1. Implement and demonstratethe FIND-Salgorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

DATASET(tenni.csv)

sky,airTemp,humidity,wind,water,forecaste,enjoySport
sunny,warm,normal,strong,warm,same,true
sunny,warm,high,strong,warm,same,true
rainy,cold,high,sttrong,warm,change,false
sunny,warm,high,strong,cool,change,true

```
import csv
with open('tenni.csv', 'r') as f:
    reader=csv.reader(f)
    your_list=list(reader)
h=[['0','0','0','0','0','0']]
for i in your_list:
    print(i)
    if i[-1]=='true':
        j=0
        for x in i:
            if x!='true':
                if x!=h[0][j] and h[0][j]=='0':
                    h[0][j]=x
                elif x!=h[0][j] and h[0][j]!='0':
                    h[0][j]='?'
                else:
                    pass
            j=j+1
    print("most specific hypothes is")
    print(h)
```

Output

'Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same',True
'Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same',True
'Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change',False
'Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change',True

Maximally Specific set
[['Sunny', 'Warm', '?', 'Strong', '?', '?']]

2. 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(tenni2.csv)

sky,airTemp,humidity,wind,water,forecaste,enjoySport
cloudy,cold,high,strong,warm,change,yes
sunny,warm,normal,strong,warm,same,yes
sunny,warm,high,strong,warm,same,yes
cloudy,cold,high,strong,warm,change,no
sunny,warm,high,strong,cool,change,yes
rain,mild,high,weak,cool,change,no
rain,cool,normal,weak,cool,same,no
overcast,cool,normal,strong,warm,same,yes

PROGRAM

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('tenni2.csv'))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
```

```
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
    print(" steps of Candidate Elimination Algorithm",i+1)
    print("Specific_h ",i+1,"\n ")
    print(specific_h)
    print("general_h ", i+1, "\n ")
    print(general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

OUTPUT

initialization of specific_h and general_h

['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

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

steps of Candidate Elimination Algorithm 8

Specific_h 8

['?' '?' '?' 'Strong' '?' '?']

general_h 8

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

Final Specific_h:

['?' '?' '?' 'Strong' '?' '?']

Final General_h:

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

3. 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.

DATASET(tennis.csv)

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

PROGRAM

```
import pandas as pd
import numpy as np
dataset= pd.read_csv('tennis.csv',names=['outlook','temperature','humidity','wind','class',])
def entropy(target_col):
    elements,counts = np.unique(target_col,return_counts = True)
    entropy = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])
    return entropy
def InfoGain(data,split_attribute_name,target_name="class"):
    total_entropy = entropy(data[target_name])
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)
    Weighted_Entropy =
```

```
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i])
.dropna()[target_name]) for i in range(len(vals))])
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain
def ID3(data,originaldata,features,target_attribute_name="class",parent_node_class = None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data)==0:
        return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attri
bute_name],return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class                                                    =
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name],ret
urn_counts=True)[1])]
        item_values = [InfoGain(data,feature,target_attribute_name) for feature in features]
#Return the information gain values for the features in the dataset
        best_feature_index = np.argmax(item_values)
        best_feature = features[best_feature_index]
        tree = {best_feature:{}}
        features = [i for i in features if i != best_feature]
        for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()
            subtree = ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
            tree[best_feature][value] = subtree
        return(tree)
tree = ID3(dataset,dataset,dataset.columns[:-1])
print(' \nDisplay Tree\n',tree)
```

OUTPUT

Display Tree

{'outlook': {'overcast': 'yes', 'rain': {'wind': {'strong': 'no', 'weak': 'yes'}}, 'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}

4. 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) # maximum of X array longitudinally y = y/100
#Sigmoid Function
def sigmoid (x):
    return (1/(1 + np.exp(-x)))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
#Variable initialization
epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
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))
# draws a random range of numbers uniformly of dim x*y
#Forward Propagation
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)
#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr
# dotproduct of nextlayererror and currentlayerop
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

OUTPUT

```
Input:
[[ 0.66666667 1. ]
[ 0.33333333 0.55555556]
[ 1. 0.66666667]]

Actual Output:
[[ 0.92]
[ 0.86]
[ 0.89]]
```

Predicted Output:

[[0.89559591]

[0.88142069]

[0.8928407]]

5. 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.

DATASET(abcd.csv)

3423313857,41.91,10,1
3423312432,58.64,20,0
3423311434,52.02,8,1
3423311328,31.25,34,0
3423312488,44.31,19,1
3423311254,49.35,40,0
3423312943,58.07,45,1
3423312536,44.22,22,0
3423311542,55.73,19,1
3423312176,46.63,43,0
3423314176,52.97,32,1
3423314202,46.25,35,0
3423311346,51.55,27,1
3423310666,57.05,26,1
3423313527,58.45,30,1
3423312182,43.42,23,1
3423313590,55.68,37,1
3423312268,55.15,18,0

PROGRAM

```
import csv
import random
import math

def loadCsv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
```

```
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset
def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio);
    trainSet = []
    copy = list(dataset);
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy));
        trainSet.append(copy.pop(index))
    return [trainSet, copy]
def separateByClass(dataset):
    separated = {}
#creates a dictionary of classes 1 and 0 where the values are the instacnes belonging to
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated
def mean(numbers):
    return sum(numbers)/float(len(numbers))
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)
def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
    del summaries[-1]
    return summaries
def summarizeByClass(dataset):
    separated = separateByClass(dataset);
    summaries = {}
    for classValue, instances in separated.items():
```

```
        summaries[classValue] = summarize(instances)
    return summaries
def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.items():#class and attribute information
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i] #take mean and sd of every attribute
        x = inputVector[i] #testvector's first attribute
        probabilities[classValue] *= calculateProbability(x, mean, stdev);#use
    return probabilities
def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():#assigns that class which has he
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions
def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0
def main():
    filename = "abcd"
    splitRatio = 0.67
```

```
    dataset=loadCsv( filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset),len(trainingSet),
len(testSet)))
    summaries = summarizeByClass(trainingSet);
    predictions = getPredictions(summaries, testSet)
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy of the classifier is: {0} %'.format(accuracy))
main()
```

OUTPUT

split 18 rows into train=12 and test=6 rows

Accuracy of the classifier is: 66.66666666666666 %

6. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

DATASET (pgm6.csv)

```
I love this sandwich,pos
This is an amazing place,pos
I feel very good about these beers,pos
This is my best work,pos
What an awesome view,pos
I do not like this restaurant,neg
I am tired of this stuff,neg
I can't deal with this,neg
He is my sworn enemy,neg
My boss is horrible,neg
This is an awesome place,pos
I do not like the taste of this juice,neg
I love to dance,pos
I am sick and tired of this place,neg
What a great holiday,pos
That is a bad locality to stay,neg
We will have good fun tomorrow,pos
I went to my enemy's house today,neg
```

PROGRAM

```
import pandas as pd
msg=pd.read_csv('pgm6.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)
#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)
#output of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print(count_vect.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df)#tabular representation
print(xtrain_dtm) #sparse matrix representation
# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
#printing accuracy metrics
from sklearn import metrics
print('Accuracy metrics')
```

```
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
```

OUTPUT

```
The dimensions of the dataset (18, 2)
0 I love this sandwich
1 This is an amazing place
2 I feel very good about these beers
3 This is my best work
4 What an awesome view
5 I do not like this restaurant
6 I am tired of this stuff
7 I can't deal with this
8 He is my sworn enemy
9 My boss is horrible
10 This is an awesome place
11 I do not like the taste of this juice
12 I love to dance
13 I am sick and tired of this place
14 What a great holiday
15 That is a bad locality to stay
16 We will have good fun tomorrow
17 I went to my enemy's house today
Name: message, dtype: object
0 1
1 1
2 1
3 1
```

4 1
5 0
6 0
7 0
8 0
9 0
10 1
11 0
12 1
13 0
14 1
15 0
16 1
17 0
Name: labelnum, dtype: int64

(5,)

(13,)

(5,)

(13,)

Accuracy metrics

Accuracy of the classifer is 0.8

Confusion matrix

[[3 1]

[0 1]]

Recall and Precison

1.0

0.5

7. Write a program to construct aBayesian 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.

PROGRAM

```
import numpy as np
from urllib.request import urlopen
import urllib
import pandas as pd
```

```
from pgmpy.inference import VariableElimination
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator
names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope',
'ca',
'thal', 'heartdisease']
heartDisease = pd.read_csv('heart.csv', names = names)
heartDisease = heartDisease.replace('?', np.nan)
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('exang',
'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),('heartdisease','restecg'),
('heartdisease','thalach'), ('heartdisease','chol')])
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)
#q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 37, 'sex' :0})
print('\n1.Probability of HeartDisease given Age=20')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])
```

OUTPUT:

```
╒═══════════════════╤═══════════════════╕
│ heartdisease      │ phi(heartdisease) │
╞═══════════════════╪═══════════════════╡
│ heartdisease_0    │ 0.5593            │
├───────────────────┼───────────────────┤
│ heartdisease_1    │ 0.4407            │
╘═══════════════════╧═══════════════════╛
```

8.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.

DATASET(pgm8data.csv)

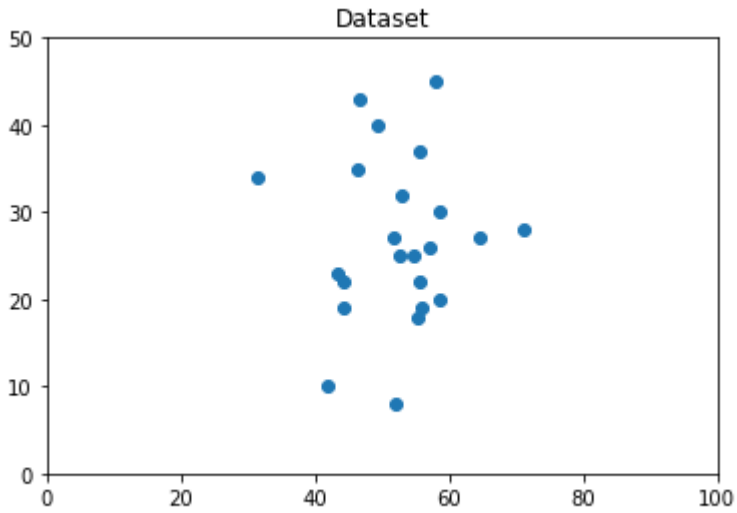
Driver_ID,Distance_Feature,Speeding_Feature
3423311935,71.24,28
3423313212,52.53,25
3423313724,64.54,27
3423311373,55.69,22
3423310999,54.58,25
3423313857,41.91,10
3423312432,58.64,20
3423311434,52.02,8
3423311328,31.25,34
3423312488,44.31,19
3423311254,49.35,40
3423312943,58.07,45
3423312536,44.22,22
3423311542,55.73,19
3423312176,46.63,43
3423314176,52.97,32
3423314202,46.25,35
3423311346,51.55,27
3423310666,57.05,26
3423313527,58.45,30
3423312182,43.42,23
3423313590,55.68,37
3423312268,55.15,18

PROGRAM

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
import pandas as pd
X=pd.read_csv("pgm8data.csv")
```

```
x1 = X['Distance_Feature'].values
x2 = X['Speeding_Feature'].values
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()
#code for EM
gmm = GaussianMixture(n_components=3)
gmm.fit(X)
em_predictions = gmm.predict(X)
print("\nEM predictions")
print(em_predictions)
print("mean:\n",gmm.means_)
print('\n')
print("Covariances\n",gmm.covariances_)
print(X)
plt.title('Exceptation Maximum')
plt.scatter(X[:,0], X[:,1],c=em_predictions,s=50)
plt.show()
#code for Kmeans
import matplotlib.pyplot as plt1
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
plt.title('KMEANS')
plt1.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt1.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```

OUTPUT



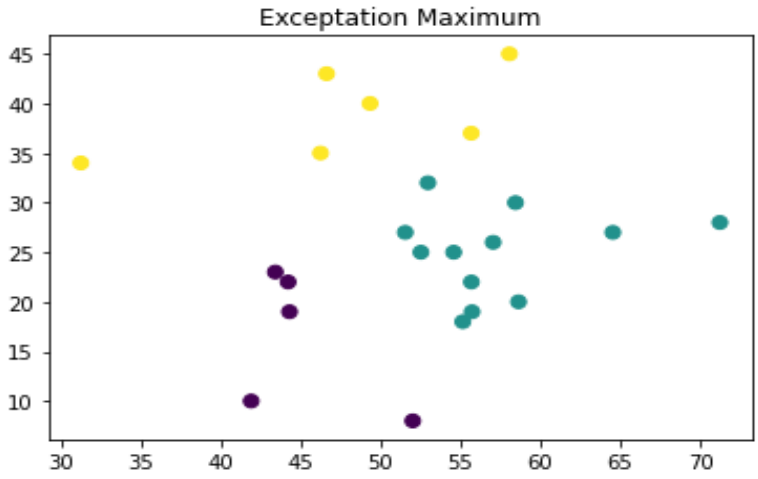
EM predictions
[1 1 1 1 1 0 1 0 2 0 2 2 0 1 2 1 2 1 1 1 0 2 1]
mean:
[[45.52364659 15.12663491]
[56.2895734 24.72458957]
[47.81649609 38.96676066]]

Covariances
[[[15.50444953 -11.70433701]
[-11.70433701 39.34887084]]

[[39.25509622 6.40503301]
[6.40503301 18.32286182]]

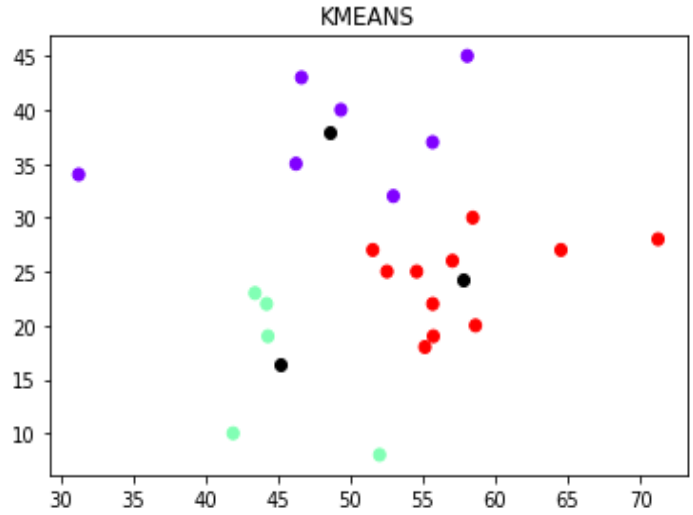
[[74.43313242 21.91196052]
[21.91196052 16.92381491]]]
[[71.24 28.]
[52.53 25.]
[64.54 27.]
[55.69 22.]
[54.58 25.]
[41.91 10.]
[58.64 20.]
[52.02 8.]
[31.25 34.]

[44.31 19.]
[49.35 40.]
[58.07 45.]
[44.22 22.]
[55.73 19.]
[46.63 43.]
[52.97 32.]
[46.25 35.]
[51.55 27.]
[57.05 26.]
[58.45 30.]
[43.42 23.]
[55.68 37.]
[55.15 18.]]



[[48.6 38.]
[45.176 16.4]
[57.74090909 24.27272727]]
[2 2 2 2 2 1 2 1 0 1 0 0 1 2 0 0 0 2 2 2 1 0 2]

```
Out[7]: <matplotlib.collections.PathCollection at 0x210f1236970>
```



9. 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.

DATASET(iris.csv): https://www.kaggle.com/

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import pandas as pd
dataset=pd.read_csv("iris.csv")
X=dataset.iloc[:, :-1].values
y=dataset.iloc[:, 4].values
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0,test_size=0.25)
classifier=KNeighborsClassifier(n_neighbors=8,p=3,metric='euclidean')
classifier.fit(X_train,y_train)
#predict the test resuts
y_pred=classifier.predict(X_test)
```

```
cm=confusion_matrix(y_test,y_pred)
print('Confusion matrix is as follows\n',cm)
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
print(" correct predicition",accuracy_score(y_test,y_pred))
print(" worng predicition",(1-accuracy_score(y_test,y_pred)))
```

Output :

```
Confusion matrix is as follows
[[13 0 0]
[ 0 15 1]
[ 0 0 9]]
Accuracy Metrics
precision recall f1-score support
Iris-setosa 1.00 1.00 1.00 13
Iris-versicolor 1.00 0.94 0.97 16
Iris-virginica 0.90 1.00 0.95 9
avg / total 0.98 0.97 0.97 38
correct predicition 0.9736842105263158
worng predicition 0.02631578947368418
```

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

PROGRAM

```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy.linalg
from scipy.stats.stats import pearsonr
```

```
def kernel(point,xmat, k):
    m,n = shape(xmat)
    weights = mat(eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
def localWeightRegression(xmat,ymat,k):
    m,n = shape(xmat)
    ypred = zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
# load data points
data = pd.read_csv('pgm10data.csv')
bill = array(data.total_bill)
tip = array(data.tip)

#preparing and add 1 in bill
mbill = mat(bill)
mtip = mat(tip)
m= shape(mbill)[1]
one = mat(ones(m))
X= hstack((one.T,mbill.T))
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
```

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

OUTPUT