PRACTICAL NO 1

Program 1- Water Jug Problem using DFS

Code:

```
start(2,0):-write(' 4lit Jug: 2 | 3lit Jug: 0|\n'),
write('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n'),
write('Goal Reached! Congrats!!\n'),
write('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n').
start(X,Y):-write(' Water Jug Game \n'),
write('Intial State: 4lit Jug- 0lit\n'),
write(' 3lit Jug- 0lit\n'),
write('Final State: 4lit Jug- 2lit\n'),
write(' 3lit Jug- 0lit\n'),
write('Follow the Rules: \n'),
write('Rule 1: Fill 4lit Jug\n'),
write('Rule 2: Fill 3lit Jug\n'),
write('Rule 3: Empty 4lit Jug\n'),
write('Rule 4: Empty 3lit Jug\n'),
write('Rule 5: Pour water from 3lit Jug to fill 4lit Jug\n'),
write('Rule 6: Pour water from 4lit Jug to fill 3lit Jug\n'),
write('Rule 7: Pour all of water from 3lit Jug to 4lit Jug\n'),
write('Rule 8: Pour all of water from 4lit Jug to 3lit Jug\n'),
write(' 4lit Jug: 0 | 3lit Jug: 0'),nl,
write(' Current Quantity :'),
write(' 4lit Jug: '),write(X),write('| 3lit Jug: '),
write(Y),write('|\n'),
write(' Enter the move::'),
read(N),
contains(X,Y,N).
contains(_,Y,1):-start(4,Y).
```

contains(X,_,2):-start(X,3).

contains(_,Y,3):-start(0,Y).

contains(X,_,4):-start(X,0).

contains(X,Y,5):-N is Y-4+X, start(4,N).

contains(X,Y,6):-N is X-3+Y, start(N,3).

contains(X,Y,7):-N is X+Y, start(N,0).

contains(X,Y,8):-N is X+Y, start(0,N).

Output Window Commands:

start(0,0).

Enter Move :: 1.

Enter Move :: 6.

Enter Move :: 4.

Enter Move :: 8.

Enter Move :: 1.

Enter Move :: 6.

Enter Move :: 4.

PRACTICAL NO 2

Problem 2 : 8 Puzzle problem using prolog

Code:

```
ids :-
   start(State),
   length(Moves, N),
   hill([State], Moves, Path), !,
   show([start|Moves], Path),
   format('~nmoves = ~w~n', [N]).

hill([State|States], [], Path) :-
   goal(State), !,
   reverse([State|States], Path).

hill([State|States], [Move|Moves], Path) :-
   move(State, Next, Move),
   not(memberchk(Next, [State|States])),
   hill([Next,State|States], Moves, Path).

show([], _).
show([Move|Moves], [State|States]) :-
   State = state(A,B,C,D,E,F,G,H,J),
   format('~n~w~n~n', [Move]),
   format('~w ~w ~w~n',[A,B,C]),
   format('~w ~w ~w~n',[D,E,F]),
   format('~w ~w ~w~n',[G,H,J]),
   show(Moves, States).

% Empty position is marked with '*'
start( state(0,1,*,2,3,4,5,6,7) ).
```

```
goal( state(*,0,1,2,3,4,5,6,7) ).

move( state(A,*,C,D,E,F,G,H,J), state(*,A,C,D,E,F,G,H,J), left ).
move( state(A,B,*,D,E,F,G,H,J), state(A,*,B,D,E,F,G,H,J), left ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,*,D,F,G,H,J), left ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,*,E,G,H,J), left ).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,*,G,J), left ).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,F,G,*,H), left ).

move( state(*,B,C,D,E,F,G,H,J), state(B,*,C,D,E,F,G,H,J), right).
move( state(A,*,C,D,E,F,G,H,J), state(A,C,*,D,E,F,G,H,J), right).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,E,*,F,G,H,J), right).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,F,*,G,H,J), right).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,D,E,F,H,*,J), right).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,G,J,*), right).

move( state(A,B,C,*,E,F,G,H,J), state(*,B,C,A,E,F,G,H,J), up).
move( state(A,B,C,D,*,F,G,H,J), state(A,*,C,D,B,F,G,H,J), up).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,*,D,E,C,G,H,J), up).
move( state(A,B,C,D,E,F,*,H,J), state(A,B,C,*,E,F,D,H,J), up).
move( state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,*,F,G,E,J), up).
move( state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,*,G,H,F), up).

move( state(*,B,C,D,E,F,G,H,J), state(D,B,C,*,E,F,G,H,J), down ).
move( state(A,*,C,D,E,F,G,H,J), state(A,E,C,D,*,F,G,H,J), down ).
move( state(A,B,*,D,E,F,G,H,J), state(A,B,F,D,E,*,G,H,J), down ).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,G,E,F,*,H,J), down ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,H,F,G,*,J), down ).
move( state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,E,J,G,H,*), down )
```

Output :

```
[4]  ?- ids.

start

0 1 *
2 3 4
5 6 7

left

0 * 1
2 3 4
5 6 7

left

* 0 1
2 3 4
5 6 7

moves = 2
true.
```

PRACTICAL NO 3

Program 3- Tic-Tac-Toe using Prolog

Code:

```
% To play a game with the computer, type
% play.

% 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,_,_].
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.

% Predicates to support playing a game with the user:
x_can_win_in_one(Board) :- move(Board, x, Newboard), win(Newboard, x).

% The predicate validate generates the computer's (playing o) reponse
% from the current Board.
validate(Board,Newboard) :-
  move(Board, o, Newboard),
  win(Newboard, o),
  !.
validate(Board,Newboard) :-
  move(Board, o, Newboard),
  not(x_can_win_in_one(Newboard)).
validate(Board,Newboard) :-
  move(Board, o, Newboard).

% 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.


play :- 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),
  validate(Newboard, Newnewboard),
  display(Newnewboard),
  playfrom(Newnewboard).
```

Output :

SWI-Prolog (AMD64, Multi-threaded, version 9.2.0)

File Edit Settings Run Debug Help

```
?- play.
You play X by entering integer positions followed by a period.
[1,2,3]
[4,5,6]
[7,8,9]

|: 1.
[x,b,b]
[b,b,b]
[b,b,b]

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

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

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

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

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

|: 4.
[x,o,o]
[x,x,b]
[x,b,o]

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

You win!
true .

?-
```

PRACTICAL NO 4

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

1. NUMPY

[1]:
```
!pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (1.26.4)
```

[5]:
```
import numpy as np
```

[7]:
```
#create an array
digits=np.array([[1,2,3],
                [4,5,6],
                [7,8,9]
                ])
```

[9]:
```
digits
```

[9]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[13]:
```
#addition of two integers
a=5
b=5
c=a+b
c
```

[13]:
```
10
```

[21]:
```
# addition of two arrays
# With two input arrays
arr = np.array([2, 6, 9])
arr1 = np.array([5, 8, 12])
arr2 = np.add(arr, arr1)
```

[23]:
```
arr2
```

[23]:
```
array([ 7, 14, 21])
```

[25]:
```
arr+arr1
```

[25]:
```
array([ 7, 14, 21])
```

```
[29]: digits.T

[29]: array([[1, 4, 7],
             [2, 5, 8],
             [3, 6, 9]])

[31]: digits.transpose()

[31]: array([[1, 4, 7],
             [2, 5, 8],
             [3, 6, 9]])
```

```
[37]: sortarray=np.array([
             [1, 7, 4],
             [2, 8, 5],
             [9, 6, 1]])
      np.sort(sortarray)

[37]: array([[1, 4, 7],
             [2, 5, 8],
             [1, 6, 9]])

[39]: np.sort(sortarray,axis=none)

      ---------------------------------------------------------------------------
      NameError                                 Traceback (most recent call last)
      Cell In[39], line 1
      ----> 1 np.sort(sortarray,axis=none)

      NameError: name 'none' is not defined

[41]: np.sort(sortarray,axis=None)

[41]: array([1, 1, 2, 4, 5, 6, 7, 8, 9])

[43]: np.sort(sortarray,axis=0)

[43]: array([[1, 6, 1],
             [2, 7, 4],
             [9, 8, 5]])
```

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- f - float
- M - datetime
- S - string

```
[5]: arr = np.array([1, 2, 3, 4], dtype='S')
     print(arr)
     print(arr.dtype)

     [b'1' b'2' b'3' b'4']
     |S1

[7]: arr

[7]: array([b'1', b'2', b'3', b'4'], dtype='|S1')

[ ]:

[9]: arr = np.array([1, 2, 3, 4], dtype='i4')

     print(arr)
     print(arr.dtype)

     [1 2 3 4]
     int32
```

2. Pandas

```
[1]: import pandas as pd

[3]: #Create a Dataframe
     data={
         'apples':[3,2,0,1],
         'oranges': [0,3,7,2]
     }
     purchases=pd.DataFrame(data)

[5]: purchases
```

[5]:

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

```
[7]: purchases=pd.DataFrame(data,index=['June','Robert,'David','Lily'])
     purchases

       Cell In[7], line 1
         purchases=pd.DataFrame(data,index=['June','Robert,'David','Lily'])
                                                                     ^
     SyntaxError: unterminated string literal (detected at line 1)

[9]: purchases=pd.DataFrame(data,index=['June','Robert','David','Lily'])
     purchases
```

[9]:

	apples	oranges
June	3	0
Robert	2	3
David	0	7
Lily	1	2

3. SciPy

[1]:
```
import numpy as np
```

[3]:
```
A=np.array([[1,2],[3,4]])
#linear algebra determinant of a matrix from scipy
from scipy import linalg
linalg.det(A)
```

[3]:
```
-2.0
```

4. Matplotlib

[1]:
```
from matplotlib import pyplot as plt
```

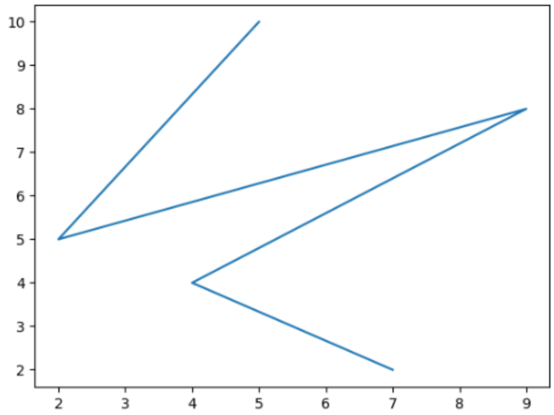
```
Matplotlib is building the font cache; this may take a moment.
```

[3]:
```
#x-axis values
x=[5,2,,9,4,7]
#y-axis values
y=[10,5,8,4,2]

plt.plot(x,y)
```

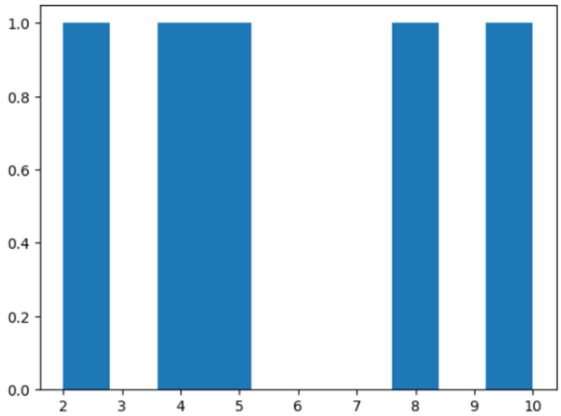
```
  Cell In[3], line 2
    x=[5,2,,9,4,7]
           ^
SyntaxError: invalid syntax
```

```
[5]: #x-axis values
     x=[5,2,9,4,7]
     #y-axis values
     y=[10,5,8,4,2]

     plt.plot(x,y)
```

```
[5]: [<matplotlib.lines.Line2D at 0x27454fba930>]
```

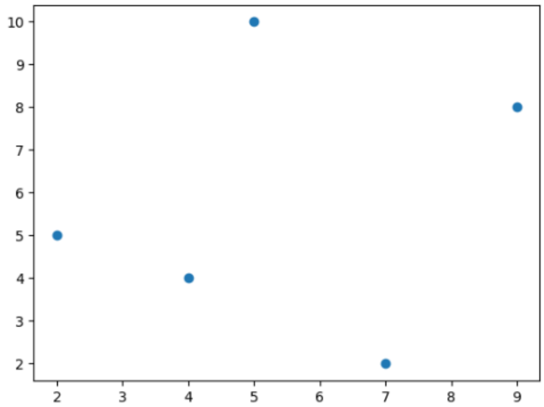


```
[7]: #histogram

     #y-axis values
     y=[10,5,8,4,2]

     #function to plot histogram
     plt.hist(y)

     #function to show the plot
     plt.show()
```



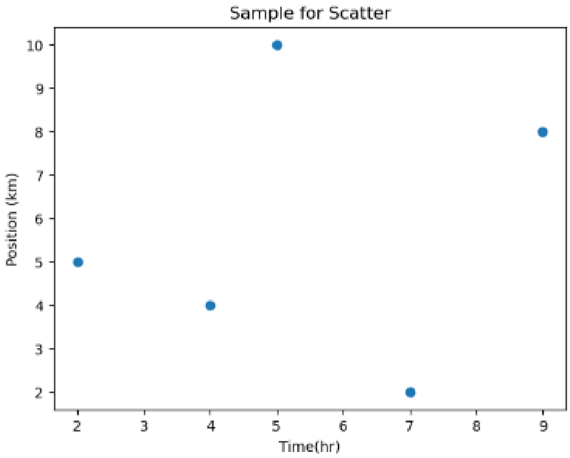
```
[11]: #scatter plot
      #x-axis values
      x=[5,2,9,4,7]
      #y-axis values
      y=[10,5,8,4,2]

      plt.scatter(x,y)
```

```
[11]: <matplotlib.collections.PathCollection at 0x2745528c3b0>
```



```
[13]: #scatter plot
      #x-axis values
      x=[5,2,9,4,7]
      #y-axis values
      y=[10,5,8,4,2]

      plt.scatter(x,y)

      plt.title("Sample for Scatter")

      #Labeling the axis
      plt.xlabel("Time(hr)")
      plt.ylabel("Position (km)")
```

```
[13]: Text(0, 0.5, 'Position (km)')
```



5.SciKit Learn

[7]:
```
import pandas as pd
from sklearn.datasets import load_wine

wine_data = load_wine()

# Convert data to pandas dataframe
wine_df = pd.DataFrame(wine_data.data, columns=wine_data.feature_names)

# Add the target label
wine_df["target"] = wine_data.target

# Take a preview
wine_df.head()
```

[7]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_dilut
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	

Building the model

Thanks to sklearn, building a machine learning model is extremely simple.

We are going to build three models to predict the class of wine:

1. Logistic regression
2. Support vector machine
3. Decision tree classifier

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

PRACTICAL NO 5

Design OR gate using perceptron

```
import numpy as np
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iterations=1):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        y_ = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.n_iterations):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation_function(linear_output)

                update = self.learning_rate * (y_[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    def activation_function(self, x):
        return np.where(x>=0, 1, 0)

    def predict(self, X):
```

```
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self.activation_function(linear_output)
        return y_predicted

# OR gate inputs and outputs
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 1])
# Initialize and train the perceptron
perceptron = Perceptron(learning_rate=0.1, n_iterations=6)
perceptron.fit(X, y)

# Test the perceptron
predictions = perceptron.predict(X)
predictions
```

Output :

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

PRACTICAL NO 6

Design AND Gate using Adaline Algorithm

```
import numpy as np
class Adaline:
    def __init__(self, input_size, learning_rate=0.1, epochs=100):
        self.weights = np.zeros(input_size)
        self.bias=0
        self.learning_rate = learning_rate
        self.epochs = epochs
    def activation(self, X): # X is input
        return X
    def predict(self,X):
        return self.activation(np.dot(X, self.weights)+self.bias)
    def train(self, X,y):
        for epoch in range(self.epochs):
            for i in range(len(X)):
                prediction=self.predict(X[i])
                error = y[i]-prediction
                self.weights+=self.learning_rate*error*X[i]
                self.bias+=self.learning_rate*error
    def evaluate(self, X):
        return np.where(self.predict(X) >=0.5,1,0)
X=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,0,0,1])
adaline=Adaline(input_size=2,learning_rate=0.1,epochs=100)
adaline.train(X,y)
predictions = adaline.evaluate(X)
for i, prediction in enumerate(predictions):
    print(f"Input: {X[i]}=>Predicted: {prediction} => Actual: {y[i]}")
```

Output-

Input: [0 0]=>Predicted: 0 => Actual: 0

Input: [0 1]=>Predicted: 0 => Actual: 0

Input: [1 0]=>Predicted: 0 => Actual: 0

Input: [1 1]=>Predicted: 1 => Actual: 1

PRACTICAL NO 7

Stochastic Gradient Descent Algorithm

(Add-on Learning)

```
import numpy as np
# Generate synthetic data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
def sgd(X, y, learning_rate=0.1, epochs=1000, batch_size=1):
    m = len(X)
    theta = np.random.randn(2, 1)

    # Add a bias term to X (X_0 = 1)
    X_bias = np.c_[np.ones((m, 1)), X]

    cost_history = []

    for epoch in range(epochs):
        # Shuffle the data at the beginning of each epoch
        indices = np.random.permutation(m)
        X_shuffled = X_bias[indices]
        y_shuffled = y[indices]

        for i in range(0, m, batch_size):
            # Select a mini-batch or a single sample
            X_batch = X_shuffled[i:i+batch_size]
            y_batch = y_shuffled[i:i+batch_size]

            # Compute the gradient
            gradients = 2 / batch_size * X_batch.T.dot(X_batch.dot(theta) - y_batch)
```

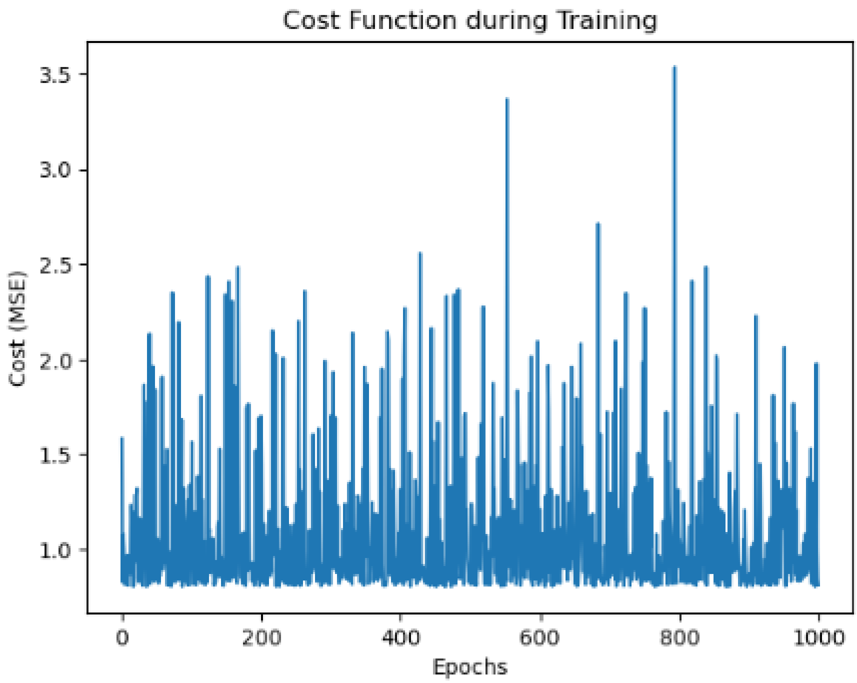
```
            # Update the parameters (theta)
            theta -= learning_rate * gradients

        # Calculate and record the cost (Mean Squared Error)
            predictions = X_bias.dot(theta)
        cost = np.mean((predictions - y) ** 2)
        cost_history.append(cost)

        # Print progress every 100 epochs
        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Cost: {cost}")

    return theta, cost_history
  # Train the model using SGD
  theta_final, cost_history = sgd(X, y, learning_rate=0.1, epochs=1000, batch_size=1)
```

```
Epoch 0, Cost: 1.581821686856939
Epoch 100, Cost: 1.5664692700155733
Epoch 200, Cost: 1.4445422391173144
Epoch 300, Cost: 1.7037674963102662
Epoch 400, Cost: 0.9101999515899212
Epoch 500, Cost: 0.8184497904316664
Epoch 600, Cost: 0.8352333304446237
Epoch 700, Cost: 0.8542729530055074
Epoch 800, Cost: 1.0508310318687628
Epoch 900, Cost: 0.8261971232182218
```

```
  import matplotlib.pyplot as plt
# Plot the cost history
plt.plot(cost_history)
plt.xlabel('Epochs')
plt.ylabel('Cost (MSE)')
```

plt.title('Cost Function during Training')

plt.show()



Plot the data and the regression line

plt.scatter(X, y, color='blue', label='Data points')

plt.plot(X, np.c_[np.ones((X.shape[0], 1)), X].dot(theta_final), color='red', label='SGD fit line')

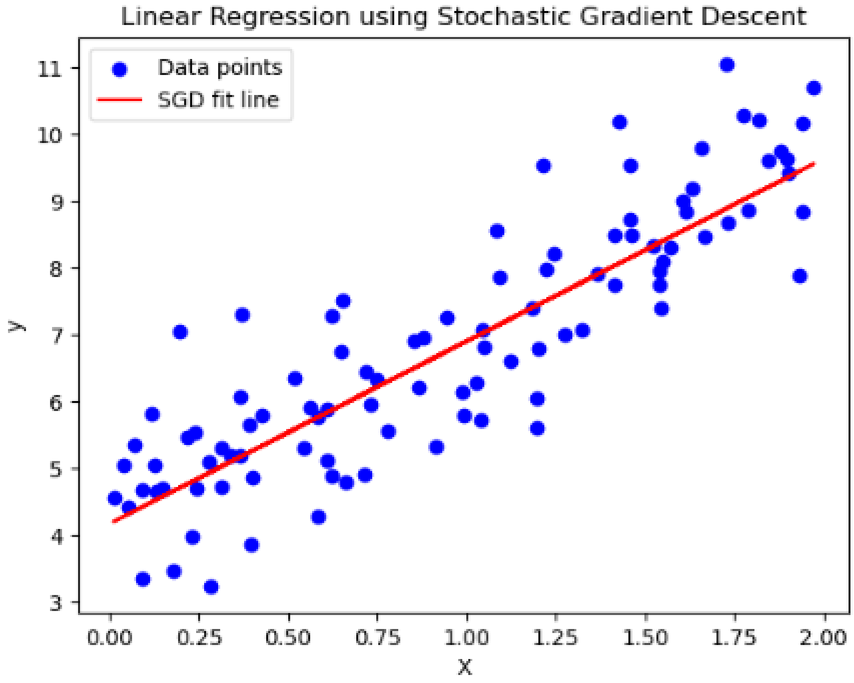
plt.xlabel('X')

plt.ylabel('y')

plt.title('Linear Regression using Stochastic Gradient Descent')

plt.legend()

plt.show()


Linear Regression using Stochastic Gradient Descent
Data points
SGD fit line
y
X
11
10
9
8
7
6
5
4
3
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

PRACTICAL NO 8

PCA

Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, MinMaxScaler, PowerTransformer, LabelEncoder

from sklearn.feature_selection import SelectKBest, f_classif, VarianceThreshold

from sklearn.decomposition import PCA


# Load the healthcare dataset

file_path = "health_dataset.csv"

df = pd.read_csv(file_path)


# Display first few rows

display(df.head())


# Identify non-numeric columns

categorical_columns = df.select_dtypes(include=['object']).columns

print("Categorical Columns:", categorical_columns)


# Convert categorical columns using Label Encoding

label_encoders = {}

for col in categorical_columns:

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le


# Ensure all columns are numeric

df = df.apply(pd.to_numeric, errors='coerce')  # Converts non-numeric values to NaN

df.fillna(0, inplace=True)  # Replace NaNs with 0
```

```
# Separate features and target
target_column = 'Disease Risk Score'
if target_column not in df.columns:
    raise ValueError("Target column not found in dataset")

X = df.drop(columns=[target_column])
y = df[target_column]

# Check for missing values in target and encode if necessary
if y.isnull().sum() > 0:
    y.fillna(y.mode()[0], inplace=True)  # Replace NaN with most frequent value

if y.dtype == 'object':
    le = LabelEncoder()
    y = le.fit_transform(y)

# Remove constant features
var_thresh = VarianceThreshold(threshold=0)
X = var_thresh.fit_transform(X)

# Feature Selection using ANOVA F-score
selector = SelectKBest(score_func=f_classif, k=min(10, X.shape[1]))  # Ensure k does not exceed feature count
X_selected = selector.fit_transform(X, y)
selected_features = [col for col, keep in zip(df.drop(columns=[target_column]).columns, selector.get_support()) if keep]
print("Selected Features:", selected_features)

# Normalization using Min-Max Scaling
scaler = MinMaxScaler()
```

```
X_normalized = scaler.fit_transform(X_selected)

# Transformation using Power Transform (Box-Cox or Yeo-Johnson)
power_transformer = PowerTransformer(method='yeo-johnson')  # Use 'box-cox' if no negative values
X_transformed = power_transformer.fit_transform(X_normalized)

# Dimensionality Reduction using PCA
pca_components = min(5, X_transformed.shape[1])  # Ensure PCA components do not exceed feature count
pca = PCA(n_components=pca_components)
X_pca = pca.fit_transform(X_transformed)
print("Explained Variance Ratio:", pca.explained_variance_ratio_)

# Convert processed data back to DataFrame
processed_df = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(X_pca.shape[1])])
processed_df['Disease Risk Score'] = y.reset_index(drop=True)

# Save processed dataset
processed_df.to_csv("processed_health_dataset.csv", index=False)
print("Processed dataset saved successfully.")
```

OUTPUT

	ID	Age	Height (cm)	Weight (kg)	Blood Pressure (BP)	Cholesterol Level	Diabetes	Physical Activity (hours/week)	Smoking Habit	Disease Risk Score	Unnamed: 10
0	1	56	159	77	122	High	1	1.00	0	59.00	NaN
1	2	69	185	77	157	Normal	0	0.53	1	48.00	NaN
2	3	46	163	93	122	Low	1	9.59	0	42.05	NaN
3	4	32	180	79	103	Normal	1	8.47	0	78.47	NaN
4	5	60	197	111	110	Normal	1	3.55	1	63.94	NaN

PRACTICAL NO 9

Statement 1: Build and train a Logistic Regression Model to do binary classification of iris flowers using the iris dataset.

In particular, the model should predict whether a particular iris flower instance belongs to the class Iris Virginica or not using

#only petal width as the input feature.

CODE

```
import numpy as np

from sklearn import datasets


iris = datasets.load_iris()

print(type(iris))

print(list(iris.keys()))

X = iris["data"][:,3:] # petal width

y = (iris["target"] == 2).astype(np.int64) # 1 if Iris-Virginica, else 0
```

Output:

<class 'sklearn.utils._bunch.Bunch'>

['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module']

Code:

```
from sklearn.linear_model import LogisticRegression


log_reg = LogisticRegression(solver="lbfgs", random_state=42)

log_reg.fit(X,y)
```

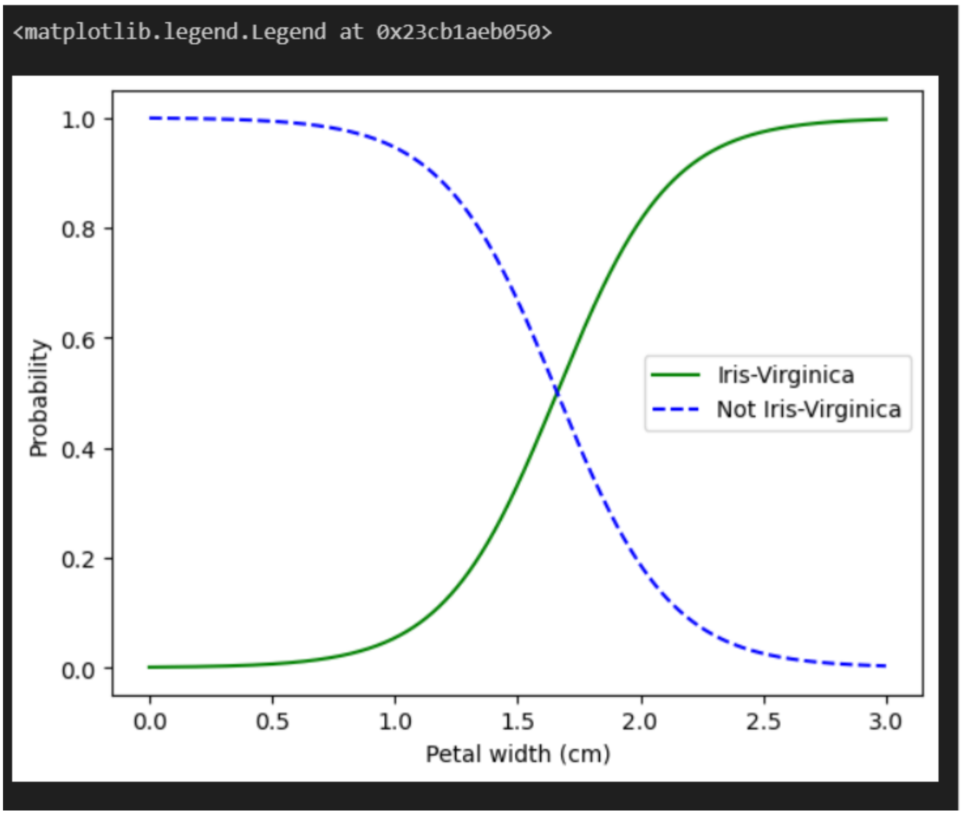
Output

```
LogisticRegression
LogisticRegression(random_state=42)
```

Code

```
import matplotlib.pyplot as plt
X_new = np.linspace(0,3,1000).reshape(-1,1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:,1],"g-")
plt.plot(X_new, y_proba[:,0], "b--")
plt.xlabel('Petal width (cm)')
plt.ylabel('Probability')
plt.legend(['Iris-Virginica','Not Iris-Virginica'])
```

Output



Code

```
log_reg.predict([[1.7],[1.5]])
```

Output

```
array([1, 0], dtype=int64)
```

Code

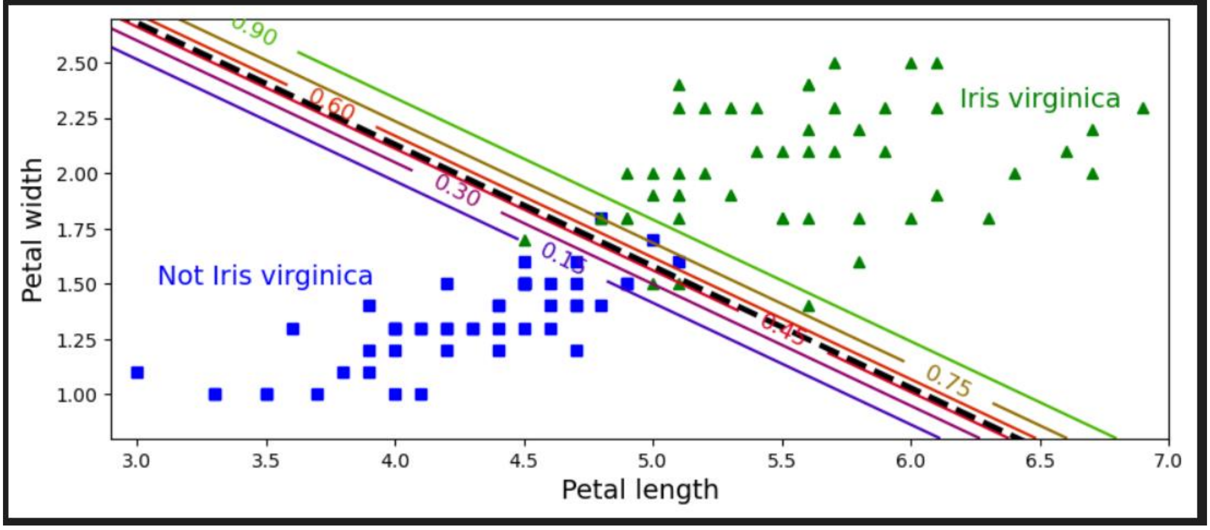
```
from sklearn.linear_model import LogisticRegression

X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.int64)

log_reg2 = LogisticRegression(solver="lbfgs", C=10**10, random_state=42)
log_reg2.fit(X, y)

x0, x1 = np.meshgrid(
        np.linspace(2.9, 7, 500).reshape(-1, 1),
        np.linspace(0.8, 2.7, 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]
print(X_new.shape)

y_proba = log_reg2.predict_proba(X_new)

plt.figure(figsize=(10, 4))
plt.plot(X[y==0, 0], X[y==0, 1], "bs")
plt.plot(X[y==1, 0], X[y==1, 1], "g^")
zz = y_proba[:, 1].reshape(x0.shape)
contour = plt.contour(x0, x1, zz, cmap=plt.cm.brg)
left_right = np.array([2.9, 7])
boundary = -(log_reg2.coef_[0][0] * left_right + log_reg2.intercept_[0]) /
log_reg2.coef_[0][1]
plt.clabel(contour, inline=1, fontsize=12)
plt.plot(left_right, boundary, "k--", linewidth=3)
plt.text(3.5, 1.5, "Not Iris virginica", fontsize=14, color="b", ha="center")
plt.text(6.5, 2.3, "Iris virginica", fontsize=14, color="g", ha="center")
plt.xlabel("Petal length", fontsize=14)
```

plt.ylabel("Petal width", fontsize=14)

plt.axis([2.9, 7, 0.8, 2.7])

Output

```
(100000, 2)

(2.9, 7.0, 0.8, 2.7)
```



PRACTICAL NO 10

LINEAR SVM

Code:

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

def plot_svc_decision_boundary(svm_clf, xmin, xmax):
    w = svm_clf.coef_[0]
    b = svm_clf.intercept_[0]

    # At the decision boundary, w0*x0 + w1*x1 + b = 0
    # => x1 = -w0/w1 * x0 - b/w1
    x0 = np.linspace(xmin, xmax, 200)
    decision_boundary = -w[0]/w[1] * x0 - b/w[1]

    margin = 1/w[1]
    gutter_up = decision_boundary + margin
    gutter_down = decision_boundary - margin

    svs = svm_clf.support_vectors_
    plt.scatter(svs[:, 0], svs[:, 1], s=180, facecolors='#FFAAAA')
    plt.plot(x0, decision_boundary, "k-", linewidth=2)
    plt.plot(x0, gutter_up, "k--", linewidth=2)
    plt.plot(x0, gutter_down, "k--", linewidth=2)
```

Code

```
from sklearn.svm import SVC
from sklearn import datasets
import numpy as np
# Load Iris dataset
```

```
iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # Select petal length and petal width
y = iris["target"]

# Select only Setosa and Versicolor classes
setosa_or_versicolor = (y == 0) | (y == 1)
X = X[setosa_or_versicolor]
y = y[setosa_or_versicolor]

# SVM Classifier model with a large but finite C value
svm_clf = SVC(kernel="linear", C=1e10)  # Large C approximates a hard margin
svm_clf.fit(X, y)

# Make a prediction
prediction = svm_clf.predict([[2.4, 3.1]])
print("Predicted class:", prediction[0])
```

Output

Predicted class: 1

Code

```
#plot the decision boundaries
import numpy as np

plt.figure(figsize=(12,3.2))

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
svm_clf.fit(X_scaled, y)

plt.plot(X_scaled[:, 0][y==1], X_scaled[:, 1][y==1], "bo")
```

plt.plot(X_scaled[:, 0][y==0], X_scaled[:, 1][y==0], "ms")

plot_svc_decision_boundary(svm_clf, -2, 2)

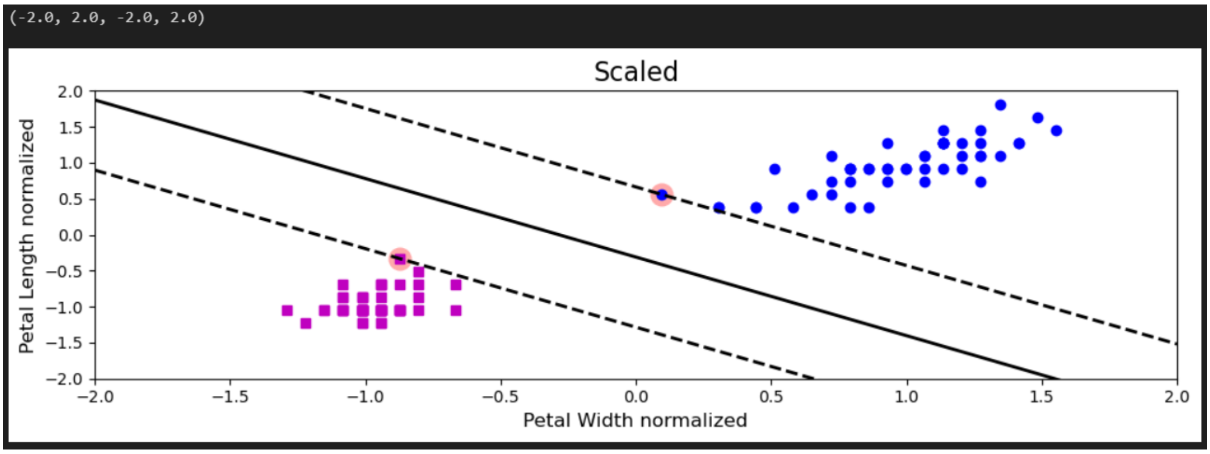
plt.xlabel("Petal Width normalized", fontsize=12)

plt.ylabel("Petal Length normalized", fontsize=12)

plt.title("Scaled", fontsize=16)

plt.axis([-2, 2, -2, 2])

Output



NON-LINEAR SVM

Code

from sklearn.datasets import make_moons

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import PolynomialFeatures

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

Code

import numpy as np

%matplotlib inline

import matplotlib

import matplotlib.pyplot as plt

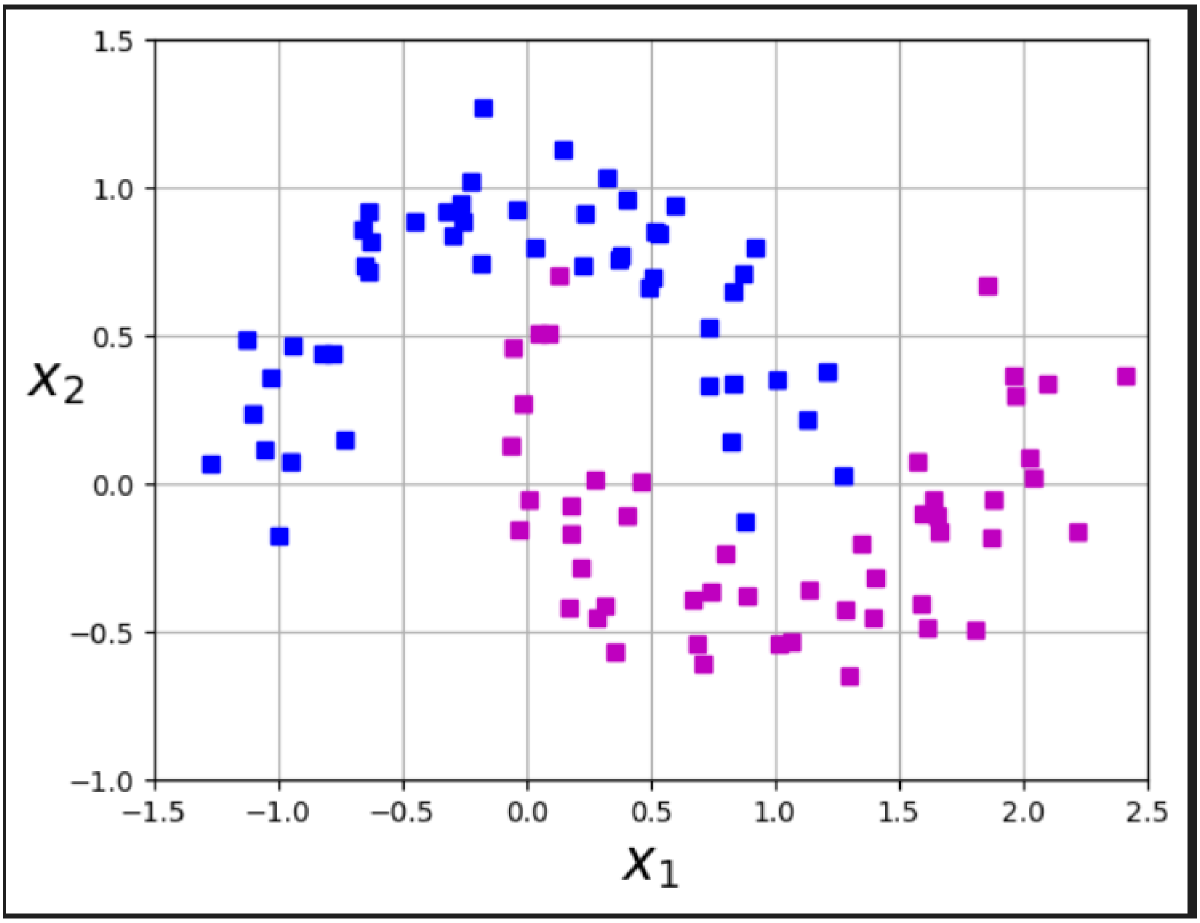
Code

from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.15, random_state=42)

```
#define a function to plot the dataset
def plot_dataset(X, y, axes):
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "ms")
    plt.axis(axes)
    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$", fontsize=20)
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)
#Let's have a look at the data we have generated
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])
plt.show()
```

Output



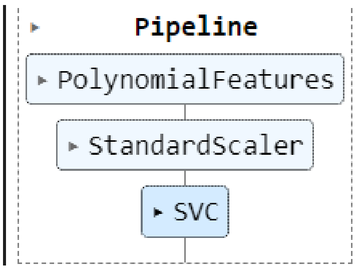
Code

```
#define a function plot the decision boundaries
def plot_predictions(clf, axes):
    #create data in continous linear space
    x0s = np.linspace(axes[0], axes[1], 100)
    x1s = np.linspace(axes[2], axes[3], 100)
    x0, x1 = np.meshgrid(x0s, x1s)
    X = np.c_[x0.ravel(), x1.ravel()]
    y_pred = clf.predict(X).reshape(x0.shape)
    y_decision = clf.decision_function(X).reshape(x0.shape)
    plt.contourf(x0, x1, y_pred, cmap=plt.cm.brg, alpha=0.2)
    plt.contourf(x0, x1, y_decision, cmap=plt.cm.brg, alpha=0.1)
```

Code

```
#C controls the width of the street
#Degree of data
#create a pipeline to create features, scale data and fit the model
polynomial_svm_clf = Pipeline((
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scalar", StandardScaler()),
    ("svm_clf", SVC(kernel="poly", degree=10, coef0=1, C=5))
))
#call the pipeline
polynomial_svm_clf.fit(X,y)
```

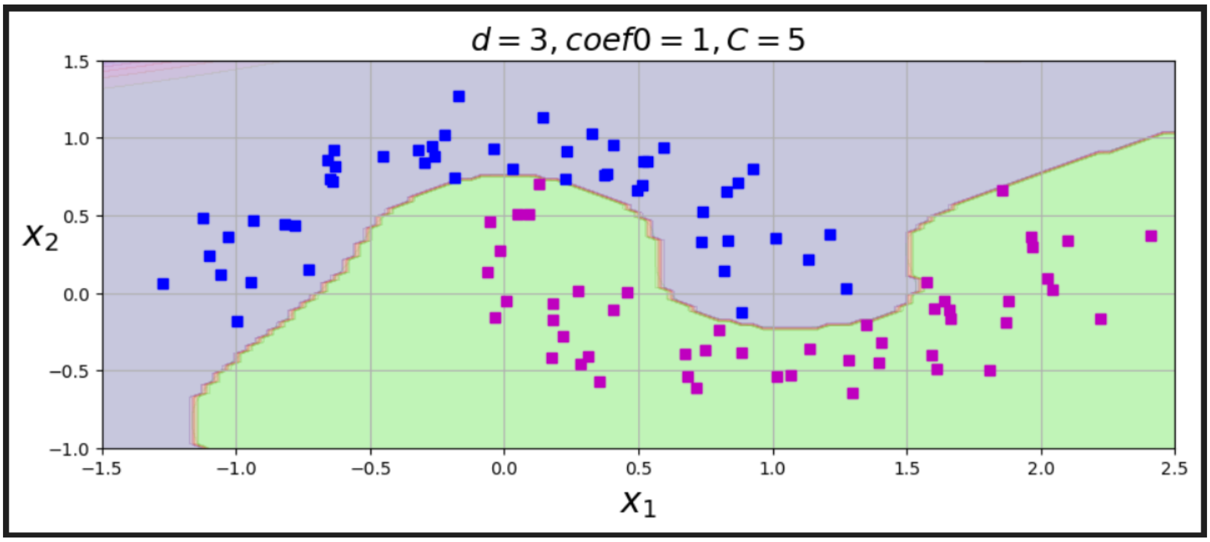
Output



Code

```
#plot the decision boundaries
plt.figure(figsize=(11, 4))
#plot the decision boundaries
plot_predictions(polynomial_svm_clf, [-1.5, 2.5, -1, 1.5])
#plot the dataset
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])
plt.title(r"$d=3, coef0=1, C=5$", fontsize=18)
plt.show()
```

Output



PRACTICAL NO 11

Implement Elbow method for K means Clustering

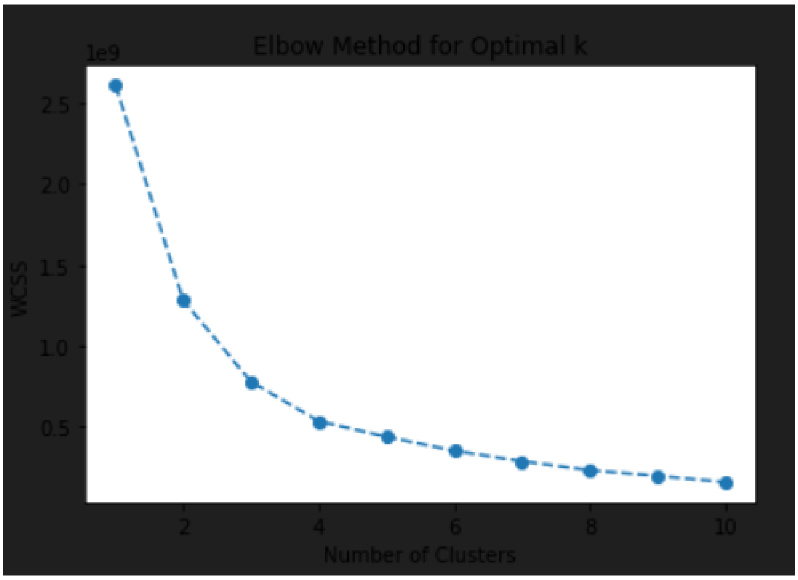
Code

```
!pip install --user threadpoolctl==3.1.0
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Load the dataset
df = pd.read_csv("clustering.csv")

# Display first few rows of the dataset
print(df.head())

# Drop missing values
df_cleaned = df.dropna()

# Selecting numerical columns for clustering
numerical_cols = df_cleaned.select_dtypes(include=[np.number]).columns
print("Numerical columns used for clustering:", numerical_cols.tolist())

# Feature selection for clustering (Modify as needed)
X = df_cleaned[numerical_cols]

# Apply the Elbow Method
wcss = []  # Within-cluster sum of squares
for i in range(1, 11):  # Trying different cluster numbers from 1 to 10
    kmeans = KMeans(n_clusters=i, random_state=42, n_init=10)
    kmeans.fit(X)
```

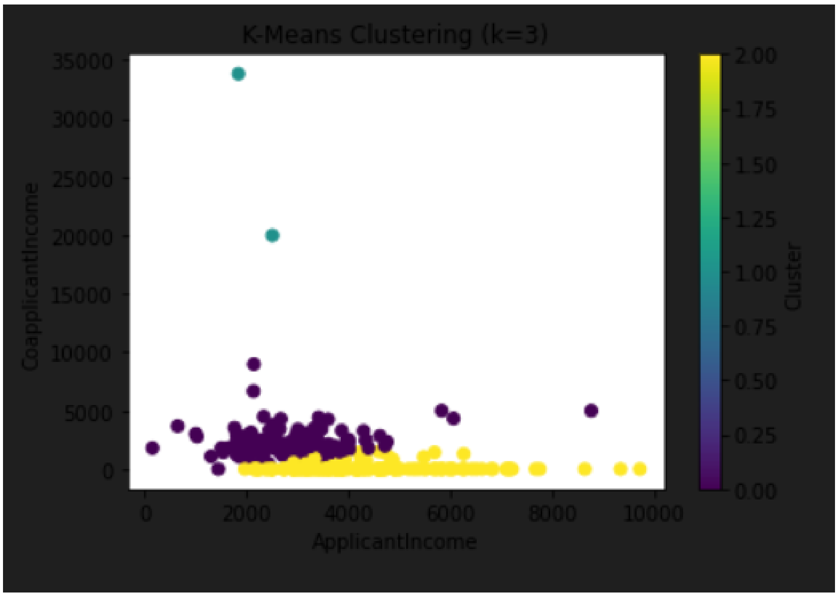
```
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method

plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal k')
plt.show()

# Choose optimal k (Modify based on the elbow plot observation)
k_optimal = 3  # Example choice, change based on your dataset

# Apply K-Means with the optimal number of clusters
kmeans = KMeans(n_clusters=k_optimal, random_state=42, n_init=10)
df_cleaned['Cluster'] = kmeans.fit_predict(X)

# Display clustered data
print(df_cleaned.head())

# Plot the clusters (for 2D visualization, choose two relevant features)
plt.scatter(df_cleaned[numerical_cols[0]], df_cleaned[numerical_cols[1]], c=df_cleaned['Cluster'], cmap='viridis')
plt.xlabel(numerical_cols[0])
plt.ylabel(numerical_cols[1])
plt.title(f'K-Means Clustering (k={k_optimal})')
plt.colorbar(label='Cluster')
plt.show()
```

Output

```
     Loan_ID Gender Married Dependents     Education Self_Employed  \
0   LP001003   Male     Yes          1      Graduate            No
1   LP001005   Male     Yes          0      Graduate           Yes
2   LP001006   Male     Yes          0  Not Graduate            No
3   LP001008   Male      No          0      Graduate            No
4   LP001013   Male     Yes          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             4583             1508.0       128.0             360.0
1             3000                0.0        66.0             360.0
2             2583             2358.0       120.0             360.0
3             6000                0.0       141.0             360.0
4             2333             1516.0        95.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Rural           N
1             1.0         Urban           Y
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```



```
df_cleaned['Cluster'] = kmeans.fit_predict(X)
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001003   Male     Yes          1      Graduate            No
1  LP001005   Male     Yes          0      Graduate           Yes
2  LP001006   Male     Yes          0  Not Graduate            No
3  LP001008   Male      No          0      Graduate            No
4  LP001013   Male     Yes          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             4583             1508.0       128.0             360.0
1             3000                0.0        66.0             360.0
2             2583             2358.0       120.0             360.0
3             6000                0.0       141.0             360.0
4             2333             1516.0        95.0             360.0

   Credit_History Property_Area Loan_Status  Cluster
0             1.0         Rural           N        2
1             1.0         Urban           Y        2
2             1.0         Urban           Y        0
3             1.0         Urban           Y        2
4             1.0         Urban           Y        0
```



PRACTICAL NO 12

Implementation of Bagging Algorithm: Random Forest

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA

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

# Split 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)

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Accuracy of Random Forest Classifier: {accuracy_rf * 100:.2f}%')

# AdaBoost Classifier
```

```
adaboost = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1), n_estimators=50, random_state=42)
adaboost.fit(X_train, y_train)
y_pred_adaboost = adaboost.predict(X_test)
accuracy_adaboost = accuracy_score(y_test, y_pred_adaboost)
print(f'Accuracy of AdaBoost Classifier: {accuracy_adaboost * 100:.2f}%')

# Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb_classifier.fit(X_train, y_train)
y_pred_gb = gb_classifier.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print(f'Accuracy of Gradient Boosting Classifier: {accuracy_gb * 100:.2f}%')

# Voting Classifier (Ensemble of Logistic Regression, Decision Tree, and Random Forest)
voting_classifier = VotingClassifier(estimators=[
    ('lr', LogisticRegression()),
    ('dt', DecisionTreeClassifier()),
    ('rf', RandomForestClassifier(n_estimators=100))
], voting='hard')
voting_classifier.fit(X_train, y_train)
y_pred_voting = voting_classifier.predict(X_test)
accuracy_voting = accuracy_score(y_test, y_pred_voting)
print(f'Accuracy of Voting Classifier: {accuracy_voting * 100:.2f}%')

# Reduce dimensions for visualization
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Scatter plot of the dataset
```

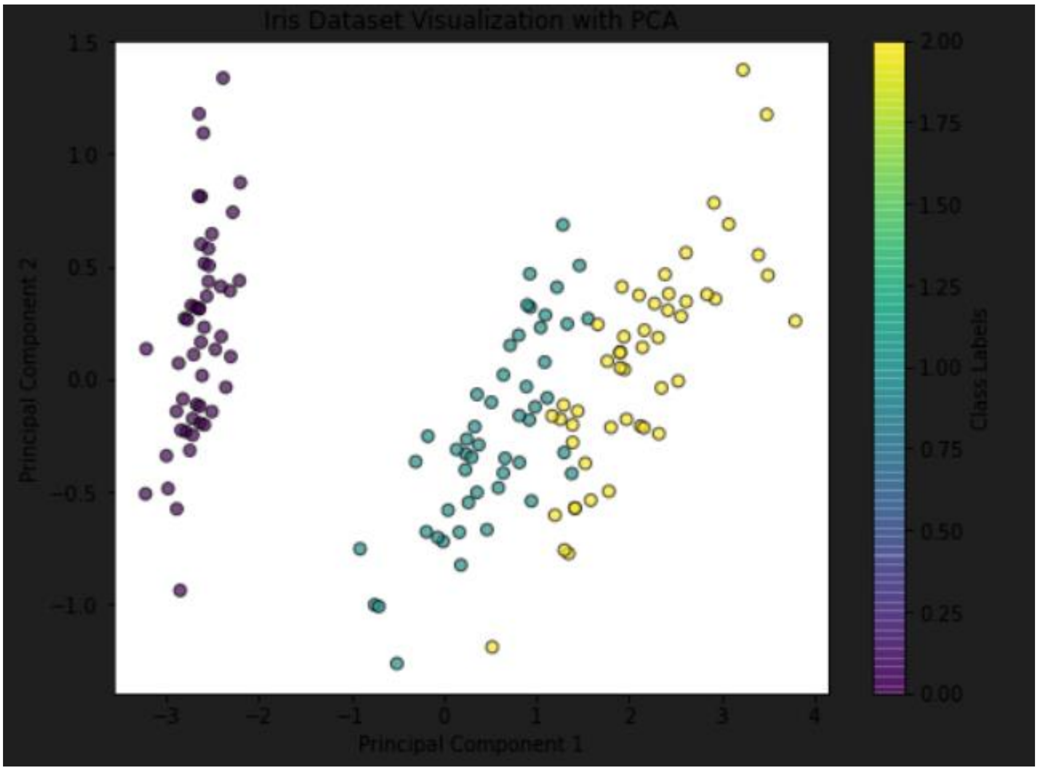
```
plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y, cmap='viridis', edgecolor='k', alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Iris Dataset Visualization with PCA')
plt.colorbar(label='Class Labels')
plt.show()
```

Output

Accuracy of Random Forest Classifier: 100.00%

Accuracy of AdaBoost Classifier: 100.00%

Accuracy of Gradient Boosting Classifier: 100.00%



PRACTICAL NO 13

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

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA

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

# Split 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 and train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Make predictions
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Random Forest Classifier: {accuracy * 100:.2f}%')
```

OUTPUT

Accuracy of Random Forest Classifier: 100.00%