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

```
import pandas as pd

d=pd.read_csv("/content/ENJOYSPORT.csv")

d=pd.DataFrame(d)

print(d)


h=[]

t=0

for i in range(len(d)):

 l=list(d.loc[i])

 if l[len(l)-1]==1:

  l.pop()

  h.extend(l)

  k=i

  break

print("H",t+1,":",h)

t=t+2

l=[]

for i in range(k+1,len(d)):

  l=list(d.loc[i])

  if l[len(l)-1]==1:

   for j in range(len(h)):

    if h[j]!=l[j]:

     h[j]='?'

  print("H",t,":",h)

  t=t+1
```

OUTPUT:-

```
H 1 : ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
H 2 : ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
H 3 : ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
H 4 : ['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.

```
from re import S

import pandas as pd

d=pd.read_csv("/content/ENJOYSPORT.csv")

# print(len(d.columns))

d=pd.DataFrame(d)


print("DATA SET: ")

print(d)

# print(len(d.columns))


s=[]

t=['?' for i in range(len(d.columns)-1)]

g=[]



l=list(d.loc[0])

l.pop()

s.extend(l)

for i in range(len(d)):

  l=list(d.loc[i])

  if l[len(l)-1]==1:

   for j in range(len(s)):

    if s[j]!=l[j]:

     s[j]='?'

   if g!=[]:

    for p in range(len(g)):

     for q in range(len(s)):

      if g[p][q]==s[q] or g[p][q]=='?':

        continue
```

```
       else:
         g.pop(p)

   else:
    for j in range(len(s)):
     if l[j]!=s[j] and s[j]!='?':
      t[j]=s[j]
      g.append(t)
      t=['?' for i in range(len(d.columns)-1)]

# print(g)
# print('\n')
# print(s)



def versionspace(g,s):
 vs=[]
 for i in g:
  for j in range(len(i)):
   if i[j]=='?' and s[j]!='?':
    m=i[:]
    m[j]=s[j]
    if m not in vs:
     vs.append(m)
 return vs
r=[]
r.append(s)
r1=versionspace(g,s)
r.extend(r1)
r.extend(g)
print("\nGeneral Hypothesis : ",g)
```

```
print("\nSpecific Hypothesis : ",s)
print("\nVersion Space :")
for i in r:
  print(i)
```

OUTPUT:-

DATA SET:

```
    Sky AirTemp Humidity Wind Water Forecast EnjoySport
0  Sunny   Warm   Normal  Strong  Warm    Same        1
1  Sunny   Warm    High  Strong  Warm    Same        1
2  Rainy   Cold    High  Strong  Warm   Change       0
3  Sunny   Warm    High  Strong  Cool   Change       1
```

General Hypothesis : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

Specific Hypothesis : ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Version Space :

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

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

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

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

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

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

3. 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
df=pd.read_csv('/content/play_tennis.csv')
df=df.iloc[:,1:]
```

```
print("DATA FRAME : ")
print(df)

train=df[:10]
yn=train.iloc[:,-1].value_counts()
y=yn['Yes']
n=yn['No']
t=len(train)
py=y/t
pn=n/t

c=list(train.iloc[:,:].columns)
dict={}
for i in c:
  dict[i]={}
  for j in train[i]:
    if j not in dict[i]:
      dict[i][j]=[]

for k in dict:
    for l in dict[k]:
      ky=len(train[(train[c[-1]]=='Yes') & (train[k]==l)][k])
      kn=len(train[(train[c[-1]]=='No') & (train[k]==l)][k])
      dict[k][l].extend([(ky/y),(kn/n)])

# print(dict)

test=df[10:]
test=test.iloc[:,:]
test1=test
print(test1)
```

```
test=test.iloc[:,:-1]


o=[]
for i in range(len(test)):
  yes=py
  no=pn
  l=list(test.iloc[i,:])
  print("Test row : ",l)


  for j in range(len(l)):
    yes=yes*dict[c[j]][l[j]][0]
    no=no*dict[c[j]][l[j]][1]
  print("Yes : ",yes,"No : ",no)


  if yes>no:
    print("Yes" )
    o.append('Yes')
  elif no>yes:
    print("No" )
    o.append('No')


test2=pd.DataFrame(test)
test2[c[-1]]=o
print(test2)
import sklearn
from sklearn.metrics import accuracy_score
actual=list(test1[c[-1]])
predicted=o
accuracy=sklearn.metrics.accuracy_score(actual,predicted)
print(accuracy)
```

OUTPUT:-

```
DATA FRAME :
     outlook  temp humidity    wind play
0      Sunny   Hot     High    Weak   No
1      Sunny   Hot     High  Strong   No
2   Overcast   Hot     High    Weak  Yes
3       Rain  Mild     High    Weak  Yes
4       Rain  Cool   Normal    Weak  Yes
5       Rain  Cool   Normal  Strong   No
6   Overcast  Cool   Normal  Strong  Yes
7      Sunny  Mild     High    Weak   No
8      Sunny  Cool   Normal    Weak  Yes
9       Rain  Mild   Normal    Weak  Yes
10     Sunny  Mild   Normal  Strong  Yes
11  Overcast  Mild     High  Strong  Yes
12  Overcast   Hot   Normal    Weak  Yes
13      Rain  Mild     High  Strong   No
     outlook  temp humidity    wind play
10     Sunny  Mild   Normal  Strong  Yes
11  Overcast  Mild     High  Strong  Yes
12  Overcast   Hot   Normal    Weak  Yes
13      Rain  Mild     High  Strong   No
Test row :  ['Sunny', 'Mild', 'Normal', 'Strong']
Yes :  0.0037037037037037025 No :  0.009375000000000001
No
Test row : ['Overcast', 'Mild', 'High', 'Strong']
Yes :  0.0037037037037037025 No :  0.0
Yes
Test row : ['Overcast', 'Hot', 'Normal', 'Weak']
Yes :  0.018518518518518514 No :  0.0
Yes
Test row :  ['Rain', 'Mild', 'High', 'Strong']
Yes :  0.005555555555555554 No :  0.009375000000000001
No
     outlook  temp humidity    wind play
10     Sunny  Mild   Normal  Strong   No
11  Overcast  Mild     High  Strong  Yes
12  Overcast   Hot   Normal    Weak  Yes
13      Rain  Mild     High  Strong   No

0.75
```

4. 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.

```
import pandas as pd

d=pd.read_csv("/content/text_classification.csv")

# d=d.iloc[:,:-1]

print("DATA FRAME : ")

print(d)
```

```
train=d[:10]
yn=train.iloc[:,-1].value_counts()
y=yn['pos']
n=yn['neg']
t=len(train)
py=y/t
pn=n/t
# print(train)
dict={}
n1,n2=0,0
for i in range(len(train)):
  dict[i]={}
  l=list(train.iloc[i])
  if(l[-1]=='pos'):
  sl=l[0].split()
  for j in sl:
      if j not in dict:
        dict[i][j]=1
        n1=n1+1
      else:
        dict[i][j]=dict[i][j]+1
  else:
    sl=l[0].split()
    for j in sl:
      if j not in dict:
        dict[i][j]=1
        n2=n2+1
      else:
        dict[i][j]=dict[i][j]+1


# print(dict)
```

```
dict2={}
dict3={}
for i in range(len(train)):
  l=list(train.iloc[i])
  if(l[-1]=='pos'):
   sl=l[0].split()
   for j in sl:
    if j not in dict2:
     dict2[j]=1
    else:
     dict2[j]=dict2[j]+1
  else:
   sl=l[0].split()
   for j in sl:
    if j not in dict3:
     dict3[j]=1
    else:
     dict3[j]=dict3[j]+1

# print(dict2) # pos words
# print(dict3)  # neg words

#total vocabulary
v=0
for i in dict3:
 if i not in dict2:
  v=v+1
v=v+len(dict2)
# print(v)
pp={}
```

```
np={}
for i in dict2:
pp[i]=(dict2[i]+1)/(v+n1)
# print(pp)
for i in dict3:
np[i]=(dict3[i]+1)/(v+n2)
# print(np)
test=d[10:]
actual=list(test.iloc[:,-1])
predicted=[]
for i in range(len(test)):
  l=list(test.iloc[i])
  sl=l[0].split()
  p_prob=py
  n_prob=pn
  for j in sl:
   if j in pp.keys():
   p_prob*=dict2[j]
   else:
   p_prob*=(1)/(v+n1)
   if j in np.keys():
   n_prob*=dict3[j]
   else:
   n_prob*=(1)/(v+n2)
 # print(l[0],p_prob,n_prob)
  if p_prob>n_prob:
    predicted.append('pos')
  else:
  predicted.append('neg')


# print(actual)
```

print(predicted)

OUTPUT:-

```
[[3 1]
 [0 4]]
accuracy 0.875
precision 0.8
recall 1.0
```

5. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis the of heart patients using standard heart disease data set. You can use Java or Python ML Library classes /API.

```
import pandas as pd

import numpy as np

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

heart=pd.read_csv("/content/heart.csv")

heart

heart = heart.replace('?', np.nan)

print('Sample instances from the dataset are given below')

print (heart.head())

print("\n Attributes and datatypes")

print (heart.dtypes)

model =BayesianModel([('age', 'target'), ('sex','target'), ('exang','target'),
('cp','target'),('target','restecg'), ('target', 'chol')])

model.fit(heart, estimator=MaximumLikelihoodEstimator)

print("\n Inferencing with Bayesian Network:")

HeartDiseasetest_infer= VariableElimination(model)

print("\n 1. Probability of HeartDisease given evidence-restecg :1 and age=40")

q1=HeartDiseasetest_infer.query(variables=['target'], evidence={'restecg':1,'age':50})

print(q1)

print("\n 2.Probability of HeartDisease given evidence- cp:2 and sex-1")

q2=HeartDiseasetest_infer.query(variables=['target'], evidence={'cp':2,'sex':1})

print(q2)
```

OUTPUT:-

```
 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence-restecg :1 and age=40
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.4729 |
+-----------+---------------+
| target(1) |        0.5271 |
+-----------+---------------+

 2.Probability of HeartDisease given evidence- cp:2 and sex-1
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.4076 |
+-----------+---------------+
| target(1) |        0.5924 |
```

6. 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 numpy as np

from collections import Counter


def  entropy(labels):

    counter = Counter(labels)

    probabilities = [count / len(labels) for count in counter.values()]

    return -np.sum(probabilities * np.log2(probabilities))


def information_gain(data, attribute, labels):

    total_entropy = entropy(labels)

    attribute_values = set(data[attribute])

    weighted_entropy = 0

    for value in attribute_values:

        subset_labels = labels[data[attribute] == value]

        weighted_entropy += len(subset_labels) / len(labels) * entropy(subset_labels)

    return total_entropy - weighted_entropy
```

```
def id3(data, attributes, labels):
    # Base cases
    if len(set(labels)) == 1:  # All instances have the same label
        return labels[0]
    if len(attributes) == 0: # No more attributes to split on
        most_common_label = Counter(labels).most_common(1)[0][0]
        return most_common_label

    # Attribute selection
    information_gains = [information_gain(data, attribute, labels) for attribute in attributes]
    best_attribute_index = np.argmax(information_gains)
    best_attribute = attributes[best_attribute_index]

    # Recursion
    attribute_values = set(data[best_attribute])
    tree = {best_attribute: {}}
    for value in attribute_values:
        subset_indices = data[best_attribute] == value
        subset_data = data[subset_indices]
        subset_labels = labels[subset_indices]
        if len(subset_data) == 0:
            most_common_label = Counter(labels).most_common(1)[0][0]
            tree[best_attribute][value] = most_common_label
        else:
            remaining_attributes = attributes[:best_attribute_index] + attributes[best_attribute_index +
1:]
            tree[best_attribute][value] = id3(subset_data, remaining_attributes, subset_labels)

    return tree
def predict(instance, tree):
    if isinstance(tree, dict)==False:
```

```
        return tree

    attribute_value = instance[list(tree.keys())[0]]
    if attribute_value not in list(tree.values())[0]:
        return None

    child = tree[list(tree.keys())[0]][attribute_value]
    return predict(instance, child)

# Example usage
data = pd.read_csv('/content/play_tennis.csv')
attributes = data.columns[1:-1].tolist()
labels = data.iloc[:, -1].values
decision_tree = id3(data, attributes, labels)

# Example prediction
instance = {'outlook': 'Sunny', 'temp': 'Cool', 'humidity': 'High', 'wind': 'Weak'}
prediction = predict(instance, decision_tree)
print("Prediction:", prediction)
print(decision_tree)
```

OUTPUT:-

```
Prediction: No
{'outlook': {'Rain': {'wind': {'Weak': 'Yes', 'Strong': 'No'}}, 'Sunny': {'humidity': {'Normal': 'Yes', 'High': 'No'}}, 'Overcast': 'Yes'}}
```

7. Build an Artificial Neural Network by implementing the Backpropagation 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=5 #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
for i in range(epoch):
    #Forward Propogation
    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
    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:-

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.91755203]
 [0.90180959]
 [0.91814824]]
```

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.

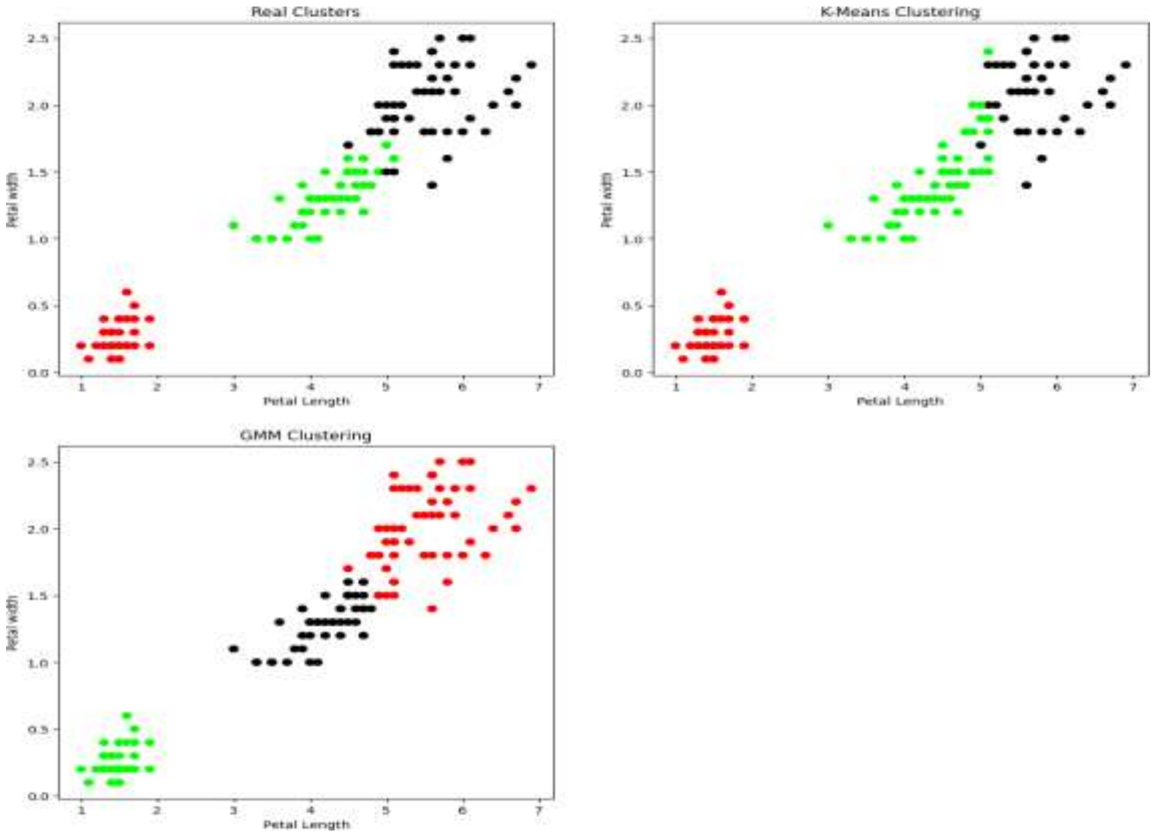
```
import matplotlib.pyplot as plt

from sklearn import datasets

import pandas as pd

from sklearn.cluster import KMeans

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']

# Build the K Means Model

model = KMeans (n_clusters=3)

model.fit(X)

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

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

# Plot the original Classifications using Petal features

plt.subplot(2, 2, 1)

plt.scatter(X.Petal_Length, X.Petal_width, c=colormap[y.Targets], s=40)

plt.title('Real Clusters')

plt.xlabel('Petal Length')

plt.ylabel('Petal width')

plt.subplot(2, 2, 2)

plt.scatter(X.Petal_Length, X.Petal_width, c=colormap[model.labels_], s=40)

plt.title('K-Means Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal width')

from sklearn import preprocessing

# transform your data such that its distribution will have a # mean value 0 and standard
deviation of 1.

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns)
```

```
from sklearn.mixture import GaussianMixture

gmm=GaussianMixture(n_components=3)

gmm.fit(xs)

gmm_y=gmm.predict(xs)

plt.subplot(2, 2, 3)

plt.scatter (X.Petal_Length, X.Petal_width, c=colormap[gmm_y], s=40)

plt.title('GMM Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')
```

OUTPUT:-

```
Observation: The GMM using EM algorithm based clustering matched the true
labels more closely than the Kmeans.
```



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

```
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

#Load dataset

iris=datasets.load_iris()

print("Iris Data set loaded...")

# print(iris)

# Split the data into train and test samples

x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)

print("Dataset is split into training and testing...")

print("size of trainng data and its label",x_train.shape,y_train.shape)

print("Size of trainng data and its label",x_test.shape, y_test.shape)

# Prints Label no. and their names

for i in range(len(iris.target_names)):

    print("Label", i, "-",str(iris.target_names[i]))

    # Create object of KNN classifier

    classifier = KNeighborsClassifier(n_neighbors=1)

#Perform Training

classifier.fit(x_train, y_train)

# Perform testing

y_pred=classifier.predict(x_test)

# Display the results

print("Results of Classification using K-nn with K=1")

for r in range(0,len(x_test)):

    print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), "Predicted-label:", str(y_pred[r]))

    print("Classification Accuracy:", classifier.score(x_test,y_test));
```

```
from sklearn.metrics import classification_report, confusion_matrix

print('Confusion Matrix')

print(confusion_matrix(y_test,y_pred))

print('Accuracy Metrics')

print(classification_report(y_test,y_pred))
```

OUTPUT:-

```
Iris Data set loaded...
Dataset is split into training and testing...
size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
KNeighborsClassifier(n_neighbors=1)
Label 1 - versicolor
KNeighborsClassifier(n_neighbors=1)
Label 2 - virginica
KNeighborsClassifier(n_neighbors=1)
Results of Classification using K-nn with K=1
 Sample: [6.1 2.9 4.7 1.4]  Actual-label: 1 Predicted-label: 1
Classification Accuracy: 0.9333333333333333
 Sample: [6.  2.2 5.  1.5]  Actual-label: 2 Predicted-label: 1
Classification Accuracy: 0.9333333333333333
 Sample: [4.5 2.3 1.3 0.3]  Actual-label: 0 Predicted-label: 0
Classification Accuracy: 0.9333333333333333
 Sample: [7.7 2.8 6.7 2. ]  Actual-label: 2 Predicted-label: 2
Classification Accuracy: 0.9333333333333333
 Sample: [6.7 3.1 4.4 1.4]  Actual-label: 1 Predicted-label: 1
Classification Accuracy: 0.9333333333333333
 Sample: [6.3 2.3 4.4 1.3]  Actual-label: 1 Predicted-label: 1
Classification Accuracy: 0.9333333333333333
 Sample: [5.1 3.7 1.5 0.4]  Actual-label: 0 Predicted-label: 0
Classification Accuracy: 0.9333333333333333
 Sample: [5.5 2.6 4.4 1.2]  Actual-label: 1 Predicted-label: 1
Classification Accuracy: 0.9333333333333333
 Sample: [5.4 3.9 1.3 0.4]  Actual-label: 0 Predicted-label: 0
Classification Accuracy: 0.9333333333333333
 Sample: [4.8 3.  1.4 0.3]  Actual-label: 0 Predicted-label: 0
Classification Accuracy: 0.9333333333333333
 Sample: [7.2 3.  5.8 1.6]  Actual-label: 2 Predicted-label: 2
Classification Accuracy: 0.9333333333333333
 Sample: [6.5 3.  5.2 2. ]  Actual-label: 2 Predicted-label: 2
Classification Accuracy: 0.9333333333333333
 Sample: [5.6 2.8 4.9 2. ]  Actual-label: 2 Predicted-label: 2
Classification Accuracy: 0.9333333333333333
 Sample: [4.9 3.1 1.5 0.2]  Actual-label: 0 Predicted-label: 0
Classification Accuracy: 0.9333333333333333
 Sample: [6.4 3.1 5.5 1.8]  Actual-label: 2 Predicted-label: 2
Classification Accuracy: 0.9333333333333333
Confusion Matrix
[[5 0 0]
 [0 4 0]
 [0 1 5]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
```

```
           1       0.80      1.00      0.89         4
           2       1.00      0.83      0.91         6

    accuracy                           0.93        15
   macro avg       0.93      0.94      0.93        15
weighted avg       0.95      0.93      0.93        15
```

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

```
import numpy as np
import pandas as pd
# kernel smoothing function
def kernel(point, xmat, k):
    m, n= np.shape(xmat)
    weights =np.mat(np.eye((m)))
    for j  in  range(m):
        diff =point-xmat[j]
        weights[j, j] = np.exp(diff* diff.T/ (-2.0 * k**2))  # smooth = exp
-(x-x0)^2/2k^2
    return weights
# function to return local weight of eah traiining example
def localweight (point, xmat, ymat, k):
    wt =kernel(point, xmat, k)
    W = ((xmat.T *(wt*xmat)).I )* (xmat.T* wt* ymat.T) # beta = (xtrans *
smooth * x)inverse *(xtrans * smooth * y)
    return W
def localweightRegression(xmat, ymat, k):
    m,n= np. shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i] *localweight(xmat[i], xmat, ymat, k)     # y = x0
* beta
    return ypred
import matplotlib.pyplot as plt

#import data
data = pd.read_csv('10-dataset.csv')

# place them in suitable data types
colA = np.array(data.total_bill)
colB= np.array(data.tip)
# print(colA)
# print(colB)

mcolA = np.mat (colA)
mcolB= np.mat (colB)

# print(mcolA)
# print(mcolB)

m = np.shape(mcolB) [1]
# print(m)
one = np.ones((1, m), dtype = int)
# print(one)

# print(one.T,mcolA.T)
# horizontal stacking
X = np.hstack((one. T, mcolA.T))
```

```
# print(X)
print(X. shape)

# predicting values using LWLR
ypred = localweightRegression(X, mcolB, 0.8)

# plotting the predicted graph
xsort = X.copy()
xsort.sort(axis=0)
plt.scatter (colA, colB, color='blue')
plt.plot(xsort[:, 1], ypred[X[:, 1].argsort(0)], color='yellow',
linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
OUTPUT:-
[[ 1.   16.99]
 [ 1.   10.34]
 [ 1.   21.01]
 [ 1.   23.68]
 [ 1.   24.59]
 [ 1.   25.29]
 [ 1.    8.77]
 [ 1.   26.88]
 [ 1.   15.04]
 [ 1.   14.78]
 [ 1.   10.27]
 [ 1.   35.26]
 [ 1.   15.42]
 [ 1.   18.43]
 [ 1.   14.83]
 [ 1.   21.58]
 [ 1.   10.33]
 [ 1.   16.29]
 [ 1.   16.97]
 [ 1.   20.65]
 [ 1.   17.92]
 [ 1.   20.29]
 [ 1.   15.77]
 [ 1.   39.42]
 [ 1.   19.82]
 [ 1.   17.81]
 [ 1.   13.37]
 [ 1.   12.69]
 [ 1.   21.7 ]
 [ 1.   19.65]
 [ 1.    9.55]
 [ 1.   18.35]
 [ 1.   15.06]
 [ 1.   20.69]
 [ 1.   17.78]
 [ 1.   24.06]
 [ 1.   16.31]
 [ 1.   16.93]
 [ 1.   18.69]
 [ 1.   31.27]
 [ 1.   16.04]
 [ 1.   17.46]
 [ 1.   13.94]
 [ 1.    9.68]
 [ 1.   30.4 ]
```

```
[ 1.   18.29]
[ 1.   22.23]
[ 1.   32.4 ]
[ 1.   28.55]
[ 1.   18.04]
[ 1.   12.54]
[ 1.   10.29]
[ 1.   34.81]
[ 1.    9.94]
[ 1.   25.56]
[ 1.   19.49]
[ 1.   38.01]
[ 1.   26.41]
[ 1.   11.24]
[ 1.   48.27]
[ 1.   20.29]
[ 1.   13.81]
[ 1.   11.02]
[ 1.   18.29]
[ 1.   17.59]
[ 1.   20.08]
[ 1.   16.45]
[ 1.    3.07]
[ 1.   20.23]
[ 1.   15.01]
[ 1.   12.02]
[ 1.   17.07]
[ 1.   26.86]
[ 1.   25.28]
[ 1.   14.73]
[ 1.   10.51]
[ 1.   17.92]
[ 1.   27.2 ]
[ 1.   22.76]
[ 1.   17.29]
[ 1.   19.44]
[ 1.   16.66]
[ 1.   10.07]
[ 1.   32.68]
[ 1.   15.98]
[ 1.   34.83]
[ 1.   13.03]
[ 1.   18.28]
[ 1.   24.71]
[ 1.   21.16]
[ 1.   28.97]
[ 1.   22.49]
[ 1.    5.75]
[ 1.   16.32]
[ 1.   22.75]
[ 1.   40.17]
[ 1.   27.28]
[ 1.   12.03]
[ 1.   21.01]
[ 1.   12.46]
[ 1.   11.35]
[ 1.   15.38]
[ 1.   44.3 ]
[ 1.   22.42]
[ 1.   20.92]
[ 1.   15.36]
```

```
[ 1.    20.49]
 [ 1.    25.21]
 [ 1.    18.24]
 [ 1.    14.31]
 [ 1.    14.  ]
 [ 1.     7.25]
 [ 1.    38.07]
 [ 1.    23.95]
 [ 1.    25.71]
 [ 1.    17.31]
 [ 1.    29.93]
 [ 1.    10.65]
 [ 1.    12.43]
 [ 1.    24.08]
 [ 1.    11.69]
 [ 1.    13.42]
 [ 1.    14.26]
 [ 1.    15.95]
 [ 1.    12.48]
 [ 1.    29.8 ]
 [ 1.     8.52]
 [ 1.    14.52]
 [ 1.    11.38]
 [ 1.    22.82]
 [ 1.    19.08]
 [ 1.    20.27]
 [ 1.    11.17]
 [ 1.    12.26]
 [ 1.    18.26]
 [ 1.     8.51]
 [ 1.    10.33]
 [ 1.    14.15]
 [ 1.    16.  ]
 [ 1.    13.16]
 [ 1.    17.47]
 [ 1.    34.3 ]
 [ 1.    41.19]
 [ 1.    27.05]
 [ 1.    16.43]
 [ 1.     8.35]
 [ 1.    18.64]
 [ 1.    11.87]
 [ 1.     9.78]
 [ 1.     7.51]
 [ 1.    14.07]
 [ 1.    13.13]
 [ 1.    17.26]
 [ 1.    24.55]
 [ 1.    19.77]
 [ 1.    29.85]
 [ 1.    48.17]
 [ 1.    25.  ]
 [ 1.    13.39]
 [ 1.    16.49]
 [ 1.    21.5 ]
 [ 1.    12.66]
 [ 1.    16.21]
 [ 1.    13.81]
 [ 1.    17.51]
 [ 1.    24.52]
 [ 1.    20.76]
```

```
[ 1.    31.71]
[ 1.    10.59]
[ 1.    10.63]
[ 1.    50.81]
[ 1.    15.81]
[ 1.     7.25]
[ 1.    31.85]
[ 1.    16.82]
[ 1.    32.9 ]
[ 1.    17.89]
[ 1.    14.48]
[ 1.     9.6 ]
[ 1.    34.63]
[ 1.    34.65]
[ 1.    23.33]
[ 1.    45.35]
[ 1.    23.17]
[ 1.    40.55]
[ 1.    20.69]
[ 1.    20.9 ]
[ 1.    30.46]
[ 1.    18.15]
[ 1.    23.1 ]
[ 1.    15.69]
[ 1.    19.81]
[ 1.    28.44]
[ 1.    15.48]
[ 1.    16.58]
[ 1.     7.56]
[ 1.    10.34]
[ 1.    43.11]
[ 1.    13.  ]
[ 1.    13.51]
[ 1.    18.71]
[ 1.    12.74]
[ 1.    13.  ]
[ 1.    16.4 ]
[ 1.    20.53]
[ 1.    16.47]
[ 1.    26.59]
[ 1.    38.73]
[ 1.    24.27]
[ 1.    12.76]
[ 1.    30.06]
[ 1.    25.89]
[ 1.    48.33]
[ 1.    13.27]
[ 1.    28.17]
[ 1.    12.9 ]
[ 1.    28.15]
[ 1.    11.59]
[ 1.     7.74]
[ 1.    30.14]
[ 1.    12.16]
[ 1.    13.42]
[ 1.     8.58]
[ 1.    15.98]
[ 1.    13.42]
[ 1.    16.27]
[ 1.    10.09]
[ 1.    20.45]
```

```
 [ 1.    13.28]
 [ 1.    22.12]
 [ 1.    24.01]
 [ 1.    15.69]
 [ 1.    11.61]
 [ 1.    10.77]
 [ 1.    15.53]
 [ 1.    10.07]
 [ 1.    12.6 ]
 [ 1.    32.83]
 [ 1.    35.83]
 [ 1.    29.03]
 [ 1.    27.18]
 [ 1.    22.67]
 [ 1.    17.82]
 [ 1.    18.78]]
(244, 2)
```