0	4	144.0	96.0	2665.0	13.9	
391	36.0	4	135.0	84.0	2370.0	13.0	
392	27.0	4	151.0	90.0	2950.0	17.3	
393	27.0	4	140.0	86.0	2790.0	15.6	
394	44.0	4	97.0	52.0	2130.0	24.6	
395	32.0	4	135.0	84.0	2295.0	11.6	
396	28.0	4	120.0	79.0	2625.0	18.6	
397	31.0	4	119.0	82.0	2720.0	19.4	

Understanding data using df.info()

The df.info() method is a quick way to look at the data types, missing values, and data size of a DataFrame.

- show_counts = True: gives a few over the total nonmissing values in each column.
- memory_usage = True: shows the total memory usage of the DataFrame elements.
- verbose = True: prints the full summary from df.info()

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car_name      398 non-null    object 
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

Understanding data using df.describe()

Prints the summary statistics of all numeric columns, such as

- count,
- mean,
- standard deviation,
- range, and
- quartiles of numeric columns.

In [13]:
```
df.describe()
```

Out[13]:

	mpg	cylinders	displacement	horsepower	weight	accelera
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.00
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.56
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.75
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.00
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.82
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.50
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.17
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.80

Modifying the quartiles

- You can also modify the quartiles using the percentiles argument.
- Here, for example, we’re looking at the 30%, 50%, and 70% percentiles of the numeric columns in DataFrame df.

In [16]:
```
df.describe(percentiles=[0.3,0.5,0.7])
```

Out[16]:

	mpg	cylinders	displacement	horsepower	weight	accelera
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.00
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.56
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.75
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.00
30%	18.000000	4.000000	112.000000	80.000000	2301.000000	14.20
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.50
70%	27.490000	6.000000	250.000000	110.000000	3424.500000	16.80
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.80

Categorical Data

In case of categorical data the df.describe()method summarizes by

- number of observations,
- number of unique elements,
- mode, and
- frequency of the mode.

In [4]:

```
df['car_name'].describe()
```

Out[4]:

```
count            398
unique           305
top       ford pinto
freq               6
Name: car_name, dtype: object
```

In [5]:

```
df.describe(include='all')
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceler
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.0
unique	NaN	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	NaN	
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.5
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.7
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.0
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.8
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.5
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.1
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.8

Understanding your data using df.shape

• The number of rows and columns of a DataFrame can be identified using the .shape attribute of the DataFrame.

• It returns a tuple (row, column) and can be indexed to get only rows, and only columns count as output.

In [12]:

```
print('(Rows, Cols): ', df.shape)
print('Rows: ', df.shape[0])
print('Cols: ', df.shape[1])
```

```
(Rows, Cols):  (398, 9)
Rows:  398
Cols:  9
```

Get all columns and column names

• Calling the df.columns attribute of a DataFrame object returns the column names in the form of an Index object.

• As a reminder, a pandas index is the address/label of the row or column.

• This can also be converted to a Python list object.

In [11]:

```
print(df.columns)
col_list=list(df.columns)
```

```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'car_name'],
      dtype='object')
```

Handling Duplicates in a DataFrame

df.duplicated(): Used to identify duplicate rows in a DataFrame. It returns a boolean Series where True indicates a row is a duplicate of a previous row,False indicates it is not.

df.drop_duplicates(): will return a copy of your DataFrame, with duplicates removed.

df.drop_duplicates(inplace=True): will modify the DataFrame object in place, with duplicates removed

df.drop_duplicates(inplace=True, keep=first): Drop duplicates in place except for the first occurrence.

df.drop_duplicates(inplace=True, keep=last): Drop duplicates in place except for the last occurrence

df.drop_duplicates(inplace=True, keep=False): Drop all duplicates.

In [13]:
```
print(df.duplicated())
```

```
0      False
1      False
2      False
3      False
4      False
       ...
393    False
394    False
395    False
396    False
397    False
Length: 398, dtype: bool
```

In [15]:
```
import pandas as pd

# Create a sample DataFrame with duplicate rows and values
data = {
    'Name': ['Alice', 'Bob', 'Alice', 'Charlie', 'Bob', 'David'],
    'Age': [25, 30, 25, 35, 30, 40],
    'City': ['New York', 'London', 'New York', 'Paris', 'London', 'Tokyo']
}

df = pd.DataFrame(data)
print("=== Handling Duplicates in a DataFrame ===")

# 1. Display the original DataFrame
print("\n1. Original DataFrame:")
print(df)
```

```
=== Handling Duplicates in a DataFrame ===

1. Original DataFrame:
      Name  Age      City
0    Alice   25  New York
1      Bob   30    London
2    Alice   25  New York
3  Charlie   35     Paris
4      Bob   30    London
5    David   40     Tokyo
```

In [16]:
```
print("\n2. Identify duplicates (duplicated()):")
print("Rows marked as duplicates (True means duplicate):")
print(df.duplicated())   # Checks for duplicate rows

print("\nDuplicate rows only:")
print(df[df.duplicated()])
```

```
2. Identify duplicates (duplicated()):
Rows marked as duplicates (True means duplicate):
0    False
1    False
2     True
3    False
4     True
5    False
dtype: bool

Duplicate rows only:
    Name  Age      City
2  Alice   25  New York
4    Bob   30    London
```

In [17]:
```
# 3. Drop duplicates using drop_duplicates() - keep first occurrence
print("\n3. Drop duplicates (keep='first'):")

df_first = df.drop_duplicates()
print(df_first)
```

```
3. Drop duplicates (keep='first'):
      Name  Age      City
0    Alice   25  New York
1      Bob   30    London
3  Charlie   35     Paris
5    David   40     Tokyo
```

In [18]:
```
# 4. Drop duplicates using drop_duplicates() - keep last occurrence
print("\n4. Drop duplicates (keep='last'):")

df_last = df.drop_duplicates(keep='last')
print(df_last)
```

```
4. Drop duplicates (keep='last'):
      Name  Age      City
2    Alice   25  New York
3  Charlie   35     Paris
4      Bob   30    London
5    David   40     Tokyo
```

In [19]:
```
# 5. Drop duplicates based on specific columns (e.g., 'Name' and 'Age')
print("\n5. Drop duplicates based on 'Name' and 'Age':")

df_subset = df.drop_duplicates(subset=['Name', 'Age'])
print(df_subset)
```

```
5. Drop duplicates based on 'Name' and 'Age':
      Name  Age      City
0    Alice   25  New York
1      Bob   30    London
3  Charlie   35     Paris
5    David   40     Tokyo
```

In [20]:
```
# 6. Drop duplicates and keep neither (drop all duplicates)
print("\n6. Drop duplicates (keep=False):")

df_none = df.drop_duplicates(keep=False)
print(df_none)
```

```
6. Drop duplicates (keep=False):
      Name  Age   City
3  Charlie   35  Paris
5    David   40  Tokyo
```

In [21]:
```
# 7. Count duplicates
print("\n7. Count duplicates:")

duplicate_count = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_count}")
```

```
7. Count duplicates:
Number of duplicate rows: 2
```

In [22]:
```
# 8. Handling duplicates in a specific column (e.g., 'Name')
print("\n8. Unique values in 'Name' column:")

unique_names = df['Name'].drop_duplicates()
print(unique_names)
```

```
8. Unique values in 'Name' column:
0      Alice
1        Bob
3    Charlie
5      David
Name: Name, dtype: object
```

Handling NaN Values

• Detect NaN Values (isna()): df.isna() returns a boolean DataFrame where True indicates a NaN value.

• df.isna().sum() counts NaN values per column.

• Drop Rows with Any NaN (dropna()): df.dropna() removes rows containing any NaN values.

• Drop Rows Where All Values Are NaN: df.dropna(how='all') removes rows where all columns are NaN (none in this example).

• Drop Rows with NaN in Specific Columns:df.dropna(subset=['Age', 'Salary']) removes rows where 'Age' or 'Salary' is NaN.

• Fill NaN with Specific Values (fillna()): Replaces NaN with a specified value (e.g., 0 for numeric columns, 'Unknown' for strings).

• Fill NaN with Column Mean: Uses df['column'].mean() to compute the mean of non- NaN values and fills NaN with that value.

• Forward Fill (fillna(method='ffill')): Propagates the last valid value forward to fill NaN.

• Interpolate NaN Values: df['column'].interpolate(method='linear') estimates NaN values by interpolating between neighboring values (works for numeric columns).

In [23]:
```
import pandas as pd
import numpy as np

# Create a sample DataFrame with NaN values
data = {
    'Name': ['Alice', 'Bob', 'Alice', 'Charlie', 'Bob', 'David'],
    'Age': [25, np.nan, 25, 35, np.nan, 40],
    'City': ['New York', 'London', np.nan, 'Paris', 'London', np.nan],
    'Salary': [50000, 60000, np.nan, np.nan, 60000, 70000]
}

df = pd.DataFrame(data)

# --- Demonstration of Handling NaN Values ---
print("=== Handling NaN Values in a DataFrame ===")

# 1. Display the original DataFrame
print("\n1. Original DataFrame with NaN values:")
print(df)
```

```
=== Handling NaN Values in a DataFrame ===

1. Original DataFrame with NaN values:
      Name   Age      City   Salary
0    Alice  25.0  New York  50000.0
1      Bob   NaN    London  60000.0
2    Alice  25.0       NaN      NaN
3  Charlie  35.0     Paris      NaN
4      Bob   NaN    London  60000.0
5    David  40.0       NaN  70000.0
```

In [24]:
```
# 2. Detect NaN values using isna()
```

```python
print("\n2. Detect NaN values (isna()):")
print(df.isna())

print("\nCount of NaN values per column:")
print(df.isna().sum())
```

```
2. Detect NaN values (isna()):
    Name    Age   City  Salary
0  False  False  False   False
1  False   True  False   False
2  False  False   True    True
3  False  False  False    True
4  False   True  False   False
5  False  False   True   False

Count of NaN values per column:
Name      0
Age       2
City      2
Salary    2
dtype: int64
```

In [25]:
```python
# 3. Drop rows with any NaN values
print("\n3. Drop rows with any NaN (dropna()):")
df_drop_rows = df.dropna()
print(df_drop_rows)
```

```
3. Drop rows with any NaN (dropna()):
    Name   Age      City   Salary
0  Alice  25.0  New York  50000.0
```

In [26]:
```python
# 4. Drop rows where all values are NaN (none in this case)
print("\n4. Drop rows where all values are NaN:")
df_drop_all = df.dropna(how='all')
print(df_drop_all)
```

```
4. Drop rows where all values are NaN:
      Name   Age      City   Salary
0    Alice  25.0  New York  50000.0
1      Bob   NaN    London  60000.0
2    Alice  25.0       NaN      NaN
3  Charlie  35.0     Paris      NaN
4      Bob   NaN    London  60000.0
5    David  40.0       NaN  70000.0
```

In [27]:
```python
# 5. Drop rows where NaN appears in specific columns (e.g., 'Age' and 'Salary'
print("\n5. Drop rows with NaN in 'Age' or 'Salary':")
df_drop_subset = df.dropna(subset=['Age', 'Salary'])
print(df_drop_subset)
```

```
5. Drop rows with NaN in 'Age' or 'Salary':
    Name   Age      City   Salary
0  Alice  25.0  New York  50000.0
5  David  40.0       NaN  70000.0
```

In [28]:
```
# 6. Fill NaN values with a specific value (e.g., 0 for numeric, 'Unknown' for
print("\n6. Fill NaN values with specific values:")
df_fill = df.copy()
df_fill['Age'] = df_fill['Age'].fillna(0)
df_fill['City'] = df_fill['City'].fillna('Unknown')
df_fill['Salary'] = df_fill['Salary'].fillna(0)
print(df_fill)
```

```
6. Fill NaN values with specific values:
      Name   Age      City   Salary
0    Alice  25.0  New York  50000.0
1      Bob   0.0    London  60000.0
2    Alice  25.0   Unknown      0.0
3  Charlie  35.0     Paris      0.0
4      Bob   0.0    London  60000.0
5    David  40.0   Unknown  70000.0
```

In [29]:
```
# 7. Fill NaN values with column mean (for numeric columns)
print("\n7. Fill NaN values with column mean (Age and Salary):")
df_fill_mean = df.copy()
df_fill_mean['Age'] = df_fill_mean['Age'].fillna(df_fill_mean['Age'].mean())
df_fill_mean['Salary'] = df_fill_mean['Salary'].fillna(df_fill_mean['Salary'].
print(df_fill_mean)
```

```
7. Fill NaN values with column mean (Age and Salary):
      Name    Age      City   Salary
0    Alice  25.00  New York  50000.0
1      Bob  31.25    London  60000.0
2    Alice  25.00       NaN  60000.0
3  Charlie  35.00     Paris  60000.0
4      Bob  31.25    London  60000.0
5    David  40.00       NaN  70000.0
```

In [35]:
```
# 8. Forward fill NaN values (propagate previous value forward)
print("\n9. Forward fill NaN values (using obj.ffill):")
df_interpolate = df.copy()
df_interpolate['Age'] = df_interpolate['Age'].ffill()
df_interpolate['Salary'] = df_interpolate['Salary'].ffill()
print(df_interpolate)
```

```
9. Forward fill NaN values (using obj.ffill):
      Name   Age      City   Salary
0    Alice  25.0  New York  50000.0
1      Bob  25.0    London  60000.0
2    Alice  25.0       NaN  60000.0
3  Charlie  35.0     Paris  60000.0
4      Bob  35.0    London  60000.0
5    David  40.0       NaN  70000.0
```

In [34]:
```
# 9. Interpolate NaN values (linear interpolation for numeric columns)
print("\n9. Interpolate NaN values (linear):")
df_interpolate = df.copy()
df_interpolate['Age'] = df_interpolate['Age'].interpolate(method='linear')
df_interpolate['Salary'] = df_interpolate['Salary'].interpolate(method='linear
print(df_interpolate)
```

```
9. Interpolate NaN values (linear):
      Name   Age      City   Salary
0    Alice  25.0  New York  50000.0
1      Bob  25.0    London  60000.0
2    Alice  25.0       NaN  60000.0
3  Charlie  35.0     Paris  60000.0
4      Bob  37.5    London  60000.0
5    David  40.0       NaN  70000.0
```

In [ ]:

📊 Matplotlib Learning Notebook

This notebook covers the fundamentals of **Matplotlib**, a powerful Python library for creating visualizations.

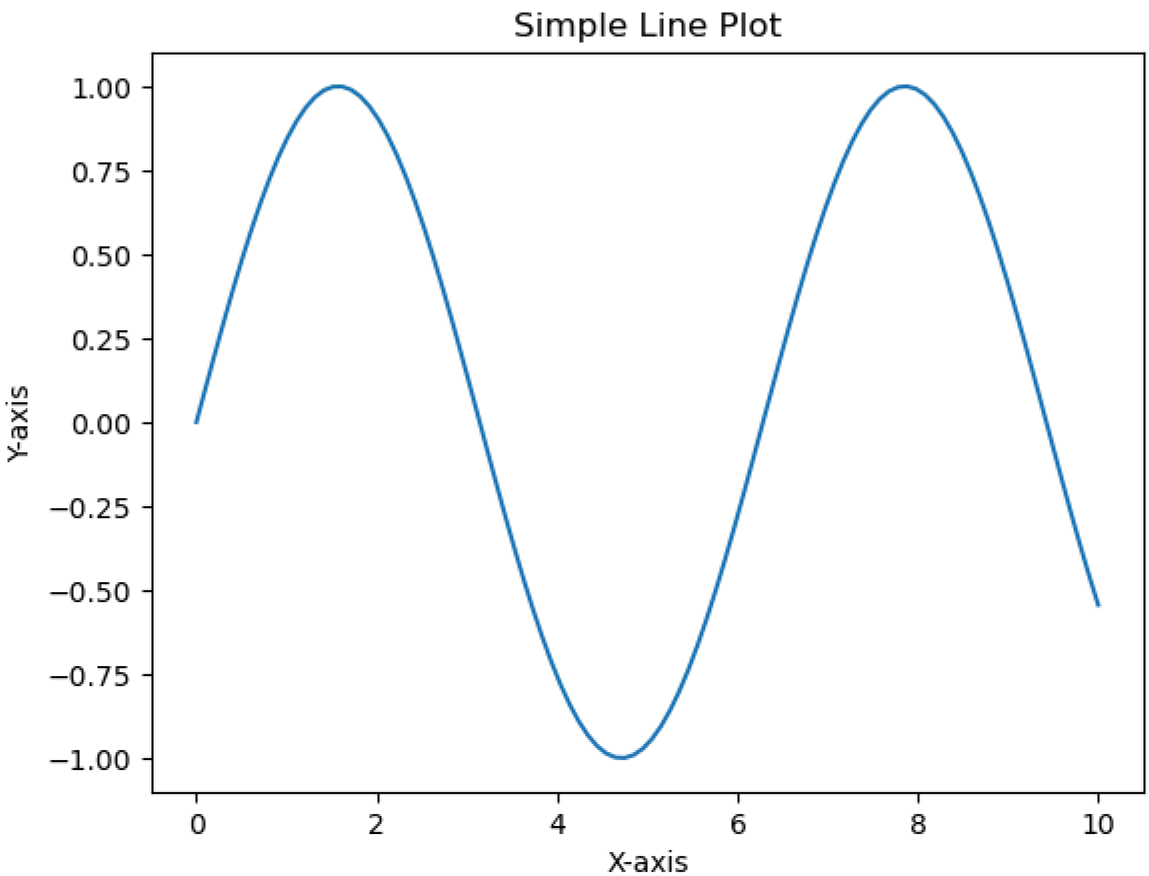
Topics Covered

- Introduction to Matplotlib
- Basic Plotting
- Formatting and Labels
- Multiple Plots
- Bar and Histogram Plots
- Subplots
- Scatter Plots
- Customization
- Saving Figures

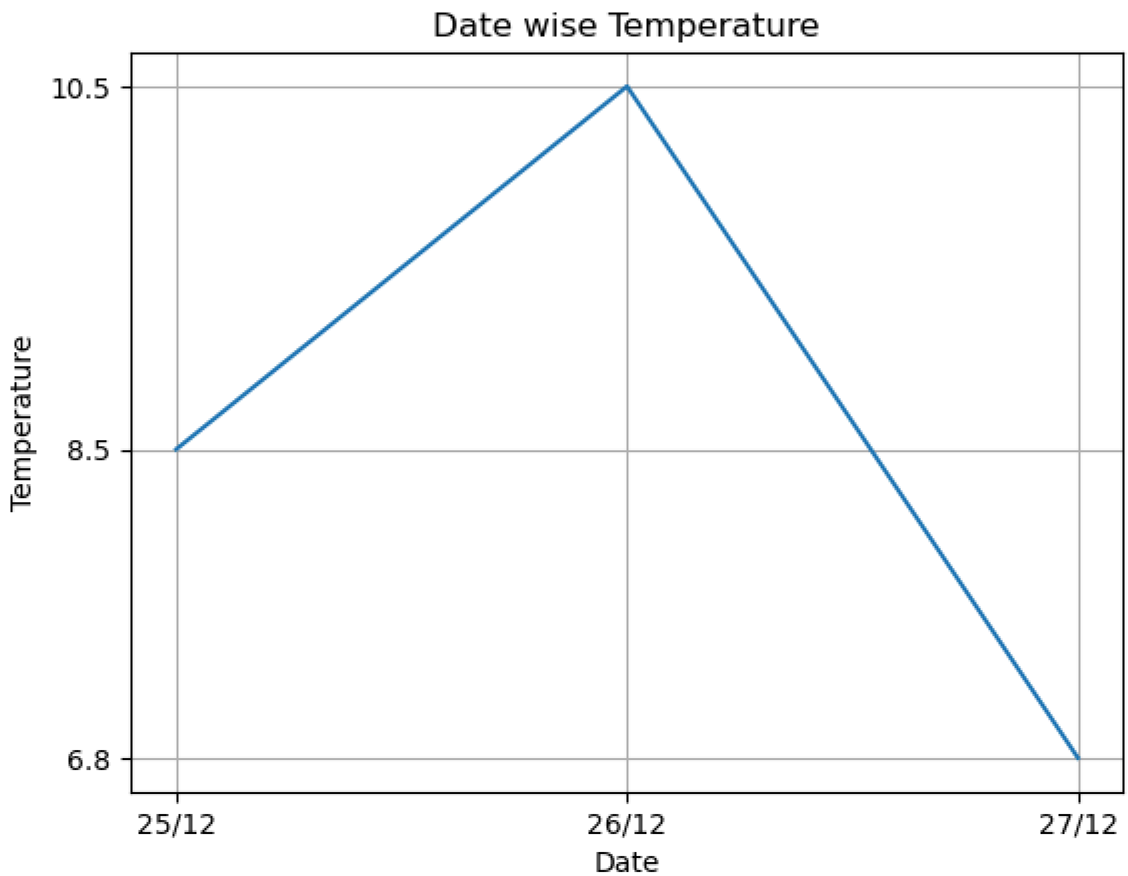
🔹 1. Introduction to Matplotlib

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y)
plt.title('Simple Line Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

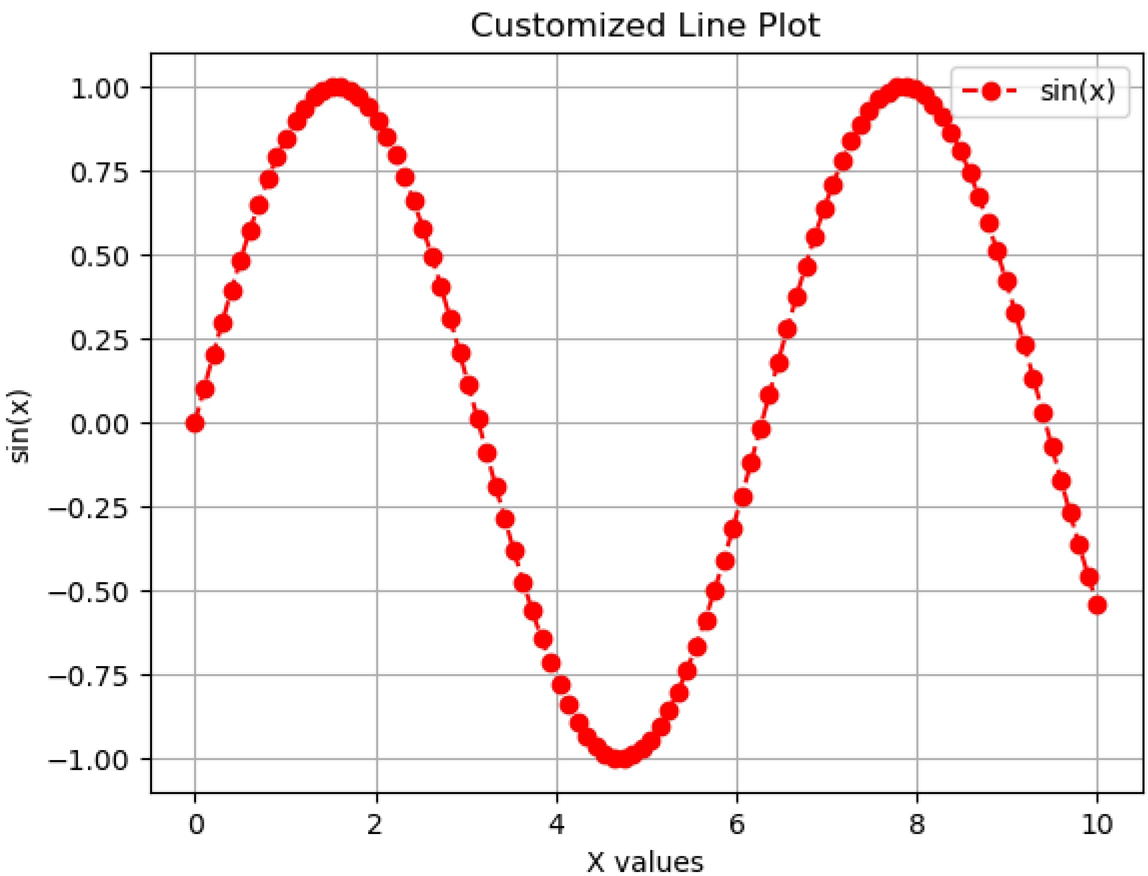


In [1]:
```
import matplotlib.pyplot as plt
date=["25/12","26/12","27/12"]
temp=[8.5,10.5,6.8]
plt.plot(date, temp)
plt.xlabel("Date") #add the Label on x-axis
plt.ylabel("Temperature") #add the Label on y-axis
plt.title("Date wise Temperature") #add the title to the chart
plt.grid(True) #add gridlines to the background
plt.yticks(temp)
plt.show()
```



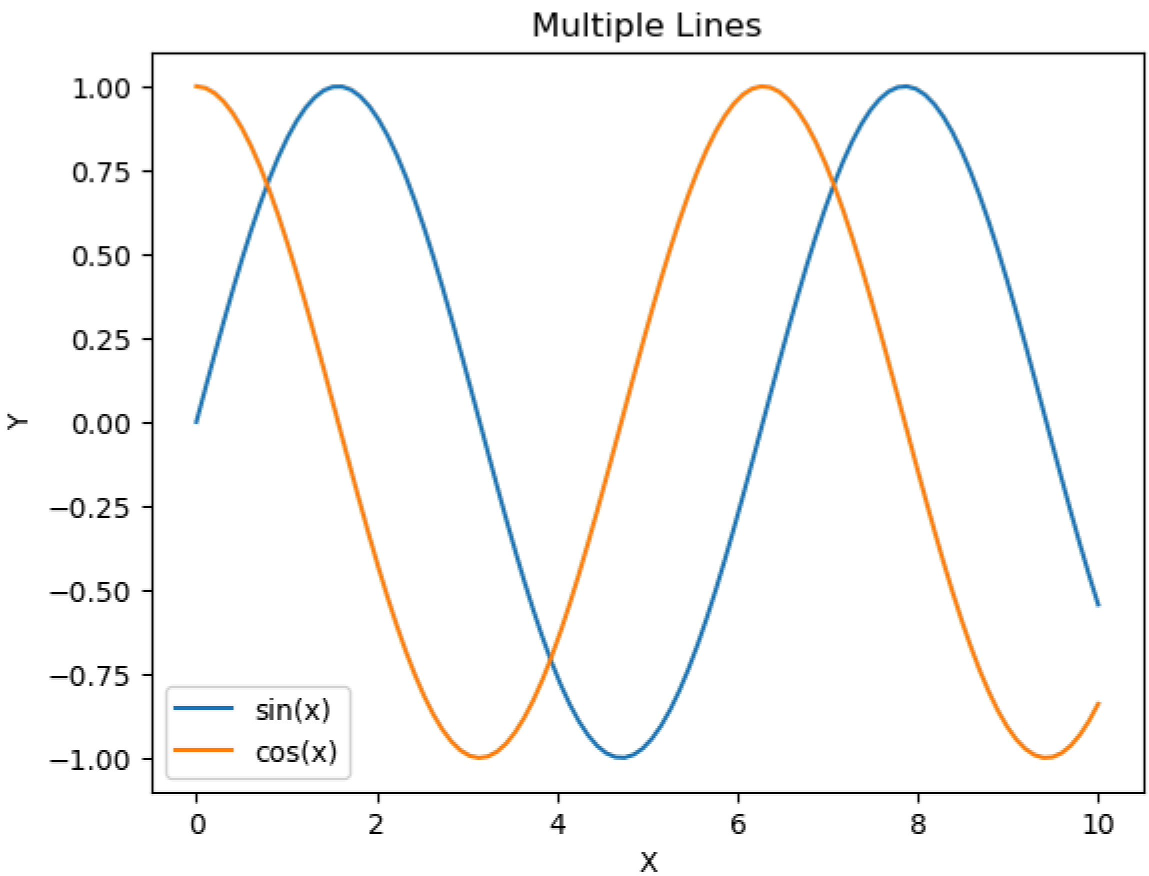
2. Plot Customization

In [2]:
```
plt.plot(x, y, color='red', linestyle='--', marker='o', label='sin(x)')
plt.title('Customized Line Plot')
plt.xlabel('X values')
plt.ylabel('sin(x)')
plt.grid(True)
plt.legend()
plt.show()
```



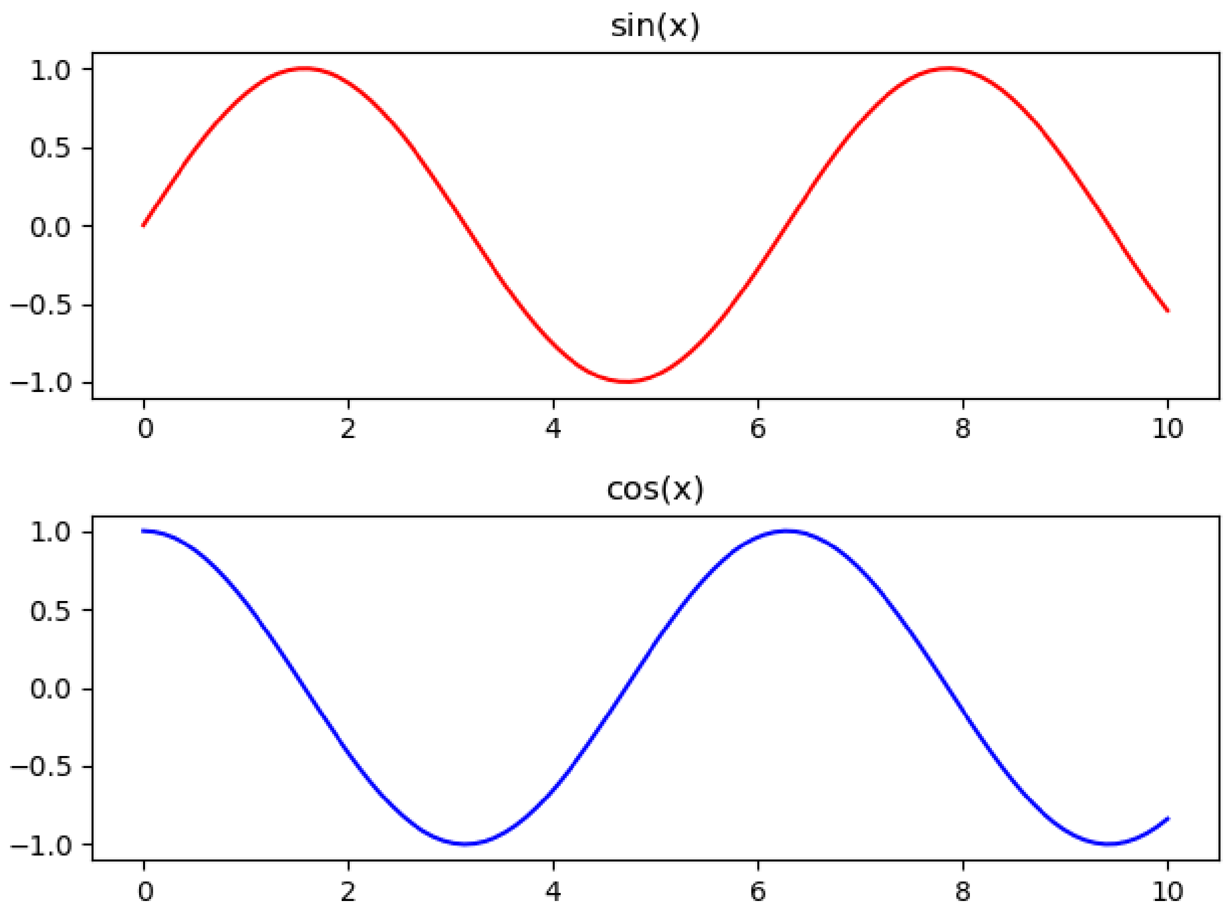
🔹 3. Multiple Plots on Same Figure

In [3]:
```
y2 = np.cos(x)
plt.plot(x, y, label='sin(x)')
plt.plot(x, y2, label='cos(x)')
plt.title('Multiple Lines')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



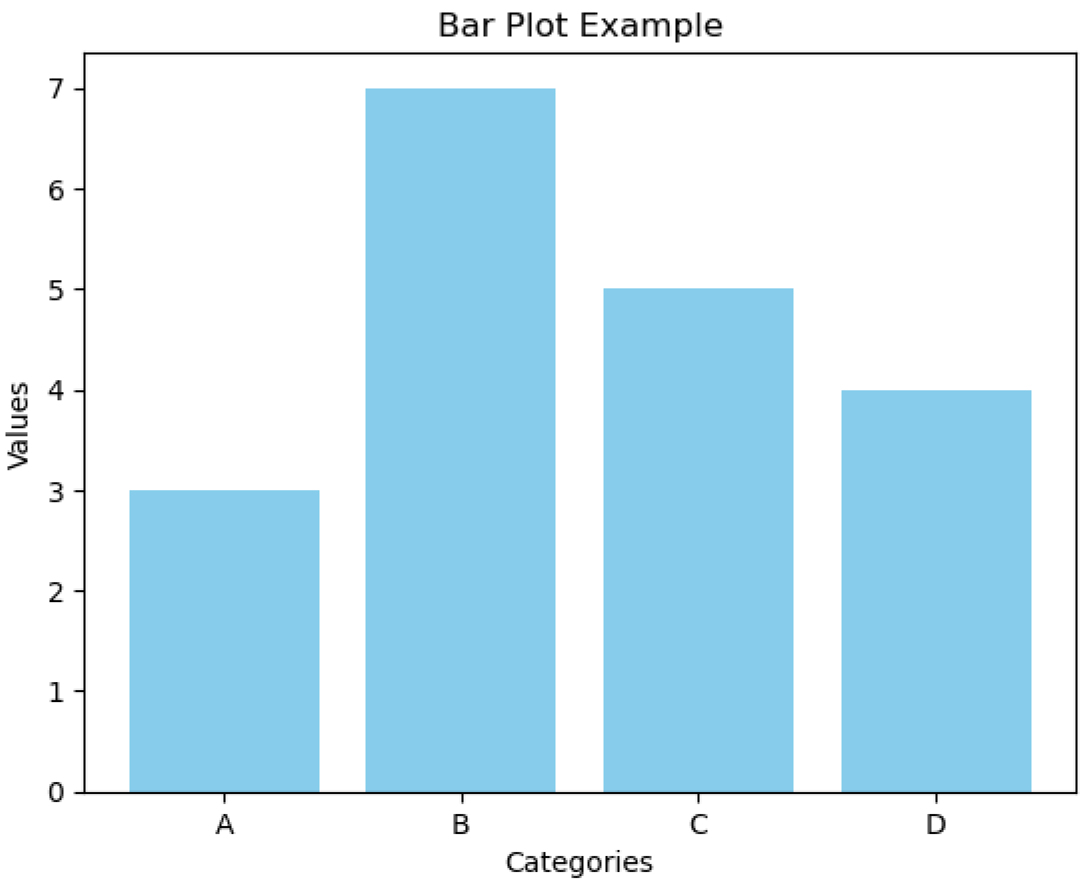
4. Subplots

In [4]:
```
fig, axs = plt.subplots(2)
axs[0].plot(x, y, 'r')
axs[0].set_title('sin(x)')
axs[1].plot(x, y2, 'b')
axs[1].set_title('cos(x)')
plt.tight_layout()
plt.show()
```



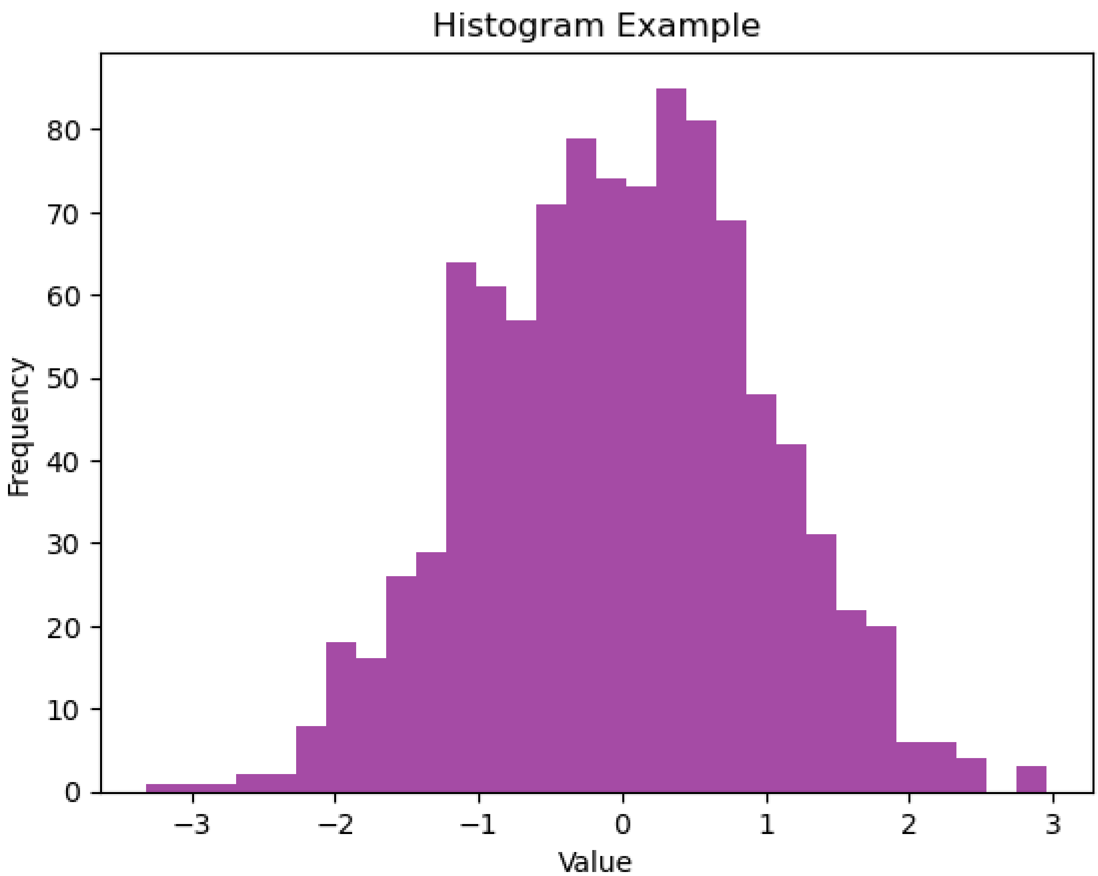
🔹 5. Bar Plot

In [5]:
```
x_vals = ['A', 'B', 'C', 'D']
y_vals = [3, 7, 5, 4]

plt.bar(x_vals, y_vals, color='skyblue')
plt.title('Bar Plot Example')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()
```



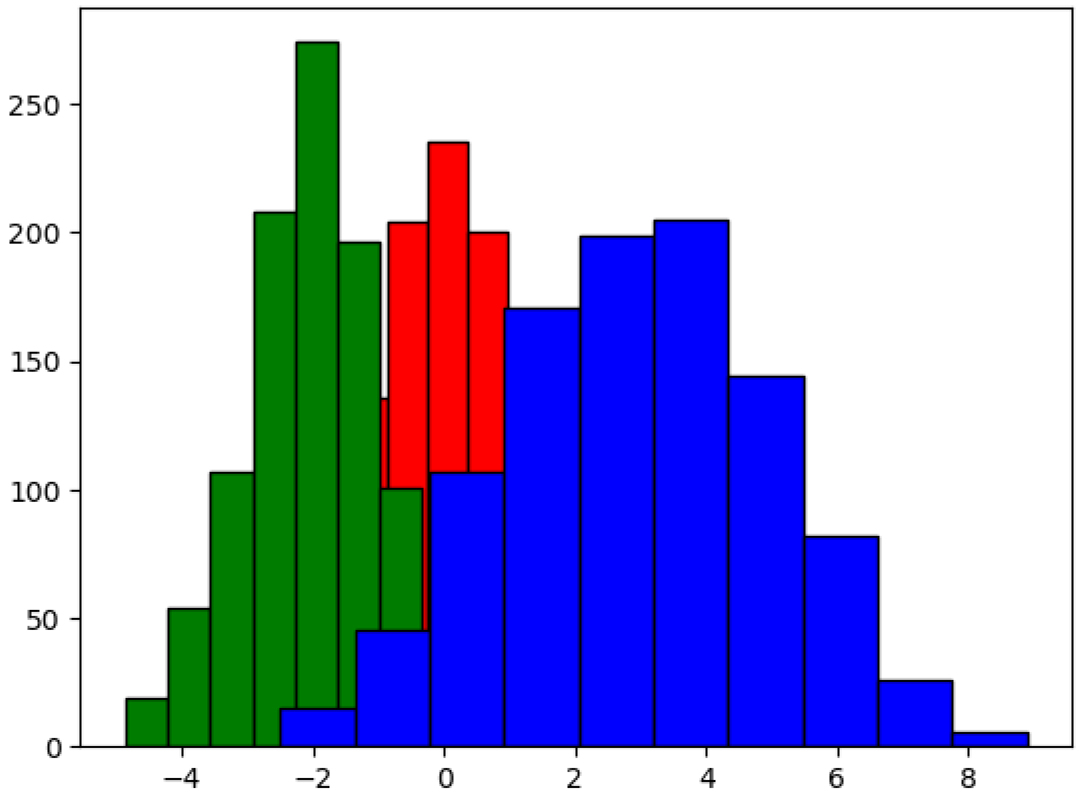
7. Histogram

In [6]:
```
data = np.random.randn(1000)
plt.hist(data, bins=30, color='purple', alpha=0.7)
plt.title('Histogram Example')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



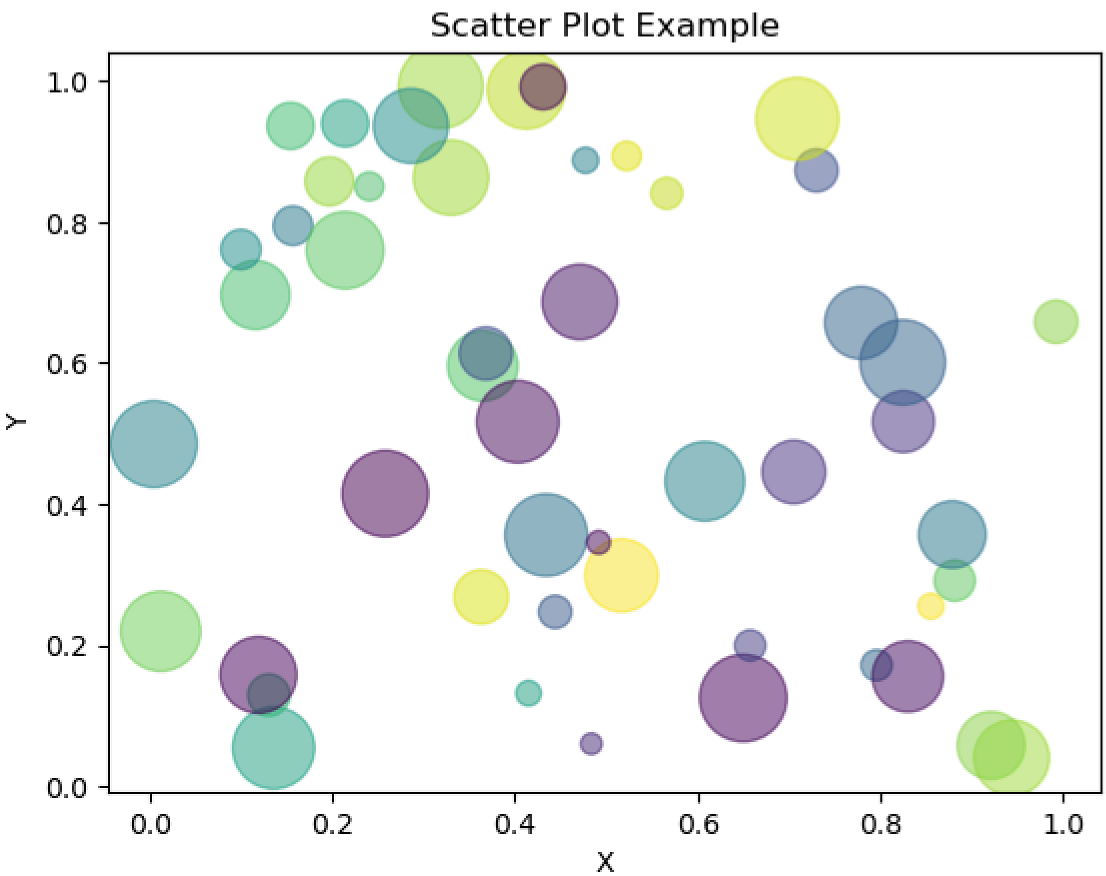
In [11]:

```
# Histogram using Numpy
import numpy as np
import matplotlib.pyplot as plt
x1=np.random.normal(0,1,1000) # normal(mean, std, size)
x2=np.random.normal(-2,1,1000)
x3=np.random.normal(3,2,1000)
plt.hist(x1,bins=10,color='red',edgecolor='black')
plt.hist(x2,bins=10,color='green',edgecolor='black')
plt.hist(x3,bins=10,color='blue',edgecolor='black')
plt.show()
```

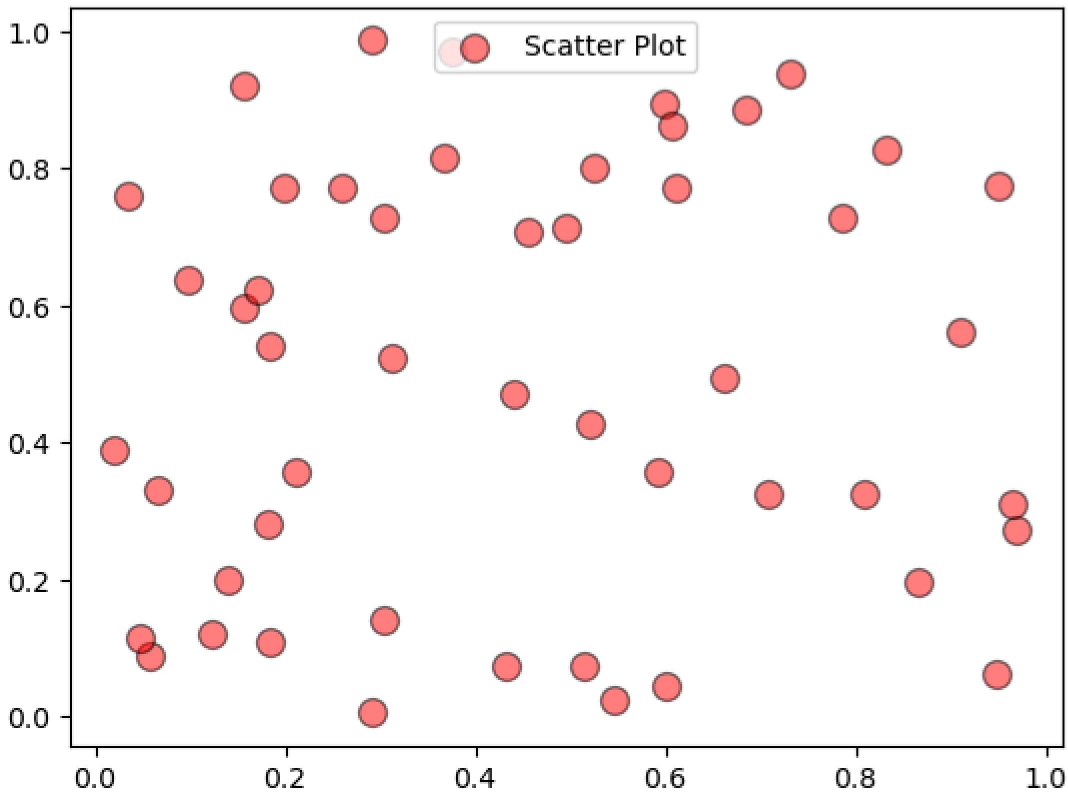


8. Scatter Plot

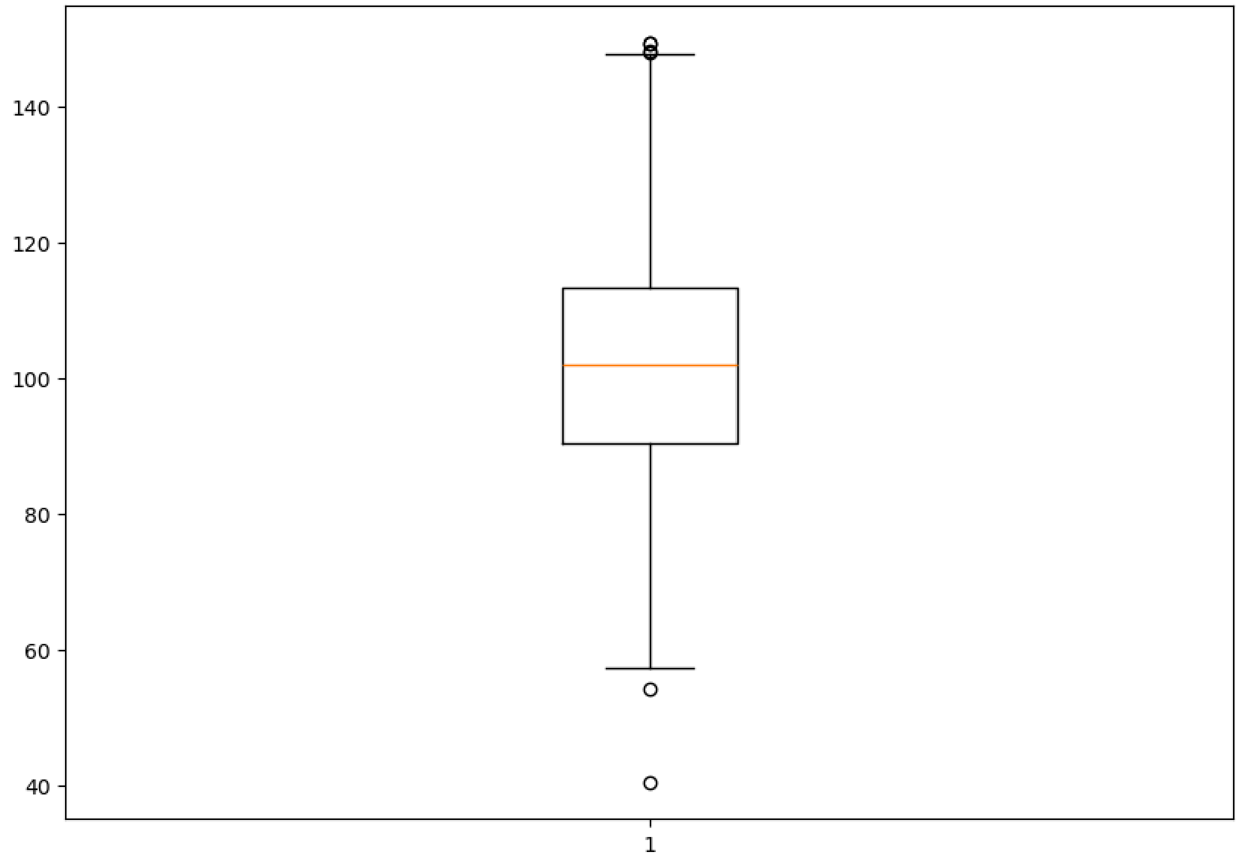
In [6]:
```
x = np.random.rand(50)
y = np.random.rand(50)
colors = np.random.rand(50)
sizes = 1000 * np.random.rand(50)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5)
plt.title('Scatter Plot Example')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



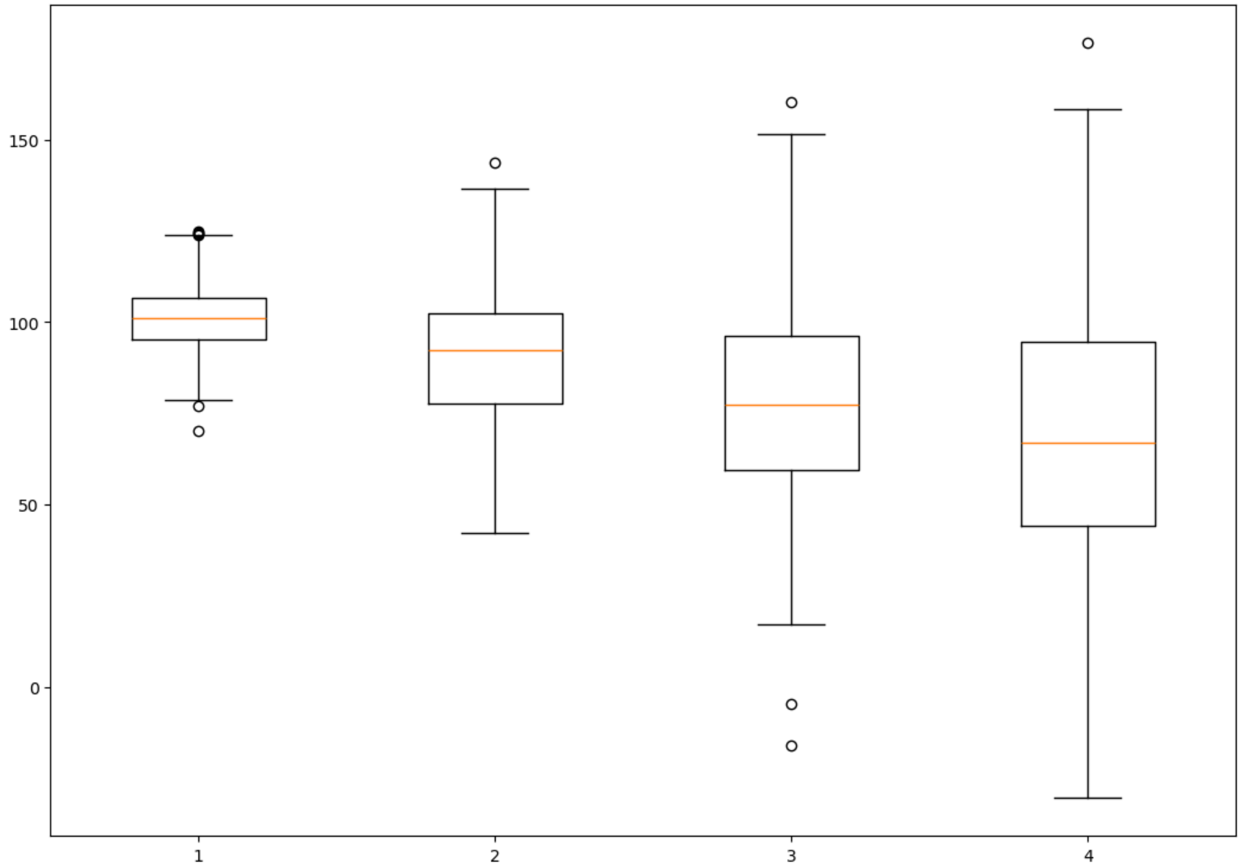
In [8]:
```
np.random.seed(42) # For reproducibility
X = np.random.rand(50) # 50 random numbers between 0 and 1
Y = np.random.rand(50)
plt.scatter(X, Y, color='red', marker='o', s=100, edgecolors='black', alpha=0.
#s-> Sets the marker size.
#edgecolors-> Sets the color of the marker edges.
#alpha-> Adjusts the transparency of the markers.
plt.legend(loc=9)
plt.show()
```



In [9]:
```
np.random.seed(10)
d = np.random.normal(100, 20, 200)
fig = plt.figure(figsize =(10, 7))
plt.boxplot(d)
plt.show()
```

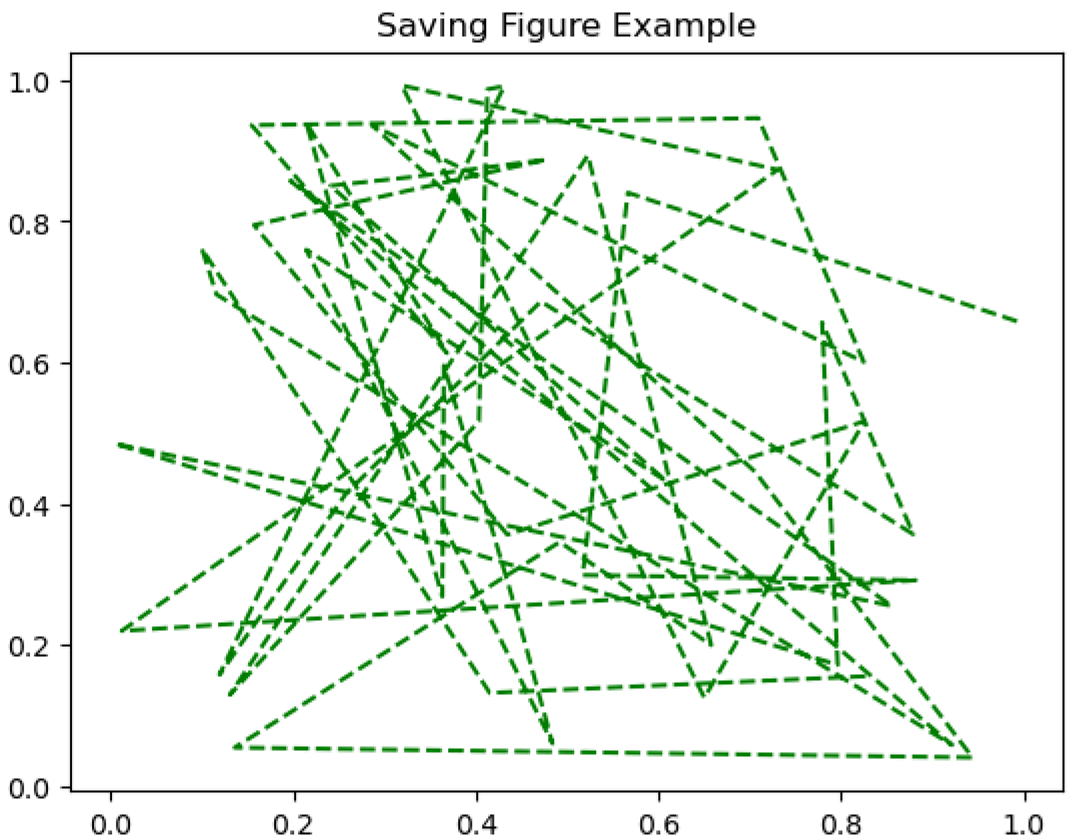


In [4]:
```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(10)
d_1 = np.random.normal(100, 10, 200)
d_2 = np.random.normal(90, 20, 200)
d_3 = np.random.normal(80, 30, 200)
d_4 = np.random.normal(70, 40, 200)
d = [d_1, d_2, d_3, d_4]
fig = plt.figure(figsize =(10, 7))
ax = fig.add_axes([0, 0, 1, 1])
bp = ax.boxplot(d)
plt.show()
```



🔷 9. Saving a Figure

In [7]:
```
plt.plot(x, y, 'g--')
plt.title('Saving Figure Example')
plt.savefig('myplot.png')  # Saves the figure as PNG
plt.show()
```



✅ Summary

- Matplotlib is versatile for data visualization.
- Learned line, bar, pie, scatter, and histogram plots.
- Customization includes color, labels, and subplots.

📘 *Next Steps:* Try using `plt.style.available` for themes and experiment with subplot layouts.

B.3 MODEL TRAINING

B.3.1 Holdout

1. The first step before starting to model, in case of supervised learning, is to load the input data, hold out a portion of the input data as test data and have the remaining portion as training data for building the model.

2. Scikit-learn provides a function to split input data set into training and test data sets. i.e., **train_test_split** function of **sklearn.model_selection**

In [1]:
```
import pandas as pd
data = pd.read_csv("btissue.csv")
data.head()
```

Out[1]:

	I0	PA500	HFS	DA	Area	A/DA	Max IP	DR	
0	524.794072	0.187448	0.032114	228.800228	6843.598481	29.910803	60.204880	220.737212	55(
1	330.000000	0.226893	0.265290	121.154201	3163.239472	26.109202	69.717361	99.084964	40(
2	551.879287	0.232478	0.063530	264.804935	11888.391830	44.894903	77.793297	253.785300	65(
3	380.000000	0.240855	0.286234	137.640111	5402.171180	39.248524	88.758446	105.198568	49:
4	362.831266	0.200713	0.244346	124.912559	3290.462446	26.342127	69.389389	103.866552	42

In [2]:
```
#import the function <train_test_split> to split input data set into training and t
from sklearn.model_selection import train_test_split
```

In [3]:
```
# shape (number of data, number of columns)
data.shape
```

Out[3]:
```
(106, 10)
```

In [4]:
```
# split the dataset into training and test data
data_train, data_test = train_test_split(data, test_size = 0.3, random_state = 123)

# test_size = the ratio of test data to the input data to 0.3 or 30%
# random_state sets the seed for random number generator
```

In [5]:
```
# len(): number of data
len(data_train)
```

Out[5]:
```
74
```

In [6]:
```
len(data_test)
```

Out[6]:
```
32
```

NOTE

1. When we do data holdout, i.e. splitting of the input data into training and test data sets, the records selected for each set are picked randomly.

2. So, it is obvious that executing the same code may result in different training data set. So the model trained will also be somewhat different.

3. In Python Scikit function **train_test_split**, the parameter **random_state** sets the starting point of the random number generator used internally to pick the records.

4. This ensures that random numbers of a specific sequence is used every time and hence the same records (i.e. records having same sequence number) are picked every time and the model is trained in the same way.

5. This is extremely critical for the reproducibility of results i.e. every time, the same machine learning program generates the same set of results.

B.3.2 K-fold cross-validation

1. Let's do k-fold cross-validation with 10 folds.

2. For creating the cross-validation, **KFold** function of **sklearn.model_ selection** can be used.

In [7]:
```
data = pd.read_csv("auto-mpg.csv")
len(data)
```

Out[7]:
```
398
```

In [8]:
```
from sklearn.model_selection import KFold
```

In [9]:
```
#kf = KFold(n_splits=10)
kf = KFold(n_splits=10, shuffle=True, random_state=123)
```

Note:

1. By default, KFold does not shuffle before splitting.

2. That means if your data is ordered (e.g., all class 0 first, then all class 1), you'll get biased folds.

3. To avoid that, add shuffle=True and a random_state for reproducibility:

In [10]:
```
for fold, (train_index, test_index) in enumerate(kf.split(data), 1):
    print(f"Fold {fold}")
    print(" Train indices:", train_index)
    print(" Test indices :", test_index, "\n")
```

```
Fold 1
 Train indices: [  0   1   2   3   4   5   6   7   8  10  12  13  14  16  17  18
19  20
  21  22  23  24  25  26  27  28  29  30  31  32  34  35  36  37  38  39
  40  43  44  45  46  47  49  50  51  52  53  55  56  57  58  59  60  61
  62  63  64  65  66  67  68  69  70  71  72  73  74  76  77  78  80  81
  82  83  84  85  86  87  88  89  90  92  93  94  95  96  97  98  99 100
 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 121
 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139
 140 141 143 144 145 146 147 148 149 151 152 153 154 155 156 157 158 159
 160 161 162 163 164 165 166 167 168 169 171 173 174 176 177 178 179 180
 181 182 183 184 185 186 187 188 191 192 193 194 195 197 198 200 201 202
 203 204 205 206 207 208 209 211 212 213 214 215 216 217 218 219 220 221
 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239
 240 241 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
 259 260 261 262 263 265 266 267 268 269 270 271 272 273 274 275 276 278
 279 280 281 283 284 285 286 287 288 289 290 291 294 296 297 298 300 301
 302 303 304 305 306 307 308 310 311 312 313 314 315 316 317 318 319 320
 321 322 323 324 325 326 327 328 329 330 331 332 333 334 336 337 338 339
 340 341 342 343 344 346 348 349 350 351 352 353 354 355 356 357 358 360
 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 379
 380 381 382 383 384 385 388 389 390 391 392 393 394 395 396 397]
 Test indices : [  9  11  15  33  41  42  48  54  75  79  91 101 102 120 142 150 1
70 172
 175 189 190 196 199 210 242 264 277 282 292 293 295 299 309 335 345 347
 359 378 386 387]

Fold 2
 Train indices: [  0   1   2   3   4   5   6   7   8   9  10  11  12  14  15  16
17  18
  19  21  22  23  25  27  28  29  30  32  33  34  35  37  38  39  40  41
  42  43  44  45  46  47  48  49  50  51  53  54  55  56  57  58  60  61
  62  63  64  65  66  67  68  69  70  71  73  74  75  76  77  78  79  80
  81  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
 100 101 102 103 104 105 106 107 108 109 110 111 112 113 115 116 117 118
 119 120 122 123 124 125 126 127 128 129 130 131 132 133 135 136 137 138
 139 140 141 142 143 144 145 146 149 150 151 152 153 154 156 157 158 160
 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 178 179
 180 181 182 183 185 186 187 188 189 190 191 193 194 195 196 197 198 199
 200 202 203 204 205 206 207 208 209 210 212 213 214 215 216 219 220 221
 222 223 224 225 226 227 228 230 231 232 233 234 236 237 238 239 240 241
 242 243 244 247 248 249 250 251 253 254 255 256 257 258 259 260 261 262
 264 265 267 268 269 270 271 272 273 274 275 277 278 279 280 281 282 283
 284 285 286 287 288 289 290 291 292 293 294 295 297 298 299 300 302 303
 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321
 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339
 340 341 342 343 345 346 347 348 349 350 351 352 353 354 355 357 358 359
 360 361 362 363 364 365 367 368 369 370 371 372 373 374 375 377 378 379
 380 381 382 383 384 385 386 387 389 390 391 392 394 395 396 397]
 Test indices : [ 13  20  24  26  31  36  52  59  72  82 114 121 134 147 148 155 1
59 177
 184 192 201 211 217 218 229 235 245 246 252 263 266 276 296 301 344 356
 366 376 388 393]

Fold 3
 Train indices: [  1   2   3   4   5   7   8   9  10  11  13  14  15  16  17  18
19  20
  22  23  24  25  26  27  28  29  31  32  33  34  35  36  37  39  40  41
  42  43  44  45  46  47  48  49  50  51  52  53  54  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
  79  80  81  82  83  84  85  86  87  88  89  90  91  92  94  96  97  98
  99 100 101 102 103 104 105 106 108 109 110 111 112 113 114 115 116 117
 118 119 120 121 122 123 124 126 127 128 129 130 131 132 133 134 135 136
 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154
```

```
 155 156 158 159 161 163 164 166 167 168 169 170 172 174 175 176 177 178
 179 180 182 183 184 185 186 187 189 190 191 192 193 194 195 196 197 198
 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
 217 218 219 221 222 224 225 226 227 229 230 231 233 234 235 236 238 239
 240 242 243 244 245 246 247 249 250 251 252 253 254 255 256 257 258 259
 260 261 262 263 264 265 266 267 268 269 270 271 272 275 276 277 278 279
 280 281 282 284 285 288 289 290 291 292 293 294 295 296 297 299 300 301
 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319
 320 321 322 323 324 325 326 328 329 330 331 332 333 334 335 337 339 340
 341 342 343 344 345 346 347 348 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 365 366 367 368 370 371 372 373 375 376 378 379 380 381
 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397]
 Test indices : [  0   6  12  21  30  38  55  93  95 107 125 157 160 162 165 171 1
73 181
 188 220 223 228 232 237 241 248 273 274 283 286 287 298 327 336 338 349
 364 369 374 377]

Fold 4
 Train indices: [  0   1   2   3   6   7   8   9  10  11  12  13  14  15  16  17
18  20
  21  23  24  25  26  27  28  30  31  32  33  34  36  37  38  39  40  41
  42  43  44  45  46  47  48  49  50  51  52  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  72  73  75  76  77  79  81  82
  83  84  85  86  87  88  89  91  92  93  95  96  97  98  99 100 101 102
 103 104 106 107 108 109 110 111 112 113 114 115 116 118 119 120 121 122
 123 124 125 126 127 129 130 131 132 133 134 135 136 137 138 139 140 141
 142 145 146 147 148 149 150 151 153 154 155 157 158 159 160 161 162 163
 164 165 166 167 168 169 170 171 172 173 174 175 176 177 180 181 182 183
 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238
 239 240 241 242 243 244 245 246 247 248 250 251 252 253 254 255 256 257
 258 259 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276
 277 278 279 280 281 282 283 285 286 287 288 289 290 291 292 293 294 295
 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313
 314 315 316 317 318 319 321 322 324 325 326 327 328 329 330 331 333 334
 335 336 337 338 339 340 341 342 343 344 345 347 348 349 350 351 352 353
 356 357 358 359 360 361 362 363 364 365 366 369 371 372 373 374 375 376
 377 378 379 381 382 385 386 387 388 389 390 391 392 393 395 396]
 Test indices : [  4   5  19  22  29  35  53  71  74  78  80  90  94 105 117 128 1
43 144
 152 156 178 179 202 249 260 284 320 323 332 346 354 355 367 368 370 380
 383 384 394 397]

Fold 5
 Train indices: [  0   1   2   3   4   5   6   8   9  10  11  12  13  14  15  16
17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  35  36  38
  39  40  41  42  43  45  46  47  48  50  51  52  53  54  55  56  57  58
  59  60  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77
  78  79  80  82  83  84  85  86  87  88  90  91  92  93  94  95  96  97
  98  99 100 101 102 103 105 106 107 109 110 111 112 113 114 116 117 118
 119 120 121 122 123 124 125 126 128 129 130 131 132 133 134 135 137 138
 139 140 141 142 143 144 146 147 148 149 150 151 152 153 154 155 156 157
 158 159 160 161 162 163 165 168 169 170 171 172 173 174 175 176 177 178
 179 180 181 182 183 184 185 186 187 188 189 190 192 193 194 195 196 197
 198 199 201 202 203 205 206 208 209 210 211 212 213 214 215 216 217 218
 219 220 221 222 223 224 225 227 228 229 230 232 233 235 237 238 239 241
 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 259 260
 261 262 263 264 265 266 267 268 269 270 271 272 273 274 276 277 278 279
 281 282 283 284 286 287 288 289 290 291 292 293 294 295 296 298 299 300
 301 302 303 304 305 307 308 309 310 311 312 313 314 316 318 319 320 321
 322 323 324 325 326 327 328 330 331 332 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 352 353 354 355 356 357 358 359 360
```

```
 361 362 363 364 365 366 367 368 369 370 371 373 374 375 376 377 378 380
 381 382 383 384 385 386 387 388 389 390 392 393 394 395 396 397]
 Test indices : [  7  34  37  44  49  61  81  89 104 108 115 127 136 145 164 166 1
67 191
 200 204 207 226 231 234 236 240 258 275 280 285 297 306 315 317 329 333
 351 372 379 391]

Fold 6
 Train indices: [  0   1   2   3   4   5   6   7   8   9  11  12  13  14  15  16
17  18
  19  20  21  22  24  25  26  27  29  30  31  32  33  34  35  36  37  38
  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56
  57  58  59  60  61  62  63  64  65  67  68  70  71  72  73  74  75  76
  77  78  79  80  81  82  83  84  86  87  88  89  90  91  92  93  94  95
  96  97  98  99 101 102 103 104 105 106 107 108 109 111 113 114 115 116
 117 118 119 120 121 123 125 126 127 128 129 130 133 134 135 136 139 140
 141 142 143 144 145 146 147 148 149 150 152 153 154 155 156 157 158 159
 160 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178
 179 180 181 182 183 184 185 186 188 189 190 191 192 193 195 196 197 198
 199 200 201 202 204 205 206 207 208 209 210 211 213 214 215 217 218 219
 220 222 223 224 225 226 228 229 230 231 232 233 234 235 236 237 238 240
 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
 259 260 262 263 264 265 266 268 269 270 271 272 273 274 275 276 277 278
 280 281 282 283 284 285 286 287 288 289 290 292 293 294 295 296 297 298
 299 301 302 304 305 306 309 310 311 312 313 314 315 316 317 318 320 322
 323 324 325 326 327 329 331 332 333 334 335 336 337 338 339 340 342 343
 344 345 346 347 348 349 350 351 352 353 354 355 356 358 359 360 361 362
 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397]
 Test indices : [ 10  23  28  66  69  85 100 110 112 122 124 131 132 137 138 151 1
61 187
 194 203 212 216 221 227 239 261 267 279 291 300 303 307 308 319 321 328
 330 341 357 363]

Fold 7
 Train indices: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  41  42  43  44  46  47  48  49  51  52  53  54  55  56
  57  58  59  60  61  64  65  66  67  68  69  70  71  72  73  74  75  76
  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  93  94  95
  96  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114
 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 131 132 133
 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151
 152 154 155 156 157 158 159 160 161 162 164 165 166 167 168 170 171 172
 173 175 176 177 178 179 180 181 183 184 186 187 188 189 190 191 192 193
 194 196 198 199 200 201 202 203 204 205 206 207 208 210 211 212 213 214
 216 217 218 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
 235 236 237 239 240 241 242 243 244 245 246 248 249 250 252 253 254 255
 256 258 259 260 261 262 263 264 265 266 267 268 270 271 273 274 275 276
 277 278 279 280 282 283 284 285 286 287 290 291 292 293 294 295 296 297
 298 299 300 301 303 304 305 306 307 308 309 311 312 314 315 317 318 319
 320 321 322 323 325 327 328 329 330 331 332 333 334 335 336 338 339 340
 341 342 344 345 346 347 348 349 350 351 353 354 355 356 357 358 359 360
 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378
 379 380 382 383 384 385 386 387 388 390 391 393 394 395 396 397]
 Test indices : [ 40  45  50  62  63  92  97 130 153 163 169 174 182 185 195 197 2
09 215
 219 238 247 251 257 269 272 281 288 289 302 310 313 316 324 326 337 343
 352 381 389 392]

Fold 8
 Train indices: [  0   1   2   4   5   6   7   9  10  11  12  13  15  17  18  19
20  21
```

```
  22  23  24  25  26  28  29  30  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  47  48  49  50  51  52  53  54  55  57  59  61  62
  63  64  66  68  69  70  71  72  73  74  75  76  78  79  80  81  82  83
  84  85  89  90  91  92  93  94  95  96  97  98  99 100 101 102 104 105
 106 107 108 110 111 112 113 114 115 117 118 120 121 122 123 124 125 126
 127 128 129 130 131 132 134 135 136 137 138 142 143 144 145 146 147 148
 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
 186 187 188 189 190 191 192 194 195 196 197 199 200 201 202 203 204 205
 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 223 224 225
 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 244
 245 246 247 248 249 251 252 253 255 257 258 260 261 263 264 266 267 268
 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286
 287 288 289 290 291 292 293 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 319 320 321 322 323 324
 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 343
 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 361 363
 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397]
 Test indices : [  3   8  14  16  27  46  56  58  60  65  67  77  86  87  88 103 1
09 116
 119 133 139 140 141 149 193 198 206 222 243 250 254 256 259 262 265 294
 318 342 360 362]

Fold 9
 Train indices: [  0   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
17  19
  20  21  22  23  24  26  27  28  29  30  31  32  33  34  35  36  37  38
  39  40  41  42  44  45  46  47  48  49  50  52  53  54  55  56  57  58
  59  60  61  62  63  65  66  67  68  69  71  72  73  74  75  77  78  79
  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97
  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115
 116 117 119 120 121 122 123 124 125 126 127 128 130 131 132 133 134 135
 136 137 138 139 140 141 142 143 144 145 147 148 149 150 151 152 153 155
 156 157 159 160 161 162 163 164 165 166 167 169 170 171 172 173 174 175
 176 177 178 179 181 182 184 185 187 188 189 190 191 192 193 194 195 196
 197 198 199 200 201 202 203 204 206 207 208 209 210 211 212 214 215 216
 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 234 235
 236 237 238 239 240 241 242 243 245 246 247 248 249 250 251 252 253 254
 256 257 258 259 260 261 262 263 264 265 266 267 269 272 273 274 275 276
 277 279 280 281 282 283 284 285 286 287 288 289 291 292 293 294 295 296
 297 298 299 300 301 302 303 305 306 307 308 309 310 311 313 315 316 317
 318 319 320 321 322 323 324 326 327 328 329 330 332 333 334 335 336 337
 338 339 340 341 342 343 344 345 346 347 348 349 351 352 354 355 356 357
 359 360 361 362 363 364 365 366 367 368 369 370 372 374 376 377 378 379
 380 381 382 383 384 385 386 387 388 389 391 392 393 394 395 396 397]
 Test indices : [  1  18  25  43  51  64  70  76 118 129 146 154 158 168 180 183 1
86 205
 213 233 244 255 268 270 271 278 290 304 312 314 325 331 350 353 358 371
 373 375 390]

Fold 10
 Train indices: [  0   1   3   4   5   6   7   8   9  10  11  12  13  14  15  16
18  19
  20  21  22  23  24  25  26  27  28  29  30  31  33  34  35  36  37  38
  40  41  42  43  44  45  46  48  49  50  51  52  53  54  55  56  58  59
  60  61  62  63  64  65  66  67  69  70  71  72  74  75  76  77  78  79
  80  81  82  85  86  87  88  89  90  91  92  93  94  95  97 100 101 102
 103 104 105 107 108 109 110 112 114 115 116 117 118 119 120 121 122 124
 125 127 128 129 130 131 132 133 134 136 137 138 139 140 141 142 143 144
 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
 163 164 165 166 167 168 169 170 171 172 173 174 175 177 178 179 180 181
 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
 200 201 202 203 204 205 206 207 209 210 211 212 213 215 216 217 218 219
```

```
 220 221 222 223 226 227 228 229 231 232 233 234 235 236 237 238 239 240
 241 242 243 244 245 246 247 248 249 250 251 252 254 255 256 257 258 259
 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277
 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295
 296 297 298 299 300 301 302 303 304 306 307 308 309 310 312 313 314 315
 316 317 318 319 320 321 323 324 325 326 327 328 329 330 331 332 333 335
 336 337 338 341 342 343 344 345 346 347 349 350 351 352 353 354 355 356
 357 358 359 360 362 363 364 366 367 368 369 370 371 372 373 374 375 376
 377 378 379 380 381 383 384 386 387 388 389 390 391 392 393 394 397]
 Test indices : [   2  17  32  39  47  57  68  73  83  84  96  98  99 106 111 113 1
23 126
 135 176 208 214 224 225 230 253 305 311 322 334 339 340 348 361 365 382
 385 395 396]
```

kf.split(data): is the modern scikit-learn way of generating train/test indices using K-Fold cross-validation.

1. kf.split(data) does not split the data directly.
2. It yields index arrays (train_index, test_index) that you can use to slice your data (or predictors/targets).

enumerate(kf.split(data), 1): adds a fold counter starting at 1 instead of 0.

In [11]:
```python
# folds = []
# for fold, (train_index, test_index) in enumerate(kf.split(data), 1):
#   folds.append((train_index, test_index))
# # Access the 3rd fold
# train_index, test_index = folds[2]  # fold 3 (Python uses 0-based index)
# data_train = data.iloc[train_index]
# data_test = data.iloc[test_index]
```

In [12]:
```python
folds = []
for train_index, test_index in kf.split(data):
  folds.append((train_index, test_index))
# Access the 3rd fold
train_index, test_index = folds[3]  # fold 3 (Python uses 0-based index)
data_train = data.iloc[train_index]
data_test = data.iloc[test_index]
```

In [13]:
```python
print('Train Index:','\n', train_index)
print('Test Index:','\n', test_index)
```

```
Train Index:
 [  0   1   2   3   6   7   8   9  10  11  12  13  14  15  16  17  18  20
  21  23  24  25  26  27  28  30  31  32  33  34  36  37  38  39  40  41
  42  43  44  45  46  47  48  49  50  51  52  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  72  73  75  76  77  79  81  82
  83  84  85  86  87  88  89  91  92  93  95  96  97  98  99 100 101 102
 103 104 106 107 108 109 110 111 112 113 114 115 116 118 119 120 121 122
 123 124 125 126 127 129 130 131 132 133 134 135 136 137 138 139 140 141
 142 145 146 147 148 149 150 151 153 154 155 157 158 159 160 161 162 163
 164 165 166 167 168 169 170 171 172 173 174 175 176 177 180 181 182 183
 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201
 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220
 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238
 239 240 241 242 243 244 245 246 247 248 250 251 252 253 254 255 256 257
 258 259 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276
 277 278 279 280 281 282 283 285 286 287 288 289 290 291 292 293 294 295
 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313
 314 315 316 317 318 319 321 322 324 325 326 327 328 329 330 331 333 334
 335 336 337 338 339 340 341 342 343 344 345 347 348 349 350 351 352 353
 356 357 358 359 360 361 362 363 364 365 366 369 371 372 373 374 375 376
 377 378 379 381 382 385 386 387 388 389 390 391 392 393 395 396]
Test Index:
 [  4   5  19  22  29  35  53  71  74  78  80  90  94 105 117 128 143 144
 152 156 178 179 202 249 260 284 320 323 332 346 354 355 367 368 370 380
 383 384 394 397]
```

In [14]:

```
print(len(data_train))
data_train.head()
```

```
358
```

Out[14]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
6	14.0	8	454.0	220	4354	9.0	70	1	chevrolet impala

In [15]:

```
print(len(data_test))
data_test.head()
```

```
40
```

Out[15]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
5	15.0	8	429.0	198	4341	10.0	70	1	ford galaxie 500
19	26.0	4	97.0	46	1835	20.5	70	2	volkswagen 1131 deluxe sedan
22	25.0	4	104.0	95	2375	17.5	70	2	saab 99e
29	27.0	4	97.0	88	2130	14.5	71	3	datsun pl510

In [16]:
```
data_train.tail(5)
```

Out[16]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
391	36.0	4	135.0	84	2370	13.0	82	1	dodge charger 2.2
392	27.0	4	151.0	90	2950	17.3	82	1	chevrolet camaro
393	27.0	4	140.0	86	2790	15.6	82	1	ford mustang gl
395	32.0	4	135.0	84	2295	11.6	82	1	dodge rampage
396	28.0	4	120.0	79	2625	18.6	82	1	ford ranger

In [17]:
```
data_test.tail(5)
```

Out[17]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
380	36.0	4	120.0	88	2160	14.5	82	3	nissan stanza xe
383	38.0	4	91.0	67	1965	15.0	82	3	honda civic
384	32.0	4	91.0	67	1965	15.7	82	3	honda civic (auto)
394	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
397	31.0	4	119.0	82	2720	19.4	82	1	chevy s-10

B.3.3 Bootstrap sampling

Bootstrap resampling is used to generate samples of any given size from the training data. It uses the concept of simple random sampling with repetition.

To generate bootstrap sample in Python, **resample** function of **sklearn.utils** library can be used.

In [18]:
```
from sklearn.utils import resample
```

In [19]:
```
import pandas as pd
data = pd.read_csv("btissue.csv")
```

In [20]:
```
import numpy as np
np.shape(data)
```

Out[20]: (106, 10)

In [21]:
```
X = data.iloc[:,0:9]
X.head()
```

Out[21]:

	I0	PA500	HFS	DA	Area	A/DA	Max IP	DR	
0	524.794072	0.187448	0.032114	228.800228	6843.598481	29.910803	60.204880	220.737212	556
1	330.000000	0.226893	0.265290	121.154201	3163.239472	26.109202	69.717361	99.084964	400
2	551.879287	0.232478	0.063530	264.804935	11888.391830	44.894903	77.793297	253.785300	656
3	380.000000	0.240855	0.286234	137.640111	5402.171180	39.248524	88.758446	105.198568	49
4	362.831266	0.200713	0.244346	124.912559	3290.462446	26.342127	69.389389	103.866552	42

In [22]:
```
X1=resample(X, n_samples=200, random_state=0)
X1.head(6)
# Generates a sample of size 200 (as mentioned in parameter n_samples)
#with repetition
```

Out[22]:

	I0	PA500	HFS	DA	Area	A/DA	Max IP	DI
44	172.515797	0.127235	0.038397	37.543673	192.218148	5.119855	19.322081	32.18982
47	370.395725	0.104720	0.000000	115.923253	1308.120430	11.284366	31.367031	112.71510
64	391.000000	0.058119	0.011170	35.780061	265.149790	7.410546	22.131472	28.11424
67	145.000000	0.117635	0.110305	21.218942	82.455562	3.885941	20.303082	6.16671
67	145.000000	0.117635	0.110305	21.218942	82.455562	3.885941	20.303082	6.16671
103	1600.000000	0.071908	-0.066323	436.943603	12655.342130	28.963331	103.732704	432.12974

B.3.4 Training the model

Once the model preparatory steps, like data holdout, etc. are over, the actual training happens.

The sklearn framework in Python provides most of the models which are generally used in machine learning.

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
# Decision Tree is one of the Classification Model
```

In [24]:
```
predictors = data.iloc[:,0:9] #Segregating the predictors
predictors.head()
```

Out[24]:

	I0	PA500	HFS	DA	Area	A/DA	Max IP	DR	
0	524.794072	0.187448	0.032114	228.800228	6843.598481	29.910803	60.204880	220.737212	55(
1	330.000000	0.226893	0.265290	121.154201	3163.239472	26.109202	69.717361	99.084964	40(
2	551.879287	0.232478	0.063530	264.804935	11888.391830	44.894903	77.793297	253.785300	65(
3	380.000000	0.240855	0.286234	137.640111	5402.171180	39.248524	88.758446	105.198568	49:
4	362.831266	0.200713	0.244346	124.912559	3290.462446	26.342127	69.389389	103.866552	42‹

In [25]:
```
target = data.iloc[:,9]  #Segregating the target/class
target.head()
```

Out[25]:
```
0    car
1    car
2    car
3    car
4    car
Name: class, dtype: object
```

In [26]:
```
from sklearn.model_selection import train_test_split
predictors_train, predictors_test, target_train, target_test = train_test_split(pre
#Holdout of data
```

In [27]:
```
predictors_train.head()
# predictors_test.head()
# target_train.head()
# target_test.head()
```

Out[27]:

	I0	PA500	HFS	DA	Area	A/DA	Max IP	DR
21	211.000000	0.053931	0.094248	30.753443	151.984578	4.942034	14.268374	27.243124
77	691.972031	0.026005	0.086568	190.676692	304.270718	1.595742	23.975718	189.163331
35	144.000000	0.120602	0.046077	19.647670	70.426239	3.584458	18.131014	7.569493
71	1385.664721	0.092328	0.089361	202.480044	8785.028733	43.387134	143.092194	143.257780
65	502.000000	0.065275	0.027925	53.239433	834.272730	15.670203	33.331142	41.514722

In [28]:
```
dtree_entropy =DecisionTreeClassifier(criterion = "entropy", random_state = 100, ma
```

In [29]:
```
# Finally the model is trained
model = dtree_entropy.fit(predictors_train,target_train)
```

B.3.5 Evaluating model performance

B.3.5.1 Supervised learning - classification

The primary measure of performance of a classification model is its accuracy.

Accuracy of a model is calculated based on correct classifications made by the model compared to total number of classifications.

In Python, **sklearn.metrics** provides **accuracy_score** functionality to evaluate the accuracy of a classification model.

Below is the code for evaluating the accuracy of a classifier model.

In [30]:
```
prediction=model.predict(predictors_test)
```

In [31]:
```
from sklearn.metrics import accuracy_score
```

In [32]:
```
accuracy_score(target_test, prediction, normalize = True)
```

Out[32]:
```
0.625
```

The **confusion_matrix** functionality of **sklearn.metrics** helps in generation the confusion matrix.

Below is the code for finding the confusion matrix of a classifier model.

In [33]:
```
from sklearn.metrics import confusion_matrix
```

In [34]:
```
target_test
#len(target_test)
```

Out[34]:
```
53     mas
28     fad
63     gla
99     adi
93     adi
90     adi
8      car
5      car
0      car
62     gla
85     adi
4      car
31     fad
87     adi
13     car
38     mas
72     con
41     mas
42     mas
24     fad
23     fad
50     mas
29     fad
54     gla
19     car
59     gla
33     fad
103    adi
82     con
9      car
79     con
84     adi
Name: class, dtype: object
```

In [35]:
```
confusion_matrix(target_test,prediction)
```

Out[35]:
```
array([[6, 0, 1, 0, 0, 0],
       [0, 7, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0],
       [0, 2, 0, 0, 0, 4],
       [0, 1, 0, 0, 1, 2],
       [0, 2, 0, 0, 0, 3]], dtype=int64)
```

B.3.5.2 Supervised learning - regression

R-squared is an effective measure of performance of a regression model.

In Python, **sklearn.metrics** provides **mean_squared_error** and **r2_score** functionality to evaluate the accuracy of a regression model.

In [36]:
```
data = pd.read_csv('auto_mpg.csv')
data
```

Out[36]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_nar
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevro cheve mali
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	bu skyla 3
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymou satell
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rel
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	fc tori
...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86.0	2790.0	15.6	82	1	fc musta
394	44.0	4	97.0	52.0	2130.0	24.6	82	2	vw pick
395	32.0	4	135.0	84.0	2295.0	11.6	82	1	dod rampa
396	28.0	4	120.0	79.0	2625.0	18.6	82	1	fc rang
397	31.0	4	119.0	82.0	2720.0	19.4	82	1	chevy

398 rows × 9 columns

In [37]:
```
data = data.dropna(axis=0, how='any')
#Remove all rows where value of any columnis ‘NaN’
data
```

Out[37]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_nar
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevro cheve mali
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	bu skyl: 3
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymou satell
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rel
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	fc tori
...	...	...	...	...	...	...	...	...	
393	27.0	4	140.0	86.0	2790.0	15.6	82	1	fc musta
394	44.0	4	97.0	52.0	2130.0	24.6	82	2	vw pick
395	32.0	4	135.0	84.0	2295.0	11.6	82	1	dod rampa
396	28.0	4	120.0	79.0	2625.0	18.6	82	1	fc rang
397	31.0	4	119.0	82.0	2720.0	19.4	82	1	chevy

392 rows × 9 columns

In [38]:

```
predictors = data.iloc[:,1:8]
#Seggretating the predictor variables
predictors
```

Out[38]:

	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	8	307.0	130.0	3504.0	12.0	70	1
1	8	350.0	165.0	3693.0	11.5	70	1
2	8	318.0	150.0	3436.0	11.0	70	1
3	8	304.0	150.0	3433.0	12.0	70	1
4	8	302.0	140.0	3449.0	10.5	70	1
...	...	...	...	...	...	...	...
393	4	140.0	86.0	2790.0	15.6	82	1
394	4	97.0	52.0	2130.0	24.6	82	2
395	4	135.0	84.0	2295.0	11.6	82	1
396	4	120.0	79.0	2625.0	18.6	82	1
397	4	119.0	82.0	2720.0	19.4	82	1

392 rows × 7 columns

In [39]:
```
target = data.iloc[:,0]
#Seggretating the target / class variable
target
```

Out[39]:
```
0      18.0
1      15.0
2      18.0
3      16.0
4      17.0
       ...
393    27.0
394    44.0
395    32.0
396    28.0
397    31.0
Name: mpg, Length: 392, dtype: float64
```

In [40]:
```
predictors_train, predictors_test,target_train, target_test = train_test_split(prec
```

In [41]:
```
from sklearn.linear_model import LinearRegression
```

In [42]:
```
model = LinearRegression()
```

In [43]:
```
# First train model / classifier with the input dataset (training data part of it)
model = model.fit(predictors_train, target_train)
```

In [44]:
```
# Make prediction using the trained model
prediction = model.predict(predictors_test)
```

In [45]:
```
from sklearn.metrics import mean_squared_error, r2_score
```

In [46]:
```
mean_squared_error(target_test, prediction)
```

Out[46]:
```
12.166313433330693
```

In [47]:
```
r2_score(target_test, prediction)
```

Out[47]:
```
0.7867839344447645
```

Unsupervised Learning:

There are two popular measures of cluster quality – **purity** and **silhouette width**.

Purity can be calculated only when class label is known for the data set subjected to clustering.

On the other hand, silhouette width can be calculated for any data set.

Purity

We will use a Lower Back Pain Symptoms data set released by Kaggle (https://www.kaggle.com/sammy123/lower-back-painsymptoms-dataset).

The data set consists of 310 observations, 13 attributes (12 numeric predictors, 1 binary class attribute).

In [48]:
```
import numpy as np
import pandas as pd
```

In [49]:
```
from sklearn.cluster import KMeans
```

In [50]:
```
from sklearn.metrics.cluster import v_measure_score
```

In [51]:
```
data = pd.read_csv("Dataset_spine.csv")
data.head()
np.shape(data)
```

Out[51]:
```
(310, 14)
```

In [52]:
```
data_woc = data.iloc[:,0:12] #with out class variable from the data set
data_woc
```

Out[52]:

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254400	0.744503	12.5661	14.5386
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	0.415186	12.8874	17.5323
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	0.474889	26.8343	17.4861
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	0.369345	23.5603	12.7074
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	0.543360	35.4940	15.9546
...	...	...	...	...	...	...	...	...	...
305	47.903565	13.616688	36.000000	34.286877	117.449062	-4.245395	0.129744	7.8433	14.7484
306	53.936748	20.721496	29.220534	33.215251	114.365845	-0.421010	0.047913	19.1986	18.1972
307	61.446597	22.694968	46.170347	38.751628	125.670725	-2.707880	0.081070	16.2059	13.5565
308	45.252792	8.693157	41.583126	36.559635	118.545842	0.214750	0.159251	14.7334	16.0928
309	33.841641	5.073991	36.641233	28.767649	123.945244	-0.199249	0.674504	19.3825	17.6963

310 rows × 12 columns

In [53]:
```
data_class = data.iloc[:,12] #Segregating the target / class variable …
data_class
```

Out[53]:
```
0      Abnormal
1      Abnormal
2      Abnormal
3      Abnormal
4      Abnormal
         ...
305      Normal
306      Normal
307      Normal
308      Normal
309      Normal
Name: Class_att, Length: 310, dtype: object
```

In [54]:
```
f1 = data_woc['Col1'].values
f1
```

Out[54]:
```
array([ 63.0278175 ,  39.05695098,  68.83202098,  69.29700807,
        49.71285934,  40.25019968,  53.43292815,  45.36675362,
        43.79019026,  36.68635286,  49.70660953,  31.23238734,
        48.91555137,  53.5721702 ,  57.30022656,  44.31890674,
        63.83498162,  31.27601184,  38.69791243,  41.72996308,
        43.92283983,  54.91944259,  63.07361096,  45.54078988,
        36.12568347,  54.12492019,  26.14792141,  43.58096394,
        44.5510115 ,  66.87921138,  50.81926781,  46.39026008,
        44.93667457,  38.66325708,  59.59554032,  31.48421834,
        32.09098679,  35.70345781,  55.84328595,  52.41938511,
        35.49244617,  46.44207842,  53.85479842,  66.28539377,
        56.03021778,  50.91244034,  48.332638  ,  41.35250407,
        40.55735663,  41.76773173,  55.28585178,  74.43359316,
        50.20966979,  30.14993632,  41.17167989,  47.65772963,
        43.34960621,  46.85578065,  43.20318499,  48.10923638,
        74.37767772,  89.68056731,  44.529051  ,  77.69057712,
        76.1472121 ,  83.93300857,  78.49173027,  75.64973136,
        72.07627839,  58.59952852,  72.56070163,  86.90079431,
        84.97413208,  55.512212  ,  72.2223343 ,  70.22145219,
        86.75360946,  58.78254775,  67.41253785,  47.74467877,
        77.10657122,  74.00554124,  88.62390839,  81.10410039,
        76.32600187,  45.44374959,  59.78526526,  44.91414916,
        56.60577127,  71.18681115,  81.65603206,  70.95272771,
        85.35231529,  58.10193455,  94.17482232,  57.52235608,
        96.65731511,  74.72074622,  77.65511874,  58.52162283,
        84.5856071 ,  79.93857026,  70.39930842,  49.78212054,
        77.40933294,  65.00796426,  65.01377322,  78.42595126,
        63.17298709,  68.61300092,  63.90063261,  84.99895554,
        42.02138603,  69.75666532,  80.98807441, 129.8340406 ,
        70.48410444,  86.04127982,  65.53600255,  60.7538935 ,
        54.74177518,  83.87994081,  80.07491418,  65.66534698,
        74.71722805,  48.06062649,  70.67689818,  80.43342782,
        90.51396072,  77.23689752,  50.06678595,  69.78100617,
        69.62628302,  81.75441933,  52.20469309,  77.12134424,
        88.0244989 ,  83.39660609,  72.05403412,  85.09550254,
        69.56348614,  89.5049473 ,  85.29017283,  60.62621697,
        60.04417717,  85.64378664,  85.58171024,  55.08076562,
        65.75567895,  79.24967118,  81.11260488,  48.0306238 ,
        63.40448058,  57.28694488,  41.18776972,  66.80479632,
        79.4769781 ,  44.21646446,  57.03509717,  64.27481758,
        92.02630795,  67.26314926, 118.1446548 , 115.9232606 ,
        53.94165809,  83.7031774 ,  56.99140382,  72.34359434,
        95.38259648,  44.25347645,  64.80954139,  78.40125389,
        56.66829282,  50.82502875,  61.41173702,  56.56382381,
        67.02766447,  80.81777144,  80.65431956,  68.72190982,
        37.90391014,  64.62400798,  75.43774787,  71.00194076,
        81.05661087,  91.46874146,  81.08232025,  60.419932  ,
        85.68094951,  82.4065243 ,  43.7182623 ,  86.472905  ,
        74.46908181,  70.25043628,  72.64385013,  71.24176388,
        63.7723908 ,  58.82837872,  74.85448008,  75.29847847,
        63.36433898,  67.51305267,  76.31402766,  73.63596236,
        56.53505139,  80.11157156,  95.48022873,  74.09473084,
        87.67908663,  48.25991962,  38.50527283,  54.92085752,
        44.36249017,  48.3189305 ,  45.70178875,  30.74193812,
        50.91310144,  38.12658854,  51.62467183,  64.31186727,
        44.48927476,  54.9509702 ,  56.10377352,  69.3988184 ,
        89.83467631,  59.72614016,  63.95952166,  61.54059876,
        38.04655072,  43.43645061,  65.61180231,  53.91105429,
        43.11795103,  40.6832291 ,  37.7319919 ,  63.92947003,
        61.82162717,  62.14080535,  69.00491277,  56.44702568,
        41.6469159 ,  51.52935759,  39.08726449,  34.64992241,
        63.02630005,  47.80555887,  46.63786363,  49.82813487,
        47.31964755,  50.75329025,  36.15782981,  40.74699612,
        42.91804052,  63.79242525,  72.95564397,  67.53818154,
```

```
       54.75251965,  50.16007802,  40.34929637,  63.61919213,
       54.14240778,  74.97602148,  42.51727249,  33.78884314,
       54.5036853 ,  48.17074627,  46.37408781,  52.86221391,
       57.1458515 ,  37.14014978,  51.31177106,  42.51561014,
       39.35870531,  35.8775708 ,  43.1919153 ,  67.28971201,
       51.32546366,  65.7563482 ,  40.41336566,  48.80190855,
       50.08615264,  64.26150724,  53.68337998,  48.99595771,
       59.16761171,  67.80469442,  61.73487533,  33.04168754,
       74.56501543,  44.43070103,  36.42248549,  51.07983294,
       34.75673809,  48.90290434,  46.23639915,  46.42636614,
       39.65690201,  45.57548229,  66.50717865,  82.90535054,
       50.67667667,  89.01487529,  54.60031622,  34.38229939,
       45.07545026,  47.90356517,  53.93674778,  61.44659663,
       45.25279209,  33.84164075])
```

In [55]:
```
f2 = data_woc['Col5'].values
#f2
```

In [56]:
```
f3 = data_woc['Col9'].values
#f3
```

In [57]:
```
X = np.array(list(zip(f1, f2, f3)))
X
```

Out[57]:
```
array([[ 63.0278175 ,  98.67291675,  14.5386    ],
       [ 39.05695098, 114.4054254 ,  17.5323    ],
       [ 68.83202098, 105.9851355 ,  17.4861    ],
       [ 69.29700807, 101.8684951 ,  12.7074    ],
       [ 49.71285934, 108.1687249 ,  15.9546    ],
       [ 40.25019968, 130.3278713 ,  12.0036    ],
       [ 53.43292815, 120.5675233 ,  10.7146    ],
       [ 45.36675362, 117.2700675 ,   7.7676    ],
       [ 43.79019026, 125.0028927 ,  11.4234    ],
       [ 36.68635286,  84.24141517,   8.738     ],
       [ 49.70660953, 108.6482654 ,  16.5097    ],
       [ 31.23238734, 120.0553988 ,   9.2589    ],
       [ 48.91555137, 119.321358  ,   7.2049    ],
       [ 53.5721702 , 110.9666978 ,  12.8127    ],
       [ 57.30022656, 116.8065868 ,  18.6222    ],
       [ 44.31890674, 124.1158358 ,  19.1756    ],
       [ 63.83498162, 112.3094915 ,  16.8116    ],
       [ 31.27601184, 129.0114183 ,  18.6089    ],
       [ 38.69791243, 123.1592507 ,  17.9575    ],
       [ 41.72996308, 116.5857056 ,  12.4637    ],
       [ 43.92283983, 134.4610156 ,  16.8965    ],
       [ 54.91944259, 125.2127163 ,  14.2195    ],
       [ 63.07361096, 106.4243295 ,  17.6891    ],
       [ 45.54078988, 117.9808303 ,   9.1019    ],
       [ 36.12568347, 115.5771163 ,  15.7438    ],
       [ 54.12492019, 121.447011  ,  13.57     ],
       [ 26.14792141, 125.2032956 ,   8.6406    ],
       [ 43.58096394, 109.271634  ,  17.97     ],
       [ 44.5510115 , 111.0729197 ,  10.2244    ],
       [ 66.87921138, 113.4770183 ,   8.2495    ],
       [ 50.81926781, 112.192804  ,   7.2481    ],
       [ 46.39026008,  98.77454633,   7.4515    ],
       [ 44.93667457, 117.9803245 ,  13.8357    ],
       [ 38.66325708, 124.914118  ,   7.5284    ],
       [ 59.59554032, 119.3303537 ,   8.3058    ],
       [ 31.48421834, 113.8331446 ,   8.1003    ],
       [ 32.09098679, 132.264735  ,  11.6395    ],
       [ 35.70345781, 137.5406125 ,  12.7274    ],
       [ 55.84328595, 123.3118449 ,   8.855     ],
       [ 52.41938511, 116.5597709 ,  18.9113    ],
       [ 35.49244617, 106.9388517 ,   9.3239    ],
       [ 46.44207842, 115.4814047 ,  12.2285    ],
       [ 53.85479842, 121.6709148 ,  11.5243    ],
       [ 66.28539377, 121.2196839 ,  10.2636    ],
       [ 56.03021778, 114.0231172 ,  12.2345    ],
       [ 50.91244034, 117.4222591 ,  18.6181    ],
       [ 48.332638  , 117.3846251 ,  11.0942    ],
       [ 41.35250407, 113.2666746 ,  16.7065    ],
       [ 40.55735663, 121.0462458 ,   8.2526    ],
       [ 41.76773173, 118.3633889 ,   7.1646    ],
       [ 55.28585178, 115.8770174 ,  15.9714    ],
       [ 74.43359316, 107.9493045 ,   9.8062    ],
       [ 50.20966979, 128.2925148 ,  13.9907    ],
       [ 30.14993632, 112.6841408 ,  10.7071    ],
       [ 41.17167989, 116.3778894 ,  12.8432    ],
       [ 47.65772963,  98.24978071,  16.9817    ],
       [ 43.34960621, 112.7761866 ,  14.2176    ],
       [ 46.85578065, 116.2509174 ,  14.6233    ],
       [ 43.20318499, 124.8461088 ,  16.2905    ],
       [ 48.10923638, 124.0564518 ,  14.1761    ],
       [ 74.37767772, 143.5606905 ,  15.3975    ],
       [ 89.68056731, 129.9554764 ,  18.6012    ],
       [ 44.529051  , 134.7117723 ,  13.7595    ],
       [ 77.69057712, 114.818751  ,  15.3209    ],
```

```
[ 76.1472121 , 123.9320096 ,   7.8098    ],
[ 83.93300857, 115.012334  ,   7.2405    ],
[ 78.49173027, 118.5303266 ,  12.6737    ],
[ 75.64973136,  95.9036288 ,  13.3444    ],
[ 72.07627839, 114.2130126 ,   9.6504    ],
[ 58.59952852, 102.0428116 ,  15.3124    ],
[ 72.56070163, 119.1937238 ,  18.3111    ],
[ 86.90079431, 135.0753635 ,   8.7655    ],
[ 84.97413208, 125.6595336 ,  17.7501    ],
[ 55.512212  , 122.648753  ,   8.6569    ],
[ 72.2223343 , 137.7366546 ,   7.4437    ],
[ 70.22145219, 148.5255624 ,   9.5742    ],
[ 86.75360946, 139.414504  ,  17.3635    ],
[ 58.78254775,  98.50115697,  11.7696    ],
[ 67.41253785, 111.12397   ,  12.8779    ],
[ 47.74467877, 117.5120039 ,  14.7577    ],
[ 77.10657122, 112.1516    ,  13.2216    ],
[ 74.00554124, 120.2059626 ,  12.5946    ],
[ 88.62390839, 121.7647796 ,  13.659     ],
[ 81.10410039, 151.8398566 ,  11.2339    ],
[ 76.32600187, 124.267007  ,   8.383     ],
[ 45.44374959, 163.0710405 ,  15.0011    ],
[ 59.78526526, 119.3191109 ,  15.0648    ],
[ 44.91414916, 130.0756599 ,  18.1846    ],
[ 56.60577127, 127.2945222 ,   8.7779    ],
[ 71.18681115, 119.8649383 ,  16.7172    ],
[ 81.65603206, 114.7698556 ,  11.2491    ],
[ 70.95272771, 116.1779325 ,  16.3676    ],
[ 85.35231529, 124.4197875 ,  11.7878    ],
[ 58.10193455, 113.5876551 ,  11.6354    ],
[ 94.17482232, 114.8901128 ,  19.1837    ],
[ 57.52235608, 140.9817119 ,   7.5666    ],
[ 96.65731511, 120.6730408 ,  13.0551    ],
[ 74.72074622, 109.3565941 ,   7.8697    ],
[ 77.65511874, 123.0557067 ,  15.0493    ],
[ 58.52162283, 115.514798  ,  10.3564    ],
[ 84.5856071 , 108.0102185 ,  15.7092    ],
[ 79.93857026, 114.787107  ,  10.922     ],
[ 70.39930842, 102.3375244 ,   9.5431    ],
[ 49.78212054, 110.8647831 ,  11.704     ],
[ 77.40933294, 118.4507311 ,  12.9197    ],
[ 65.00796426, 116.5811088 ,  13.8536    ],
[ 65.01377322,  94.73852542,   8.9024    ],
[ 78.42595126, 138.5541111 ,  16.6264    ],
[ 63.17298709, 110.6440206 ,  11.073     ],
[ 68.61300092, 123.4311742 ,  17.7171    ],
[ 63.90063261, 114.1292425 ,  16.645     ],
[ 84.99895554, 126.9129899 ,   9.0119    ],
[ 42.02138603, 111.5857819 ,  10.8504    ],
[ 69.75666532,  96.49136982,  17.4517    ],
[ 80.98807441, 141.0881494 ,  13.5136    ],
[129.8340406 , 107.690466  ,  11.1514    ],
[ 70.48410444, 114.1900488 ,  10.4727    ],
[ 86.04127982, 122.0929536 ,  12.008     ],
[ 65.53600255, 136.4403015 ,  13.7801    ],
[ 60.7538935 , 113.0533309 ,  17.3268    ],
[ 54.74177518, 117.6432188 ,  19.2053    ],
[ 83.87994081, 124.6460723 ,  11.1917    ],
[ 80.07491418, 110.7099121 ,  10.3082    ],
[ 65.66534698, 109.1627768 ,  11.528     ],
[ 74.71722805, 107.1822176 ,   8.2316    ],
[ 48.06062649,  95.44375749,  11.5825    ],
[ 70.67689818, 103.0083545 ,  14.8568    ],
[ 80.43342782, 116.4389807 ,  18.2886    ],
```

```
[ 90.51396072, 100.8921596 ,  16.3289    ],
[ 77.23689752, 110.6903772 ,  12.5804    ],
[ 50.06678595,  99.71245318,  18.8412    ],
[ 69.78100617, 118.9306656 ,   8.7164    ],
[ 69.62628302, 116.8030913 ,  15.4963    ],
[ 81.75441933, 119.4250857 ,  18.1885    ],
[ 52.20469309, 136.9725168 ,   8.5783    ],
[ 77.12134424, 110.6111484 ,  17.0071    ],
[ 88.0244989 , 116.6015376 ,  12.1692    ],
[ 83.39660609, 110.4665164 ,   7.6245    ],
[ 72.05403412, 107.1723576 ,  10.8824    ],
[ 85.09550254, 109.062312  ,   8.4727    ],
[ 69.56348614, 105.0673556 ,  18.2659    ],
[ 89.5049473 , 134.6342912 ,   8.9861    ],
[ 85.29017283, 110.6607005 ,  12.5462    ],
[ 60.62621697, 117.2255542 ,   8.8097    ],
[ 60.04417717, 105.1316639 ,  16.5256    ],
[ 85.64378664, 105.1440758 ,   8.0109    ],
[ 85.58171024, 114.8660487 ,  11.2204    ],
[ 55.08076562, 109.9153669 ,   9.0278    ],
[ 65.75567895, 104.3949585 ,   7.4483    ],
[ 79.24967118,  98.62251165,  14.5804    ],
[ 81.11260488,  94.01878339,   8.3428    ],
[ 48.0306238 , 125.3509625 ,  14.8226    ],
[ 63.40448058, 111.9160075 ,  13.1978    ],
[ 57.28694488, 116.7353868 ,  17.6755    ],
[ 41.18776972, 103.3488802 ,  10.5374    ],
[ 66.80479632,  82.45603817,  12.5023    ],
[ 79.4769781 , 118.5886691 ,   8.384     ],
[ 44.21646446, 108.6295666 ,  14.7325    ],
[ 57.03509717, 103.0486975 ,  14.1033    ],
[ 64.27481758,  95.25245421,   7.0378    ],
[ 92.02630795, 115.72353   ,   9.8318    ],
[ 67.26314926,  97.8010854 ,   9.6738    ],
[118.1446548 ,  81.0245406 ,  10.9266    ],
[115.9232606 , 104.6986033 ,   8.8519    ],
[ 53.94165809, 124.3978211 ,  17.5404    ],
[ 83.7031774 , 125.4801739 ,  15.1125    ],
[ 56.99140382, 109.978045  ,  18.5683    ],
[ 72.34359434,  70.08257486,  15.7134    ],
[ 95.38259648,  89.3075466 ,   7.2124    ],
[ 44.25347645,  98.27410705,  12.4275    ],
[ 64.80954139, 111.679961  ,   7.175     ],
[ 78.40125389, 104.7312342 ,   7.1135    ],
[ 56.66829282,  93.69220863,  18.1719    ],
[ 50.82502875,  78.99945411,  16.3725    ],
[ 61.41173702, 103.4045971 ,  15.698     ],
[ 56.56382381,  98.77711506,  13.9634    ],
[ 67.02766447, 100.7154129 ,  13.34      ],
[ 80.81777144,  89.47183446,  12.6822    ],
[ 80.65431956, 120.1034928 ,  18.4914    ],
[ 68.72190982, 125.0185168 ,  16.0009    ],
[ 37.90391014, 157.848799  ,  17.5433    ],
[ 64.62400798,  90.298468  ,  18.5975    ],
[ 75.43774787, 106.8295898 ,  11.0399    ],
[ 71.00194076, 125.1642324 ,  14.5544    ],
[ 81.05661087, 125.430176  ,  10.3301    ],
[ 91.46874146, 117.3078968 ,  12.1023    ],
[ 81.08232025,  90.07187999,  13.4438    ],
[ 60.419932  , 109.0330745 ,  18.2437    ],
[ 85.68094951, 120.8407069 ,  13.238     ],
[ 82.4065243 , 117.0422439 ,  16.189     ],
[ 43.7182623 ,  88.43424213,  16.9317    ],
[ 86.472905  ,  97.4041888 ,  10.416     ],
```

```
[ 74.46908181, 146.4660009 ,  12.3988    ],
[ 70.25043628, 119.2370072 ,  11.9015    ],
[ 72.64385013, 116.9634162 ,  14.5781    ],
[ 71.24176388, 110.703107  ,  15.243     ],
[ 63.7723908 ,  89.82274067,  10.4232    ],
[ 58.82837872, 135.6294176 ,  15.9381    ],
[ 74.85448008, 115.2087008 ,   9.351     ],
[ 75.29847847, 118.8833881 ,  12.2607    ],
[ 63.36433898, 130.9992576 ,  10.1943    ],
[ 67.51305267, 145.6010328 ,  19.324     ],
[ 76.31402766, 132.2672855 ,  13.6555    ],
[ 73.63596236,  98.72792982,   9.4553    ],
[ 56.53505139, 101.7233343 ,  14.4887    ],
[ 80.11157156, 125.5936237 ,  16.1451    ],
[ 95.48022873,  96.68390337,  14.5939    ],
[ 74.09473084, 128.4057314 ,  10.1368    ],
[ 87.67908663, 120.9448288 ,  13.0473    ],
[ 48.25991962,  94.88233607,  15.0636    ],
[ 38.50527283, 127.6328747 ,  12.7802    ],
[ 54.92085752, 125.8466462 ,  11.3014    ],
[ 44.36249017, 129.220682  ,   7.808     ],
[ 48.3189305 , 128.9803079 ,  14.635     ],
[ 45.70178875, 130.1783144 ,  18.3694    ],
[ 30.74193812, 142.4101072 ,  18.0205    ],
[ 50.91310144, 118.151531  ,  13.0974    ],
[ 38.12658854, 132.114805  ,  11.3293    ],
[ 51.62467183, 129.385308  ,  13.8149    ],
[ 64.31186727, 106.1777511 ,  11.4337    ],
[ 44.48927476, 113.7784936 ,  13.4464    ],
[ 54.9509702 , 126.9703283 ,  10.5263    ],
[ 56.10377352, 116.2285032 ,  16.5844    ],
[ 69.3988184 , 103.5825398 ,  13.1408    ],
[ 89.83467631, 100.5011917 ,  17.7556    ],
[ 59.72614016, 125.1742214 ,   8.7872    ],
[ 63.95952166, 142.3601245 ,  10.957     ],
[ 61.54059876, 118.6862678 ,   7.5117    ],
[ 38.04655072, 123.8034132 ,  11.1235    ],
[ 43.43645061, 137.4396942 ,  11.0132    ],
[ 65.61180231, 124.1280012 ,  17.6222    ],
[ 53.91105429, 118.1930354 ,  17.7521    ],
[ 43.11795103, 128.5177217 ,  15.1873    ],
[ 40.6832291 , 139.1184721 ,  13.6632    ],
[ 37.7319919 , 135.740926  ,  10.2016    ],
[ 63.92947003, 113.0659387 ,  11.1443    ],
[ 61.82162717, 121.779803  ,  13.6082    ],
[ 62.14080535, 133.2818339 ,  16.3819    ],
[ 69.00491277, 126.6116215 ,   7.4752    ],
[ 56.44702568, 139.1896903 ,  16.1623    ],
[ 41.6469159 , 116.5551679 ,   8.7771    ],
[ 51.52935759, 126.7185156 ,  11.087     ],
[ 39.08726449, 131.5844199 ,  12.6508    ],
[ 34.64992241, 123.9877408 ,  15.279     ],
[ 63.02630005, 114.5066078 ,  14.0752    ],
[ 47.80555887, 125.3911378 ,  12.0502    ],
[ 46.63786363, 119.3776026 ,  11.0671    ],
[ 49.82813487, 121.4355585 ,  15.5507    ],
[ 47.31964755, 120.5769719 ,  15.313     ],
[ 50.75329025, 122.343516  ,  14.5673    ],
[ 36.15782981, 135.9369096 ,  17.1414    ],
[ 40.74699612, 139.2471502 ,  16.527     ],
[ 42.91804052, 121.6068586 ,   8.5981    ],
[ 63.79242525, 119.5503909 ,   8.0233    ],
[ 72.95564397, 111.2340468 ,  12.4063    ],
[ 67.53818154, 123.6322597 ,   9.9145    ],
```

```
       [ 54.75251965, 123.0379985 ,  14.9326    ],
       [ 50.16007802, 131.8024914 ,  16.4723    ],
       [ 40.34929637, 128.0099272 ,  13.9897    ],
       [ 63.61919213, 117.0897469 ,   9.9518    ],
       [ 54.14240778, 122.2090834 ,   9.3492    ],
       [ 74.97602148, 105.6453997 ,  11.3042    ],
       [ 42.51727249, 128.9056892 ,  18.7123    ],
       [ 33.78884314, 128.3253556 ,  11.3129    ],
       [ 54.5036853 , 111.7911722 ,  18.7158    ],
       [ 48.17074627, 135.6233101 ,  14.418     ],
       [ 46.37408781, 121.2476572 ,   8.8496    ],
       [ 52.86221391, 123.0912395 ,  15.3118    ],
       [ 57.1458515 , 113.8061775 ,   9.5092    ],
       [ 37.14014978, 125.0143609 ,  18.2159    ],
       [ 51.31177106, 126.4722584 ,  10.5134    ],
       [ 42.51561014, 120.631941  ,  15.9657    ],
       [ 39.35870531, 117.8187599 ,   9.8168    ],
       [ 35.8775708 , 126.9239062 ,  11.0116    ],
       [ 43.1919153 , 123.4674001 ,  17.7684    ],
       [ 67.28971201, 137.5917777 ,  13.7148    ],
       [ 51.32546366, 131.3061224 ,  12.3509    ],
       [ 65.7563482 , 129.3935728 ,  17.5228    ],
       [ 40.41336566, 119.3356546 ,  16.4107    ],
       [ 48.80190855, 139.1504066 ,  19.2659    ],
       [ 50.08615264, 119.1346221 ,  15.1042    ],
       [ 64.26150724, 115.3882683 ,  15.3692    ],
       [ 53.68337998, 113.9137026 ,  11.4937    ],
       [ 48.99595771, 126.3981876 ,   7.5426    ],
       [ 59.16761171, 121.0356423 ,  14.7433    ],
       [ 67.80469442, 119.6856451 ,  15.4016    ],
       [ 61.73487533, 120.9201997 ,  14.9831    ],
       [ 33.04168754, 120.3886112 ,  11.2466    ],
       [ 74.56501543, 105.417304  ,  15.6236    ],
       [ 44.43070103, 131.7176127 ,   7.9912    ],
       [ 36.42248549, 126.0768612 ,  10.87      ],
       [ 51.07983294, 115.8037111 ,  12.6652    ],
       [ 34.75673809, 127.1398495 ,  11.2762    ],
       [ 48.90290434, 137.1082886 ,   9.9608    ],
       [ 46.23639915, 128.0636203 ,  15.1769    ],
       [ 46.42636614, 130.3500956 ,  10.6369    ],
       [ 39.65690201, 131.922009  ,  18.3533    ],
       [ 45.57548229, 116.7970069 ,  12.9572    ],
       [ 66.50717865, 128.9029049 ,  11.8978    ],
       [ 82.90535054, 110.7089577 ,  12.3646    ],
       [ 50.67667667, 116.5879699 ,   8.007     ],
       [ 89.01487529, 111.4810746 ,  13.3731    ],
       [ 54.60031622, 118.3433212 ,  11.4198    ],
       [ 34.38229939, 128.3001991 ,  16.6255    ],
       [ 45.07545026, 147.8946372 ,  14.3037    ],
       [ 47.90356517, 117.4490622 ,  14.7484    ],
       [ 53.93674778, 114.365845  ,  18.1972    ],
       [ 61.44659663, 125.6707246 ,  13.5565    ],
       [ 45.25279209, 118.5458418 ,  16.0928    ],
       [ 33.84164075, 123.9452436 ,  17.6963    ]])
```

In [58]:

```
kmeans = KMeans(n_clusters = 2, random_state = 123)
```

In [59]:

```
model = kmeans.fit(X)
```

```
C:\Users\ITER\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureW
arning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set th
e value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\ITER\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWar
ning: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment vari
able OMP_NUM_THREADS=2.
  warnings.warn(
```

In [60]:
```
cluster_labels = kmeans.predict(X)
```

In [61]:
```
v=v_measure_score(cluster_labels, data_class) #Output 0.12267025741680369
print(v)
```

```
0.1226702574168037
```

Silhouette Width

For calculating **Silhouette Width**, **silhouette_score** function of **sklearn.metrics** library can be used.

Below is the code for calculating Silhouette Width.

In [62]:
```
from sklearn.metrics import silhouette_score
```

In [63]:
```
sil = silhouette_score(X, cluster_labels, metric ="euclidean",sample_size = len(dat
print(sil)
# “X” is a feature matrix for the feature subset selected for clustering and “data”
```

```
0.4028110643981811
```

Feature_Engineering_Lab

October 25, 2025

0.1 # Feature Engineering with Python

0.2 Objective

In this lab, we explore **Feature Engineering**, which involves: 1. Feature Construction – creating new features 2. Feature Extraction – transforming or reducing features 3. Feature Selection – selecting the most relevant subset of features

1 1. Feature Construction

Feature Construction involves **creating new or modified features** that help ML models perform better.

[1]:
```python
import pandas as pd
# Example: Creating Derived Features
data = {
    'length': [20, 25, 30, 22],
    'breadth': [15, 20, 18, 25],
    'price': [200000, 250000, 300000, 220000]
}
df = pd.DataFrame(data)
df['area'] = df['length'] * df['breadth']
df
```

[1]:

	length	breadth	price	area
0	20	15	200000	300
1	25	20	250000	500
2	30	18	300000	540
3	22	25	220000	550

[2]:
```python
# Example: Encoding Nominal Variables
data = {
    'city': ['A', 'B', 'C', 'A'],
    'parents_athlete': ['Y', 'N', 'N', 'Y'],
    'chance_of_win': ['Y', 'N', 'Y', 'N']
}
df = pd.DataFrame(data)
pd.get_dummies(df, drop_first=True)
```

[2]:

	city_B	city_C	parents_athlete_Y	chance_of_win_Y
0	False	False	True	True
1	True	False	False	False
2	False	True	False	True
3	False	False	True	False

Explanation: # get_dummies() converts each categorical column into binary (0/1) columns. # drop_first=True, drops the first category of each variable to avoid multicollinearity (important in linear models).

[3]:

```python
# Example: Encoding Ordinal Variables
data = {'grade': ['A', 'B', 'C', 'D', 'A']}
df = pd.DataFrame(data)
grade_map = {'A': 1, 'B': 2, 'C': 3, 'D': 4}
df['num_grade'] = df['grade'].map(grade_map)
df
```

[3]:

	grade	num_grade
0	A	1
1	B	2
2	C	3
3	D	4
4	A	1

Explanation: # Uses .map() to apply the grade_map to the 'grade' column. # Creates a new column 'num_grade' with the corresponding numeric values.

[4]:

```python
# Example: Binning Continuous Variables
import numpy as np
df = pd.DataFrame({'price': [200000, 350000, 600000, 800000]})
bins = [0, 300000, 600000, np.inf]
labels = ['Low', 'Medium', 'High']
df['price_category'] = pd.cut(df['price'], bins=bins, labels=labels)
df
```

[4]:

	price	price_category
0	200000	Low
1	350000	Medium
2	600000	Medium
3	800000	High

Explanation: # bins defines the edges of the price ranges: - 0–300000 → Low - 300000–600000 → Medium - 600000+ → High # labels assigns names to each bin. # Uses pd.cut() to categorize each price into one of the defined bins. # Adds a new column 'price_category' with the corresponding label. Note: By default, pd.cut() is right-inclusive, so 600000 falls into the Medium bin.

2 2. Feature Extraction

Feature Extraction reduces the number of features while preserving important information.

[1]:
```python
# PCA Example
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

# Load iris dataset
iris = load_iris()
X = iris.data

# Apply PCA (reduce to 2 components)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

print("Original shape:", X.shape)
print("Reduced shape:", X_pca.shape)
```

```
Original shape: (150, 4)
Reduced shape: (150, 2)
```

[2]:
```python
# SVD Example
import numpy as np
from sklearn.decomposition import TruncatedSVD

# Random matrix (simulating text data)
X = np.random.rand(5, 4)

# Apply Truncated SVD
svd = TruncatedSVD(n_components=2)
X_svd = svd.fit_transform(X)

print("Original:", X.shape)
print("Reduced:", X_svd.shape)
```

```
Original: (5, 4)
Reduced: (5, 2)
```

[ ]:
```
Explanation:
```

[3]:
```python
# LDA Example
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)

print("Original shape:", X.shape)
```

```
print("Reduced shape:", X_lda.shape)
```

```
Original shape: (150, 4)
Reduced shape: (150, 2)
```

3 3. Feature Selection

Feature Selection identifies the most relevant features.

[16]:
```
# Filter Method (Chi-Square)
from sklearn.feature_selection import SelectKBest, chi2
X_new = SelectKBest(chi2, k=2).fit_transform(iris.data, iris.target)
print('Original:', iris.data.shape, 'Reduced:', X_new.shape)
```

```
Original: (150, 4) Reduced: (150, 2)
```

[21]:
```
# Alternate Method (Chi-Square)
from sklearn.feature_selection import SelectKBest, chi2

# Separate features and target using iloc
X = df.iloc[:, :-1]  # all columns except last
y = df.iloc[:, -1]   # last column

# Apply Chi-Square test
X_new = SelectKBest(chi2, k=2).fit_transform(X, y)

print("Original shape:", X.shape)
print("Reduced shape:", X_new.shape)
```

```
Original shape: (150, 4)
Reduced shape: (150, 2)
```

Explanation: .iloc ensures we select the independent variables (X) and dependent variable (y) directly by index position, regardless of column names.

[29]:
```
# Wrapper Method - Recursive Feature Elimination (RFE)
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=200)

# Apply RFE for 2 features
rfe = RFE(model, n_features_to_select=2)
fit = rfe.fit(X, y)

print("Selected Features:", fit.support_)
print("Feature Ranking:", fit.ranking_)
```

```
Selected Features: [False False  True False False False False False  True False]
Feature Ranking: [5 7 1 3 8 9 2 4 1 6]
```

[27]:

```python
# Embedded Method (Lasso)
from sklearn.linear_model import LassoCV
from sklearn.datasets import load_diabetes

# Load diabetes dataset as DataFrame
diabetes = load_diabetes(as_frame=True)
df_d = diabetes.frame

# Separate features and target using iloc
X = df_d.iloc[:, :-1]
y = df_d.iloc[:, -1]

# Apply LassoCV
lasso = LassoCV(cv=5)
lasso.fit(X, y)

print("Coefficients:", lasso.coef_)
print("Number of selected features:", sum(lasso.coef_ != 0))
```

```
Coefficients: [  -6.49469328 -235.99308032  521.7443693   321.0607768
-569.43813385
  302.45319289   -0.          143.69851474  669.92267515   66.83551067]
Number of selected features: 9
```

[ ]:

```python

```

CHAPTER 5

Probability Overview

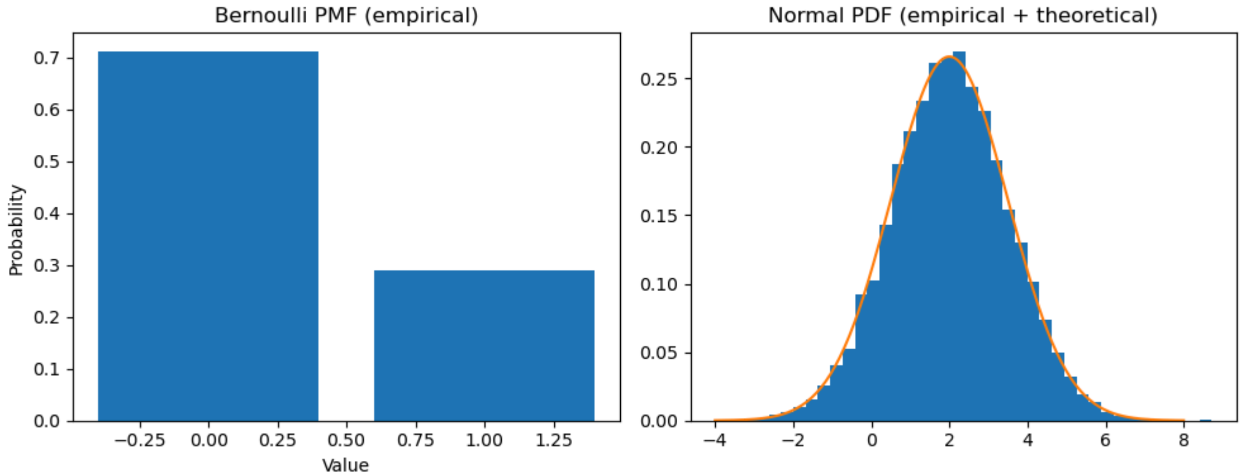
In [11]:
```
# Standard imports
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from math import factorial, sqrt, pi
np.random.seed(42)
```

1. Random Variables

Demonstration: discrete (Bernoulli) and continuous (Normal) random variables; compute mean & variance.

In [2]:
```
# Bernoulli (discrete) example
p = 0.3
n_samples = 10000
bern = np.random.binomial(1, p, size=n_samples)  # Bernoulli as Binomial(n=1)
print('Bernoulli sample mean (approx p):', bern.mean())
print('Bernoulli sample var (approx p(1-p)):', bern.var())

# Normal (continuous) example
mu, sigma = 2.0, 1.5
norm_samples = np.random.normal(mu, sigma, size=n_samples)
print('\nNormal sample mean (approx mu):', norm_samples.mean())
print('Normal sample var (approx sigma^2):', norm_samples.var())

# Plot PDFs / PMFs
fig, axes = plt.subplots(1,2, figsize=(10,4))
# Bernoulli PMF
vals, counts = np.unique(bern, return_counts=True)
axes[0].bar(vals, counts/len(bern))
axes[0].set_title('Bernoulli PMF (empirical)')
axes[0].set_xlabel('Value')
axes[0].set_ylabel('Probability')

# Normal PDF (empirical histogram + theoretical PDF)
axes[1].hist(norm_samples, bins=40, density=True)
x = np.linspace(mu-4*sigma, mu+4*sigma, 200)
axes[1].plot(x, stats.norm.pdf(x, mu, sigma))
axes[1].set_title('Normal PDF (empirical + theoretical)')
plt.tight_layout()
plt.show()
```

```
Bernoulli sample mean (approx p): 0.2887
Bernoulli sample var (approx p(1-p)): 0.20535231000000004

Normal sample mean (approx mu): 2.0185330937808708
Normal sample var (approx sigma^2): 2.2487603282724753
```

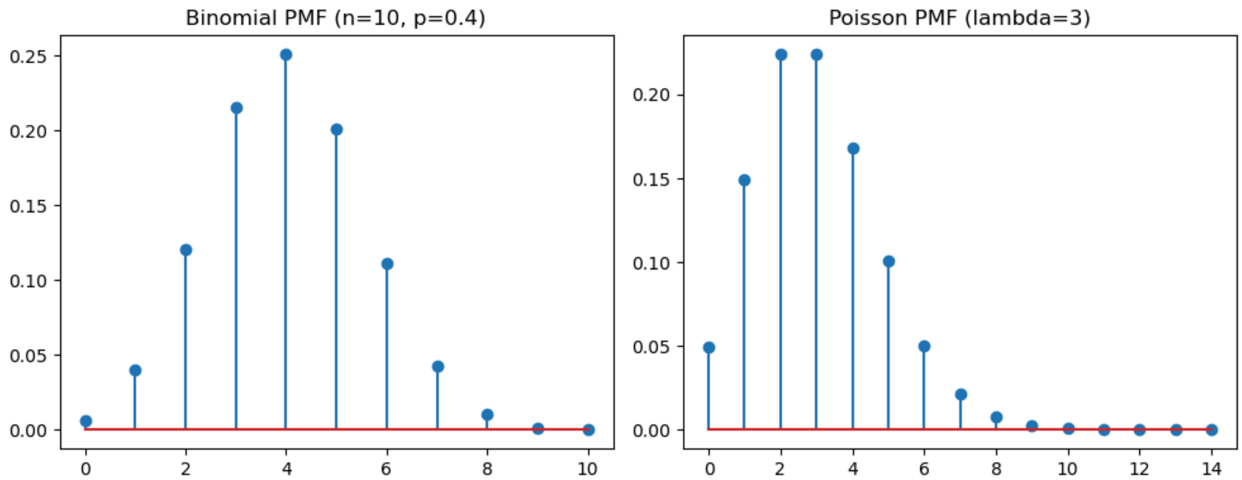


2. Common Discrete Distributions

Examples: Binomial, Poisson — theoretical PMF and sampling.

In [3]:
```python
# Binomial example (n, p)
n, p = 10, 0.4
k = np.arange(0, n+1)
pmf_binom = stats.binom.pmf(k, n, p)

print('Binomial theoretical mean, var:', stats.binom.mean(n, p), stats.binom.v

# Poisson example
lam = 3.0
k_p = np.arange(0, 15)
pmf_pois = stats.poisson.pmf(k_p, lam)
print('Poisson theoretical mean,var:', stats.poisson.mean(lam), stats.poisson.

# Plot PMFs
fig, ax = plt.subplots(1,2, figsize=(10,4))
ax[0].stem(k, pmf_binom)
ax[0].set_title('Binomial PMF (n=10, p=0.4)')
ax[1].stem(k_p, pmf_pois)
ax[1].set_title('Poisson PMF (lambda=3)')
plt.tight_layout()
plt.show()
```

```
Binomial theoretical mean, var: 4.0 2.3999999999999995
Poisson theoretical mean,var: 3.0 3.0
```

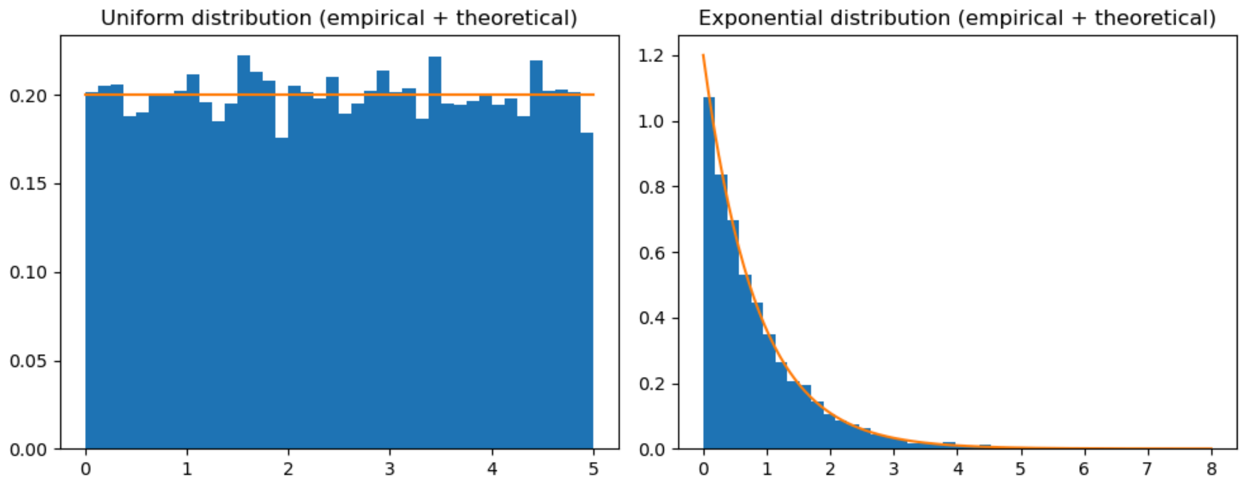


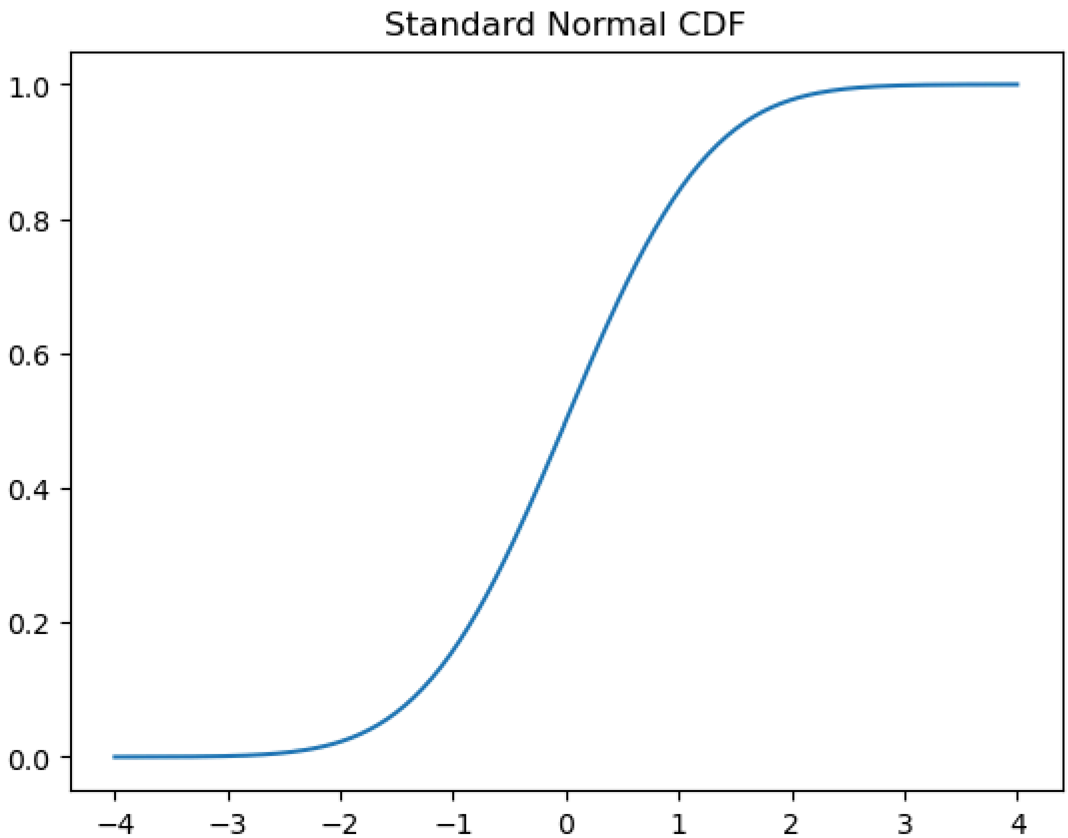
3. Common Continuous Distributions

Examples: Uniform, Normal, Exponential — theoretical PDF and sampling.

In [10]:

```
# Uniform(a,b)
a, b = 0, 5
uni_samps = np.random.uniform(a, b, size=10000)
x = np.linspace(a, b, 200)
# Exponential
lam = 1.2
exp_samps = np.random.exponential(1/lam, size=10000)
x_exp = np.linspace(0, 8, 200)

fig, ax = plt.subplots(1,2, figsize=(10,4))
ax[0].hist(uni_samps, bins=40, density=True)
ax[0].plot(x, stats.uniform.pdf(x, a, b-a))
ax[0].set_title('Uniform distribution (empirical + theoretical)')

ax[1].hist(exp_samps, bins=40, density=True)
ax[1].plot(x_exp, stats.expon.pdf(x_exp, scale=1/lam))
ax[1].set_title('Exponential distribution (empirical + theoretical)')
plt.tight_layout()
plt.show()

# Normal shown previously; we can show standard normal CDF example
xs = np.linspace(-4,4,200)
plt.plot(xs, stats.norm.cdf(xs))
plt.title('Standard Normal CDF')
plt.show()
```





4. Multiple Random Variables

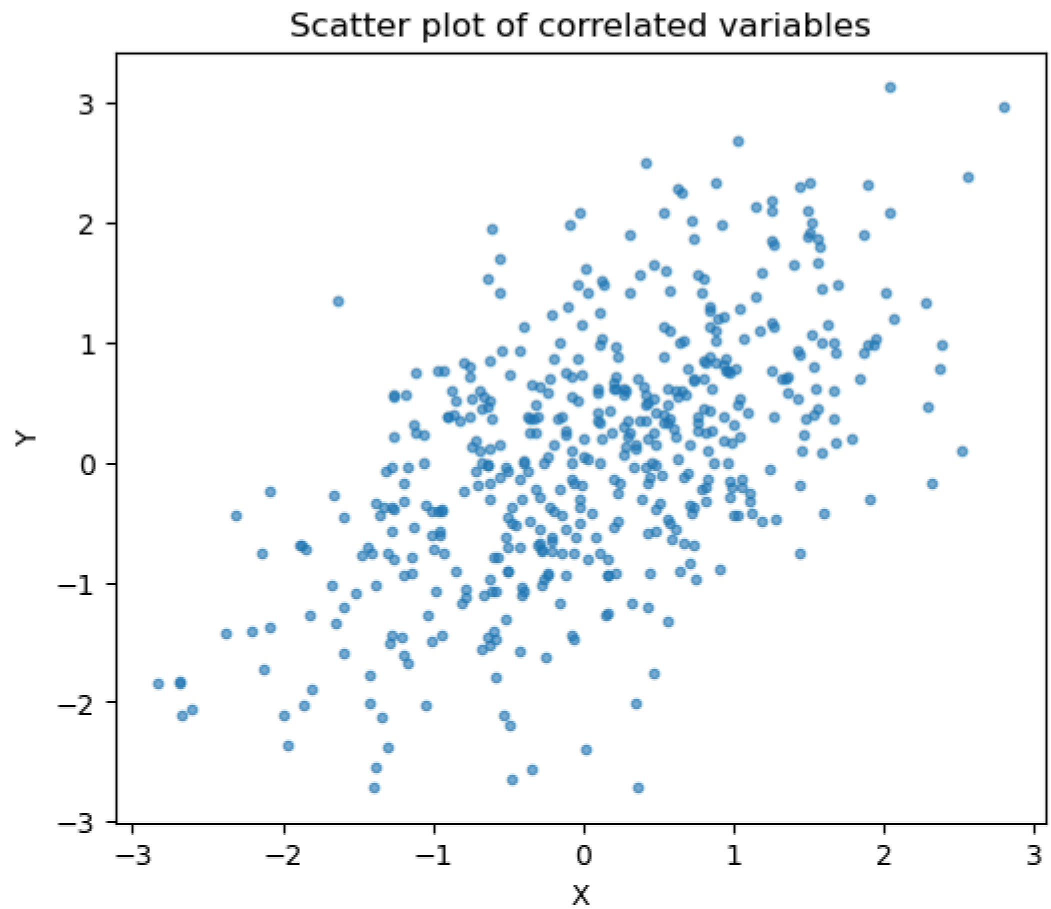
Joint, marginal, covariance, and correlation examples.

In [5]:

```
# Simulate correlated variables using a covariance matrix
mean = [0, 0]
cov = [[1.0, 0.6], [0.6, 1.0]]  # covariance matrix with positive correlation
samples = np.random.multivariate_normal(mean, cov, size=5000)
x = samples[:,0]
y = samples[:,1]
```

```
print('Empirical covariance:', np.cov(x, y)[0,1])
print('Empirical correlation:', np.corrcoef(x, y)[0,1])

# Scatter plot
plt.figure(figsize=(6,5))
plt.scatter(x[:500], y[:500], s=10, alpha=0.6)
plt.title('Scatter plot of correlated variables')
plt.xlabel('X'); plt.ylabel('Y')
plt.show()
```

```
Empirical covariance: 0.580270155733943
Empirical correlation: 0.5853545701662343
```



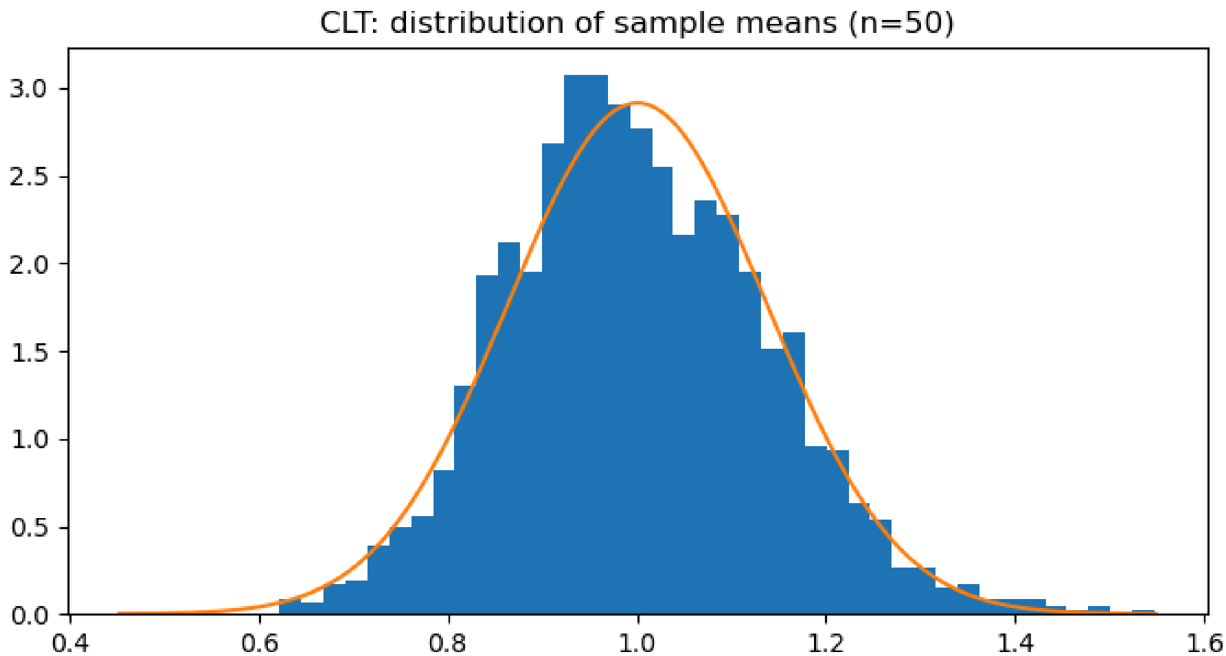
5. Central Limit Theorem (CLT)

Simulate sample means from an exponential distribution and observe convergence to normality.

In [6]:

```
# CLT demonstration
pop = np.random.exponential(scale=1.0, size=200000)  # population (skewed)
sample_size = 50
n_trials = 2000
means = [np.mean(np.random.choice(pop, sample_size)) for _ in range(n_trials)]
```

```python
plt.figure(figsize=(8,4))
plt.hist(means, bins=40, density=True)
# overlay normal with same mean and variance
mu_hat = np.mean(means)
sigma_hat = np.std(means)
x = np.linspace(mu_hat-4*sigma_hat, mu_hat+4*sigma_hat, 200)
plt.plot(x, stats.norm.pdf(x, mu_hat, sigma_hat))
plt.title(f'CLT: distribution of sample means (n={sample_size})')
plt.show()

print('Sample means mean:', mu_hat, 'std:', sigma_hat)
```




```
Sample means mean: 1.0005962542680136 std: 0.13692310275781513
```

CHAPTER 6

In [1]:
```
import pandas as pd
```

In [2]:
```
from sklearn.model_selection import train_test_split
```

In [3]:
```
from sklearn.metrics import accuracy_score
```

In [4]:
```
from sklearn.naive_bayes import GaussianNB
```

In [5]:
```
data=pd.read_csv("apndcts.csv")
```

In [6]:
```
data
```

Out[6]:

	At1	At2	At3	At4	At5	At6	At7	class
0	0.213	0.554	0.207	0.000	0.000	0.749	0.220	1
1	0.458	0.714	0.468	0.111	0.102	0.741	0.436	1
2	0.102	0.518	0.111	0.056	0.022	0.506	0.086	1
3	0.187	0.196	0.105	0.056	0.029	0.133	0.085	1
4	0.236	0.804	0.289	0.111	0.066	0.756	0.241	1
...	...	...	...	...	...	...	...	...
101	0.449	0.875	0.523	0.083	0.076	0.920	0.487	0
102	0.102	0.000	0.022	0.000	0.000	0.000	0.017	0
103	0.409	0.875	0.482	0.306	0.259	0.914	0.443	0
104	0.427	0.804	0.474	0.056	0.048	0.836	0.437	0
105	0.462	0.911	0.551	0.167	0.154	0.931	0.500	0

106 rows × 8 columns

In [7]:
```
predictors=data.iloc[:,0:7]#segregating the predictor variables
predictors
```

Out[7]:

	At1	At2	At3	At4	At5	At6	At7
0	0.213	0.554	0.207	0.000	0.000	0.749	0.220
1	0.458	0.714	0.468	0.111	0.102	0.741	0.436
2	0.102	0.518	0.111	0.056	0.022	0.506	0.086
3	0.187	0.196	0.105	0.056	0.029	0.133	0.085
4	0.236	0.804	0.289	0.111	0.066	0.756	0.241
...	...	...	...	...	...	...	...
101	0.449	0.875	0.523	0.083	0.076	0.920	0.487
102	0.102	0.000	0.022	0.000	0.000	0.000	0.017
103	0.409	0.875	0.482	0.306	0.259	0.914	0.443
104	0.427	0.804	0.474	0.056	0.048	0.836	0.437
105	0.462	0.911	0.551	0.167	0.154	0.931	0.500

106 rows × 7 columns

In [8]:

```
target=data.iloc[:,7] #Segregating the target/class variables
target
```

Out[8]:

```
0      1
1      1
2      1
3      1
4      1
      ..
101    0
102    0
103    0
104    0
105    0
Name: class, Length: 106, dtype: int64
```

In [9]:

```
predictors_train, predictors_test, target_train, target_test = train_test_spli
#Holdout of data
```

In [10]:

```
gnb=GaussianNB()
```

In [11]:

```
#first train model/classifier with input daraset
model=gnb.fit(predictors_train,target_train)
```

In [12]:

```
#Make prediction using the trained model
prediction=model.predict(predictors_test)
print("Predicted class:", prediction)
```

```
Predicted class: [0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0
0]
```

In [13]:
```
probs = model.predict_proba(predictors_test)
print("Posterior probabilities:", probs.shape)
print(probs)
print("Class Priors:", model.class_prior_)
```

```
Posterior probabilities: (32, 2)
[[9.85612083e-01 1.43879172e-02]
 [9.99944285e-01 5.57154223e-05]
 [9.73891114e-01 2.61088858e-02]
 [9.40825699e-01 5.91743007e-02]
 [3.80104260e-01 6.19895740e-01]
 [9.93291639e-01 6.70836139e-03]
 [7.90565072e-02 9.20943493e-01]
 [4.39004730e-05 9.99956100e-01]
 [2.77345625e-01 7.22654375e-01]
 [9.99999999e-01 1.21289154e-09]
 [9.98989311e-01 1.01068872e-03]
 [8.17393837e-01 1.82606163e-01]
 [9.84609895e-01 1.53901053e-02]
 [7.52693268e-01 2.47306732e-01]
 [1.50784637e-04 9.99849215e-01]
 [9.90066613e-01 9.93338661e-03]
 [9.98864400e-01 1.13560030e-03]
 [9.99332648e-01 6.67351759e-04]
 [9.99993183e-01 6.81675774e-06]
 [9.37395577e-01 6.26044231e-02]
 [9.70769057e-01 2.92309430e-02]
 [9.78957674e-01 2.10423257e-02]
 [3.26049352e-01 6.73950648e-01]
 [9.59361247e-01 4.06387528e-02]
 [6.02637122e-02 9.39736288e-01]
 [9.99985914e-01 1.40859383e-05]
 [9.76697221e-01 2.33027795e-02]
 [9.99756266e-01 2.43733635e-04]
 [1.00000000e+00 1.06059500e-10]
 [5.28392448e-05 9.99947161e-01]
 [9.52751545e-01 4.72484546e-02]
 [9.92092170e-01 7.90783039e-03]]
Class Priors: [0.81081081 0.18918919]
```

In [14]:
```
accuracy_score(target_test, prediction, normalize = True)
```

Out[14]:
```
0.90625
```

In [15]:
```
#Another problem (optional)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets
from sklearn import metrics

# Load sample data
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
#print(X)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando

# Create and train model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Take a sample
sample = X[0].reshape(1, -1)
#print(sample)
# Predicted class
pred = model.predict(sample)
print("Predicted class:", pred)

# Posterior probabilities P(class | features)
probs = model.predict_proba(sample)
print("Posterior probabilities:", probs)
# Evaluate
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Predicted class: [0]
Posterior probabilities: [[1.00000000e+00 7.82732978e-17 1.66528708e-24]]
Accuracy: 0.9777777777777777
```

In [ ]:

```

```

CHAPTER 7

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Define Training Data# ------------------------------
X = np.array([[1,2],[2,3],[3,3],[6,5],[7,7],[8,6]]) # Feature values (2D point
y = np.array([0,0,0,1,1,1]) # Corresponding class labels (0 or 1)

#Step 2: Create and Train Model#
k = 3 # Number of nearest neighbors

knn = KNeighborsClassifier(n_neighbors=k) # Initialize KNN classifier
knn.fit(X, y) # Train the classifier using the dataset (X, y)

# Step 3: Predict for a New Sample
sample = np.array([[6,5]]) # New data point to classify
predicted_class = knn.predict(sample)
print("Predicted class for [6,5]:", predicted_class)
```

```
Predicted class for [6,5]: [1]
```

In [2]:

```
import pandas as pd   # pandas: library for data manipulation and analysis, pr
import numpy as np    # numpy: library for numerical computing, arrays, and ma

# Load CSV
df = pd.read_csv("btissue.csv")  # pd.read_csv(): reads a CSV file and loads i
print("\n--- Dataset Preview ---")
print(df.head())  # df.head(): displays the first 5 rows of the DataFrame to q

# Basic info
print("\n--- Dataset Info ---")
print(df.info()) # df.info(): shows summary of DataFrame including column name

# Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())  # df.isnull(): returns DataFrame of True/False for N
                           # .sum(): sums True values column-wise to show numb

# Summary statistics
print("\n--- Summary Statistics ---")
print(df.describe())
```

```
--- Dataset Preview ---
           I0     PA500       HFS          DA          Area       A/DA  \
0  524.794072  0.187448  0.032114  228.800228   6843.598481  29.910803   
1  330.000000  0.226893  0.265290  121.154201   3163.239472  26.109202   
2  551.879287  0.232478  0.063530  264.804935  11888.391830  44.894903   
3  380.000000  0.240855  0.286234  137.640111   5402.171180  39.248524   
4  362.831266  0.200713  0.244346  124.912559   3290.462446  26.342127   

      Max IP          DR           P class  
0  60.204880  220.737212  556.828334   car  
1  69.717361   99.084964  400.225776   car  
2  77.793297  253.785300  656.769449   car  
3  88.758446  105.198568  493.701814   car  
4  69.389389  103.866552  424.796503   car  

--- Dataset Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106 entries, 0 to 105
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   I0      106 non-null    float64
 1   PA500   106 non-null    float64
 2   HFS     106 non-null    float64
 3   DA      106 non-null    float64
 4   Area    106 non-null    float64
 5   A/DA    106 non-null    float64
 6   Max IP  106 non-null    float64
 7   DR      106 non-null    float64
 8   P       106 non-null    float64
 9   class   106 non-null    object 
dtypes: float64(9), object(1)
memory usage: 8.4+ KB
None

Missing values per column:
I0        0
PA500     0
HFS       0
DA        0
Area      0
A/DA      0
Max IP    0
DR        0
P         0
class     0
dtype: int64

--- Summary Statistics ---
               I0       PA500         HFS          DA          Area  \
count   106.000000  106.000000  106.000000  106.000000    106.000000   
mean    784.251618    0.120133    0.114691  190.568642   7335.155161   
std     753.950075    0.068596    0.101347  190.801448  18580.314212   
min     103.000000    0.012392   -0.066323   19.647670     70.426239   
```

```
25%     250.000000    0.067413    0.043982    53.845470     409.647141
50%     384.936489    0.105418    0.086568   120.777303    2219.581163
75%    1487.989626    0.169602    0.166504   255.334809    7615.204968
max    2800.000000    0.358316    0.467748  1063.441427  174480.476200

             A/DA      Max IP          DR            P
count  106.000000  106.000000  106.000000   106.000000
mean    23.473784   75.381258  166.710575   810.638127
std     23.354672   81.345838  181.309580   763.019135
min      1.595742    7.968783   -9.257696   124.978561
25%      8.180321   26.893773   41.781258   270.215238
50%     16.133657   44.216040   97.832557   454.108153
75%     30.953294   83.671755  232.990070  1301.559438
max    164.071543  436.099640  977.552367  2896.582483
```

In [3]:

```python
X = df.iloc[:, :-1].values  # Features

y = df.iloc[:, -1].values   # Target

print("\nFeature shape:", X.shape)

print("Target shape:", y.shape)
```

```
Feature shape: (106, 9)
Target shape: (106,)
```

In [4]:

```python
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3,rando

print("\nTrain shape:", X_train.shape)
# X_train.shape: number of rows and columns in training features
print("Test shape:", X_test.shape)
# X_test.shape: number of rows and columns in testing features
```

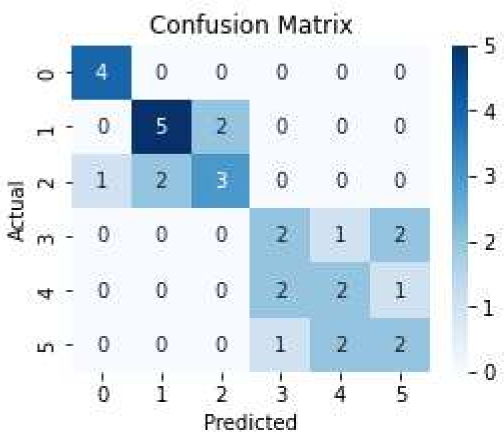
```
Train shape: (74, 9)
Test shape: (32, 9)
```

In [5]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_


knn = KNeighborsClassifier(n_neighbors=3)

model=knn.fit(X_train, y_train)

y_pred = model.predict(X_test)


print("\n--- KNN Results ---")
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='macro', zero_divi
    # Average precision across all classes
print("Recall:", recall_score(y_test, y_pred, average='macro', zero_division=0
    # Average recall across all classes
print("F1-score:", f1_score(y_test, y_pred, average='macro', zero_division=0))
    # Average F1-score across all classes


    # Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
    # Matrix showing counts of actual vs predicted labels
plt.figure(figsize=(4,3))
    # Set figure size
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    # Heatmap with annotation, integer format, blue colormap
plt.title("Confusion Matrix")
    # Set title dynamically based on model
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
    # Display confusion matrix
```

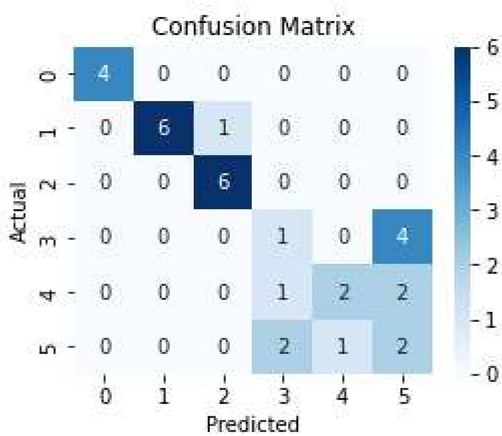
```
--- KNN Results ---
Accuracy: 0.5625
Precision: 0.5523809523809523
Recall: 0.569047619047619
F1-score: 0.5581048581048581
```



In [6]:
```
# Decision Tree
# Libraries: sklearn.tree
from sklearn.tree import DecisionTreeClassifier
# DecisionTreeClassifier: implements a decision tree for classification tasks

dt = DecisionTreeClassifier(max_depth=5, random_state=42)

model=dt.fit(X_train, y_train)
# .fit(): trains the decision tree on the training data (X_train, y_train)

y_pred_dt = model.predict(X_test)
# .predict(): predicts labels for the test data
```

```
print("\n--- Decision Tree Results ---")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))

print("Feature importances:", dt.feature_importances_)
print("Precision:", precision_score(y_test, y_pred_dt, average='macro', zero_d
    # Average precision across all classes
print("Recall:", recall_score(y_test, y_pred_dt, average='macro', zero_divisio
    # Average recall across all classes
print("F1-score:", f1_score(y_test, y_pred_dt, average='macro', zero_division=
    # Average F1-score across all classes


    # Confusion Matrix
cm = confusion_matrix(y_test, y_pred_dt)
    # Matrix showing counts of actual vs predicted labels
plt.figure(figsize=(4,3))
    # Set figure size
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    # Heatmap with annotation, integer format, blue colormap
plt.title("Confusion Matrix")
    # Set title dynamically based on model
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
    # Display confusion matrix
```

```
--- Decision Tree Results ---
Accuracy: 0.65625
Feature importances: [0.01813106 0.18496225 0.         0.11103646 0.03867959
0.02175727
 0.06962327 0.17091545 0.38489465]
Precision: 0.6706349206349206
Recall: 0.6428571428571429
F1-score: 0.6460113960113959
```

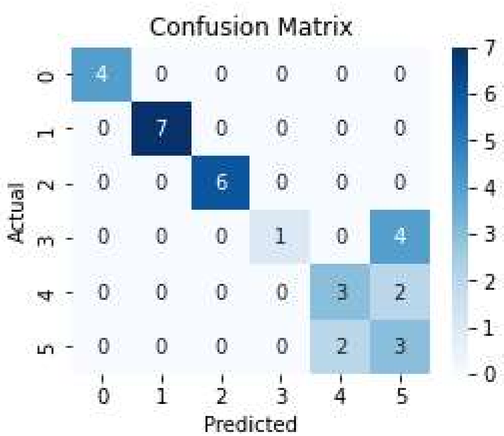


In [7]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
# First train model / classifier with the input dataset (training data part of
model = rf.fit(X_train, y_train)
# Make prediction using the trained model
```

```
y_pred_rf = model.predict(X_test)
print("\n--- RF Results ---")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf, average='macro', zero_d
    # Average precision across all classes
print("Recall:", recall_score(y_test, y_pred_rf, average='macro', zero_divisio
    # Average recall across all classes
print("F1-score:", f1_score(y_test, y_pred_rf, average='macro', zero_division=
    # Average F1-score across all classes


    # Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
    # Matrix showing counts of actual vs predicted labels
plt.figure(figsize=(4,3))
    # Set figure size
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    # Heatmap with annotation, integer format, blue colormap
plt.title("Confusion Matrix")
    # Set title dynamically based on model
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
    # Display confusion matrix
```

```
--- RF Results ---
Accuracy: 0.75
Accuracy: 0.75
Precision: 0.8222222222222221
Recall: 0.7333333333333334
F1-score: 0.7269841269841271
```

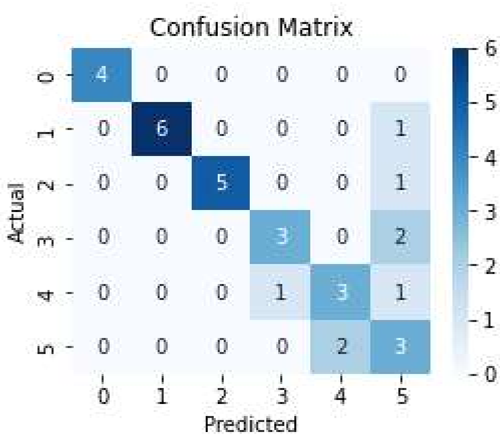


In [8]:

```
from sklearn.svm import SVC


svm = SVC(kernel='linear', random_state=42)

model=svm.fit(X_train, y_train)

y_pred_svm = model.predict(X_test)
```

```python
print("\n--- SVM Results ---")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm, average='macro', zero_
    # Average precision across all classes
print("Recall:", recall_score(y_test, y_pred_svm, average='macro', zero_divisi
    # Average recall across all classes
print("F1-score:", f1_score(y_test, y_pred_svm, average='macro', zero_divisior
    # Average F1-score across all classes


    # Confusion Matrix
cm = confusion_matrix(y_test, y_pred_svm)
    # Matrix showing counts of actual vs predicted labels
plt.figure(figsize=(4,3))
    # Set figure size
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    # Heatmap with annotation, integer format, blue colormap
plt.title("Confusion Matrix")
    # Set title dynamically based on model
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
    # Display confusion matrix
```

```
--- SVM Results ---
Accuracy: 0.75
Accuracy: 0.75
Precision: 0.7875
Recall: 0.7484126984126984
F1-score: 0.76006216006216
```



In [ ]:

In [ ]:

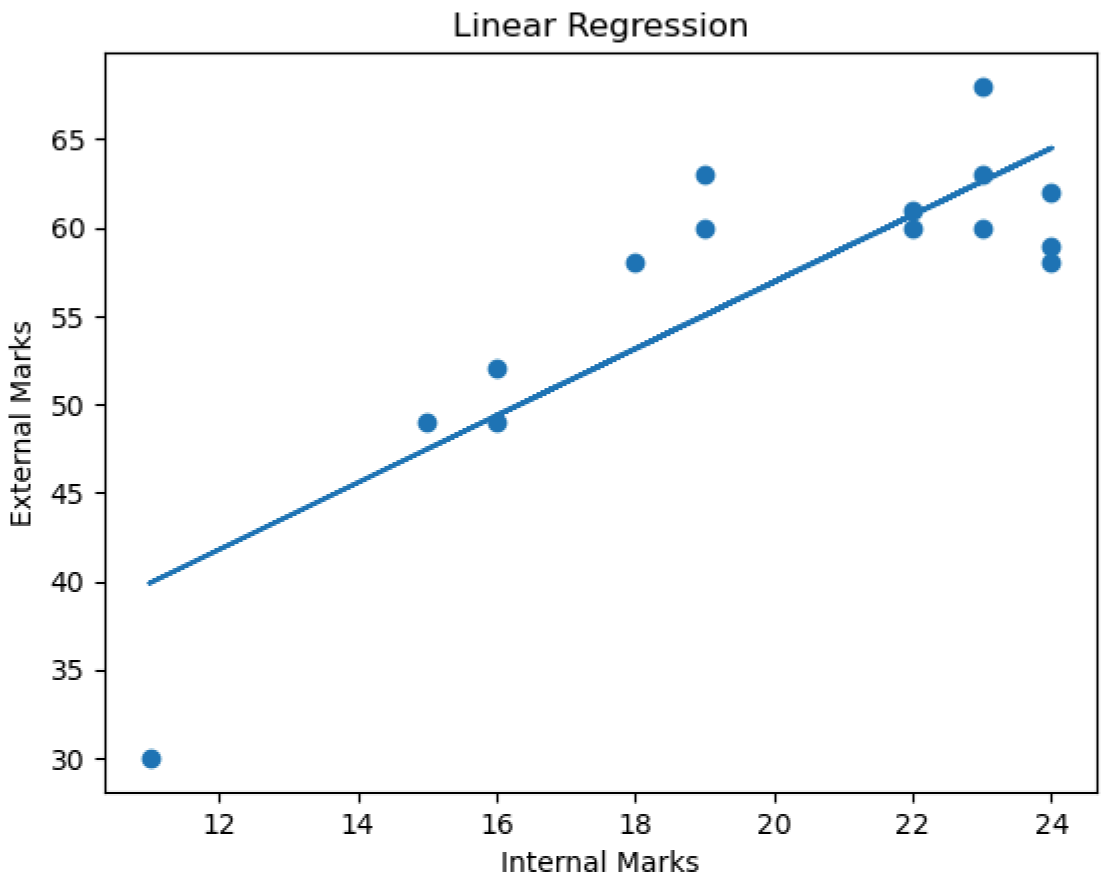
In [ ]:

CHAPTER 8

1. Linear Regression

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Dataset
X = np.array([15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23]).res
Y = np.array([49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68])

# Fit model
lr = LinearRegression().fit(X, Y)

# Plot
plt.scatter(X, Y)
plt.plot(X, lr.predict(X))
plt.xlabel("Internal Marks")
plt.ylabel("External Marks")
plt.title("Linear Regression")
plt.show()

# Output
print("Intercept:", lr.intercept_)
print("Slope:", lr.coef_[0])
print("R² Score:", lr.score(X, Y))
```




```
Intercept: 19.047297297297284
Slope: 1.8939482961222098
R² Score: 0.7088287858527735
```

2. Multiple Linear Regression There are 2 more features: number of study hours & assignment score (random simulated but fixed for reproducibility, based on same samples)

In [2]:
```
import numpy as np
from sklearn.linear_model import LinearRegression

# Dataset
X1 = np.array([15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23])
Y  = np.array([49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68])

np.random.seed(0)
X2 = np.random.randint(2,10,15)   # Study hours
X3 = np.random.randint(20,40,15)  # Assignment score

# Prepare multi-feature input
X_multi = np.column_stack([X1, X2, X3])

# Fit model
mlr = LinearRegression().fit(X_multi, Y)

# Output
```

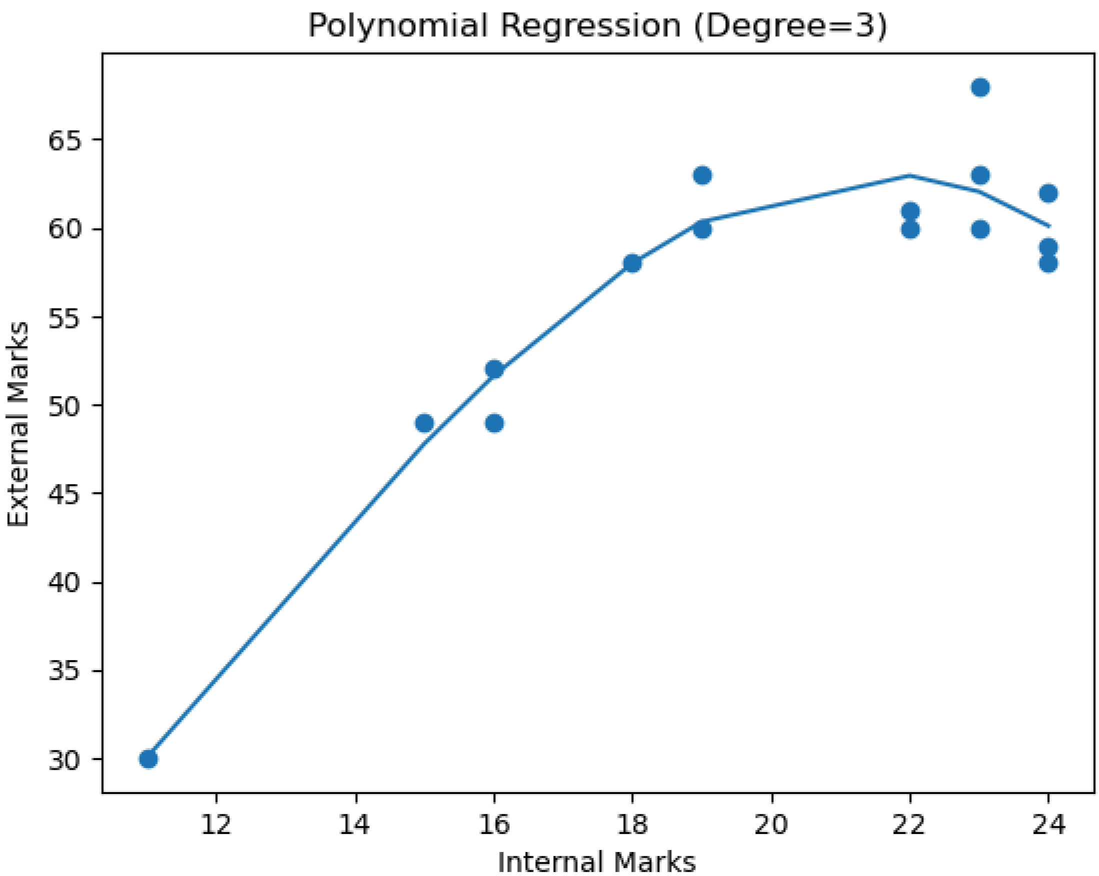
```
print("Intercept:", mlr.intercept_)
print("Coefficients:", mlr.coef_)
print("R² Score:", mlr.score(X_multi, Y))
```

```
Intercept: 21.66988785796515
Coefficients: [ 1.91242802  0.7935008  -0.24952654]
R² Score: 0.7749329840692462
```

3. Polynomial Regression (degree = 3)

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Dataset
X = np.array([15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23]).res
Y = np.array([49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68])

# Transform
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)

# Fit
pr = LinearRegression().fit(X_poly, Y)

# Plot
x_sorted = X[np.argsort(X.squeeze())]
y_sorted = pr.predict(X_poly)[np.argsort(X.squeeze())]

plt.scatter(X, Y)
plt.plot(x_sorted, y_sorted)
plt.xlabel("Internal Marks")
plt.ylabel("External Marks")
plt.title("Polynomial Regression (Degree=3)")
plt.show()

# Output
print("Intercept:", pr.intercept_)
print("Polynomial Coefficients:", pr.coef_)
print("R² Score:", pr.score(X_poly, Y))
```

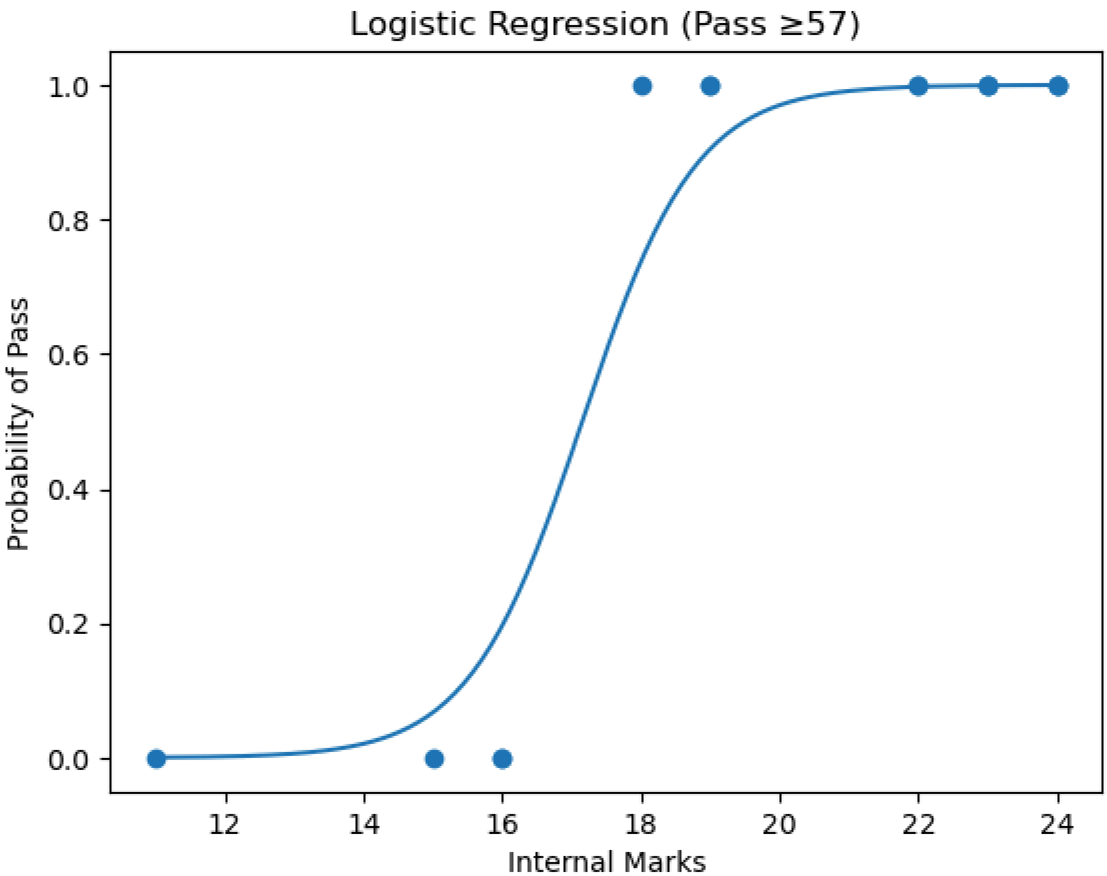



```
Intercept: 0.8055959038261022
Polynomial Coefficients: [ 0.         -1.09287736  0.5033446  -0.01478554]
R² Score: 0.9320732674279738
```

4. Logistic Regression Converting the same external marks into a pass/fail class using threshold ≥ 57 (as a probability regression application)

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Dataset
X = np.array([15, 23, 18, 23, 24, 22, 22, 19, 19, 16, 24, 11, 24, 16, 23]).res
Y = np.array([49, 63, 58, 60, 58, 61, 60, 63, 60, 52, 62, 30, 59, 49, 68])

# Convert to binary class
y_bin = (Y >= 57).astype(int)

# Fit
logr = LogisticRegression().fit(X, y_bin)

# Plot sigmoid curve
x_range = np.linspace(X.min(), X.max(), 200).reshape(-1,1)
plt.scatter(X, y_bin)
plt.plot(x_range, logr.predict_proba(x_range)[:,1])
plt.xlabel("Internal Marks")
```

```
plt.ylabel("Probability of Pass")
plt.title("Logistic Regression (Pass ≥57)")
plt.show()

# Output
print("Accuracy:", logr.score(X, y_bin))
print("Predicted Probabilities:", logr.predict_proba(X))
```




```
Accuracy: 1.0
Predicted Probabilities: [[9.32645472e-01 6.73545278e-02]
 [7.96275346e-04 9.99203725e-01]
 [2.62502138e-01 7.37497862e-01]
 [7.96275346e-04 9.99203725e-01]
 [2.35133509e-04 9.99764866e-01]
 [2.69296481e-03 9.97307035e-01]
 [2.69296481e-03 9.97307035e-01]
 [9.50602938e-02 9.04939706e-01]
 [9.50602938e-02 9.04939706e-01]
 [8.03403239e-01 1.96596761e-01]
 [2.35133509e-04 9.99764866e-01]
 [9.99452427e-01 5.47573036e-04]
 [2.35133509e-04 9.99764866e-01]
 [8.03403239e-01 1.96596761e-01]
 [7.96275346e-04 9.99203725e-01]]
```