Table of Contents :

1. implement the Tic-Tac-Toe game using Python. ............................................................... 2

2. implement the 8-puzzle problem using Python ............................................................... 7

3. water jug problem using the given rules to achieve the goal state. ................................ 10

4. implement the AO\* algorithm to determine the optimal cost path. .............................. 12

5. Implement and demonstrate the FIND-S algorithm. ...................................................... 15

6. demonstrate the Candidate-Elimination algorithm. ....................................................... 16

7. demonstrate the working of the decision tree based ID3 algorithm. ............................. 18

8. Artificial Neural Network by implementing the Back propagation algorithm ................. 21

9. program to construct a Bayesian network considering medical data. ............................ 24

10. Apply EM algorithm to cluster a set of data and clustering using k-Means algorithm. .. 27

11. implement k-Nearest Neighbour algorithm to classify the iris data set. ....................... 29

12. Implement the non-parametric Locally Weighted Regression algorithm ..................... 34

1. Write a program to implement the Tic-Tac-Toe game using Python.

**Program Code :**

# a Program to Implement Tic-Tac-Toe game using Python

import os

import time

board = ["", "", "", "", "", "", "", "", ""]

player = 1

win = 1 draw = -1

running = 0

Game = running

mark = "X"

def DrawBoard(): print("%c | %c | %c" , (board[0], board[1], board[2])) print("-------------")

print("%c | %c | %c", (board[3], board[4], board[5])) print("-------------")

print("%c | %c | %c" , (board[6], board[7], board[8]))

def CheckPosition(x): if board[x - 1] == "":

return True else:

return False

def CheckWin(): global Game if board[0] == board[1] and board[1] == board[2] and board[0] != "": Game = win elif board[3] == board[4] and board[4] == board[5] and board[3] != "": Game = win elif board[6] == board[7] and board[7] == board[8] and board[6] != "": Game = win elif board[0] == board[3] and board[3] == board[6] and board[0] != "": Game = win elif board[1] == board[4] and board[4] == board[7] and board[1] != "": Game = win elif board[2] == board[5] and board[5] == board[8] and board[2] != "":

Game = win

elif ( board[0] != "" and board[1] != "" and board[2] != "" and board[3] != "" and board[4] != "" and board[5] != "" and board[6] != "" and board[7] != "" and board[8] != ""

):

Game = draw else:

Game = running

print("Tic - Tac - Toe Game") print("Player 1 [x] --- Player 2 [o] \n") print() print() print("Please wait")

time.sleep(3)

while Game == running:

os.system("cls") # Use os.system("cls") on Windows DrawBoard() if player % 2 != 0: print("Player 1's chance") mark = "X"

# else:

print("Player 2's chance") mark = "O"

choice = int(input("Enter position between [1-9]: ")) if CheckPosition(choice): board[choice - 1] = mark player += 1 CheckWin()

os.system("cls") # Use os.system("cls") on Windows DrawBoard() if Game == draw: print("Game Draw") elif Game == win: player -= 1

if player % 2 == 0: print("Player 1 won") else:

print("Player 2 won")

**Output:**

Tic - Tac - Toe Game

Player 1 [x] --- Player 2 [o]

Please wait

('', '', '')

------------- ('', '', '')

------------- ('', '', '')

Player 1's chance

Enter position between [1-9]: 1

('X', '', '')

------------- Player 1's chance

Enter position between [1-9]: 1

('X', '', '')

------------- ('', '', '')

-------------

('', '', '')

Player 2's chance

Enter position between [1-9]: 1

('X', '', '')

------------- ('', '', '')

------------- ('', '', '')

Player 2's chance

('', '', '')

-------------

('', '', '') Player 2's chance ------------- ('', '', '')

Player 2's chance

('', '', '')

Player 2's chance

Enter position between [1-9]: 4

('X', '', '')

-------------

Player 2's chance

Enter position between [1-9]: 4

('X', '', '')

------------- ('O', '', '')

Enter position between [1-9]: 4

('X', '', '')

-------------

('O', '', '')

-------------

('', '', '') Player 1's chance

Enter position between [1-9]: 2

('X', 'X', '')

('O', '', '')

------------- ('', '', '') Player 1's chance

Enter position between [1-9]: 2

('X', 'X', '')

-------------

------------- ('', '', '') Player 1's chance

Enter position between [1-9]: 2

('X', 'X', '')

-------------

Enter position between [1-9]: 2

('X', 'X', '')

-------------

('O', '', '')

-------------

-------------

('O', '', '')

-------------

('O', '', '')

------------- ('', '', '') Player 2's chance

('', '', '')

Player 2's chance

Enter position between [1-9]: 5

('X', 'X', '')

------------- ('O', 'O', '')

Enter position between [1-9]: 5

('X', 'X', '')

------------- ('O', 'O', '')

------------- ('', '', '')

('X', 'X', '')

------------- ('O', 'O', '')

------------- ('', '', '')

Player 1's chance

Enter position between [1-9]: 3

('X', 'X', 'X')

------------- ('', '', '')

Player 1's chance

Enter position between [1-9]: 3

('X', 'X', 'X')

------------- Player 1's chance

Enter position between [1-9]: 3

('X', 'X', 'X')

-------------

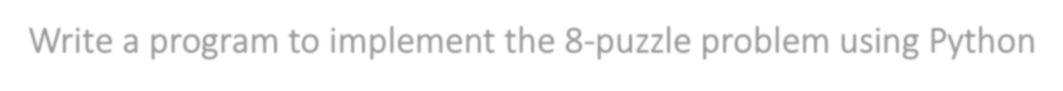
Enter position between [1-9]: 3

('X', 'X', 'X')

------------- ('O', 'O', '')

------------- ('', '', '') Player 1 won

2.Write a program to implement the 8-puzzle problem using Python



**Program Code :** import numpy as np import pandas as pd import os

def bfs(src, target): queue = []

queue.append(src)

exp = []

while len(queue) > 0: source = queue.pop(0) exp.append(source)

print(source) if source == target: print("Success") return

poss\_moves\_to\_do = possible\_move(source, exp) for move in poss\_moves\_to\_do: if move not in exp and move not in queue:

queue.append(move)

def possible\_move(state, visited\_states):

b = state.index(0)

d = [] if b not in [0, 1, 2]:

d.append('U') # Up if b not in [6, 7, 8]:

d.append('D') # Down

if b not in [0, 3, 6]:

d.append('L') # Left if b not in [2, 5, 8]:

d.append('R') # Right

pos\_moves\_it\_can = []

for i in d:

pos\_moves\_it\_can.append(gen(state, i, b))

return [move\_it\_can for move\_it\_can in pos\_moves\_it\_can if move\_it\_can not in visited\_states]

def gen(state, m, b): temp = state.copy()

if m == 'D': temp[b + 3], temp[b] = temp[b], temp[b + 3]

if m == 'U': temp[b - 3], temp[b] = temp[b], temp[b - 3]

if m == 'L': temp[b - 1], temp[b] = temp[b], temp[b - 1]

if m == 'R':

temp[b + 1], temp[b] = temp[b], temp[b + 1] return temp

src = [1, 2, 3, 4, 5, 6, 0, 7, 8] target = [1, 2, 3, 4, 5, 6, 7, 8, 0] bfs(src, target)

**Output :**

[1, 2, 3, 4, 5, 6, 0, 7, 8]

[1, 2, 3, 0, 5, 6, 4, 7, 8]

[1, 2, 3, 4, 5, 6, 7, 0, 8]

[0, 2, 3, 1, 5, 6, 4, 7, 8]

[1, 2, 3, 5, 0, 6, 4, 7, 8]

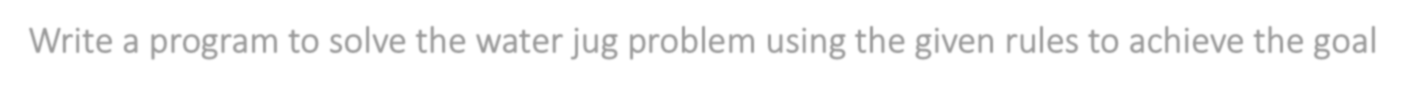
[1, 2, 3, 4, 0, 6, 7, 5, 8]

[1, 2, 3, 4, 5, 6, 7, 8, 0]

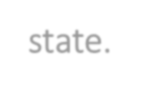
Success



3.



Write a program to solve the water jug problem using the given rules to achieve the goal



state.

**Program Code :**

x = 0 y = 0

1. = 4
2. = 3 print("Initial State = (0,0)") print("Capacities = (4,3)") print("Goal State = (2,y)") while x != 2:

r = int(input("Enter the rule: "))

if r == 1: x = m elif r == 2: y = n elif r == 3: x = 0 elif r == 4: y = 0 elif r == 5: t = n - y y = n x -= t elif r == 6: t = m - x x = m y -= t elif r == 7: y += x

x = 0 elif r == 8: x += y y = 0 else: print("Invalid Rule") break print(x, y)

**Output :**

Initial State = (0,0)

Capacities = (4,3)

Goal State = (2,y)

# Enter the rule: 1

4 0

# Enter the rule: 5

1 3

Enter the rule: 4

1 0

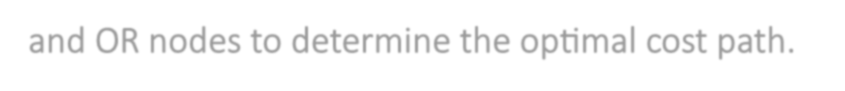
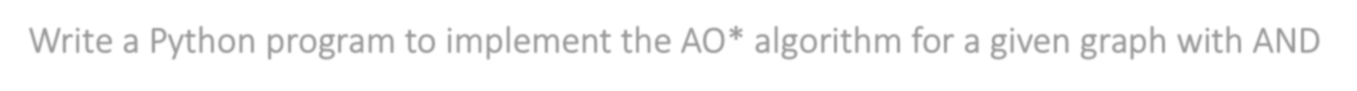
Enter the rule: 8

[0 1](#_Toc54424)

[Enter the rule: 1 4 1](#_Toc54425)

[Enter the rule: 5 2 3](#_Toc54426)

4.Write a Python program to implement the AO\* algorithm for a given graph with AND and OR nodes to determine the optimal cost path.



**Program Code :**

# Write a Program to implement AO\* Algorithm using Python

M='D'

I='C'

H='B'

A='A'

L='AND'

F='OR'

E=len

D=print

def J(n):

R=False;I=True;global X;D('Expanding Node:',n);G=[];H=[]

if n in B:

if L in B[n]:G=B[n][L] if F in B[n]:H=B[n][F]

if E(G)==0 and E(H)==0:return M=R;Q={} while not M:

if E(Q)==E(G)+E(H):S,T=O(G,H,{});M=I;P(n,S);C[n]=T;continue U,A=O(G,H,Q);N=R

if E(A)>1:

if A[0]in B:N=I;J(A[0]) if A[1]in B:N=I;J(A[1]) elif A in B:N=I;J(A)

if N:

V,W=O(G,H,{}) if A==W:M=I;P(n,V);C[n]=A else:M=I;P(n,U);C[n]=A Q[A]=1 return K(n)

def O(and\_nodes,or\_nodes,marked): G=marked;C={} for B in and\_nodes:

if not B[0]+B[1]in G:A=0;A=A+K(B[0])+K(B[1])+2;C[B[0]+B[1]]=A for D in or\_nodes: if not D in G:A=0;A=A+K(D)+1;C[D]=A E=999999;H=None

for F in C: if C[F]<E:E=C[F];H=F return[E,H] def K(n):return N[n] def P(n,cost):N[n]=cost;return def G(node):

B='->';A=node;D(C[A],end='');A=C[A]

if E(A)>1:

if A[0]in C:D(B,end='');G(A[0]) if A[1]in C:D(B,end='');G(A[1]) elif A in C:D(B,end='');G(A)

N={A:-1,H:4,I:2,M:3,'E':6,'F':8,'G':2,'H':0,'I':0,'J':0}

B={A:{L:[(I,M)],F:[H]},H:{F:['E','F']},I:{F:['G'],L:[('H','I')]},M:{F:['J']}}

C={}

Q=J(A)

D('Nodes which gives optimal cost are')

G(A)

D('\nOptimal Cost is :: ',Q)

**Output :**

Expanding Node: A Expanding Node: B

Expanding Node: C

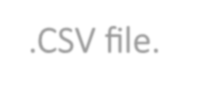
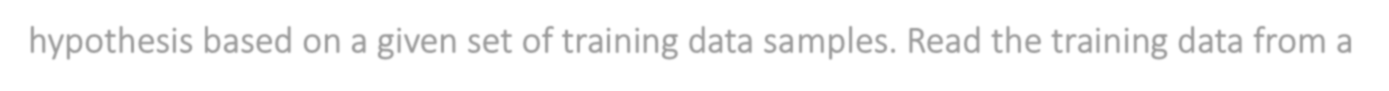
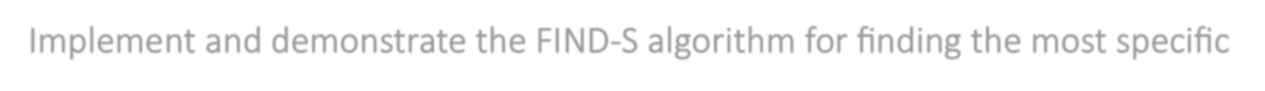
Expanding Node: D

Nodes which gives optimal cost are

CD->HI->J

Optimal Cost is :: 5

5.Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.



**Dataset :**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sky | Temperature | Humidity | Wind | Water | Forcast | EnjoySport |
| sunny | warm | normal | strong | warm | same | yes |
| sunny | warm | high | strong | warm | same | yes |
| rainy | cold | high | strong | warm | change | no |
| sunny | warm | high | strong | cool | change | yes |

**Program Code**

import csv num\_attribute = 6

a = [] with open('enjoysport.csv', 'r') as file:

reader = csv.reader(file)

a = list(reader)

hypothesis = a[0][:-1]

for i in a: if i[-1] == 'yes': for j in range(num\_attribute): if i[j] != hypothesis[j]: hypothesis[j] = '?' print(hypothesis)

print("\nThe Maximally Specific Hypothesis for a given Training Examples :\n") print(hypothesis)

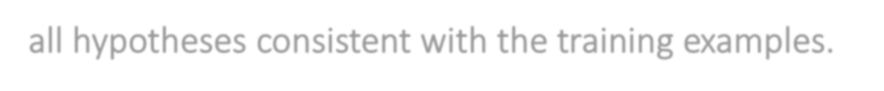
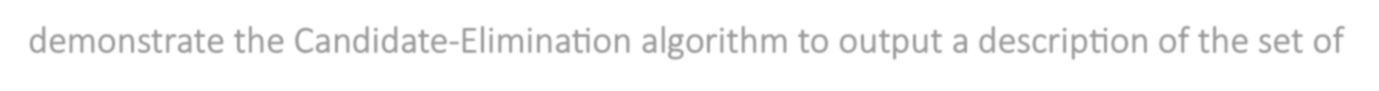
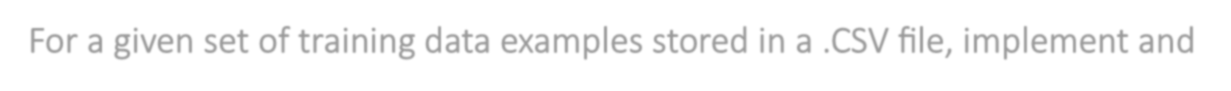
**Output :**

['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally Specific Hypothesis for a given Training Examples :

['sunny', 'warm', '?', 'strong', '?', '?']

6.For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.



**Dataset:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sky | Temperature | Humidity | Wind | Water | Forcast | EnjoySport |
| sunny | warm | normal | strong | warm | same | yes |
| sunny | warm | high | strong | warm | same | yes |
| rainy | cold | high | strong | warm | change | no |
| sunny | warm | high | strong | cool | change | yes |

**Program Code:**

import csv with open("program06.csv") as f:

csv\_file = csv.reader(f) data = list(csv\_file)

s = data[1][:-1] g = [["?" for \_ in range(len(s))] for \_ in range(len(s))]

for example in data[1:]: attributes = example[:-1] label = example[-1]

if label.lower() == "yes": for j in range(len(s)): if attributes[j] != s[j]:

s[j] = "?" g[j][j] = "?"

elif label.lower() == "no": for j in range(len(s)): if attributes[j] != s[j]: g[j][j] = s[j] else:

g[j][j] = "?"

gh = [h for h in g if any(val != "?" for val in h)] print("\nSteps of Candidate Elimination Algorithm", len(data) - 1) print(s) print(g) print("\nFinal specific hypothesis:\n", s) print("\nFinal general hypothesis:\n", gh)

**Output :**

Steps of Candidate Elimination Algorithm 4

['sunny', 'warm', '?', 'strong', '?', '?']

[['sunny', '?', '?', '?', '?', '?']

['?', 'warm', '?', '?', '?', '?'],

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']

['?', '?', '?', '?', '?', '?']]

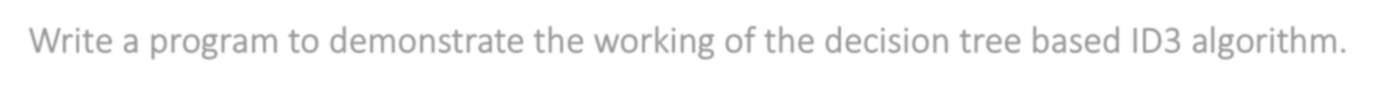
Final specific hypothesis:

['sunny', 'warm', '?', 'strong', '?', '?']

Final general hypothesis:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

7.Write a program to demonstrate the working of the decision tree based ID3 algorithm.



**Dataset:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Tempeature | Humidity | Windy | PlayTennis |
| sunny | hot | high | FALSE | no |
| sunny | hot | high | TRUE | no |
| overcast | hot | high | FALSE | yes |
| rainy | mild | high | FALSE | yes |
| rainy | cool | normal | FALSE | yes |
| rainy | cool | normal | TRUE | no |
| overcast | cool | normal | TRUE | yes |
| sunny | mild | high | FALSE | no |
| sunny | cool | normal | FALSE | yes |
| rainy | mild | normal | FALSE | yes |
| sunny | mild | normal | TRUE | yes |
| overcast | mild | high | TRUE | yes |
| overcast | hot | normal | FALSE | yes |
| rainy | mild | high | TRUE | no |

**Program Code:**

import pandas as pd from sklearn import tree

from sklearn.preprocessing import LabelEncoder from sklearn.tree import DecisionTreeClassifier

# Load and display the data data = pd.read\_csv("program07.csv")

print("The first 5 values of data is \n", data.head())

# Split data into features and labels X = data.iloc[:, :-1]

print("\nThe first 5 values of Train data is \n", X.head()) y = data.iloc[:, -1]

print("\nThe first 5 values of Train output is \n", y.head())

# Encode categorical features le\_outlook = LabelEncoder()

X.Outlook = le\_outlook.fit\_transform(X.Outlook)

le\_Temperature = LabelEncoder()

X.Temperature = le\_Temperature.fit\_transform(X.Temperature) le\_Humidity = LabelEncoder()

X.Humidity = le\_Humidity.fit\_transform(X.Humidity)

le\_Windy = LabelEncoder()

X.Windy = le\_Windy.fit\_transform(X.Windy)

print("\nNow the Train data is", X.head())

# Encode target variable le\_PlayTennis = LabelEncoder() y = le\_PlayTennis.fit\_transform(y)

print("\nNow the Train output is\n", y)

# Train the classifier classifier = DecisionTreeClassifier()

classifier.fit(X, y)

def labelEncoderForInput(list1):

list1[0] = le\_outlook.transform([list1[0]])[0] list1[1] = le\_Temperature.transform([list1[1]])[0] list1[2] = le\_Humidity.transform([list1[2]])[0] list1[3] = le\_Windy.transform([list1[3]])[0]

return [list1]

# Prepare input data inp1 = ["rainy", "cool", "high", "FALSE"]

pred1 = labelEncoderForInput(inp1)

# Predict and inverse transform y\_pred = classifier.predict(pred1)

print(

"\nfor input {0}, we obtain {1}!".format(

inp1, le\_PlayTennis.inverse\_transform(y\_pred)[0]

)

)

**Output:**

The first 5 values of data is

Outlook Temperature Humidity Windy PlayTennis

1. sunny hot high False no
2. sunny hot high True no
3. overcast hot high False yes
4. rainy mild high False yes
5. rainy cool normal False yes

The first 5 values of Train data is

Outlook Temperature Humidity Windy

1. sunny hot high False
2. sunny hot high True
3. overcast hot high False
4. rainy mild high False
5. rainy cool normal False

The first 5 values of Train output is

1. no
2. no
3. yes
4. yes
5. yes

Name: PlayTennis, dtype: object

Now the Train data is Outlook Temperature Humidity Windy

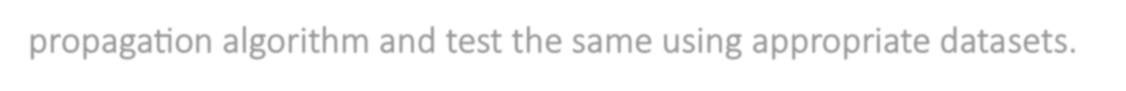
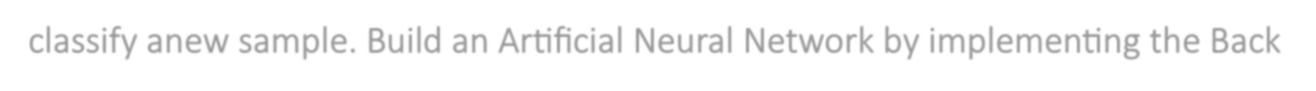
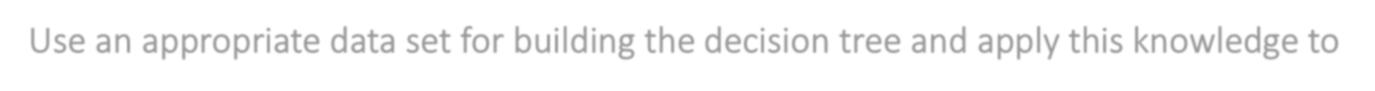
1. 2 1 0 0
2. 2 1 0 1
3. 0 1 0 0
4. 1 2 0 0
5. 1 0 1 0

Now the Train output is

[0 0 1 1 1 0 1 0 1 1 1 1 1 0]

for input [np.int64(1), np.int64(0), np.int64(0), np.int64(1)], we obtain no!

8.Use an appropriate data set for building the decision tree and apply this knowledge to classify anew sample. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate datasets.



**Program :**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X, axis=0) # maximum of X array longitudinally y = y/100

def sigmoid(x):

return 1/(1 + np.exp(-x))

def derivatives\_sigmoid(x):

return x \* (1 - x)

epoch = 5 lr = 0.1

inputlayer\_neurons = 2 hiddenlayer\_neurons = 3

output\_neurons = 1

wh = np.random.uniform(size=(inputlayer\_neurons, hiddenlayer\_neurons)) bh = np.random.uniform(size=(1, hiddenlayer\_neurons))

wout = np.random.uniform(size=(hiddenlayer\_neurons, output\_neurons)) bout = np.random.uniform(size=(1, output\_neurons))

for i in range(epoch): hinp1 = np.dot(X, wh) hinp = hinp1 + bh hlayer\_act = sigmoid(hinp) outinp1 = np.dot(hlayer\_act, wout) outinp = outinp1+bout output = sigmoid(outinp) EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act) d\_hiddenlayer = EH \* hiddengrad wout += hlayer\_act.T.dot(d\_output) \* lr

wh += X.T.dot(d\_hiddenlayer) \* lr print("\n-----------Epoch-", i+1, "Starts----------") print("Input: \n" + str(X)) print("Actual Output: \n" + str(y)) print("Predicted Output: \n", output)

print("-----------Epoch-", i+1, "Ends----------\n")

**Output:**

-----------Epoch- 1 Starts---------- Input:

[[0.66666667 1. ]

[0.33333333 0.55555556] [1. 0.66666667]] Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.87835449]

[0.87005118]

1. 87806243]]

-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts---------- Input:

[[0.66666667 1. ]

[0.33333333 0.55555556] [1. 0.66666667]] Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.8784591 ] [0.87015343]

1. 87816575]]

-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts---------- Input:

[[0.66666667 1. ]

[0.33333333 0.55555556] [1. 0.66666667]] Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.87856283]

[0.87025482]

[0.87826819]]

-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts---------- Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

1. 0.66666667]] Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.87866568]

[0.87035536]

[0.87836977]]

-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts---------- Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]] Actual Output:

[[0.92]

[0.86]

[0.89]]

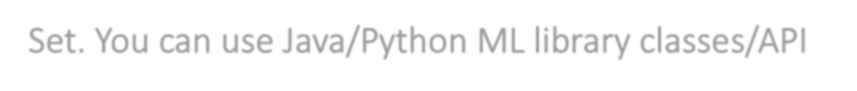
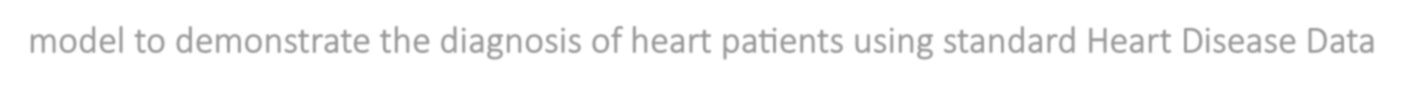
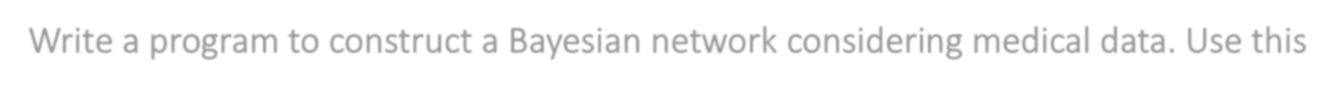
Predicted Output:

[[0.87876766]

[0.87045505] [0.8784705 ]]

-----------Epoch- 5 Ends----------

9.Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API



**Dataset:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| age |  | gender | family | diet |  | lifestyle | cholestr ol | heartdiseas e |
|  | 0 | 0 | 1 |  | 1 | 3 | 0 | 1 |
|  | 0 | 1 | 1 |  | 1 | 3 | 0 | 1 |
|  | 1 | 0 | 0 |  | 0 | 2 | 1 | 1 |
|  | 4 | 0 | 1 |  | 1 | 3 | 2 | 0 |
|  | 3 | 1 | 1 |  | 0 | 0 | 2 | 0 |
|  | 2 | 0 | 1 |  | 1 | 1 | 0 | 1 |
|  | 4 | 0 | 1 |  | 0 | 2 | 0 | 1 |
|  | 0 | 0 | 1 |  | 1 | 3 | 0 | 1 |
|  | 3 | 1 | 1 |  | 0 | 0 | 2 | 0 |
|  | 1 | 1 | 0 |  | 0 | 0 | 2 | 1 |
|  | 4 | 1 | 0 |  | 1 | 2 | 0 | 1 |
|  | 4 | 0 | 1 |  | 1 | 3 | 2 | 0 |
|  | 2 | 1 | 0 |  | 0 | 0 | 0 | 0 |
|  | 2 | 0 | 1 |  | 1 | 1 | 0 | 1 |
|  | 3 | 1 | 1 |  | 0 | 0 | 1 | 0 |
|  | 0 | 0 | 1 |  | 0 | 0 | 2 | 1 |
|  | 1 | 1 | 0 |  | 1 | 2 | 1 | 1 |
|  | 3 | 1 | 1 |  | 1 | 0 | 1 | 0 |
|  | 4 | 0 | 1 |  | 1 | 3 | 2 | 0 |

**Program Code:**

import pandas as pd

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.inference import VariableElimination

data = pd.read\_csv('program9.csv'); heartdisease = pd.DataFrame(data);

print(heartdisease);

model = BayesianModel([

('age' , 'lifestyle'),

('gender' , 'lifestyle'),

('family' , 'heartdisease'),

('diet' , 'cholestrol'),

('lifestyle' , 'diet'),

('cholestrol' , 'heartdisease'),

('diet' , 'cholestrol')

]);

model.fit(heartdisease, estimator = MaximumLikelihoodEstimator);

heartdisease\_infer = VariableElimination(model);

print("""For age Enter {

SuperSenioerCitizen : 0, SenioerCitizen : 1, middleage : 2, youth : 3, teen : 4

}""");

print('''For gender Enter {

Male : 0,

Female : 1

}''');

print('''For Family History Enter : {

Yes : 1,

No : 0

}''');

print('''For Diet Enter {

High : 0,

Medium : 1

}''');

print('''For Life Style Enter {

Athelte : 0,

Active : 1,

Moderate : 2,

Sedentary : 3

}''');

print('''For Cholestrol Enter {

High : 0,

BorderLine : 1,

Normal : 2

}''');

q = heartdisease\_infer.query( variables=['heartdisease'],

evidence = {

'age' : int(input('Enter Age : ')),

'gender' : int(input('Enter Gender : ')),

'family' : int(input('Enter Family History : ')),

'diet' : int(input('Enter Diet : ')),

'lifestyle' : int(input('Enter Life Style : ')),

'cholestrol' : int(input('Enter Cholestrol : ')),

}); print(q);

**Output:** query for 'heartdisease' given 'cholestrol=2':

+-----------------+---------------------+

| heartdisease | phi(heartdisease) |

+=================+=====================+ |

heartdisease(0) | 0.5072 |

+-----------------+---------------------+ | heartdisease(1) | 0.4928 | +-----------------+---------------------+ query for 'heartdisease' given 'diet=1':

+-----------------+---------------------+

| heartdisease | phi(heartdisease) |

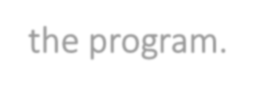
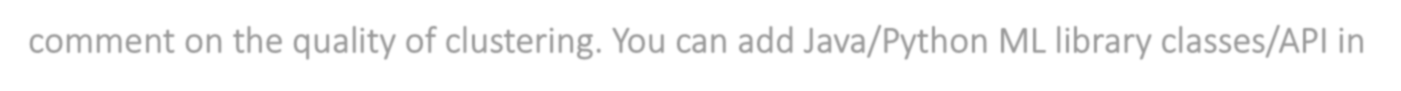
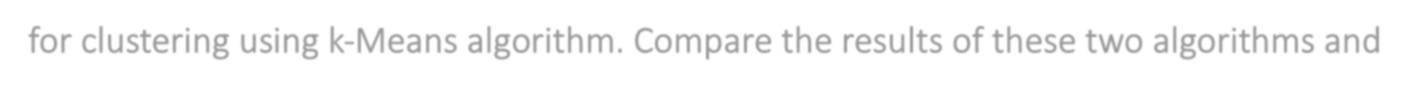
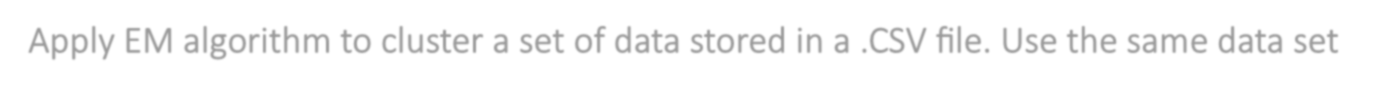
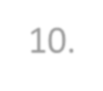
+=================+=====================+

| heartdisease(0) | 0.4952 | +-----------------+---------------------+

| heartdisease(1) | 0.5048 |

+-----------------+---------------------+

10.Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.



**Program Code:**

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler from sklearn.mixture import GaussianMixture from sklearn.datasets import load\_iris import pandas as pd import numpy as np

import matplotlib.pyplot as plt

dataset = load\_iris();

x = pd.DataFrame(dataset.data);

x.columns = ['Sepal\_Length' , 'Sepal\_Width' ,'Petal\_Length' , 'Petal\_Width']; y = pd.DataFrame(dataset.target) y.columns = ['Target'];

plt.figure(figsize=(14,7));

colormap = np.array(['red' , 'lime' , 'black']);

plt.subplot(1,3,1);

plt.scatter(x.Petal\_Length , x.Petal\_Width , c =colormap[y.Target]); plt.title('Real');

plt.subplot(1,3,2);

model = KMeans(n\_clusters=3);

model.fit(x);

predy = np.choose(model.labels\_ , [0,1,2]); plt.scatter(x.Petal\_Length , x.Petal\_Width , c=colormap[predy]); plt.title('KMeans');

scaler = StandardScaler();

scaler.fit(x);

xsa = scaler.transform(x)

xs = pd.DataFrame(xsa , columns = x.columns); gmm = GaussianMixture(n\_components=3); gmm.fit(xs)

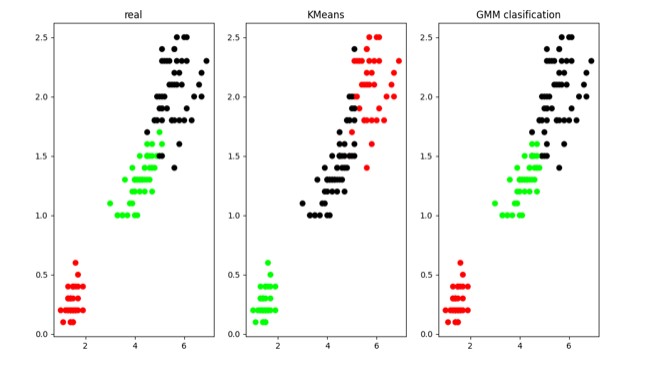
y\_cluster\_gmm = gmm.predict(xs);

plt.subplot(1,3,3);

plt.scatter(x.Petal\_Length , x.Petal\_Width , c=colormap[y\_cluster\_gmm]);

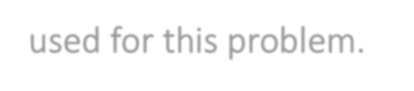
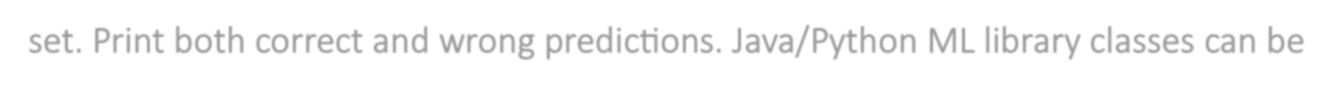
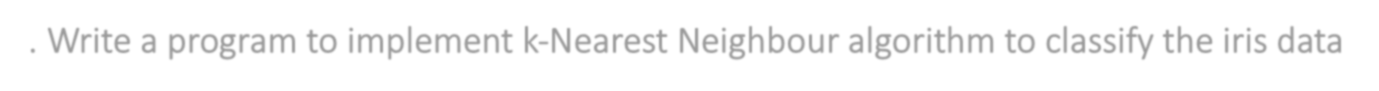
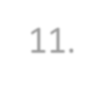
plt.title('GMM classification'); plt.show();

**Output:**





11.. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.



**Program Code:**

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report , confusion\_matrix from sklearn import datasets from sklearn.neighbors import KNeighborsClassifier

iris = datasets.load\_iris() x = iris.data

y = iris.target

print("Feature :" , iris.feature\_names) print(x)

print("Labesl " , iris.target\_names) print(y)

x\_train , x\_test , y\_train , y\_test = train\_test\_split(x,y) classifier = KNeighborsClassifier() classifier.fit(x\_train , y\_train) y\_pred = classifier.predict(x\_test) print('confusion matrx')

print(confusion\_matrix(y\_test, y\_pred)) print('Accuracy metics')

print(classification\_report(y\_test , y\_pred))

**Output:**

Feature : ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'] [[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

1. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

1. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]

[4.8 3.4 1.6 0.2]

[4.8 3. 1.4 0.1]

[4.3 3. 1.1 0.1]

[5.8 4. 1.2 0.2]

[5.7 4.4 1.5 0.4]

[5.4 3.9 1.3 0.4]

[5.1 3.5 1.4 0.3]

[5.7 3.8 1.7 0.3]

[5.1 3.8 1.5 0.3]

[5.4 3.4 1.7 0.2]

[5.1 3.7 1.5 0.4]

[4.6 3.6 1. 0.2]

[5.1 3.3 1.7 0.5]

[4.8 3.4 1.9 0.2]

[5. 3. 1.6 0.2]

1. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

* 1. 3.2 1.6 0.2]
  2. 3.1 1.6 0.2]
  3. 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

* 1. 4.2 1.4 0.2]

[4.9 3.1 1.5 0.2]

1. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.6 1.4 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

1. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]

[6.3 3.3 4.7 1.6]

[4.9 2.4 3.3 1. ]

[6.6 2.9 4.6 1.3] [5.2 2.7 3.9 1.4]

1. 2. 3.5 1. ]

[5.9 3. 4.2 1.5]

1. 2.2 4. 1. ]
   1. 2.9 4.7 1.4]

[5.6 2.9 3.6 1.3]

[6.7 3.1 4.4 1.4]

[5.6 3. 4.5 1.5]

[5.8 2.7 4.1 1. ]

* 1. 2.2 4.5 1.5]

[5.6 2.5 3.9 1.1]

[5.9 3.2 4.8 1.8]

[6.1 2.8 4. 1.3]

* 1. 2.5 4.9 1.5]

[6.1 2.8 4.7 1.2]

* 1. 2.9 4.3 1.3]

[6.6 3. 4.4 1.4]

[6.8 2.8 4.8 1.4]

[6.7 3. 5. 1.7]

[6. 2.9 4.5 1.5]

[5.7 2.6 3.5 1. ]

[5.5 2.4 3.8 1.1]

[5.5 2.4 3.7 1. ]

[5.8 2.7 3.9 1.2]

[6. 2.7 5.1 1.6]

[5.4 3. 4.5 1.5]

1. 3.4 4.5 1.6]

[6.7 3.1 4.7 1.5]

[6.3 2.3 4.4 1.3]

[5.6 3. 4.1 1.3]

[5.5 2.5 4. 1.3]

[5.5 2.6 4.4 1.2]

[6.1 3. 4.6 1.4]

[5.8 2.6 4. 1.2]

1. 2.3 3.3 1. ]
   1. 2.7 4.2 1.3]
   2. 3. 4.2 1.2]
   3. 2.9 4.2 1.3]

[6.2 2.9 4.3 1.3]

[5.1 2.5 3. 1.1]

[5.7 2.8 4.1 1.3]

[6.3 3.3 6. 2.5]

* 1. 2.7 5.1 1.9]

[7.1 3. 5.9 2.1]

[6.3 2.9 5.6 1.8]

[6.5 3. 5.8 2.2] [7.6 3. 6.6 2.1]

[4.9 2.5 4.5 1.7]

[7.3 2.9 6.3 1.8]

[6.7 2.5 5.8 1.8]

[7.2 3.6 6.1 2.5]

[6.5 3.2 5.1 2. ]

[6.4 2.7 5.3 1.9]

[6.8 3. 5.5 2.1]

* 1. 2.5 5. 2. ]
  2. 2.8 5.1 2.4]
  3. 3.2 5.3 2.3]
  4. 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

1. 2.2 5. 1.5]

[6.9 3.2 5.7 2.3]

[5.6 2.8 4.9 2. ]

[7.7 2.8 6.7 2. ]

[6.3 2.7 4.9 1.8]

* 1. 3.3 5.7 2.1]

[7.2 3.2 6. 1.8]

[6.2 2.8 4.8 1.8]

[6.1 3. 4.9 1.8]

[6.4 2.8 5.6 2.1]

[7.2 3. 5.8 1.6]

[7.4 2.8 6.1 1.9]

[7.9 3.8 6.4 2. ]

[6.4 2.8 5.6 2.2]

[6.3 2.8 5.1 1.5]

[6.1 2.6 5.6 1.4]

[7.7 3. 6.1 2.3]

* 1. 3.4 5.6 2.4]
  2. 3.1 5.5 1.8]

1. 3. 4.8 1.8]

[6.9 3.1 5.4 2.1]

* 1. 3.1 5.6 2.4]

[6.9 3.1 5.1 2.3]

[5.8 2.7 5.1 1.9]

* 1. 3.2 5.9 2.3]

[6.7 3.3 5.7 2.5]

[6.7 3. 5.2 2.3]

[6.3 2.5 5. 1.9]

[6.5 3. 5.2 2. ]

[6.2 3.4 5.4 2.3] [5.9 3. 5.1 1.8]]

Labesl ['setosa' 'versicolor' 'virginica'] [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. 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
    2. 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
    3. 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]

confusion matrx

[[11 0 0]

[ 0 10 2]

[ 0 0 15]]

Accuracy metics

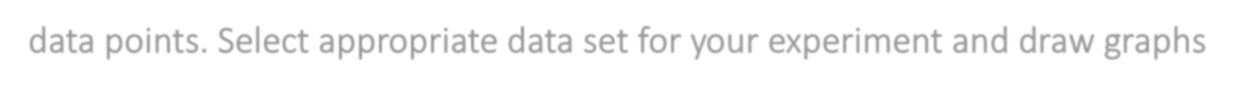
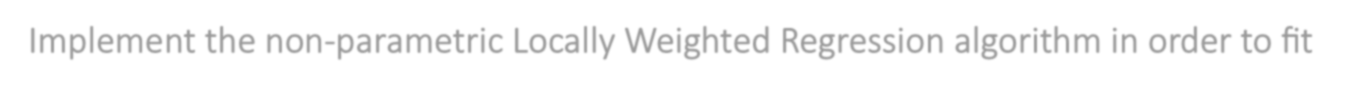
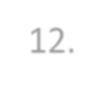
precision recall f1-score support

* + - 1. 1.00 1.00 1.00 11
      2. 1.00 0.83 0.91 12
      3. 0.88 1.00 0.94 15

accuracy 0.95 38 macro avg 0.96 0.94 0.95 38

weighted avg 0.95 0.95 0.95 38

12.Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs



**Program Code:**

import numpy as np import math

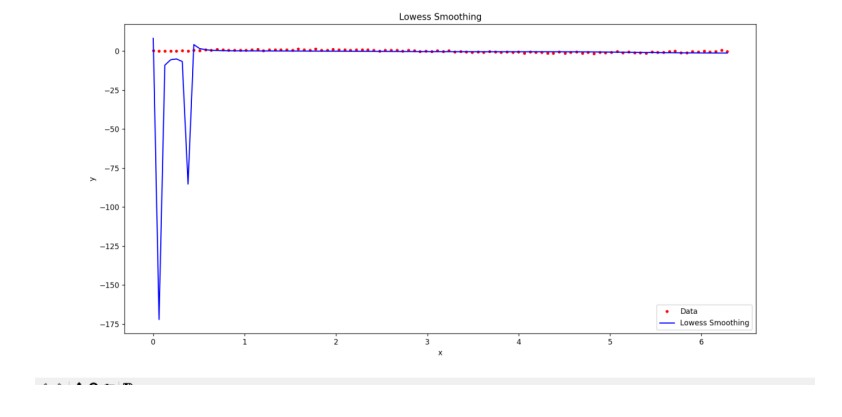
import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

n = 100

xpoints = np.linspace(0,2 \* math.pi , n).reshape(-1,1); ypoints = np.sin(xpoints); linreg = LinearRegression(); linreg.fit(xpoints , ypoints); prediction = linreg.predict(xpoints) plt.scatter(xpoints , ypoints , color='red'); plt.plot(xpoints , prediction) plt.show();

**Output:**



**Open Ended Experiment**

**Rainfall Predection System:-**

'''

# Cleaning the data

# read data in pandas dataframe

data = pd.read\_csv("austin\_weather.csv")

# drop (delete) the unnecessary columns in the data

data = data.drop(

['Events', 'Date', 'SeaLevelPressureHighInches', 'SeaLevelPressureLowInches'], axis=1)

data = data.replace('T', 0.0)

data = data.replace('-', 0.0)

# Save the data in a csv file

data.to\_csv('austin\_final.csv')

'''

# import the libraries

import pandas as pd

import numpy as np

import sklearn as sk

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# read the cleaned data

data = pd.read\_csv("austin\_final.csv")

X = data.drop(['PrecipitationSumInches'], axis=1)

Y = data['PrecipitationSumInches']

Y = Y.values.reshape(-1, 1)

day\_index = 798

days = [i for i in range(Y.size)]

clf = LinearRegression()

clf.fit(X, Y)

inp = np.array([[74], [60], [45], [67], [49], [43], [33], [45],

[57], [29.68], [10], [7], [2], [0], [20], [4], [31]])

inp = inp.reshape(1, -1)

# Print output

print('The precipitation in inches for the input is:', clf.predict(inp))

print('The precipitation trend graph: ')

plt.scatter(days, Y, color='g')

plt.scatter(days[day\_index], Y[day\_index], color='r')

plt.title('Precipitation level')

plt.xlabel('Days')

plt.ylabel('Precipitation in inches')

# Plot a graph of precipitation levels vs n# of days

plt.show()

x\_f = X.filter(['TempAvgF', 'DewPointAvgF', 'HumidityAvgPercent',

'SeaLevelPressureAvgInches', 'VisibilityAvgMiles',

'WindAvgMPH'], axis=1)

print('Preciptiation Vs Selected Attributes Graph: ')

for i in range(x\_f.columns.size):

plt.subplot(3, 2, i+1)

plt.scatter(days, x\_f[x\_f.columns.values[i][:100]], color='g')

plt.scatter(days[day\_index], x\_f[x\_f.columns.values[i]]

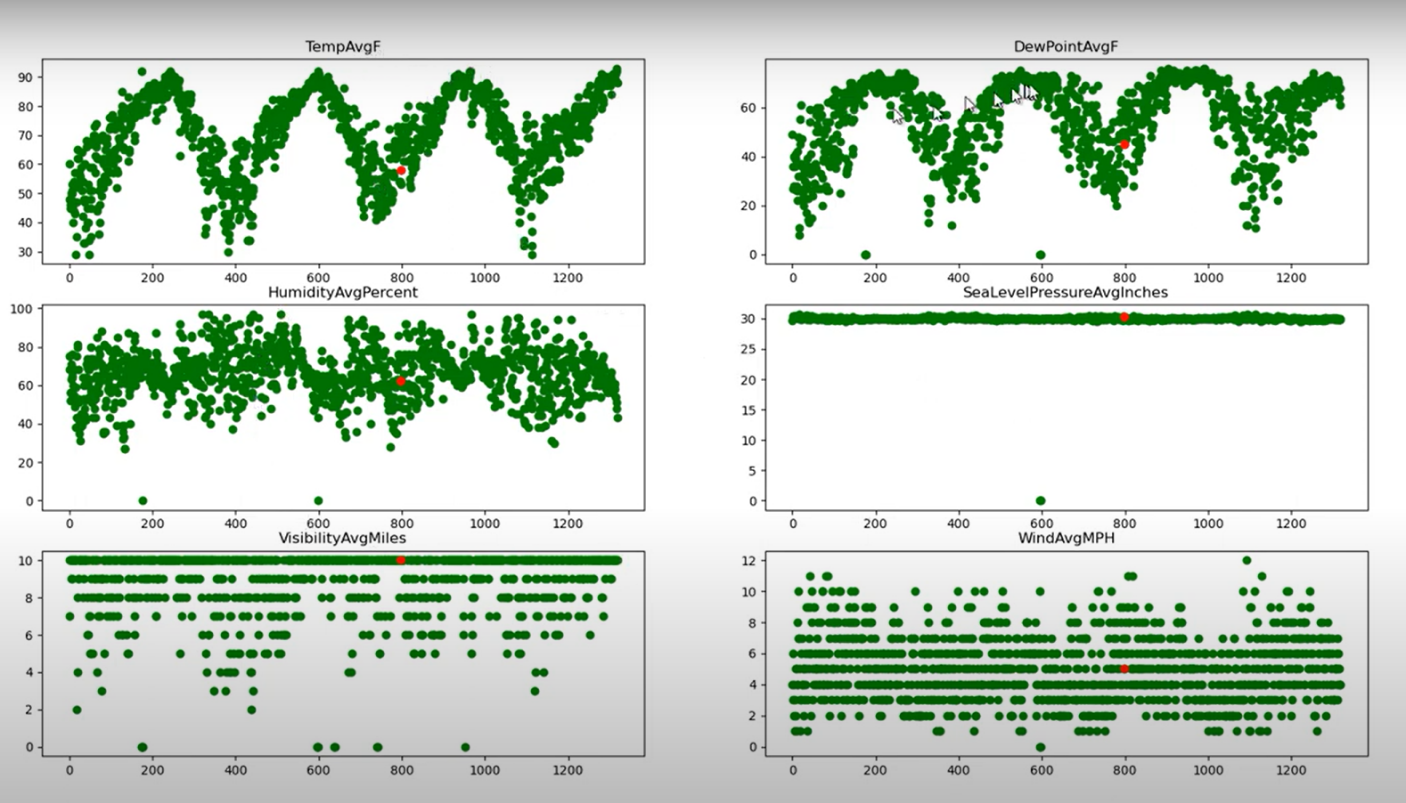
[day\_index], color='r')

plt.title(x\_f.columns.values[i])

# plot a graph with a few features vs precipitation to observe the trends

plt.show()

**Result:-**

****