1. Implement A* Search algorithm.

```
def aStarAlgo(start_node, stop_node):
open_set = set(start_node)
closed_set = set()
g = {}
parents = {}
g[start_node] = 0
parents[start_node] = start_node
while len(open_set) > 0:
n = None
for v in open_set:
if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
n = v
if n == stop_node or Graph_nodes[n] == None:
pass
else:
for (m, weight) in get_neighbors(n):
if m not in open_set and m not in closed_set:
open_set.add(m)
parents[m] = n
g[m] = g[n] + weight
else:
if g[m] > g[n] + weight:
g[m] = g[n] + weight
parents[m] = n
if m in closed_set:
closed_set.remove(m)
open_set.add(m)
if n == None:
```

```
print('Path does not exist!')
return None
if n == stop_node:
path = []
while parents[n] != n:
path.append(n)
n = parents[n]
path.append(start_node)
path.reverse()
print('Path found: {}'.format(path))
return path
open_set.remove(n)
closed_set.add(n)
print('Path does not exist!')
return None
def get_neighbors(v):
if v in Graph_nodes:
return Graph_nodes[v]
else:
return None

def heuristic(n):
H_dist = {
'A': 11,
'B': 6,
'C': 5,
'D': 7,
'E': 3,
'F': 6,
```

```
'G': 5,
'H': 3,
'I': 1,
'J': 0
}
return H_dist[n]


Graph_nodes = {
'A': [('B', 6), ('F', 3)],
'B': [('A', 6), ('C', 3), ('D', 2)],
'C': [('B', 3), ('D', 1), ('E', 5)],
'D': [('B', 2), ('C', 1), ('E', 8)],
'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
'F': [('A', 3), ('G', 1), ('H', 7)],
'G': [('F', 1), ('I', 3)],
'H': [('F', 7), ('I', 2)],
'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}
aStarAlgo('A', 'J')
```

Output:

Path found: ['A', 'F', 'G', 'I', 'J']

2. Implement AO* Search algorithm.

class Graph:

def __init__(self, graph, heuristicNodeList, startNode): #instantiate graph object with graph topology, heuristic values, start node

self.graph = graph

self.H=heuristicNodeList

self.start=startNode

self.parent={}

self.status={}

self.solutionGraph={}

def applyAOStar(self):

self.aoStar(self.start, False)

def getNeighbors(self, v):

return self.graph.get(v,'')

def getStatus(self,v):

return self.status.get(v,0)

def setStatus(self,v, val):

self.status[v]=val

def getHeuristicNodeValue(self, n):

return self.H.get(n,0)

def setHeuristicNodeValue(self, n, value):

self.H[n]=value

```
def printSolution(self):

print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE:",self.start)

print("------------------------------------------------------------")

print(self.solutionGraph)

print("------------------------------------------------------------")


def computeMinimumCostChildNodes(self, v):

minimumCost=0

costToChildNodeListDict={}

costToChildNodeListDict[minimumCost]=[]

flag=True

for nodeInfoTupleList in self.getNeighbors(v):

cost=0

nodeList=[]

for c, weight in nodeInfoTupleList:

cost=cost+self.getHeuristicNodeValue(c)+weight

nodeList.append(c)

if flag==True:

minimumCost=cost

costToChildNodeListDict[minimumCost]=nodeList

flag=False

else:

if minimumCost>cost:

minimumCost=cost

costToChildNodeListDict[minimumCost]=nodeList

return minimumCost, costToChildNodeListDict[minimumCost]


def aoStar(self, v, backTracking):
```

```
print("HEURISTIC VALUES :", self.H)
print("SOLUTION GRAPH :", self.solutionGraph)
print("PROCESSING NODE :", v)
print("-----------------------------------------------------------------------------------------")
if self.getStatus(v) >= 0:
minimumCost, childNodeList = self.computeMinimumCostChildNodes(v)
print(minimumCost, childNodeList)
self.setHeuristicNodeValue(v, minimumCost)
self.setStatus(v,len(childNodeList))
solved=True
for childNode in childNodeList:
self.parent[childNode]=v
if self.getStatus(childNode)!=-1:
solved=solved & False
if solved==True:
self.setStatus(v,-1)
self.solutionGraph[v]=childNodeList
if v!=self.start:
self.aoStar(self.parent[v], True)
if backTracking==False:
for childNode in childNodeList:
self.setStatus(childNode,0)
self.aoStar(childNode, False)

print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
graph1 = {
'A': [[('B', 1), ('C', 1)], [('D', 1)]],
'B': [[('G', 1)], [('H', 1)]],
```

```
'C': [[('J', 1)]],
'D': [[('E', 1), ('F', 1)]],
'G': [[('I', 1)]]
}

G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

Output

```
Graph - 1
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
-----------------------------------------------------------------------------------------
6 ['G']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
```

PROCESSING NODE : G

----

8 ['I']

HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : B

----

8 ['H']

HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : A

----

12 ['B', 'C']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}

SOLUTION GRAPH : {}

PROCESSING NODE : I

----

0 []

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': []}

PROCESSING NODE : G

----

1 ['I']

HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I']}

PROCESSING NODE : B

----

2 ['G']

HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

----

6 ['B', 'C']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : C

----

2 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : A

----

6 ['B', 'C']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}

PROCESSING NODE : J

----

0 []

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}

PROCESSING NODE : C

----

1 ['J']

HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}

SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}

PROCESSING NODE : A

----

5 ['B', 'C']

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}

----

3. 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.

```
import csv
with open("sk3.csv") as f:
csv_file=csv.reader(f)
data=list(csv_file)

s=data[1][:-1]
g=[['?' for i in range(len(s))] for j in range(len(s))]
for i in data:
if i[-1]=="Yes":
for j in range(len(s)):
if i[j]!=s[j]:
s[j]='?'
g[j][j]='?'
elif i[-1]=="No":
for j in range(len(s)):
if i[j]!=s[j]:
g[j][j]=s[j]
else:
g[j][j]="?"
print("\steps of candidate elimation algorithm", data.index(i)+1)
```

```
print(s)
print(g)
gh=[]
for i in g:
for j in i:
if j!='?':
gh.append(i)
break
print("\n final specific hypothesis:\n",s)
print("\n final general hypothesis:\n",gh)
```

.csv file

```
Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Rainy,Cold,High,Strong,Warm,Same,No
Sunny,Warm,High,Strong,Cool,Change,Yes
```

Output

```
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

final specific hypothesis:
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

final general hypothesis:
[['Sunny', '?', '?', '?', '?', '?']]

final specific hypothesis:
```

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

final general hypothesis:

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

final specific hypothesis:

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

final general hypothesis:

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

final specific hypothesis:

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

final general hypothesis:

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

final specific hypothesis:

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

final general hypothesis:

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

final specific hypothesis:

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

final general hypothesis:

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

4. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import pandas as pd

import math

def base_entropy(dataset):

p = 0

n = 0

target = dataset.iloc[:, -1]

targets = list(set(target))

for i in target:

if i == targets[0]:

p = p + 1

else:

n = n + 1

if p == 0 or n == 0:

return 0

elif p == n:

return 1

else:

entropy = 0 - (

((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))

return entropy

def entropy(dataset, feature, attribute):

p = 0

n = 0

target = dataset.iloc[:, -1]

targets = list(set(target))

for i, j in zip(feature, target):

if i == attribute and j == targets[0]:
```

```
p = p + 1
elif i == attribute and j == targets[1]:
n = n + 1
if p == 0 or n == 0:
return 0
elif p == n:
return 1
else:
entropy = 0 - (
((p / (p + n)) * (math.log2(p / (p + n))) + (n / (p + n)) * (math.log2(n/ (p + n)))))
return entropy
def counter(target, attribute, i):
p = 0
n = 0
targets = list(set(target))
for j, k in zip(target, attribute):
if j == targets[0] and k == i:
p = p + 1
elif j == targets[1] and k == i:
n = n + 1
return p, n
def Information_Gain(dataset, feature):
Distinct = list(set(feature))
Info_Gain = 0
for i in Distinct:
Info_Gain = Info_Gain + feature.count(i) / len(feature) * entropy(dataset,feature, i)
Info_Gain = base_entropy(dataset) - Info_Gain
return Info_Gain
def generate_childs(dataset, attribute_index):
```

```
distinct = list(dataset.iloc[:, attribute_index])

childs = dict()

for i in distinct:

childs[i] = counter(dataset.iloc[:, -1], dataset.iloc[:, attribute_index], i)

return childs

def modify_data_set(dataset,index, feature, impurity):

size = len(dataset)

subdata = dataset[dataset[feature] == impurity]

del (subdata[subdata.columns[index]])

return subdata

def greatest_information_gain(dataset):

max = -1

attribute_index = 0

size = len(dataset.columns) - 1

for i in range(0, size):

feature = list(dataset.iloc[:, i])

i_g = Information_Gain(dataset, feature)

if max < i_g:

max = i_g

attribute_index = i

return attribute_index

def construct_tree(dataset, tree):

target = dataset.iloc[:, -1]

impure_childs = []

attribute_index = greatest_information_gain(dataset)

childs = generate_childs(dataset, attribute_index)

tree[dataset.columns[attribute_index]] = childs

targets = list(set(dataset.iloc[:, -1]))

for k, v in childs.items():
```

```
if v[0] == 0:
tree[k] = targets[1]
elif v[1] == 0:
tree[k] = targets[0]
elif v[0] != 0 or v[1] != 0:
impure_childs.append(k)
for i in impure_childs:
sub = modify_data_set(dataset,attribute_index,
dataset.columns[attribute_index], i)
tree = construct_tree(sub, tree)
return tree
def main():
df = pd.read_csv("sk4.csv")
tree = dict()
result = construct_tree(df, tree)
for key, value in result.items():
print(key, " => ", value)
if __name__ == "__main__":
main()
```

.csv file

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes

Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Output

```
Sunny     Hot    High    Weak   No  => Rain      Cool   Normal   Weak  Yes
Sunny     Hot    High    Strong No  => Rain      Cool   Normal   Weak  Yes
Overcast  Hot    High    Weak   Yes => Rain      Cool   Normal   Weak  Yes
Rain      Mild   High    Weak   Yes => Rain      Cool   Normal   Weak  Yes
Rain      Cool   Normal  Weak   Yes => Rain      Cool   Normal   Weak  Yes
Rain      Cool   Normal  Strong No  => Rain      Cool   Normal   Weak  Yes
Overcast  Cool   Normal  Strong Yes => Rain      Cool   Normal   Weak  Yes
Sunny     Mild   High    Weak   No  => Rain      Cool   Normal   Weak  Yes
Sunny     Cool   Normal  Weak   Yes => Rain      Cool   Normal   Weak  Yes
Rain      Mild   Normal  Weak   Yes => Rain      Cool   Normal   Weak  Yes
Sunny     Mild   Normal  Strong Yes => Rain      Cool   Normal   Weak  Yes
Overcast  Mild   High    Strong Yes => Rain      Cool   Normal   Weak  Yes
Overcast  Hot    Normal  Weak   Yes => Overcast  Hot    Normal   Weak  Yes
Rain      Mild   High    Strong No  => Rain      Cool   Normal   Weak  Yes
```

5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets

```
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)
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 ("-----------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")

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

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:
[[0.84039994]
[0.82699888]
[0.84180893]]
-----------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:
[[0.84092204]
[0.82751715]
```

[0.84232896]]

-----------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:

[[0.84143652]

[0.82802795]

[0.84284138]]

-----------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:

[[0.84194354]

[0.82853144]

[0.84334636]]

-----------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:

[[0.84244325]

[0.82902779]

[0.84384404]]

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

Input:

[[0.66666667 1.   ]

[0.33333333 0.55555556]

[1.   0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.84244325]

[0.82902779]

[0.84384404]]

6. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd

PlayTennis=pd.read_csv("sk6.csv")
print("Given dataset:\n",PlayTennis,"\n")

from sklearn.preprocessing import LabelEncoder
Le=LabelEncoder()

PlayTennis['outlook']=Le.fit_transform(PlayTennis['outlook'])
PlayTennis['temp']=Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity']=Le.fit_transform(PlayTennis['humidity'])
PlayTennis['wind']=Le.fit_transform(PlayTennis['wind'])
PlayTennis['play']=Le.fit_transform(PlayTennis['play'])

print("the encoded dataset is:\n",PlayTennis)

X=PlayTennis.drop(['play'],axis=1)
y=PlayTennis['play']

from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.20)
print("\n X_train:\n",X_train)
print("\n y_train:\n",y_train)
print("\n X_test:\n",X_test)
print("\n y_test:\n",y_test)
classifier=GaussianNB()
classifier.fit(X_train, y_train)
accuracy=accuracy_score(classifier.predict(X_test), y_test)
print("\n Accuracy is:",accuracy)
```

.csv file

```
outlook, temp, humidity, wind, play
Sunny, Hot, High, Weak, No
Sunny, Hot, High, Strong, No
Overcast, Hot, High, Weak, Yes
Rainy, Cool, Normal, Weak, Yes
Rainy, Cool, Normal, Strong, No
Overcast, Cool, Normal, Strong, Yes
Sunny, Mild, High, Weak, No
Sunny, Cool, Normal, Weak, Yes
Rainy, Mild, Normal, Weak, Yes
```

Output

```
Given dataset:
outlook  temp humidity    wind play
0    Sunny   Hot    High    Weak   No
1    Sunny   Hot    High  Strong   No
```

2 Overcast Hot High Weak Yes
3 Rainy Cool Normal Weak Yes
4 Rainy Cool Normal Strong No
5 Overcast Cool Normal Strong Yes
6 Sunny Mild High Weak No
7 Sunny Cool Normal Weak Yes
8 Rainy Mild Normal Weak Yes

the encoded dataset is:

	outlook	temp	humidity	wind	play
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	0	1	1	1
4	1	0	1	0	0
5	0	0	1	0	1
6	2	2	0	1	0
7	2	0	1	1	1
8	1	2	1	1	1

X_train:

	outlook	temp	humidity	wind
5	0	0	1	0
3	1	0	1	1
6	2	2	0	1
7	2	0	1	1
0	2	1	0	1
1	2	1	0	0
4	1	0	1	0

y_train:

```
5    1
3    1
6    0
7    1
0    0
1    0
4    0
Name: play, dtype: int32
```

X_test:

```
outlook  temp  humidity  wind
2        0     1         0     1
8        1     2         1     1
```

y_test:

```
2    1
8    1
Name: play, dtype: int32
Accuracy is: 0.0
```

7. 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

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
Y = pd.DataFrame(iris.target)
Y.columns = ['Targets']

print(X)
print(Y)
colormap = np.array(['red', 'lime', 'black'])

plt.subplot(1,2,1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[Y.Targets], s=40)
plt.title('Real Clustering')

model1 = KMeans(n_clusters=3)
model1.fit(X)
```

```
plt.subplot(1,2,2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model1.labels_], s=40)
plt.title('K Mean Clustering')
plt.show()

model2 = GaussianMixture(n_components=3)
model2.fit(X)

plt.subplot(1,2,1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model2.predict(X)], s=40)
plt.title('EM Clustering')
plt.show()

print("Actual Target is:\n", iris.target)
print("K Means:\n",model1.labels_)
print("EM:\n",model2.predict(X))
print("Accuracy of KMeans is ",sm.accuracy_score(Y,model1.labels_))
print("Accuracy of EM is ",sm.accuracy_score(Y, model2.predict(X)))
```

output

```
Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
0     5.1     3.5     1.4     0.2
1     4.9     3.0     1.4     0.2
2     4.7     3.2     1.3     0.2
3     4.6     3.1     1.5     0.2
4     5.0     3.6     1.4     0.2
..    ...     ...     ...     ...
```

145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

[150 rows x 4 columns]

Targets

0	0
1	0
2	0
3	0
4	0
..	...
145	2
146	2
147	2
148	2
149	2

[150 rows x 1 columns]

Actual Target is:

[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]

K Means:

[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 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2

2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2

2 0]

EM:

[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 0 2 0 2 0 2

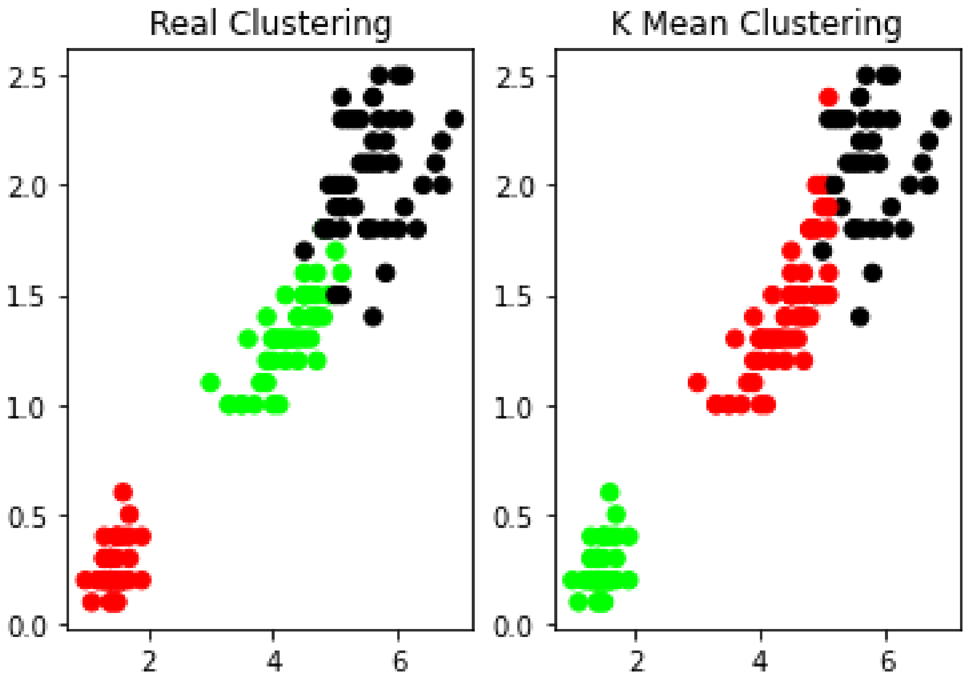
2 2 2 0 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0

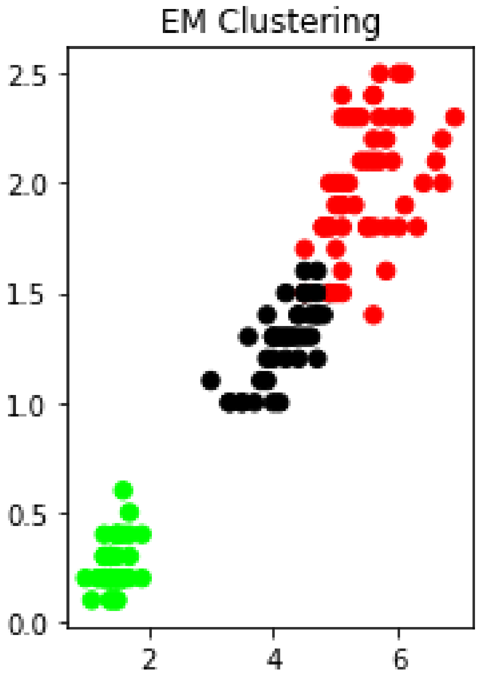
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0]

Accuracy of KMeans is 0.44

Accuracy of EM is 0.03333333333333333





8. 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

```
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report,confusion_matrix

from sklearn import datasets


iris=datasets.load_iris()


x=iris.data

y=iris.target

print('sepal-length','sepal-width','petal-length','petal-width')

print(x)

print('class:0-Iris-Setosa,1-Iris-Versicolour,2-Iris-Virginica')

print(y)
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Matrics')
print(classification_report(y_test, y_pred))
```

<u>output</u>

```
sepal-length  sepal-width  petal-length  petal-width
[[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]
[5.  3.6 1.4 0.2]
[5.4 3.9 1.7 0.4]
[4.6 3.4 1.4 0.3]
[5.  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]

[5. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

[4.7 3.2 1.6 0.2]

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.2]

[5. 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]

[5. 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]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
[6.  2.2 4.  1. ]
[6.1 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. ]
[6.2 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]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
```

[6.4 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]

[6. 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]

[5. 2.3 3.3 1.]

[5.6 2.7 4.2 1.3]

[5.7 3. 4.2 1.2]

[5.7 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]

[5.8 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]

[5.7 2.5 5. 2.]

[5.8 2.8 5.1 2.4]

[6.4 3.2 5.3 2.3]

[6.5 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

[6. 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]

[6.7 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]

[6.3 3.4 5.6 2.4]

[6.4 3.1 5.5 1.8]

[6. 3. 4.8 1.8]

[6.9 3.1 5.4 2.1]

[6.7 3.1 5.6 2.4]

[6.9 3.1 5.1 2.3]

[5.8 2.7 5.1 1.9]

[6.8 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]]

class:0-lris-Setosa,1-lris-Versicolour,2-lris-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

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]

Confusion Matrix

[[13 0 0]

[0 16 1]

[0 2 13]]

Accuracy Matrics

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	0.89	0.94	0.91	17
2	0.93	0.87	0.90	15
accuracy			0.93	45
macro avg	0.94	0.94	0.94	45
weighted avg	0.93	0.93	0.93	45

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

```
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
import statsmodels.api as sm

x=[i/5.0 for i in range(30)]
y=[1,2,1,2,1,1,3,4,5,4,5,6,5,6,7,8,9,10,11,11,12,11,11,10,12,11,11,10,9,13]

lowess=sm.nonparametric.lowess(y,x)
lowess_x=list(zip(*lowess))[0]
lowess_y=list(zip(*lowess))[1]
f=interp1d(lowess_x,lowess_y,bounds_error=False)

xnew=[i/10.0 for i in range(100)]
ynew=f(xnew)
plt.plot(x,y,'o')
```

```
plt.plot(lowess_x,lowess_y,'+')

plt.plot(xnew,ynew,'-')

plt.show()
```

output