SGPC's
Guru Nanak Institute of Management Studies
(Management Institute of G N Khalsa College), Matunga, Mumbai – 400 019
INDEX

Subject: MCAL21 - Artificial Intelligence and Machine Learning

SR.NO	PRACTICAL PROBLEM STATEMENT	DATE
1	Implementation of Logic programming using PROLOG: BFS for tic-tac-toe problem	14/5/2024
2	Introduction to Python Programming: Learn the different libraries – a) NumPy, b) Pandas, c) Matplotlib, d) Scikit Learn.	16/3/2024, 30/03/2024
3	Implementation of a) Linear Regression, b) Logistic regression, c) KNN- classification	19/04/2024, 07/05/2024
4	Implementation of dimensionality reduction techniques: a) Features Extraction and Selection, b) Normalization, c) Transformation, d) Principal Components Analysis.	30/03/2024
5	Implementation of clustering algorithm: a) K-Means clustering b) K-medoid clustering	14/05/2024
6	Implementation of Classifying data using a) Support Vector Machines (SVMs).	07/05/2024
7	Implementation of Bagging Algorithm: a) Decision Tree, b) Random Forest.	14/05/2024
8	Implementation of Boosting Algorithms: a) AdaBoost, b) Stochastic Gradient Boosting, c) Voting Ensemble.	15/05/2024
9	Steps for Deployment of Machine Learning Models.	15/05/2024

Practical 1

Aim: Implementation of Logic programming using LISP /PROLOG: BFS for tic-tac-toe problems.

Theory:

1. Representing the Tic-Tac-Toe Game State:
   Represent the Tic-Tac-Toe board as a list of lists or a 1-dimensional list where each element represents a cell on the board.
   Use symbols like 'x', 'o', and 'e' (for empty) to represent the contents of each cell.
2. Defining Legal Moves:
   Define the rules for legal moves, such as placing a symbol ('x' or 'o') in an empty cell.
   Check if a move is legal by verifying if the target cell is empty.
3. Generating Successor States:
   Generate successor states by applying legal moves to the current state.
   Store the successor states in a queue for BFS traversal.
4. BFS Search Algorithm:
   Use a queue to store states to be explored.
   Start with the initial state and iteratively explore successor states.
   Stop when a winning state is found or when all states have been explored.
5. Checking for Winning State:
   Check if a player has won by examining rows, columns, and diagonals for a complete sequence of their symbol.
6. Putting It All Together:
   Define a function to start the game and invoke BFS to find the winning state.

Code:

```
% A Tic-Tac-Toe program in Prolog
% To play a game with the computer, type
% playo.
% To watch the computer play a game with itself, type
% selfgame.
% original at
https://courses.cs.washington.edu/courses/cse341/03sp/slides/PrologEx/tictactoe.pl.txt
```

```
% Predicates that define the winning conditions:
win(Board, Player) :- rowwin(Board, Player).
win(Board, Player) :- colwin(Board, Player).
win(Board, Player) :- diagwin(Board, Player).
rowwin(Board, Player) :- Board = [Player,Player,Player,_,_,_,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,Player,Player,Player,_,_,_].
rowwin(Board, Player) :- Board = [_,_,_,_,_,_,Player,Player,Player].
colwin(Board, Player) :- Board = [Player,_,_,Player,_,_,Player,_,_].
colwin(Board, Player) :- Board = [_,Player,_,_,Player,_,_,Player,_].
colwin(Board, Player) :- Board = [_,_,Player,_,_,Player,_,_,Player].
diagwin(Board, Player) :- Board = [Player,_,_,_,Player,_,_,_,Player].
diagwin(Board, Player) :- Board = [_,_,Player,_,Player,_,Player,_,_].
% Helping predicate for alternating play in a "self" game:
other(x,o).
other(o,x).
game(Board, Player) :- win(Board, Player), !, write([player, Player, wins]).
game(Board, Player) :-
  other(Player,Otherplayer),
  move(Board,Player,Newboard),
  !,
  display(Newboard),
  game(Newboard,Otherplayer).
move([b,B,C,D,E,F,G,H,I], Player, [Player,B,C,D,E,F,G,H,I]).
move([A,b,C,D,E,F,G,H,I], Player, [A,Player,C,D,E,F,G,H,I]).
move([A,B,b,D,E,F,G,H,I], Player, [A,B,Player,D,E,F,G,H,I]).
move([A,B,C,b,E,F,G,H,I], Player, [A,B,C,Player,E,F,G,H,I]).
move([A,B,C,D,b,F,G,H,I], Player, [A,B,C,D,Player,F,G,H,I]).
move([A,B,C,D,E,b,G,H,I], Player, [A,B,C,D,E,Player,G,H,I]).
move([A,B,C,D,E,F,b,H,I], Player, [A,B,C,D,E,F,Player,H,I]).
move([A,B,C,D,E,F,G,b,I], Player, [A,B,C,D,E,F,G,Player,I]).
move([A,B,C,D,E,F,G,H,b], Player, [A,B,C,D,E,F,G,H,Player]).
display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,
 write([G,H,I]),nl,nl.
selfgame :- game([b,b,b,b,b,b,b,b,b],x).
% Predicates to support playing a game with the user:
x_can_win_in_one(Board) :- move(Board, x, Newboard), win(Newboard, x).
% The predicate orespond generates the computer's (playing o) reponse
% from the current Board.

orespond(Board,Newboard) :-
```

```
  move(Board, o, Newboard),
  win(Newboard, o),
  !.
orespond(Board,Newboard) :-
  move(Board, o, Newboard),
  not(x_can_win_in_one(Newboard)).
orespond(Board,Newboard) :-
  move(Board, o, Newboard).
orespond(Board,Newboard) :-
  not(member(b,Board)),
  !,
  write('Cats game!'), nl,
  Newboard = Board.
% The following translates from an integer description
% of x's move to a board transformation.
xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).
xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).
xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).
xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).
xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).
xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).
xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).
xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).
xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).
xmove(Board, _, Board) :- write('Illegal move.'), nl.
% The 0-place predicate playo starts a game with the user.
playo :- explain, playfrom([b,b,b,b,b,b,b,b,b]).
explain :-
  write('You play X by entering integer positions followed by a period.'),
  nl,
  display([1,2,3,4,5,6,7,8,9]).
playfrom(Board) :- win(Board, x), write('You win!').
playfrom(Board) :- win(Board, o), write('I win!').
playfrom(Board) :- read(N),
  xmove(Board, N, Newboard),
  display(Newboard),
  orespond(Newboard, Newnewboard),
  display(Newnewboard),
  playfrom(Newnewboard).
```

Output:

```
% c:/users/letsc/onedrive/desktop/game compiled 0.00 sec, -4 clauses
?- playo.
You play X by entering integer positions followed by a period.
[1,2,3]
[4,5,6]
[7,8,9]

|: 5.
[b,b,b]
[b,x,b]
[b,b,b]

[o,b,b]
[b,x,b]
[b,b,b]

|: 9
|: .
[o,b,b]
[b,x,b]
[b,b,x]

[o,o,b]
[b,x,b]
[b,b,x]

|: 3.
[o,o,x]
[b,x,b]
[b,b,x]

[o,o,x]
[o,x,b]
[b,b,x]

|: 7.
[o,o,x]
[o,x,b]
[x,b,x]

[o,o,x]
[o,x,o]
[x,b,x]

You win!
true .
```

Practical 2

Aim: Introduction to Python Programming: Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn.

Theory:

A. NumPy (Numerical Python):
   a. NumPy is a fundamental package for scientific computing with Python.
   b. It provides support for arrays, which are the core data structure for numerical computations.
   c. NumPy arrays are more efficient than Python lists for numerical operations and allow for vectorized operations.

B. Pandas:
   a. Pandas is a powerful library for data manipulation and analysis.
   b. It provides data structures like DataFrame and Series that are designed for working with structured or tabular data.
   c. Pandas simplifies tasks such as reading and writing data, data cleaning, filtering, grouping, and data aggregation.

C. Matplotlib:
   a. Matplotlib is a plotting library for creating static, interactive, and animated visualizations in Python.
   b. It provides a MATLAB-like interface for creating plots and graphs.
   c. Matplotlib supports a wide variety of plot types, including line plots, scatter plots, bar plots, histograms, and 3D plots.
   d. It's highly customizable, allowing users to control every aspect of their plots.

D. Scikit-learn:
   a. Scikit-learn is a machine learning library that provides simple and efficient tools for data mining and data analysis.
   b. It includes various algorithms for classification, regression, clustering, dimensionality reduction, and model selection.
   c. Scikit-learn is built on top of NumPy, SciPy, and Matplotlib, making it easy to integrate into existing Python workflows.

Code & Output:
2.A Numpy

```
!pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
```

```
import numpy as np
List1=[1,2,5,4]
List1
```

```
[1, 2, 5, 4]
```

```
Array1=np.array(List1)
Array1
```

```
array([1, 2, 5, 4])
```

```
List2=[[1,2,5,4],[7,8,9,5]]
Array2=np.array(List2)
print(Array2)
```

```
[[1 2 5 4]
 [7 8 9 5]]
```

```
toyprices=[5,8,3,6]
for i in range(len(toyprices)):
 toyprices[i]-=2
print(toyprices)
```

```
[3, 6, 1, 4]
```

```
Toyprices=np.array([1,2,3,4,5])
print(Toyprices-2)
```

```
[-1  0  1  2  3]
```

```
a=np.array([1,2,3,4])
b=np.array([(1.5,2,3,6),(8,3,6,7)],dtype=float)
```

```
c=np.array([(1.5,2,3,6),(8,3,6,7),(4,1.3,5,7)],dtype=float)
print(a)
print(b)
print(c)
```

```
[1 2 3 4]
[[1.5 2.  3.  6. ]
 [8.  3.  6.  7. ]]
[[1.5 2.  3.  6. ]
 [8.  3.  6.  7. ]
 [4.  1.3 5.  7. ]]
```

```
b.shape
```

```
(2, 4)
```

```
c.size
```

```
12
```

```
c.dtype
```

```
dtype('float64')
```

```
print(np.zeros((3,4)))
print(np.ones((2,3,4),dtype=np.int16))
d = np.arange(10,25,5)
print(d)
print(np.linspace(0,2,9))
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
[10 15 20]
[0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
```

```
import pandas as pd
Mydict={'red':2000,'blue':1000,'yellow':500,'orange':1000}
Myseries=pd.Series(Mydict)
Myseries
```

```
red       2000
blue      1000
yellow     500
orange    1000
dtype: int64
```

```
Colors=['red','yellow','orange','blue','green']
Myseries=pd.Series(Mydict,index=Colors)
Myseries
```

```
red       2000.0
yellow     500.0
orange    1000.0
blue      1000.0
green        NaN
dtype: float64
```

```
Mydict2={'red':200,'yellow':1000,'balck':700}
Myseries2=pd.Series(Mydict2)
Myseries2
```

```
red        200
yellow    1000
balck      700
dtype: int64
```

```
Myseries2 +Myseries
```

```
balck        NaN
blue         NaN
green        NaN
orange       NaN
red       2200.0
yellow    1500.0
dtype: float64
```

```
import numpy as np
Data={'color':['blue','green','yellow','red','white'],
'object':['ball','pen','pencil','paper','mug'],
'price':[1,2,3,4,5]}
Frame=pd.DataFrame(Data)
Frame
```

	color	object	price
0	blue	ball	1
1	green	pen	2
2	yellow	pencil	3
3	red	paper	4
4	white	mug	5

```
Frame2=pd.DataFrame(np.arange(16).reshape((4,4)),
index=['red','blue','yellow','white'],
columns=['ball','pen','pencil','paper'])
Frame2
```

	ball	pen	pencil	paper
red	0	1	2	3
blue	4	5	6	7
yellow	8	9	10	11
white	12	13	14	15

```
Nest={'red':{2012:22,2013:33},
'white':{2011:13,2012:22,2013:16},
'blue':{2011:17,2012:27,2013:18}}
```

```
{'red': {2012: 22, 2013: 33},
 'white': {2011: 13, 2012: 22, 2013: 16},
 'blue': {2011: 17, 2012: 27, 2013: 18}}
```

2.B Pandas

```
import numpy as np
import pandas as pd
df = pd.DataFrame(np.random.randn(8, 4), columns = ['A', 'B', 'C', 'D'])
df
```

	A	B	C	D
0	-0.197607	-0.824076	-0.306035	-0.499815
1	0.725671	0.393881	-1.024236	-1.567358
2	1.820249	-1.960516	-1.541196	1.445468
3	0.044291	-0.437421	0.634758	0.892199
4	1.192635	-1.408094	-0.724867	1.001425
5	-0.236830	-0.817671	-0.165732	0.587316
6	-0.133015	0.725248	0.391606	-1.497607
7	0.620363	0.489333	-1.701170	1.869447

```
s = df.iloc[3]
s
```

```
A    0.044291
B   -0.437421
C    0.634758
D    0.892199
Name: 3, dtype: float64
```

```
pd.concat([df, pd.DataFrame([s])], ignore_index=True)
```

	A	B	C	D
0	-0.197607	-0.824076	-0.306035	-0.499815
1	0.725671	0.393881	-1.024236	-1.567358
2	1.820249	-1.960516	-1.541196	1.445468
3	0.044291	-0.437421	0.634758	0.892199
4	1.192635	-1.408094	-0.724867	1.001425
5	-0.236830	-0.817671	-0.165732	0.587316
6	-0.133015	0.725248	0.391606	-1.497607
7	0.620363	0.489333	-1.701170	1.869447
8	0.044291	-0.437421	0.634758	0.892199

```
df = pd.DataFrame({
   'brand' : ['Yum Yum', 'Yum Yum', 'Indomie' , 'Indomie', 'Indomie'],
   'style': ['cup','cup','cup','pack', 'pack'],
   'rating' :[4,4,3.5,5,5]})
df
```

	brand	style	rating
0	Yum Yum	cup	4.0
1	Yum Yum	cup	4.0
2	Indomie	cup	3.5
3	Indomie	pack	5.0
4	Indomie	pack	5.0

```
import numpy as np
```

```
import pandas as pd
data1=pd.read_csv("C:/Users/Student/Desktop/data1.csv")
data2=pd.read_csv("C:/Users/Student/Desktop/data2.csv")
data1
```

	x1	x2
0	a	11
1	b	1
2	d	100

```
data2
```

	x1	x2
0	a	22
1	b	Nan
2	c	55

```
pd.merge(data1,data2,how='left',on='x1')
pd.merge(data2,data1,how='left',on='x1')
```

	x1	x2_x	x2_y
0	a	22	11.0
1	b	Nan	1.0
2	c	55	NaN

```
pd.merge(data1,data2,how='right',on='x1')
```

	x1	x2_x	x2_y
0	a	11.0	22
1	b	1.0	Nan
2	c	NaN	55

```
pd.merge(data1,data2,how='outer',on='x1')
```

	x1	x2_x	x2_y
0	a	11.0	22
1	b	1.0	Nan
2	d	100.0	NaN
3	c	NaN	55

```
import pandas as pd
data={'A':[1,2,3,1,2],
   'B':['a','b','c','a','b']}
df=pd.DataFrame(data)
df
```

	A	B
0	1	a
1	2	b
2	3	c
3	1	a
4	2	b

```
duplicate_rows=df.duplicated()
```

```
print("Duplicate rows: \n",duplicate_rows)
```

```
Duplicate rows:
 0    False
1    False
2    False
3     True
4     True
dtype: bool
```

```
data={'Name':['Harshal','Shivesh','Vinit','Rohit','Harshal'],
    'Age':[23,22,21,22,23],
   'Salary':[50000,60000,70000,55000,65000]}
df=pd.DataFrame(data)
grouped=df.groupby('Name')
for name,group in grouped:
   print(group)
   print()
```

```
      Name  Age  Salary
0  Harshal   23   50000
4  Harshal   23   65000

    Name  Age  Salary
3  Rohit   22   55000

      Name  Age  Salary
1  Shivesh   22   60000

    Name  Age  Salary
2  Vinit   21   70000
```

```
grouped=df.groupby('Name')
mean_salary= grouped['Salary'].mean()
mean_salary
```

```
Name
Harshal    57500.0
Rohit      55000.0
Shivesh    60000.0
Vinit      70000.0
Name: Salary, dtype: float64
```

```
grouped_multiple=df.groupby(['Name','Age'])
for name,group in grouped_multiple:
    print(group)
    print()
```

```
      Name  Age  Salary
0  Harshal   23   50000
4  Harshal   23   65000

    Name  Age  Salary
3  Rohit   22   55000

      Name  Age  Salary
1  Shivesh   22   60000

    Name  Age  Salary
2  Vinit   21   70000
```

```
aggregated_data=grouped['Salary'].agg(['mean','sum'])
print(aggregated_data)
```

```
            mean     sum
Name
Harshal  57500.0  115000
Rohit    55000.0   55000
Shivesh  60000.0   60000
Vinit    70000.0   70000
```

```
def salary_range(series):
    return series.max()-series.min()
salary_range_per_group=grouped['Salary'].agg(salary_range)
print(salary_range_per_group)
```

```
Name
Harshal    15000
Rohit          0
Shivesh        0
Vinit          0
Name: Salary, dtype: int64
```

```
z_score=grouped['Salary'].transform(lambda x:(x-x.mean())/x.std())
df['Salary_ZScore'] =z_score
print(z_score)
```

```
0   -0.707107
1         NaN
2         NaN
3         NaN
4    0.707107
Name: Salary, dtype: float64
```

```
data={'A':[1,2,None,4],
    'B':[None,5,6,7],
    'C':[8,9,10,11]}
df=pd.DataFrame(data)
print(df.isna())
```

```
       A      B      C
0  False   True  False
1  False  False  False
2   True  False  False
3  False  False  False
```

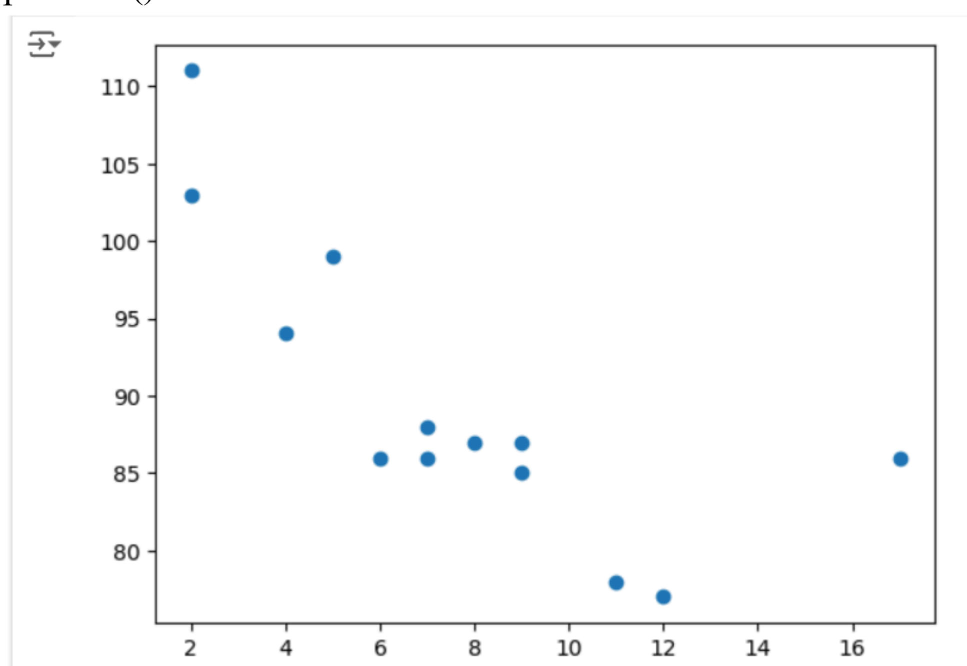
```
df_cleaned_rows = df.dropna()
df_cleaned_columns = df.dropna(axis = 1)
df_cleaned_rows
df_cleaned_columns
df_filled = df.fillna(df.mean())
df_flled_specfic = df.fillna(value = 0)
```

```
df_flled_specfic
df_filled_ffill = df.fillna(method = 'ffill')
df_filled_bfill = df.fillna(method = 'bfill')
df_filled_ffill
Df_filled_bfill
```

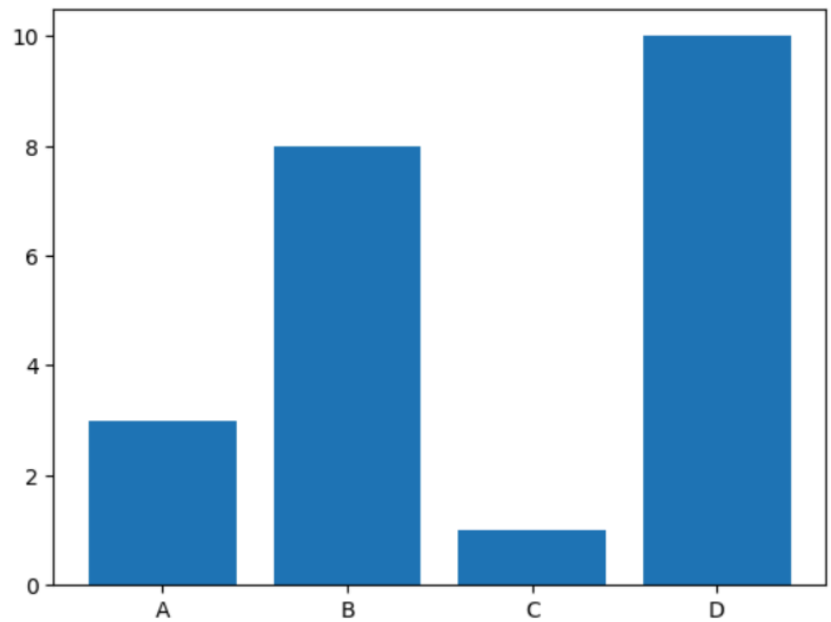
	A	B	C
0	1.0	5.0	8
1	2.0	5.0	9
2	4.0	6.0	10
3	4.0	7.0	11

2.C Matplotlib

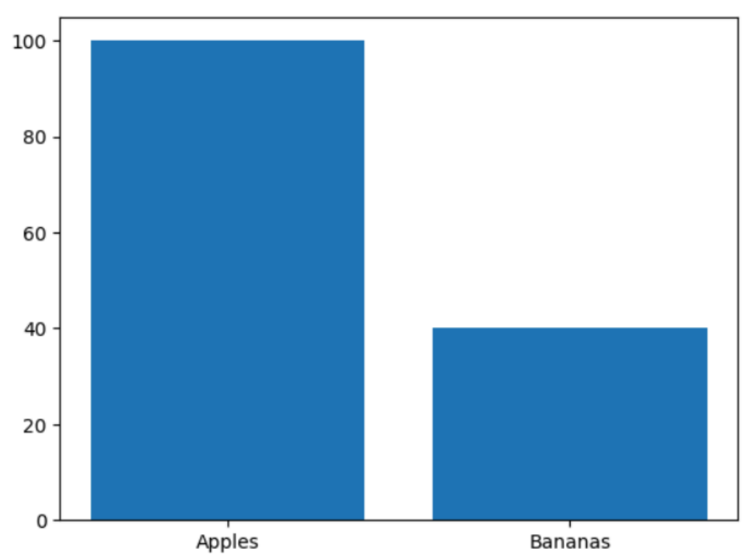
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()
```



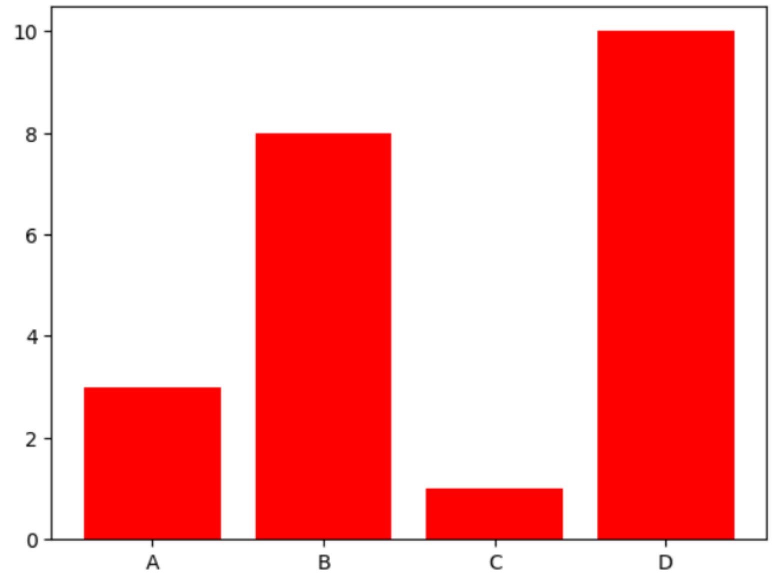
```
x = np.array(["A","B","C","D"])
y = np.array([3,8,1,10])
plt.bar(x, y)
plt.show()
```



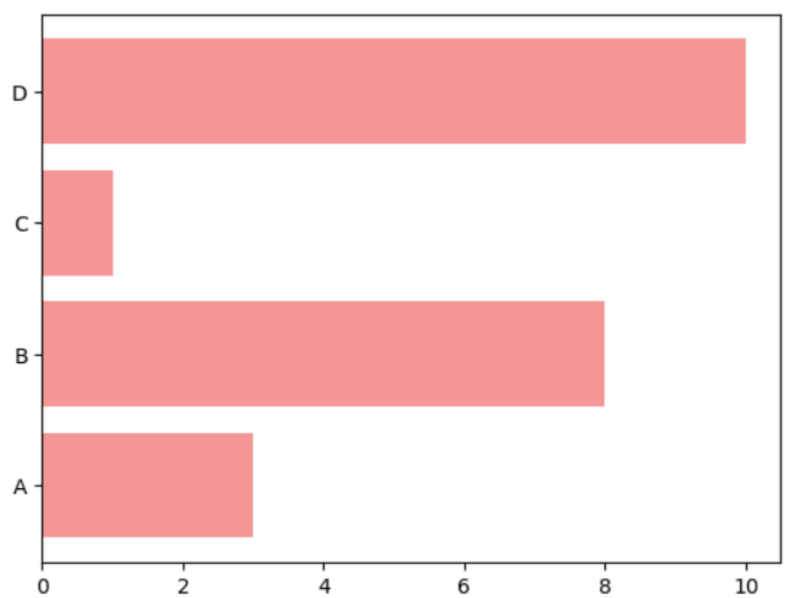

```
x = np.array(["Apples","Bananas"])
y = np.array([100,40])
plt.bar(x, y)
plt.show()
```



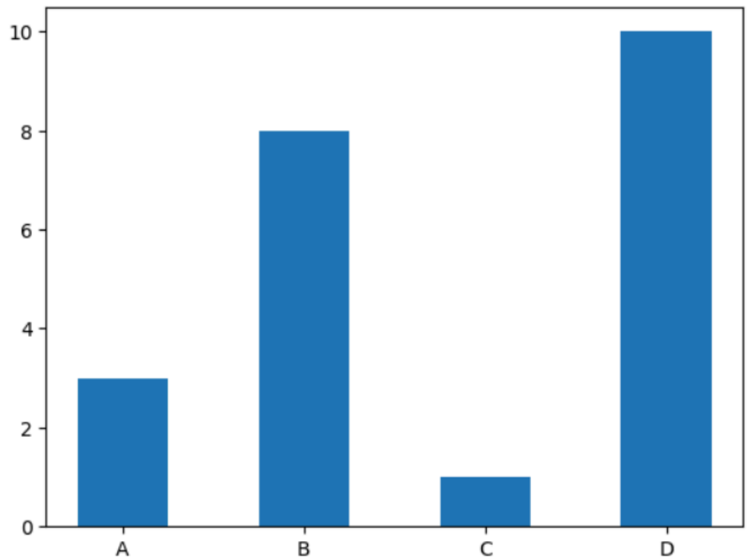
```
x = np.array(["A","B","C","D"])
y = np.array([3,8,1,10])
plt.bar(x, y,color="red")
plt.show()
```



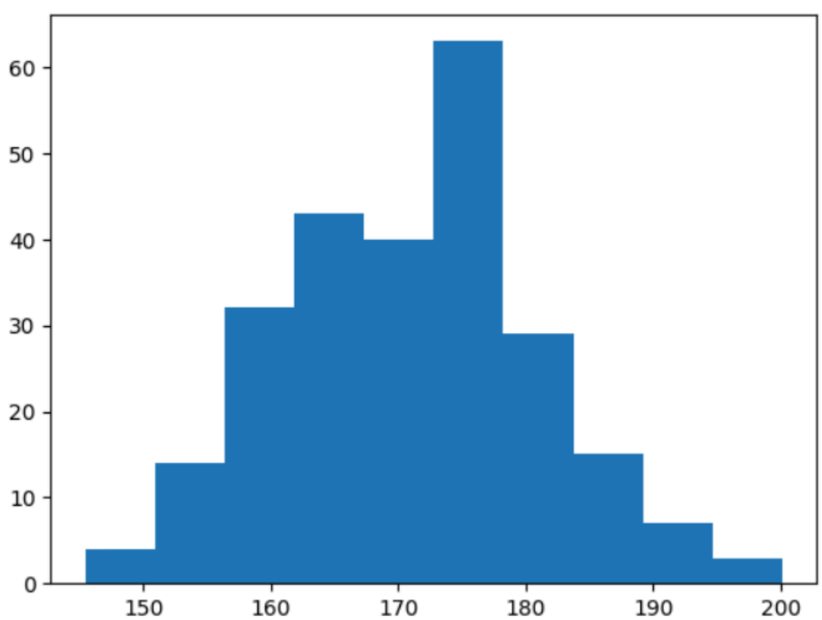

```
x = np.array(["A","B","C","D"])
y = np.array([3,8,1,10])
plt.barh(x, y,color="#f69A97")
plt.show()
```



```
x = np.array(["A","B","C","D"])
y = np.array([3,8,1,10])
plt.bar(x, y,width=0.5)
plt.show()
```

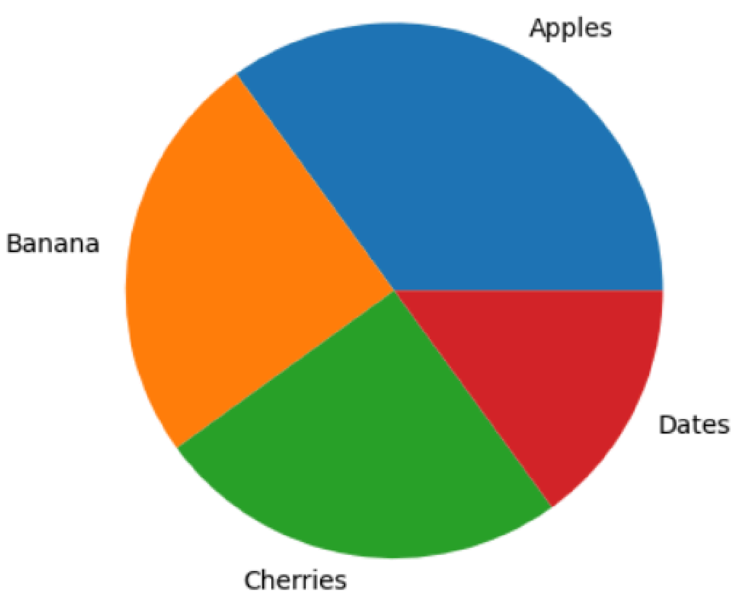



```
x = np.random.normal(170,10,250)
plt.hist(x)
plt.show()
```

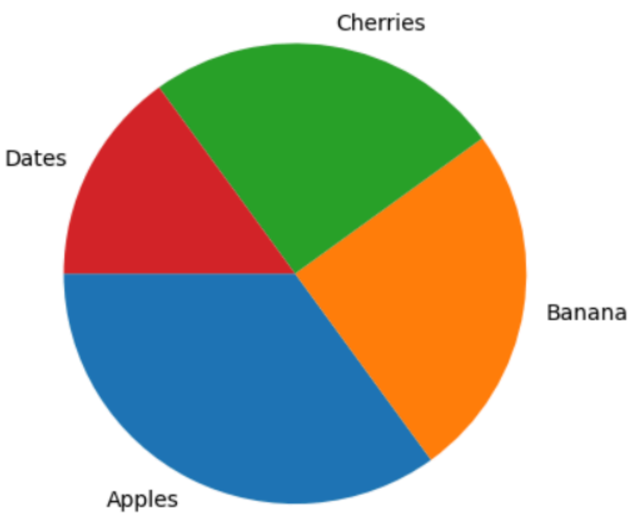


```
x = np.array([35,25,25,15])
plt.pie(x)
plt.show()
```

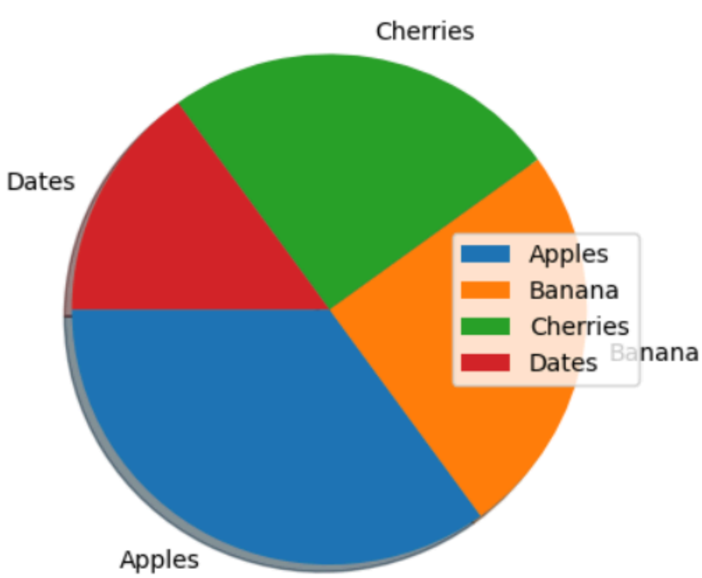
```
x = np.array([35,25,25,15])
mylabel=["Apples","Banana","Cherries","Dates"]
plt.pie(x,labels=mylabel)
plt.show()
```



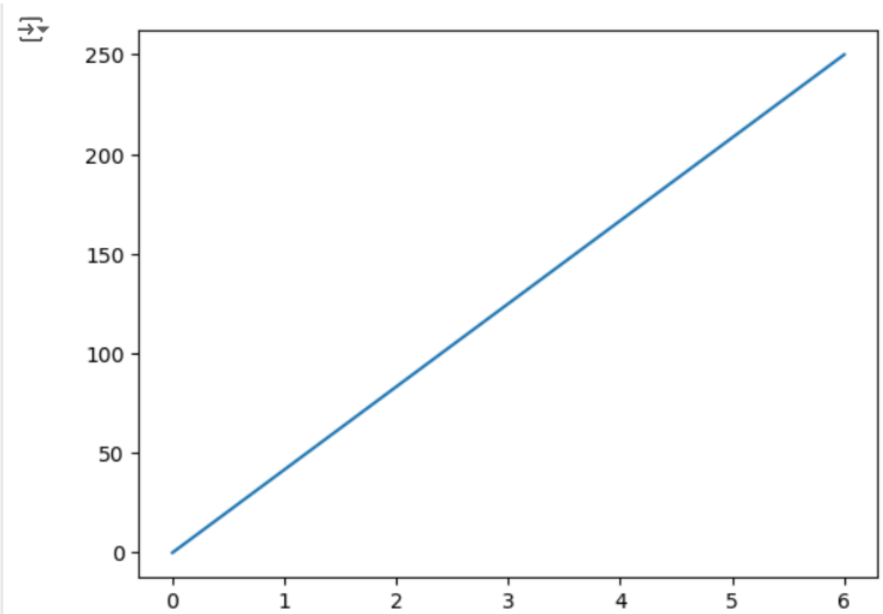
```
x = np.array([35,25,25,15])
mylabel=["Apples","Banana","Cherries","Dates"]
plt.pie(x,labels=mylabel,startangle=180)
plt.show()
```



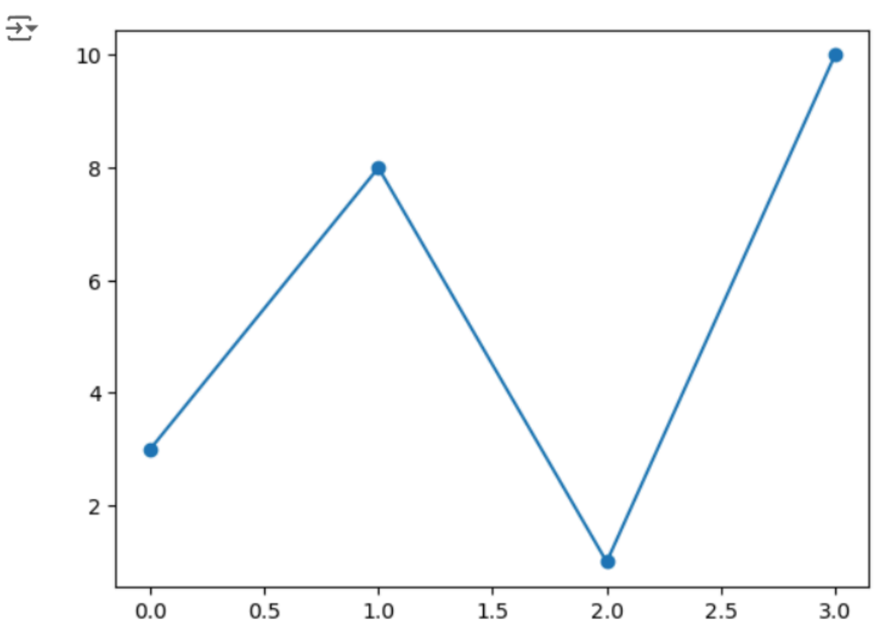

```
x = np.array([35,25,25,15])
mylabel=["Apples","Banana","Cherries","Dates"]
plt.pie(x,labels=mylabel,startangle=180,shadow=True)
plt.legend()
plt.show()
```



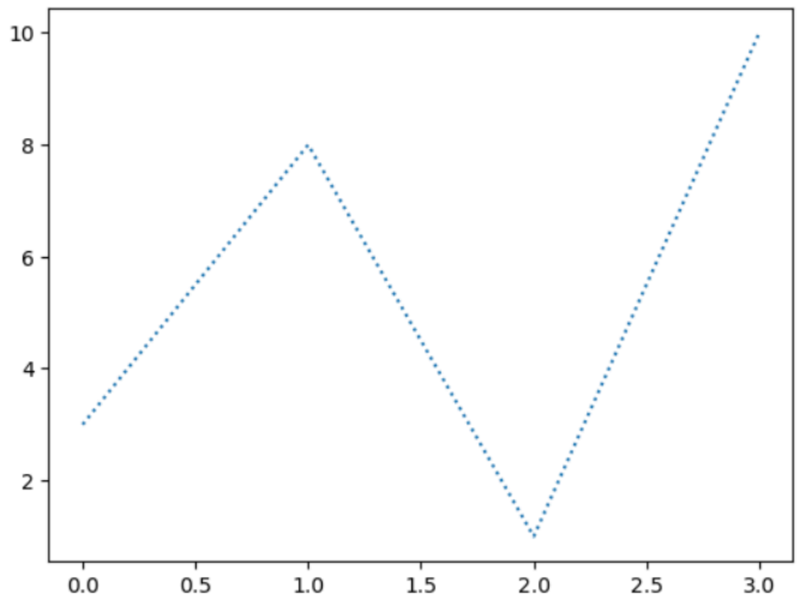
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```



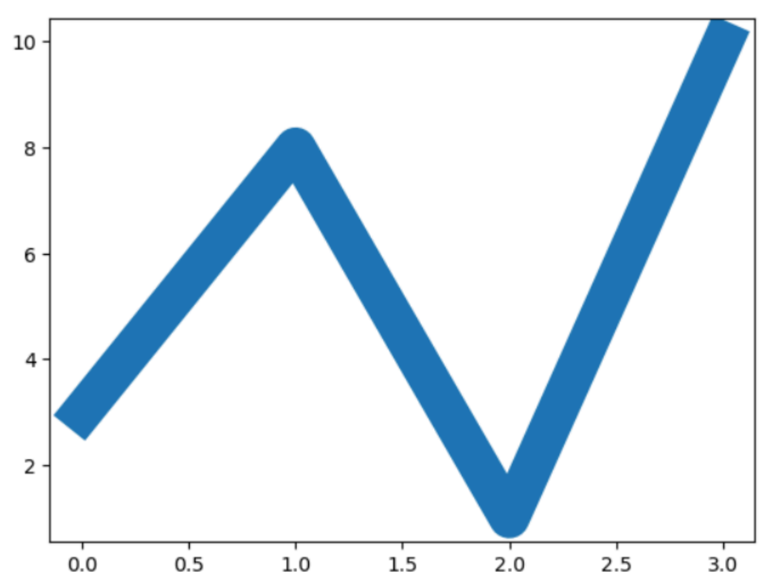

```
ypoints = np.array([3,8,1,10])
plt.plot(ypoints, marker = 'o')
plt.show()
```



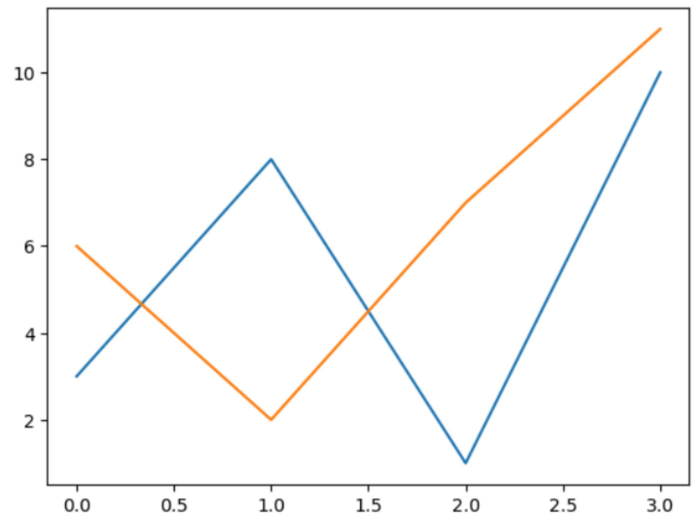
```
ypoints = np.array([3,8,1,10])
plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```



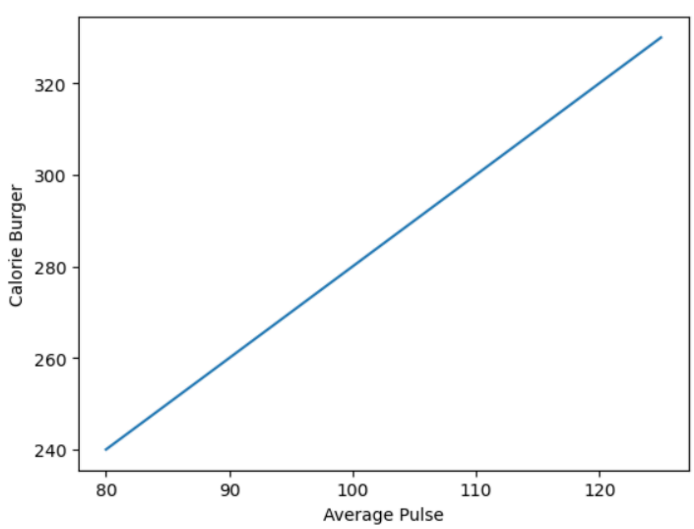

```
ypoints = np.array([3,8,1,10])
#plt.plot(ypoints, color = 'r')
plt.plot(ypoints, c ='#4CAF50')
#plt.plot(ypoints, color = 'hotpink')
plt.plot(ypoints, linewidth = '20.5')
plt.show()
```



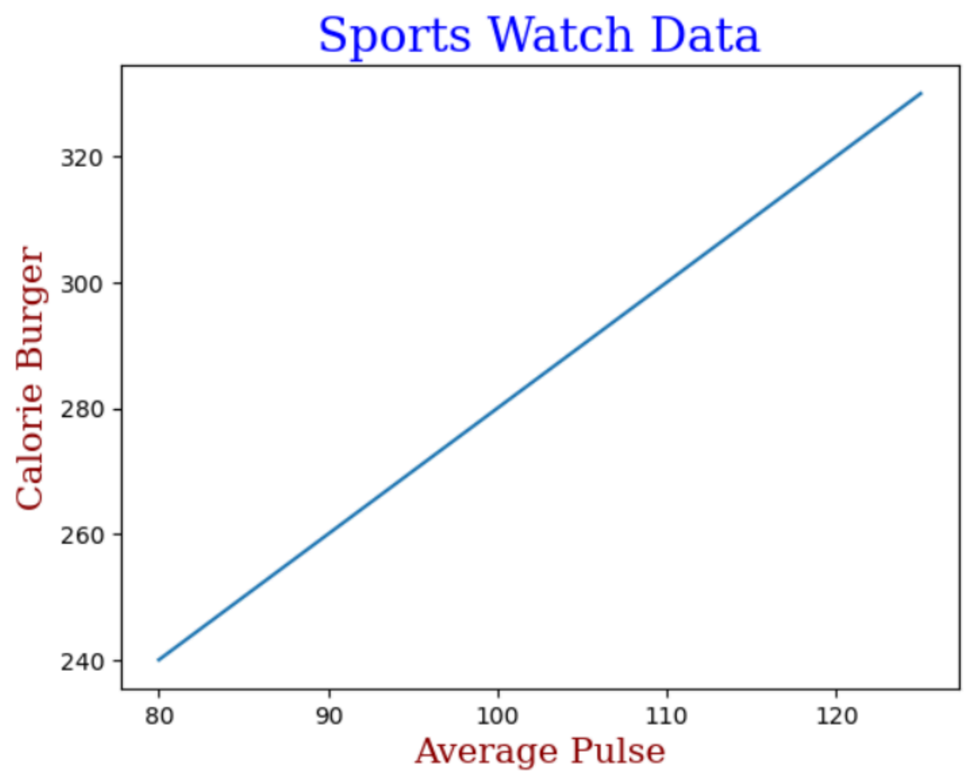
```
y1 = np.array([3,8,1,10])
y2 = np.array([6,2,7,11])
plt.plot(y1)
plt.plot(y2)
plt.show()
```




```
x = np.array([80,85,90,95,100,105,110,115,120,125])
y = np.array([240,250,260,270,280,290,300,310,320,330])
plt.plot(x,y)
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burger")
#plt.title("Sports Watch Data")
plt.show()
```

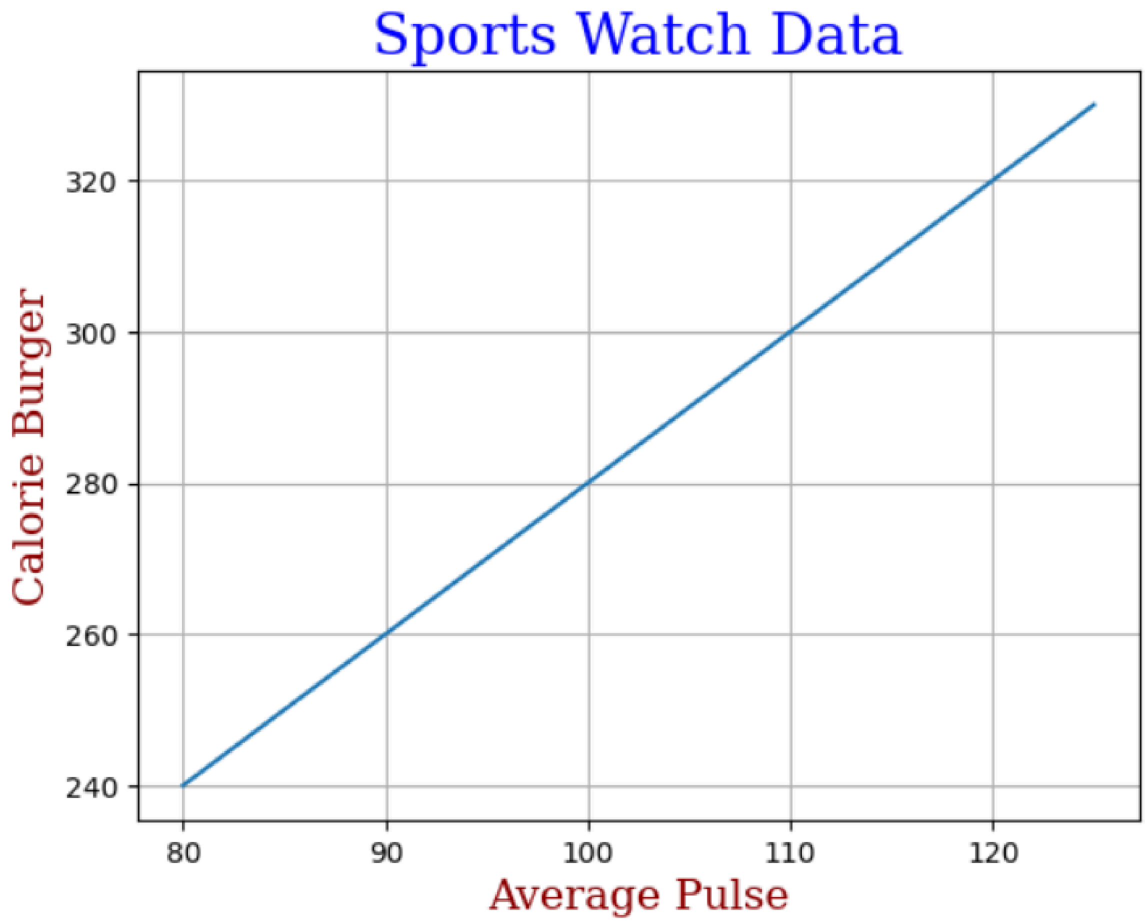


```
x = np.array([80,85,90,95,100,105,110,115,120,125])
y = np.array([240,250,260,270,280,290,300,310,320,330])
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burger", fontdict = font2)
plt.plot(x,y)
plt.show()
```



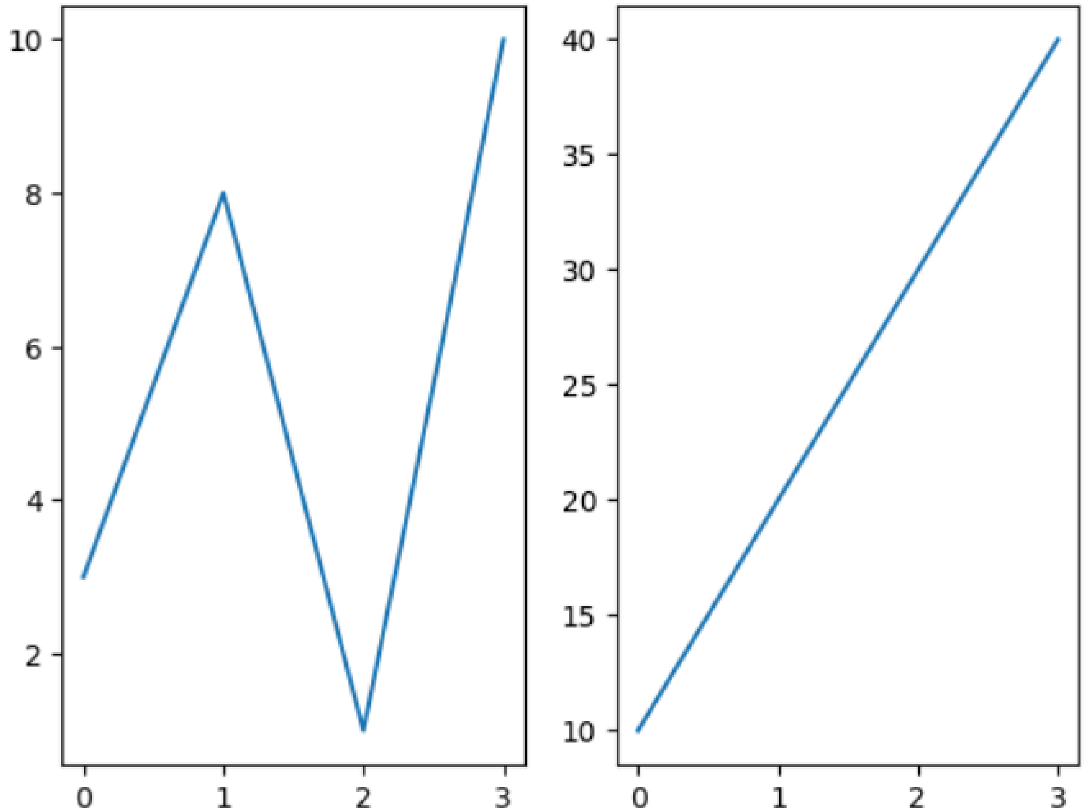

```
x = np.array([80,85,90,95,100,105,110,115,120,125])
y = np.array([240,250,260,270,280,290,300,310,320,330])
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}
plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burger", fontdict = font2)
plt.plot(x,y)
plt.grid()
```

```
plt.show()
```

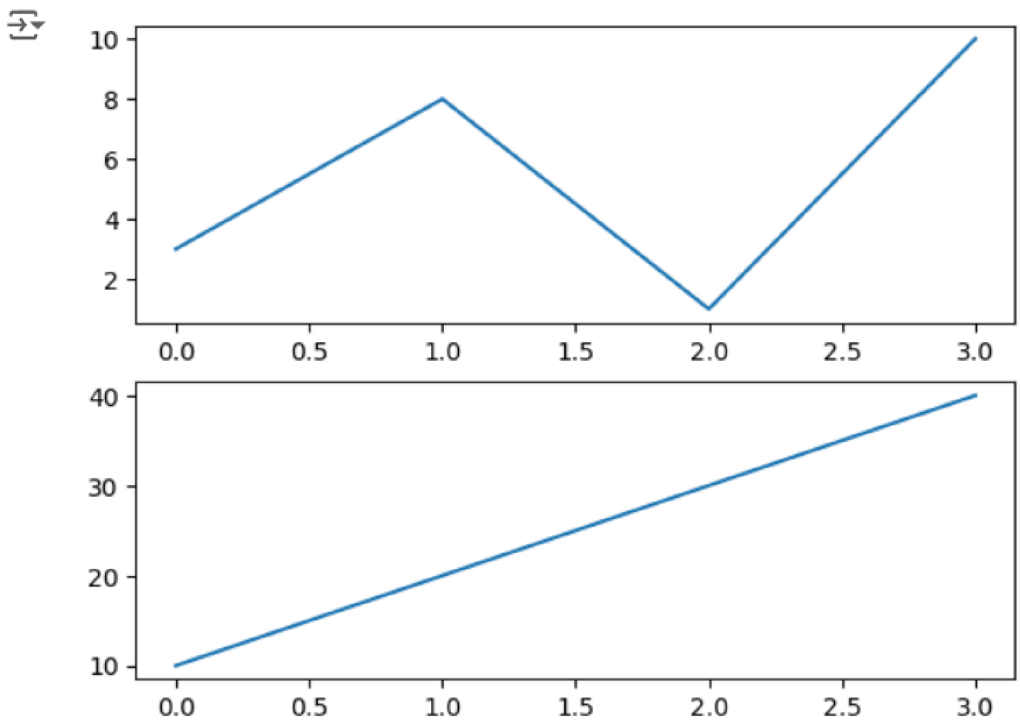



```
x = np.array([0,1,2,3])
y = np.array([3,8,1,10])
plt.subplot(1,2,1)
plt.plot(x,y)
#plot2
x = np.array([0,1,2,3])
y = np.array([10,20,30,40])
plt.subplot(1,2,2)
plt.plot(x,y)
```

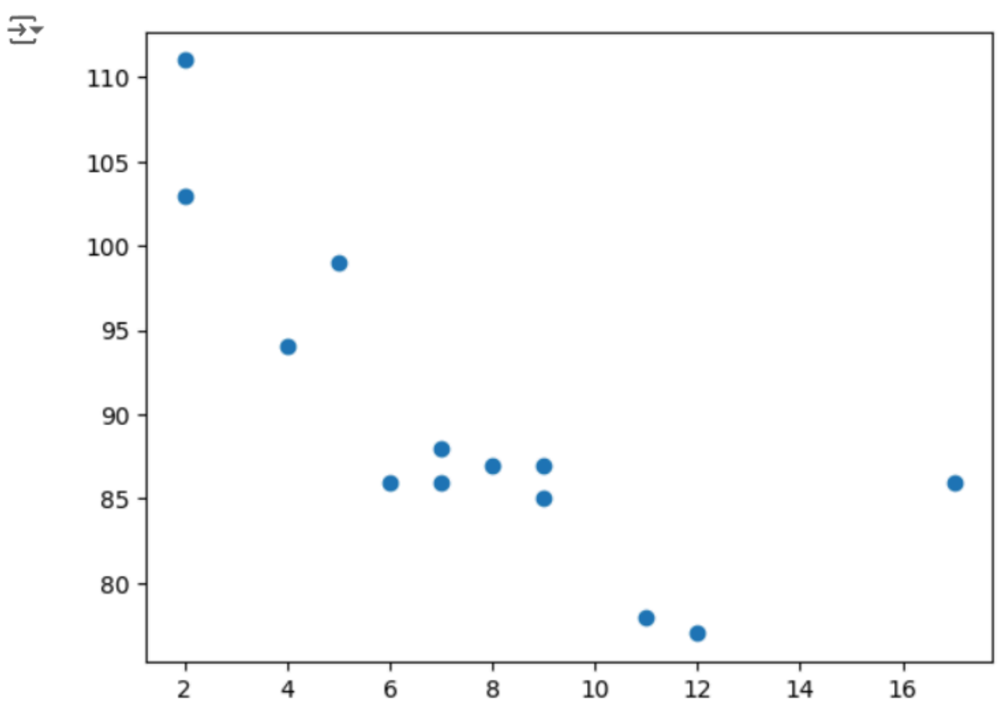
```
[<matplotlib.lines.Line2D at 0x15ada4e47d0>]
```




```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 1, 1)
plt.plot(x, y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 1, 2)
plt.plot(x, y)
plt.show()
```



x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()



Practical 3

Aim: Implementation of Linear Regression, Logistic regression, KNN-classification.

Theory:

A. Linear Regression

Linear regression is a statistical method for modeling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the input variables (X) and the single output variable (Y).

B. Logistic Regression

Logistic regression is a statistical method for analyzing datasets in which there are one or more independent variables that determine an outcome, which is categorical. It is used when the dependent variable is binary (e.g., success/failure, yes/no).

C. K-Nearest Neighbors (KNN) Classification

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm used for classification and regression. For classification, it assigns the class of a given data point based on the majority class among its K nearest neighbors.

Algorithm:

- Choose the number of neighbors K.
- Calculate the distance (e.g., Euclidean distance) between the new data point and all training data points.
- Select the K nearest data points.
- Assign the class by majority vote among the K nearest neighbors.

Code & Output:

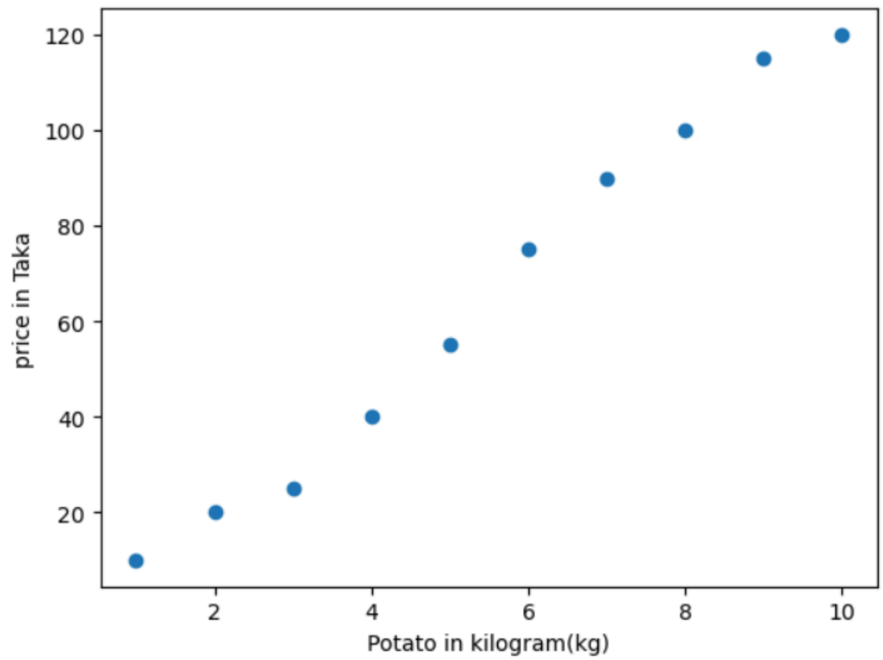
A. Linear Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
df = pd.read_csv('Potato.csv')
df
```

	potato_kg	price
0	1	10
1	2	20
2	3	25
3	4	40
4	5	55
5	6	75
6	7	90
7	8	100
8	9	115
9	10	120

```
%matplotlib inline
plt.xlabel("Potato in kilogram(kg)")
plt.ylabel('price in Taka')
plt.scatter(df.potato_kg, df.price)
```

<matplotlib.collections.PathCollection at 0x133c489e250>



X = df[['potato_kg']]
y = df['price']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
X_train

	potato_kg
1	2
0	1
9	10
2	3
4	5
7	8
5	6
6	7

X_test

	potato_kg
3	4
8	9

y_train

```
1     20
0     10
9    120
2     25
4     55
7    100
5     75
6     90
Name: price, dtype: int64
```

y_test

```
3     40
8    115
Name: price, dtype: int64
```

reg=LinearRegression()
reg.fit(X_train, y_train)

```
▾ LinearRegression
LinearRegression()
```

reg.predict(X_test)

```
array([ 45.55555556, 110.83333333])
```

reg.score(X_test, y_test)

```
0.9828532235939643
```

```
x = input('To know the potato price, Enter the potato kilogram up to 1: ')
import numpy as np
array = np.array(x)
fvalu = array.astype(float)
fvalu_2D = np.array([[fvalu]])
my_prediction = reg.predict(fvalu_2D)
price = my_prediction.item()
print('So', x, 'kilogram potato price is =', price, 'Taka')
```

```
To know the potato price, Enter the potato kilogram up to 1:  3
So 3 kilogram potato price is = 32.5 Taka
```

B. Logistic Regression

```
import pandas as pd
pima = pd.read_csv("diabetes - diabetes.csv")
pima.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
feature_cols =
['Pregnancies','Insulin','BMI','Age','Glucose','BloodPressure','DiabetesPedigreeFunc
tion']
X = pima[feature_cols]
y = pima.Outcome
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
```

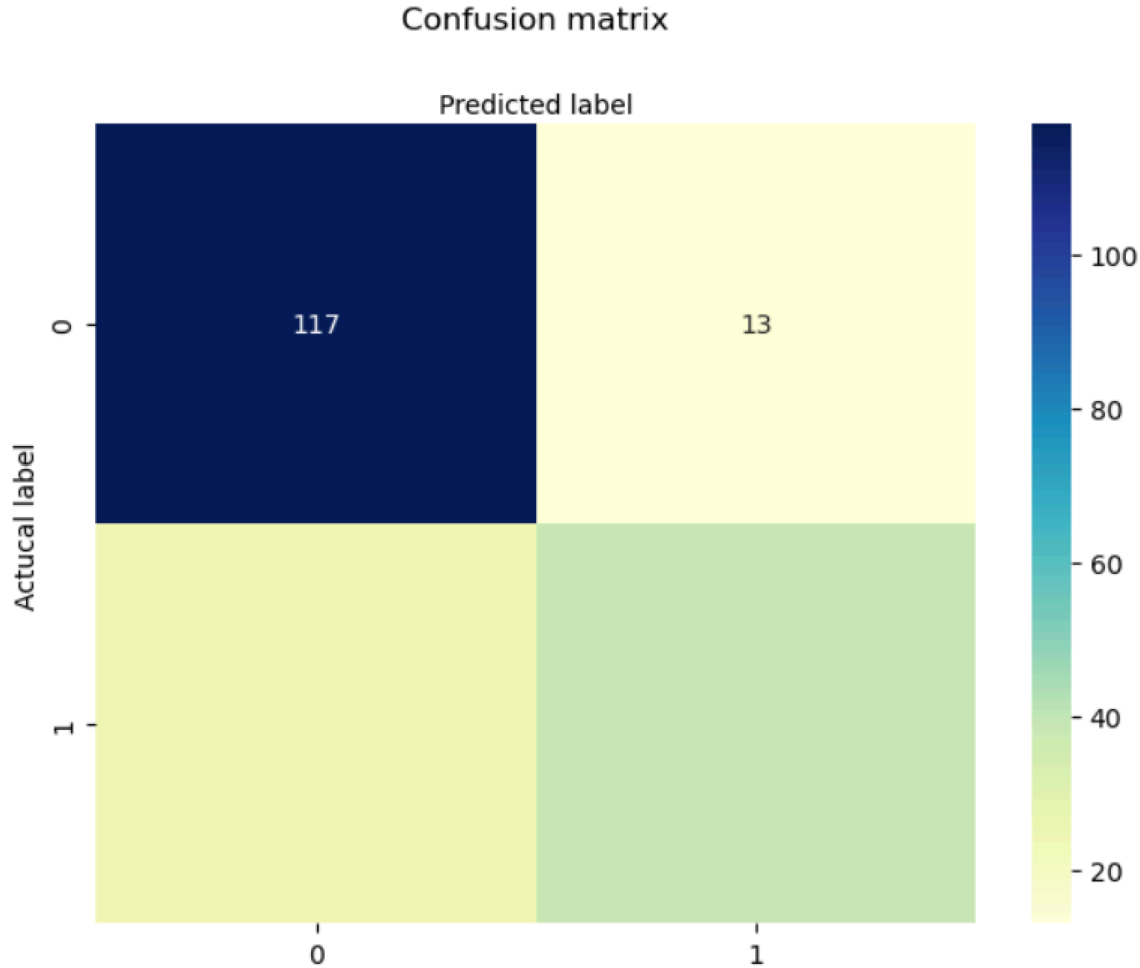
```
C:\Users\Student\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

```
y_pred = logreg.predict(X_test)
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test,y_pred)
cnf_matrix
```

```
array([[117,  13],
       [ 24,  38]], dtype=int64)
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
class_names = [0, 1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cnf_matrix), annot = True, cmap = "YlGnBu", fmt =
'g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y = 1.1)
plt.ylabel('Actucal label')
plt.xlabel('Predicted label')
```

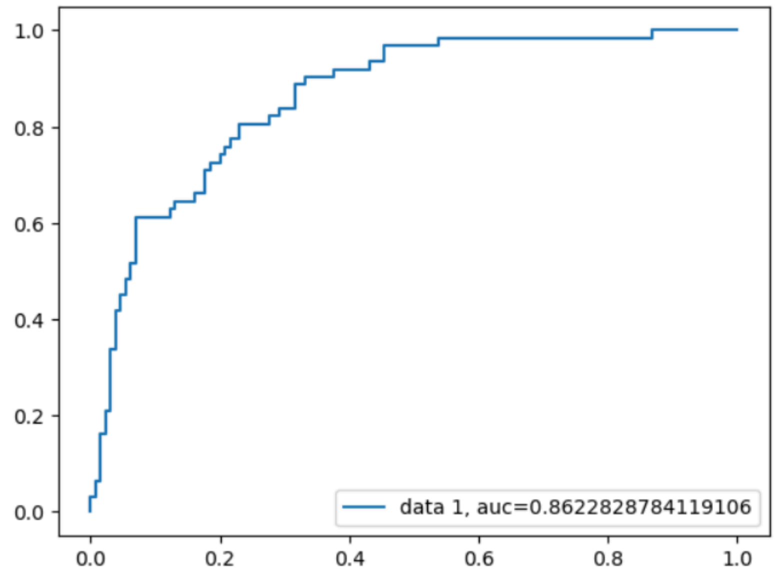
```
Text(0.5, 427.9555555555555, 'Predicted label')
```



print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
print("Precision:",metrics.precision_score(y_test,y_pred))
print("Recall:",metrics.recall_score(y_test,y_pred))

```
Accuracy: 0.8072916666666666
Precision: 0.7450980392156863
Recall: 0.6129032258064516
```

y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()



C. KNN-Classification

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
iris = load_iris()
X, y = iris.data, iris.target
print(X.shape)
print(y.shape)
```

```
(150, 4)
(150,)
```

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,)
knn = KNeighborsClassifier()
param_grid = {
   'n_neighbors':np.arange(1,10),
   'weights':['uniform','distance'],}
```

```
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

```
best_k = grid_search.best_params_['n_neighbors']
best_weights = grid_search.best_params_['weights']
print(f"Best K value: {best_k}")
print(f"Best weight function: {best_weights}")
```

```
Best K value: 8
Best weight function: distance
```

```
best_knn = KNeighborsClassifier(n_neighbors=best_k, weights=best_weights)
best_knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=8, weights='distance')
```

```
y_pred = best_knn.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

```
[[12  0  0]
 [ 0  6  1]
 [ 0  0 11]]
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Test accuracy: {accuracy:.2f}")
```

```
Test accuracy: 0.97
```

Practical 4

Aim: Implementation of dimensionality reduction techniques: Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis.

Theory:

Dimensionality reduction is a crucial step in data preprocessing that aims to reduce the number of input variables in a dataset while retaining as much information as possible. This can help improve the performance of machine learning models, make the models more interpretable, and reduce computational costs. Here are some key techniques for dimensionality reduction:

A. Feature Extraction & Selection

Feature extraction involves creating new features from the original dataset that encapsulate the relevant information. This can sometimes lead to better performance because it focuses on the most important aspects of the data.

Feature selection involves selecting a subset of the original features based on certain criteria, such as their relevance to the target variable or their statistical properties. This can help to improve model performance by removing irrelevant or redundant features.

B. Normalization

Normalization is a preprocessing step that adjusts the scale of the features to a common range, typically [0, 1] or [-1, 1]. This can improve the performance of machine learning algorithms, especially those that are sensitive to the scale of input data, such as gradient descent-based methods.

C. Transformation

Transformation techniques modify the features to make them more suitable for modeling. This can involve linear or non-linear transformations.

D. Principal Component Analysis (PCA)

PCA is a widely used dimensionality reduction technique that transforms the original features into a new set of orthogonal (uncorrelated) features called principal components. The first principal component captures the most variance in the data, the second captures the second most, and so on. PCA helps to reduce dimensionality by selecting the top principal components that explain the most variance, thereby compressing the dataset while preserving essential information.

Code & Output:

```
from sklearn.datasets import load_breast_cancer
breast = load_breast_cancer()
breast
breast.data
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

```
breast.data.shape
```

```
(569, 30)
```

```
breast_labels=breast.target
breast_labels
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

```
import numpy as np
label=np.reshape(breast_labels,(569,1))
label
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
```

```
final_breast_data=np.concatenate([breast.data,label],axis=1)
final_breast_data
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 4.601e-01, 1.189e-01,
        0.000e+00],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 2.750e-01, 8.902e-02,
        0.000e+00],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 3.613e-01, 8.758e-02,
        0.000e+00],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 2.218e-01, 7.820e-02,
        0.000e+00],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 4.087e-01, 1.240e-01,
        0.000e+00],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 2.871e-01, 7.039e-02,
        1.000e+00]])
```

```
import pandas as pd
breast_dataset=pd.DataFrame(final_breast_data)
breast_dataset
```

	0	1	2	3	4	5	6	7	8	9	...	21	22	23	24	25	26	27	28	29	30
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601	0.11890	0.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902	0.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758	0.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638	0.17300	0.0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115	0.0
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637	0.0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820	0.0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400	0.0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039	1.0

569 rows × 31 columns

```
features=breast.feature_names
features
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

features_labels=np.append(features,'label')

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension', 'label'], dtype='<U23')
```

breast_dataset.columns=features_labels

breast_dataset.head()

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883

5 rows × 31 columns

breast_dataset['label'].replace(0,'NO',inplace=True)

breast_dataset['label'].replace(1,'YES',inplace=True)

breast_dataset.tail()

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884

5 rows × 31 columns

from sklearn.preprocessing import StandardScaler

x=breast_dataset.loc[:,features].values

x=StandardScaler().fit_transform(x)

x

```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

np.mean(x),np.std(x)

```
(-6.118909323768877e-16, 1.0)
```

feast_cols=['features'+str(i) for i in range(x.shape[1])]
normalised_breast=pd.DataFrame(x,columns=feast_cols)
normalised_breast.tail()

	features0	features1	features2	features3	features4	features5	features6	features7	features8	features9
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589	-0.931027
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664	-1.058611
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117	-0.895587
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194	1.043695
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752	-1.114873	-1.261820	-0.820070	-0.561032

5 rows × 30 columns

from sklearn.decomposition import PCA
pca_breast=PCA(n_components=2)
principalComponents_breast=pca_breast.fit_transform(x)
principal_breast_Df=pd.DataFrame(data=principalComponents_breast,columns=['
principal component 1','principal component 2'])
principal_breast_Df.tail()

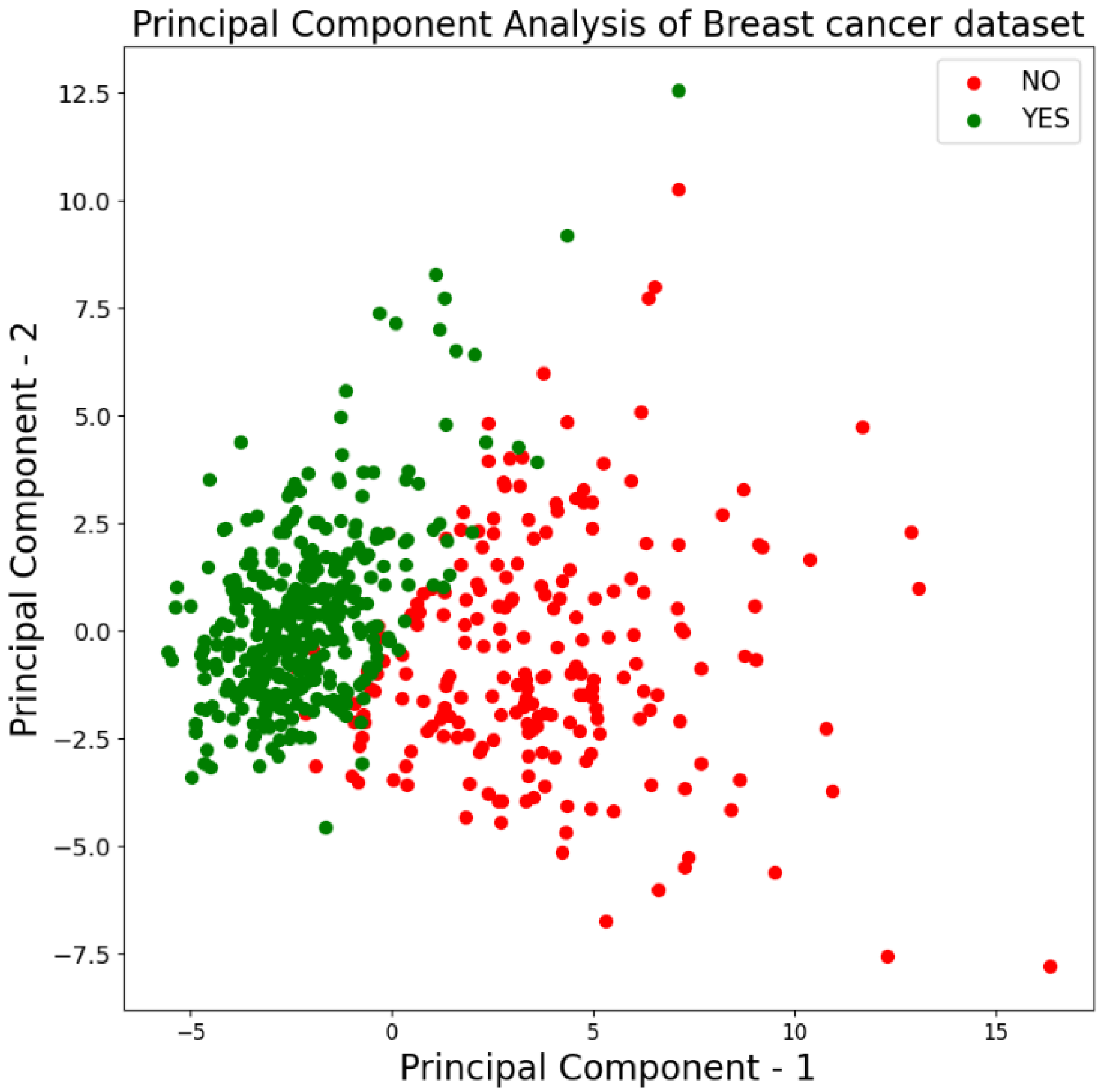
	principal component 1	principal component 2
564	6.439315	-3.576817
565	3.793382	-3.584048
566	1.256179	-1.902297
567	10.374794	1.672010
568	-5.475243	-0.670637

```
print('Explained variation per princial component :{}'.format(pca_breast.explained_variance_ratio_))
```

```
Explained variation per princial component :[0.44272026 0.18971182]
```

```
import matplotlib.pyplot as plt
plt.figure()
plt.figure(figsize=(10,10))
plt.xticks(fontsize=12)
plt.yticks(fontsize=14)
plt.xlabel('Principal Component - 1',fontsize=20)
plt.ylabel('Principal Component - 2',fontsize=20)
plt.title('Principal Component Analysis of Breast cancer dataset',fontsize=20)
targets=['NO','YES']
colors=['r','g']
for target,color in zip(targets,colors):
  indicatesToKeep=breast_dataset['label']==target
  plt.scatter(principal_breast_Df.loc[indicatesToKeep,'principal component 1'],principal_breast_Df.loc[indicatesToKeep,'principal component 2'],c=color,s=50)
plt.legend(targets,prop={'size':15})
```

```
<matplotlib.legend.Legend at 0x792834ec0430>
<Figure size 640x480 with 0 Axes>
```



Practical 5

Aim: Implementation of K-Means and K-medoid clustering algorithm.

Theory:

K-Means and K-Medoids are both partitioning clustering techniques used to group a set of objects into k clusters. While they share some similarities, they have distinct differences in their approach to clustering.

A. K-Means Clustering:

K-Means is a widely used clustering algorithm that aims to partition n observations into k clusters, where each observation belongs to the cluster with the nearest mean.

Steps of K-Means:

- Initialization: Randomly select k centroids (initial cluster centers).
- Assignment: Assign each data point to the nearest centroid, forming k clusters.
- Update: Calculate the new centroids by taking the mean of all data points assigned to each cluster.
- Repeat: Repeat the assignment and update steps until the centroids no longer change significantly (convergence).

B. K-Medoids Clustering:

K-Medoids (or Partitioning Around Medoids, PAM) is similar to K-Means but uses medoids instead of means as cluster centers. A medoid is an actual data point within the dataset, which makes the algorithm more robust to outliers and noise.

Steps of K-Medoids:

- Initialization: Randomly select k data points as the initial medoids.
- Assignment: Assign each data point to the nearest medoid, forming k clusters.
- Update: For each cluster, select a new medoid by minimizing the total dissimilarity (sum of distances) between the medoid and all other points in the cluster.
- Repeat: Repeat the assignment and update steps until the medoids no longer change significantly (convergence).

Code & Output:

5.A K-Mean Clustering:

```
import pandas as pd
import  numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
data=pd.read_csv("C:/Users/Student/Desktop/crime_data.csv")
data.head()
```

	Murder	Assault	UrbanPop	Rape
0	13.2	236	58	21.2
1	10.0	263	48	44.5
2	8.1	294	80	31.0
3	8.8	190	50	19.5
4	9.0	276	91	40.6

```
data.isnull().any()
```

```
Murder      False
Assault     False
UrbanPop    False
Rape        False
dtype: bool
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Murder    50 non-null     float64
 1   Assault   50 non-null     int64  
 2   UrbanPop  50 non-null     int64  
 3   Rape      50 non-null     float64
dtypes: float64(2), int64(2)
memory usage: 1.7 KB
```

data

[ ]

	Murder	Assault	UrbanPop	Rape
0	13.2	236	58	21.2
1	10.0	263	48	44.5
2	8.1	294	80	31.0
3	8.8	190	50	19.5
4	9.0	276	91	40.6
5	7.9	204	78	38.7
6	3.3	110	77	11.1
7	5.9	238	72	15.8
8	15.4	335	80	31.9
9	17.4	211	60	25.8
10	5.3	46	83	20.2

df_index_length = len(data.index)
df_index_length

```
50
```

```
df_length=len(data)
df_length
```

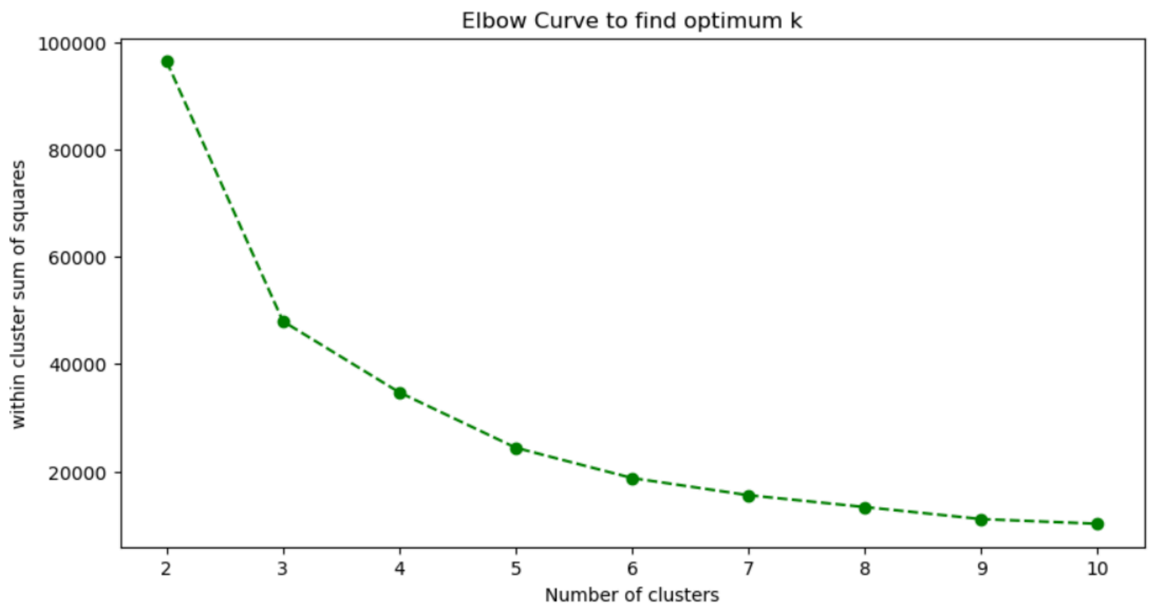
```
50
```

```
def minmaxscaler(x):
    scaler=MinMaxScaler()
    scaler.fit(x)
    scaled_data=scaler.transform(x)
    scaler_df=pd.DataFrame(scaled_data,columns=x.columns,index=x.index)
    return scaler_df
norm_data=data.copy()
minmaxscaler(norm_data)
```

	Murder	Assault	UrbanPop	Rape
0	0.746988	0.654110	0.440678	0.359173
1	0.554217	0.746575	0.271186	0.961240
2	0.439759	0.852740	0.813559	0.612403
3	0.481928	0.496575	0.305085	0.315245
4	0.493976	0.791096	1.000000	0.860465
5	0.427711	0.544521	0.779661	0.811370
6	0.150602	0.222603	0.762712	0.098191
7	0.307229	0.660959	0.677966	0.219638
8	0.879518	0.993151	0.813559	0.635659
9	1.000000	0.568493	0.474576	0.478036
10	0.271084	0.003425	0.864407	0.333333

```
k = list(range(2,11))
sum_of_squared_distances = []
for i in k :
```

```
    kmeans = KMeans(n_clusters = i)
    kmeans.fit(norm_data)
    sum_of_squared_distances.append(kmeans.inertia_)
plt.figure(figsize=(10,5))
plt.plot(k, sum_of_squared_distances, 'go--')
plt.xlabel('Number of clusters')
plt.ylabel('within cluster sum of squares')
plt.title('Elbow Curve to find optimum k')
```




```
kmeans4 = KMeans(n_clusters = 4)
kmeans4.fit(norm_data)
y_pred = kmeans4.fit_predict(norm_data)
print(y_pred)
data['Cluster'] = y_pred+1
```

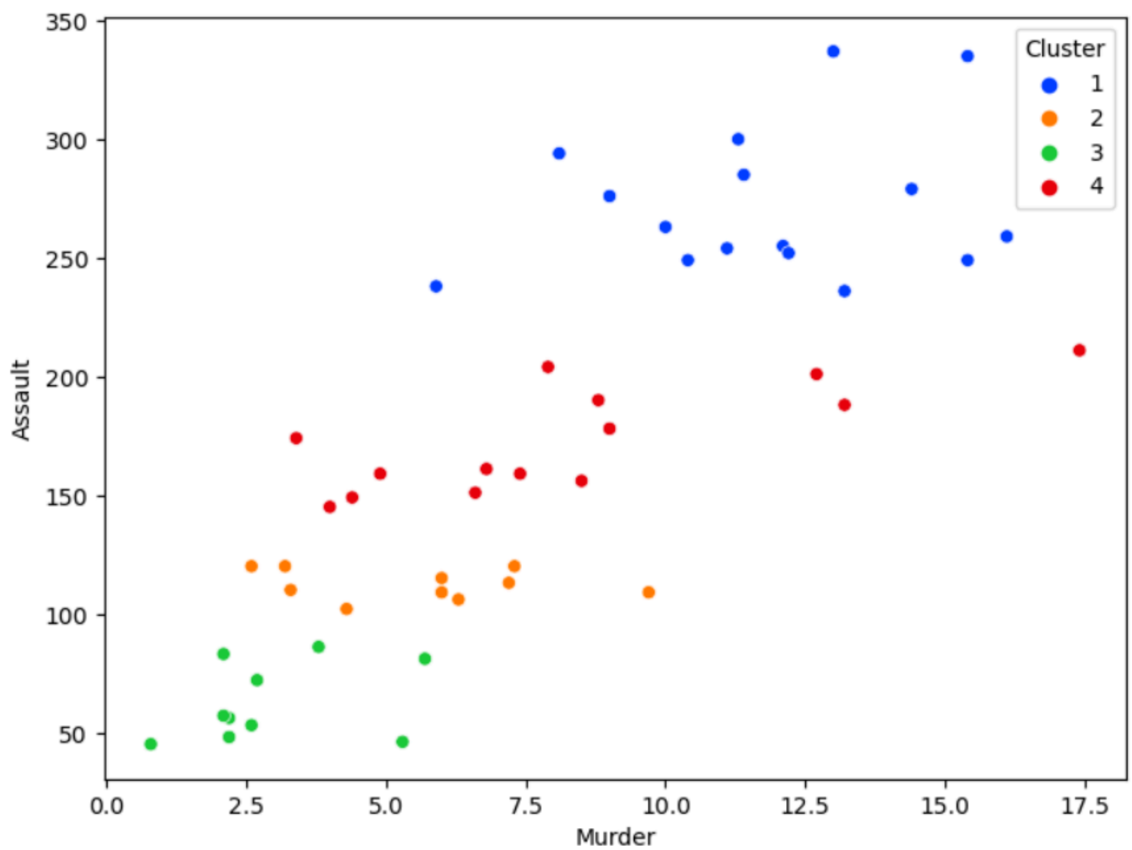
```
[0 0 0 3 0 3 1 0 0 3 2 1 0 1 2 1 1 0 2 0 3 0 2 0 3 1 1 0 2 3 0 0 0 2 1 3 3
 1 3 0 2 3 3 1 2 3 3 2 2 3]
```

```
centroids = kmeans4.cluster_centers_
centroids = pd.DataFrame(centroids,
columns=['Murder','Assault','UrbanPop','Rape'])
centroids.index = np.arange(1, len(centroids)+1)
centroids
```

	Murder	Assault	UrbanPop	Rape
1	11.812500	272.562500	68.312500	28.375000
2	5.590000	112.400000	65.600000	17.270000
3	2.950000	62.700000	53.900000	11.510000
4	8.214286	173.285714	70.642857	22.842857

```
import seaborn as sns
plt.figure(figsize= (8,6))
sns.set_palette("pastel")
sns.scatterplot(x=data['Murder'], y = data['Assault'], hue=data['Cluster'],
palette='bright')
```

```
<Axes: xlabel='Murder', ylabel='Assault'>
```




```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
%matplotlib inline
from sklearn.datasets import make_blobs
data = make_blobs(n_samples=300, n_features=2, centers=5,
cluster_std=1.8,random_state=101)
data[0].shape
```

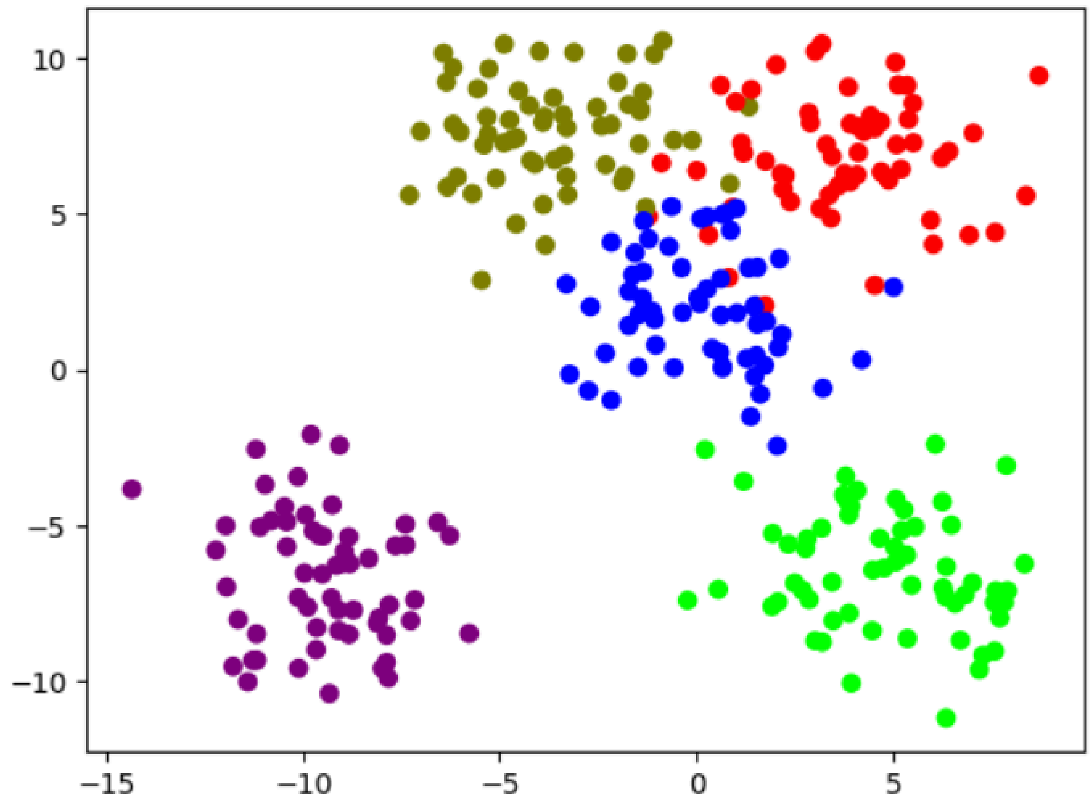
```
(300, 2)
```

```
data[1]
```

```
array([4, 0, 2, 4, 4, 0, 2, 2, 0, 3, 1, 0, 0, 2, 1, 1, 4, 4, 0, 1, 1, 4,
       4, 0, 0, 2, 1, 2, 2, 4, 0, 0, 4, 0, 0, 2, 2, 3, 3, 4, 4, 4, 2, 1,
       2, 1, 0, 4, 4, 4, 1, 1, 0, 4, 0, 3, 1, 0, 2, 3, 1, 1, 3, 1, 4, 4,
       4, 1, 1, 1, 2, 2, 1, 1, 4, 1, 3, 3, 3, 1, 3, 2, 3, 0, 2, 1, 1, 1,
       2, 0, 3, 2, 1, 2, 1, 4, 3, 0, 2, 3, 0, 4, 0, 0, 3, 3, 3, 4, 2, 2,
       2, 1, 0, 0, 0, 3, 4, 4, 1, 1, 2, 0, 3, 4, 1, 1, 1, 4, 4, 2, 3, 3,
       3, 3, 3, 3, 4, 1, 0, 0, 0, 4, 3, 4, 2, 3, 2, 1, 3, 3, 4, 4, 2, 3,
       1, 2, 4, 2, 1, 4, 2, 4, 4, 1, 3, 1, 0, 3, 0, 0, 0, 3, 1, 3, 2, 1,
       3, 0, 0, 4, 1, 2, 4, 2, 2, 0, 0, 0, 3, 4, 2, 0, 2, 3, 2, 3, 4, 0,
       2, 0, 3, 4, 2, 0, 3, 2, 0, 4, 3, 4, 4, 1, 2, 4, 1, 3, 1, 0, 3, 1,
       0, 2, 3, 0, 1, 3, 2, 4, 4, 3, 1, 3, 3, 4, 0, 0, 0, 2, 0, 2, 4, 1,
       1, 3, 4, 0, 2, 4, 2, 4, 1, 2, 3, 0, 3, 2, 1, 0, 0, 0, 2, 3, 2, 0,
       4, 2, 2, 1, 4, 1, 1, 1, 3, 3, 0, 4, 3, 3, 4, 1, 4, 3, 2, 0, 2, 4,
       1, 2, 4, 3, 2, 2, 1, 0, 3, 1, 3, 2, 1, 0])
```

```
plt.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='brg')
```

<matplotlib.collections.PathCollection at 0x1cd4c42f550>



from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5)
kmeans.fit(data[0])

```
C:\Users\Student\anacond
  warnings.warn(
C:\Users\Student\anacond
  warnings.warn(
```

KMeans
KMeans(n_clusters=5)

kmeans.cluster_centers_

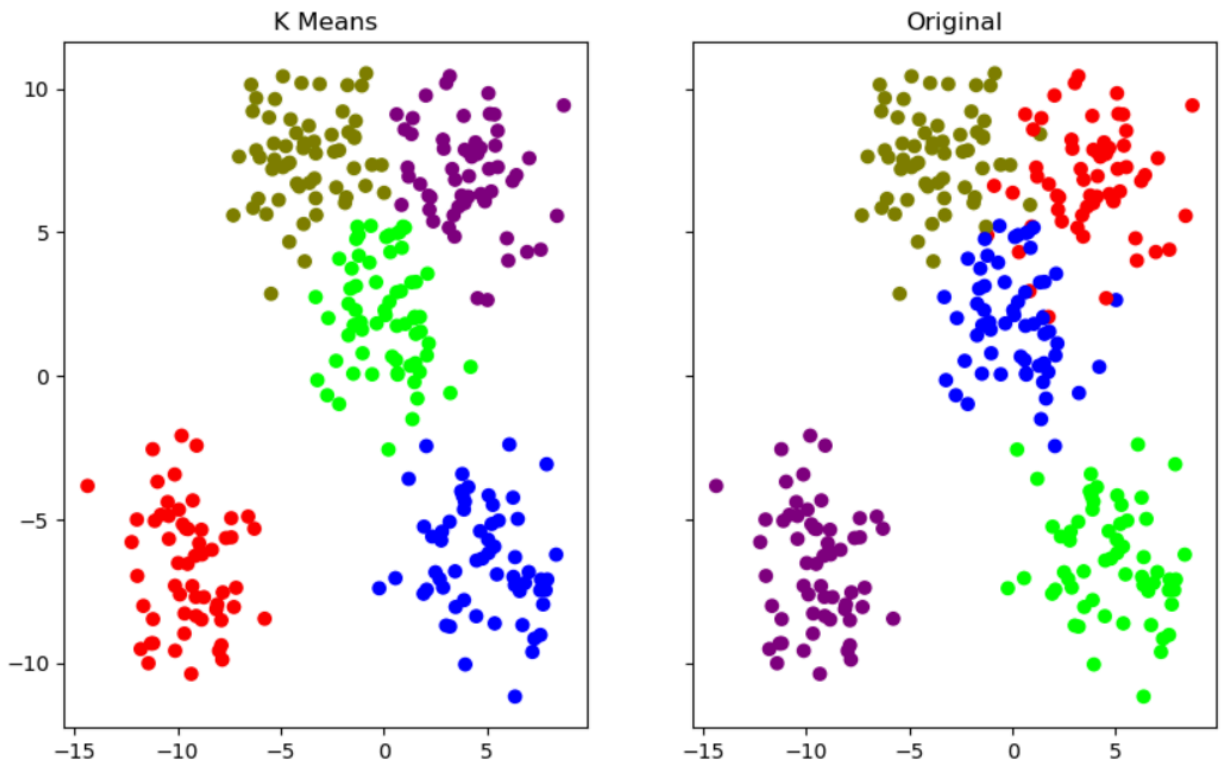
```
array([[ 4.78617945, -6.4146509 ],
       [ 4.07267325,  7.01145994],
       [-9.46255618, -6.63414105],
       [-3.74000213,  7.61266002],
       [ 0.04663137,  2.09954982]])
```

kmeans.labels_

```
array([0, 4, 1, 0, 4, 4, 1, 1, 4, 1, 2, 4, 4, 1, 2, 2, 0, 0, 1, 2, 2, 0,
       0, 4, 4, 1, 2, 1, 1, 0, 4, 4, 0, 4, 4, 1, 3, 3, 3, 0, 0, 0, 1, 2,
       4, 2, 4, 0, 0, 0, 2, 2, 4, 0, 4, 3, 2, 4, 1, 3, 2, 2, 3, 2, 0, 0,
       0, 2, 2, 2, 1, 1, 2, 2, 0, 2, 3, 3, 3, 2, 3, 4, 3, 4, 1, 2, 2, 2,
       1, 4, 3, 1, 2, 4, 2, 0, 3, 4, 1, 3, 4, 0, 4, 4, 3, 3, 3, 0, 1, 4,
       3, 2, 4, 4, 4, 4, 0, 0, 2, 2, 1, 4, 3, 0, 2, 2, 2, 0, 0, 1, 3, 3,
       3, 3, 3, 3, 0, 2, 4, 4, 4, 0, 3, 0, 1, 3, 1, 2, 3, 3, 0, 0, 1, 3,
       2, 1, 0, 1, 2, 0, 1, 0, 0, 2, 3, 2, 4, 3, 4, 4, 4, 3, 2, 3, 1, 2,
       3, 4, 4, 0, 2, 1, 0, 1, 1, 4, 4, 4, 3, 0, 1, 4, 1, 3, 1, 1, 0, 4,
       1, 0, 3, 0, 1, 4, 3, 1, 4, 0, 3, 0, 0, 2, 4, 0, 2, 3, 2, 4, 3, 2,
       4, 1, 3, 4, 2, 3, 1, 0, 0, 3, 2, 3, 3, 0, 4, 4, 4, 1, 4, 1, 0, 2,
       2, 3, 0, 4, 1, 0, 1, 0, 2, 1, 3, 4, 3, 1, 2, 4, 4, 4, 1, 3, 1, 4,
       0, 1, 1, 2, 0, 2, 2, 2, 3, 3, 4, 0, 3, 3, 0, 2, 0, 3, 1, 4, 1, 0,
       2, 1, 0, 3, 1, 1, 2, 4, 3, 2, 3, 1, 2, 4])
```

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0],data[0][:,1],c=kmeans.labels_,cmap='brg')
ax2.set_title('Original')
ax2.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='brg')

```
<matplotlib.collections.PathCollection at 0x1cd4c484090>
```

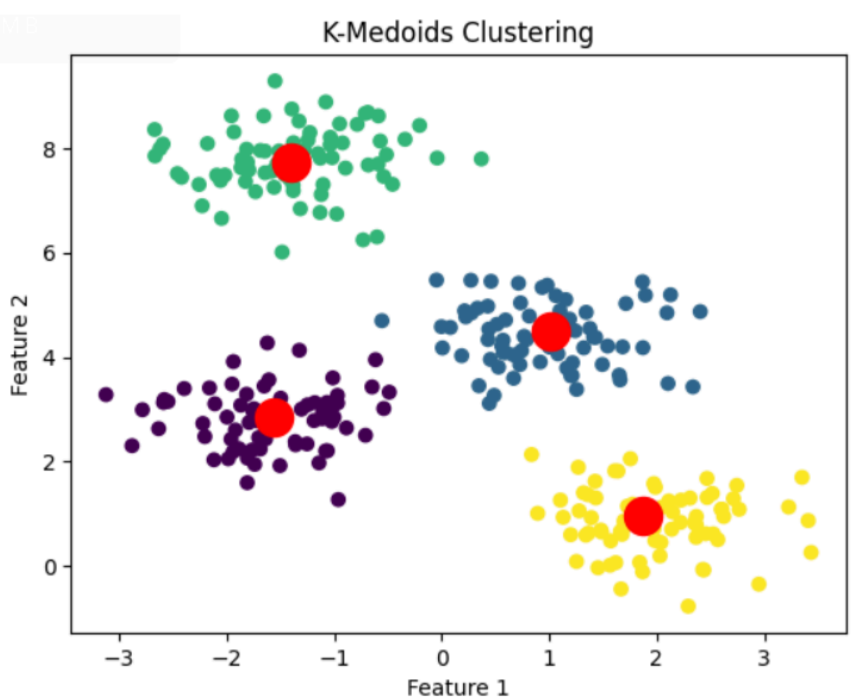


5.B K-Medoid Clustering:

```
!pip install scikit-learn-extra
from sklearn_extra.cluster import KMedoids
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
x,_=make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
KMedoids = KMedoids(n_clusters=4, random_state=0)
KMedoids.fit(x)
```

```
KMedoids
KMedoids(n_clusters=4, random_state=0)
```

```
medoids = KMedoids.cluster_centers_
labels = KMedoids.labels_
plt.scatter(x[:,0], x[:,1], c=labels, cmap='viridis')
plt.scatter(medoids[:,0], medoids[:,1], c= 'red',marker='o', s=300)
plt.title('K-Medoids Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show
```



Practical 6

Aim: Implementation of Classifying data using Support Vector Machines (SVMs).

Theory:

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression tasks. SVMs are particularly effective in high-dimensional spaces and are known for their ability to find the optimal boundary between classes.

Key Concepts of Support Vector Machines

1. Hyperplane

A hyperplane is a decision boundary that separates different classes in the feature space. For a dataset with n features, the hyperplane is an $n-1$ dimensional subspace. In a two-dimensional space, the hyperplane is a line; in a three-dimensional space, it is a plane.

2. Support Vectors

Support vectors are the data points that are closest to the hyperplane. These points are critical because they determine the position and orientation of the hyperplane. The SVM algorithm aims to find the hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest support vectors.

3. Margin

The margin is the distance between the hyperplane and the nearest data points from each class. SVM seeks to maximize this margin, which helps to improve the model's generalization to unseen data.

SVM for Classification

Linear SVM

In the case of linearly separable data, SVM finds a hyperplane that separates the data into two classes with the maximum margin. The objective is to solve the following optimization problem:

Non-Linear SVM

For non-linearly separable data, SVM uses the kernel trick to map the input features into a higher-dimensional space where a linear hyperplane can be found. Common kernel functions include:

Code & Output:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('Social_Network_Ads - Social_Network_Ads.csv')

dataset

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

X = dataset.iloc[:,[2,3]].values

y = dataset.iloc[:,4].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

X_train

```
array([[ 0.58164944, -0.88670699],
       [-0.60673761,  1.46173768],
       [-0.01254409, -0.5677824 ],
       [-0.60673761,  1.89663484],
       [ 1.37390747, -1.40858358],
       [ 1.47293972,  0.99784738],
       [ 0.08648817, -0.79972756],
       [-0.01254409, -0.24885782],
       [-0.21060859, -0.5677824 ],
       [-0.21060859, -0.19087153],
       [-0.30964085, -1.29261101],
       [-0.30964085, -0.5677824 ],
       [ 0.38358493,  0.09905991],
       [ 0.8787462 , -0.59677555],
       [ 2.06713324, -1.17663843],
```

X_test

```
array([[-0.80480212,  0.50496393],
       [-0.01254409, -0.5677824 ],
       [-0.30964085,  0.1570462 ],
       [-0.80480212,  0.27301877],
       [-0.30964085, -0.5677824 ],
       [-1.10189888, -1.43757673],
       [-0.70576986, -1.58254245],
       [-0.21060859,  2.15757314],
       [-1.99318916, -0.04590581],
       [ 0.8787462 , -0.77073441],
       [-0.80480212, -0.59677555],
       [-1.00286662, -0.42281668],
       [-0.11157634, -0.42281668],
```

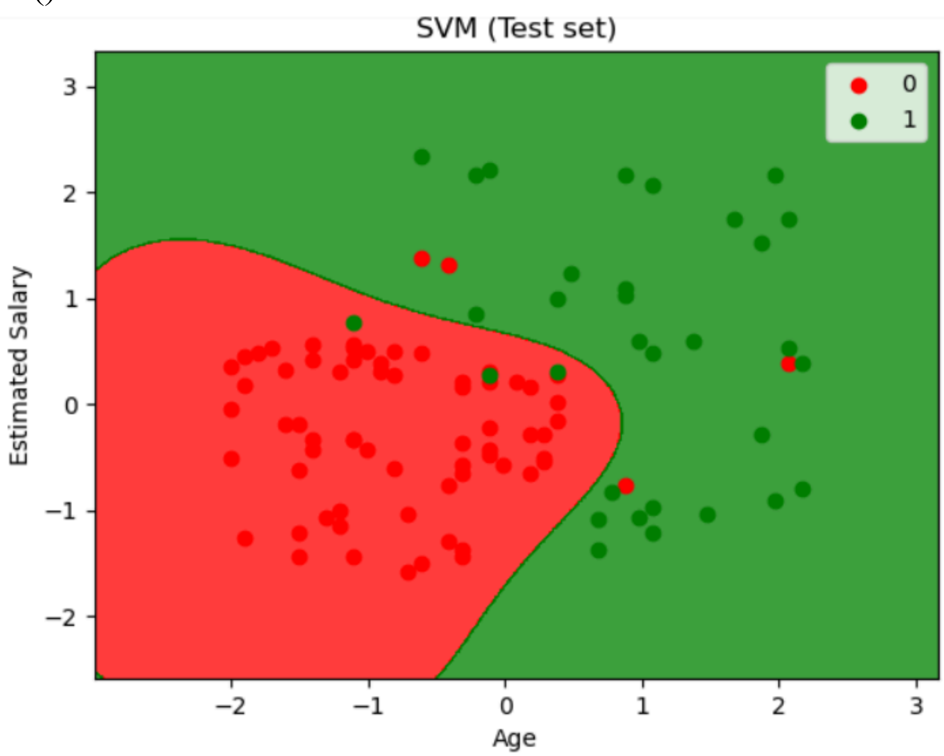
```
from sklearn.svm  import SVC
classifier = SVC(kernel = 'rbf',random_state = 0)
classifier.fit(X_train, y_train)
```

```
SVC
SVC(random_state=0)
```

```
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test,y_pred)
print(cm)
print()
print("**********************************")
print("Accuracy:")
accuracy_score(y_test,y_pred)
```

```
[[64  4]
 [ 3 29]]

**********************************
Accuracy:
0.93
```

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start=X_set[:, 0].min() - 1, stop=X_set[:,
0].max() + 1, step=0.01),
                     np.arange(start=X_set[:, 1].min() - 1, stop=X_set[:, 1].max() + 1,
step=0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(['red', 'green']))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c=ListedColormap(['red', 'green'])(i), label=j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Practical 7

Aim: Implementation of Bagging Algorithm: Decision Tree, Random Forest

.

Theory:

Bagging, short for Bootstrap Aggregating, is an ensemble learning technique that aims to improve the stability and accuracy of machine learning algorithms. It reduces variance and helps to avoid overfitting by combining the predictions of multiple models.

A. Decision Trees

Decision trees are highly flexible and powerful models but can suffer from high variance, meaning that small changes in the training data can lead to significantly different trees. Bagging helps mitigate this by training multiple decision trees on different subsets of the data and aggregating their predictions.

Steps for Bagging with Decision Trees:

- Create Bootstrap Samples: Generate B bootstrap samples from the training dataset.
- Train Decision Trees: Train a decision tree on each bootstrap sample. These trees are often referred to as base learners or weak learners.
- Aggregate Predictions: For regression, average the predictions of all the trees. For classification, use majority voting to determine the final class.

B. Random Forest

Random Forest is an extension of the bagging technique specifically designed for decision trees. It introduces additional randomness when building each tree to further reduce correlation among the trees and improve overall performance.

Steps for Building a Random Forest:

- Create Bootstrap Samples: Generate B bootstrap samples from the training dataset.
- Train Decision Trees:For each bootstrap sample, train a decision tree.
- At each node in the tree, randomly select a subset of features and choose the best feature to split the data based on this subset.
- Aggregate Predictions: For regression, average the predictions from all trees. For classification, use majority voting to determine the final class.

Code & Output:

7.A Decision Tree:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import plot_tree
df = sns.load_dataset('iris')
df.shape
df.isnull().any()
```

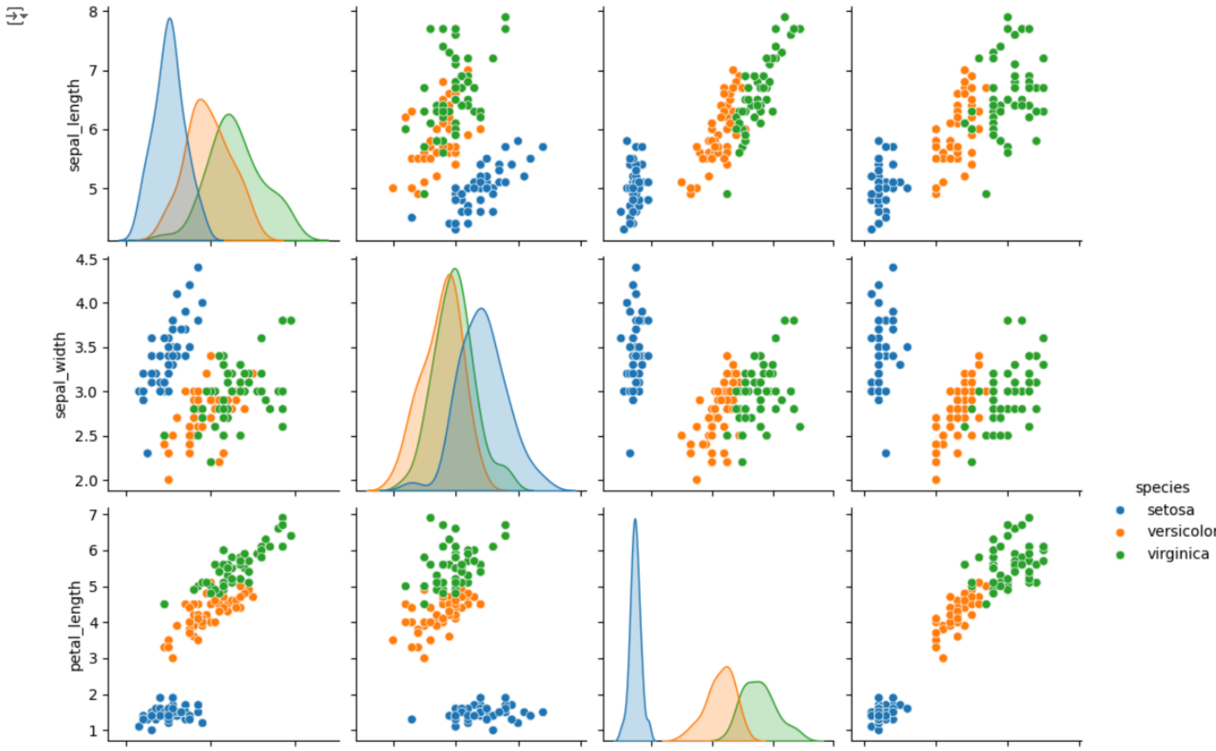
```
sepal_length    False
sepal_width     False
petal_length    False
petal_width     False
species         False
dtype: bool
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

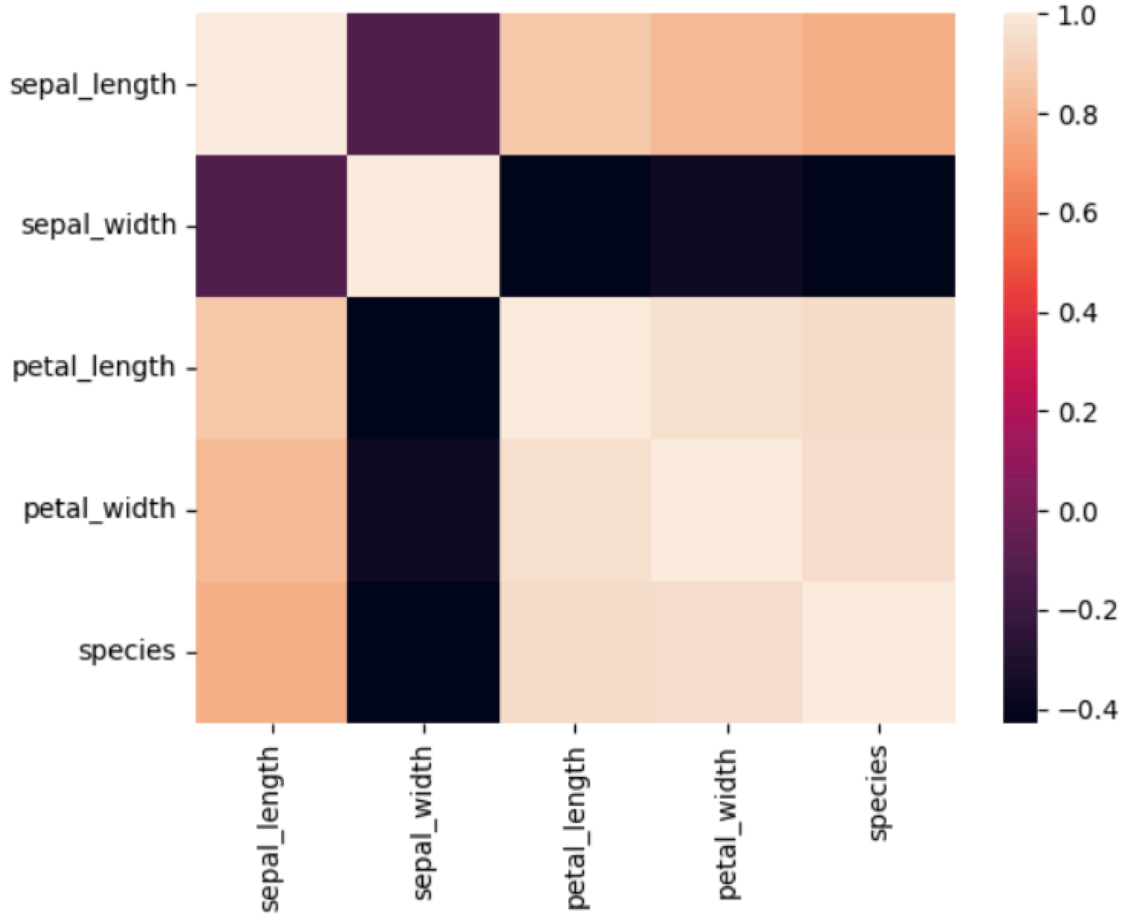
```
sns.pairplot(data = df, hue = 'species')
```

[ ] <seaborn.axisgrid.PairGrid at 0x78a224fd6ec0>




```
df1 = sns.load_dataset('iris')

# Import label encoder
from sklearn import preprocessing

# label_encoder object knows
# how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df1['species']= label_encoder.fit_transform(df1['species'])
df1['species'].unique()
sns.heatmap(df1.corr())
```

<Axes: >



target = df['species']
df1 = df.copy()
df1 = df1.drop('species' , axis=1)
X = df1
target

```
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
           ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

```
le = LabelEncoder()
target = le.fit_transform(target)
target
```

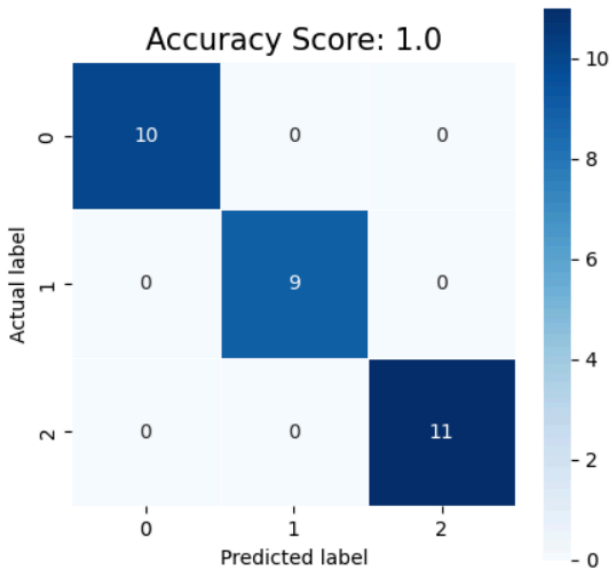
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

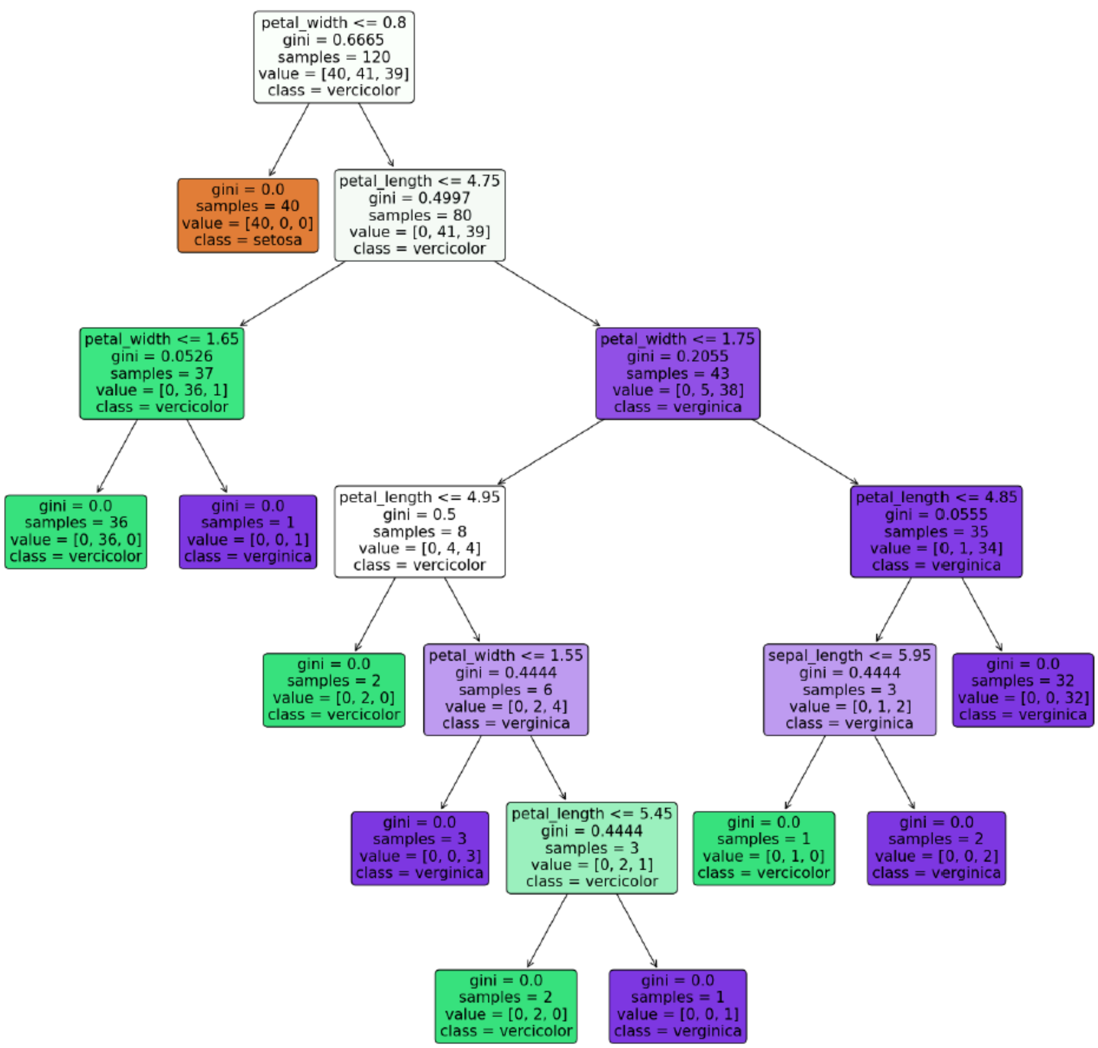
```
y = target
X_train, X_test, y_train,y_test = train_test_split(X, y, test_size = 0.2, random_state
=42)
print("training split input- ", X_train.shape)
print()
print("Testing split input - ", X_test.shape)
```

```
training split input-  (120, 4)

Testing split input -  (30, 4)
```

```
dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
print('Decision Tree Classifier Created')
y_pred = dtree.predict(X_test)
cm = confusion_matrix(y_test,y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5,annot=True,square = True,cmap='Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score: {0}'.format(dtree.score(X_test,y_test))
plt.title(all_sample_title, size = 15)
plt.figure(figsize=(20,20))
dec_tree = plot_tree(decision_tree = dtree, feature_names = df1.columns,
class_names=["setosa","vercicolor","verginica"],filled = True,precision = 4,
rounded = True )
```

Decision Tree Classifier Created





7.B Random Forest:

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

df=pd.read_csv("/content/audi.csv")

df
```

	model	year	price	transmission	mileage	fuelType	tax	mpg	engineSize
0	A1	2017	12500	Manual	15735	Petrol	150	55.4	1.4
1	A6	2016	16500	Automatic	36203	Diesel	20	64.2	2.0
2	A1	2016	11000	Manual	29946	Petrol	30	55.4	1.4
3	A4	2017	16800	Automatic	25952	Diesel	145	67.3	2.0
4	A3	2019	17300	Manual	1998	Petrol	145	49.6	1.0

```
X=df.iloc[:,[0,1,3,4,5,6,7,8]].values
Y=df.iloc[:,[2]].values

print(X)
```

```
[[' A1' 2017 'Manual' ... 150 55.4 1.4]
 [' A6' 2016 'Automatic' ... 20 64.2 2.0]
 [' A1' 2016 'Manual' ... 30 55.4 1.4]
 ...
 [' A3' 2020 'Manual' ... 150 49.6 1.0]
 [' Q3' 2017 'Automatic' ... 150 47.9 1.4]
 [' Q3' 2016 'Manual' ... 150 47.9 1.4]]
```

```
print(Y)
```

```
[[12500]
 [16500]
 [11000]
 ...
 [17199]
 [19499]
 [15999]]
```

```
from sklearn.preprocessing import LabelEncoder
```

```
le1=LabelEncoder()
X[:,0]=le1.fit_transform(X[:,0])
X[:,-4]=le1.fit_transform(X[:,-4])

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[2])],remainder
='passthrough')
X=ct.fit_transform(X)
print(X)
```

```
[[0.0 1.0 1.0 ... 150 55.4 1.4]
 [0.0 1.0 0.0 ... 20 64.2 2.0]
 [0.0 1.0 1.0 ... 30 55.4 1.4]
 ...
 [0.0 1.0 1.0 ... 150 49.6 1.0]
 [0.0 1.0 0.0 ... 150 47.9 1.4]
 [0.0 1.0 1.0 ... 150 47.9 1.4]]
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(X)

print(X)
```

```
[[0.0 1.0 1.0 ... 150 55.4 1.4]
 [0.0 1.0 0.0 ... 20 64.2 2.0]
 [0.0 1.0 1.0 ... 30 55.4 1.4]
 ...
 [0.0 1.0 1.0 ... 150 49.6 1.0]
 [0.0 1.0 0.0 ... 150 47.9 1.4]
 [0.0 1.0 1.0 ... 150 47.9 1.4]]
```

Splitting Dataset Into Training and testing

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

Model Training

from sklearn.ensemble import RandomForestRegressor
regression=RandomForestRegressor(random_state=0)
regression.fit(X_train,Y_train)

```
<ipython-input-29-be73de189193>:3: DataC
  regression.fit(X_train,Y_train)
        RandomForestRegressor
RandomForestRegressor(random_state=0)
```

Y_pred=regression.predict(X_test)

Testing Result

print(np.concatenate((Y_pred.reshape(len(Y_pred),1),Y_test.reshape(len(Y_test),1)),1))

```
[[14281.03 14998.  ]
 [23437.9  21950.  ]
 [27312.82 28990.  ]
 ...
 [46503.55 45995.  ]
 [31380.3  30500.  ]
 [ 9953.98  8400.  ]]
```

Calculating Accuracy

```
0.9535109633010225
```

from sklearn.metrics import r2_score,mean_absolute_error
r2_score(Y_test,Y_pred)

mean_absolute_error(Y_test,Y_pred)

print(Y_pred)

```
[14281.03 23437.9  27312.82 ... 46503.55 31380.3   9953.98]
```

ReShape to 2D

print(Y_test)

```
[[14998]
 [21950]
 [28990]
 ...
 [45995]
 [30500]
 [ 8400]]
```

Y_pred=np.reshape(Y_pred,(-1,1))

Y_pred

```
array([[14281.03],
       [23437.9 ],
       [27312.82],
       ...,
       [46503.55],
       [31380.3 ],
       [ 9953.98]])
```

Making Pandas DataFrame

mydata=np.concatenate((Y_test,Y_pred),axis=1)
dataframe=pd.DataFrame(mydata,columns=['Real Price','Predicated Price'])

print(dataframe)

```
      Real Price  Predicated Price
0        14998.0          14281.03
1        21950.0          23437.90
2        28990.0          27312.82
3        25489.0          27193.03
4        30950.0          32341.00
...          ...               ...
2129     23700.0          38913.94
2130     18000.0          16746.08
2131     45995.0          46503.55
2132     30500.0          31380.30
2133      8400.0           9953.98

[2134 rows x 2 columns]
```

Practical 8

Aim: Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.

Theory:

A. AdaBoost (Adaptive Boosting)

AdaBoost is one of the first boosting algorithms that corrects the errors of its predecessors iteratively.

How it works:

- Initialization: Assign equal weights to all training samples.
- Training: Train a weak learner (usually a decision tree stump) on the training data.
- Error Calculation: Calculate the weighted error of the weak learner.
- Update Weights: Increase the weights of the misclassified samples so that the next weak learner focuses more on these harder cases.
- Iteration: Repeat steps 2-4 for a predefined number of iterations or until the error is minimized.

B. Stochastic Gradient Boosting (Gradient Boosting Machines, GBM)

Stochastic Gradient Boosting is an enhancement of traditional gradient boosting that introduces randomness to improve generalization.

How it works:

- Initialization: Start with an initial prediction, usually the mean of the target values.
- Iterative Training:
- Compute the residuals (errors) of the current model.
- Train a weak learner on a random subset of the training data (this is the stochastic part).
- Update the model by adding the weak learner's predictions scaled by a learning rate.
- Update Model: Combine the current model with the new weak learner to minimize the overall error.
- Iteration: Repeat steps 2-3 for a predefined number of iterations.

C. Voting Ensemble
Voting Ensemble is a straightforward ensemble method that aggregates the predictions of multiple models by voting.

Types of Voting:

- Hard Voting: Each model casts a vote for a class, and the class with the majority votes is selected as the final prediction.
- Soft Voting: Each model provides a probability distribution over classes, and the probabilities are averaged to make the final prediction.

How it works:

- Training: Train multiple different models (e.g., decision trees, SVMs, logistic regression) on the same dataset.
- Aggregation:
- For hard voting, use majority voting.
- For soft voting, average the predicted probabilities.
- Prediction: The final output is determined by the aggregated votes or averaged probabilities.

Code & Output:

8.A AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
abc = AdaBoostClassifier(n_estimators=50, learning_rate=1)
model = abc.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn import metrics
svc = SVC(probability=True, kernel='linear')
abc = AdaBoostClassifier(n_estimators=50, estimator=svc,learning_rate=1)
model = abc.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

```
Accuracy: 0.9777777777777777
```

8.B Stochastic Gradient Boosting

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
#Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target
#Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
gb_clf = GradientBoostingClassifier(n_estimators = 100, max_depth = 3, learning_rate = 0.1, subsample = 0.8, random_state = 42)
gb_clf.fit(X_train, y_train)

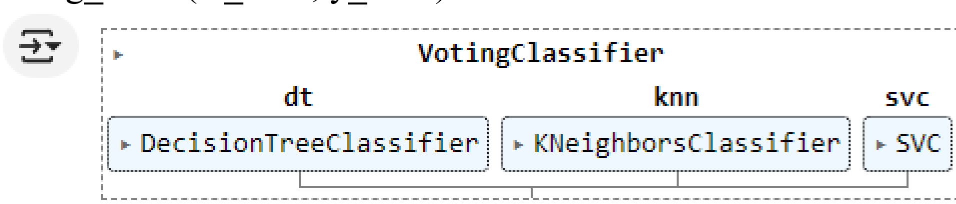
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42, subsample=0.8)
```

y_pred = gb_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)

```
Accuracy:  1.0
```

8.C Voting Ensemble

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
# Load the Iris Dataset
iris = load_iris()
X, y = iris.data, iris.target
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 42)
# Initialize the base Classifiers
clf1 = DecisionTreeClassifier(random_state = 42)
clf2 = KNeighborsClassifier()
clf3 = SVC(probability = True)
# Initialize the VotingClassifier with the base classifiers
voting_clf = VotingClassifier(estimators = [('dt', clf1), ('knn', clf2), ('svc', clf3)],
voting = 'hard')
voting_clf.fit(X_train, y_train)
```




```
# Make predicitons on the test data
y_pred = voting_clf.predict(X_test)
# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Voting Ensemble Accuracy: ", accuracy)
```

```
Voting Ensemble Accuracy:  1.0
```

Practical 9

Aim: Step for Deployment of Machine Learning Models.

Theory:

Deploying a machine learning model in the Runway IDE involves several steps to prepare your model and integrate it with Runway's platform. Here's a general outline of the process:

1. Prepare Your Model:
    - Ensure your model is trained and serialized into a format compatible with Runway's requirements. Runway currently supports models trained in TensorFlow, PyTorch, and ONNX formats.
    - Serialize your model and save it to a file format supported by Runway (.pb for TensorFlow, .onnx for ONNX, etc.).
    - Make sure your model is capable of accepting input data and producing output predictions according to Runway's expected format.

2. Set Up Your Workspace:
    - Log in to the Runway IDE and create a new workspace or open an existing one where you want to deploy your model.
    - Familiarize yourself with the workspace environment and understand how to navigate and interact with it.

3. Import Your Model:
    - In the Runway IDE, navigate to the "Models" tab.
    - Click on the "Import Model" button and follow the instructions to upload your serialized model file.
    - Once imported, your model will appear in the list of available models within the workspace.

4. Configure Your Model:
    - Select your imported model from the list and configure its settings according to your requirements.
    - Specify input and output details, such as data types, dimensions, and labels, to ensure compatibility with your model's input and output formats.

5. Connect Inputs and Outputs:

- In the workspace editor, define the inputs and outputs for your model by creating nodes and connecting them as needed.
- Inputs represent data sources or triggers that will activate your model, while outputs represent the results or actions produced by your model.

6. Test Your Model:
   - Use the built-in testing and debugging features of the Runway IDE to verify that your model is working as expected.
   - Test different input scenarios and examine the corresponding output predictions to ensure accuracy and reliability.

7. Deploy Your Model:
   - Once you're satisfied with your model's performance, deploy it within the Runway IDE by clicking on the appropriate button or command.
   - Follow the deployment wizard or prompts to finalize the deployment settings and launch your model into production.

8. Monitor and Manage:
   - Monitor the performance of your deployed model within the Runway IDE, keeping an eye on metrics, logs, and any potential issues or errors.
   - Make adjustments and updates to your model as needed, iterating on its design and configuration to optimize performance and usability.

9. Integrate with Other Tools:
   - Explore integration options to connect your deployed model with other tools, services, or workflows within the Runway IDE or external environments.
   - Leverage APIs, webhooks, or custom scripts to facilitate data exchange, automation, or communication with external systems.

10. Documentation and Collaboration:
    - Document your model deployment process, configurations, and usage instructions to facilitate collaboration and knowledge sharing within your team or community.
    - Share your deployed model with others and gather feedback to improve its functionality and usability over time.