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

Source Code:

```
import pandas as pd
import numpy as np
df=pd.read_csv('finds.csv')
print(df)
df=df[df[df.columns[-1]]=='YES']
def findS(df):
  h=['pi']*(len(df.columns)-1)
  d=np.array(df)
  for i in d:
    if 'pi' in h:
      for j in range(len(h)):
        h[j]=i[j]
    else:
      for j in range(len(h)):
        if h[j]!=i[j]:
          h[j]='?'
  print(h)
findS(df)
```

Output:

```
       Sky  AirTemp Humidity   Wind  Water   Forecast EnjoySport
0 Sunny     Warm    Normal  Strong   Warm      Same        YES
1 Sunny     Warm      High  Strong   Warm      Same        YES
2 Rainy     Cold      High  Strong   Warm    Change         NO
3 Sunny     Warm      High  Strong   Cool    Change        YES
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

2. **Aim:** 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.

Source code:

```
import pandas as pd
import numpy as np
df=pd.read_csv('candidate.csv')
print(df)
def specified(s,p):
  if 'pi' in s:
    for j in range(len(s)):
      s[j]=p[j]
  else:
    for j in range(len(s)):
      if s[j]!=p[j]:
        s[j]='?'
  return s
def accept(l,df):
  for j in range(len(df)):
    if df.iloc[j,-1]=='YES':
      for k in range(len(l)):
        if l[k]!='?' and df.iloc[j,k]!=l[k]:
          return 0
    else:
      c=0
      for k in range(len(l)):
        if l[k]=='?' or df.iloc[j,k]==l[k]:
          c=c+1
      if c==len(l):
        return 0
  return 1
def generalised(df,i):
  g1=[]
  for j in range(len(df.columns)-1):
    m=list(np.unique(df[df.columns[j]]))
    m.remove(df.iloc[i,j])
    l=['?']*(len(df.columns)-1)
    for k in m:
      l[j]=k
      if accept(l,df):
        g1.append(l)
  return g1
def learned(s,df,g):
  lvs=[]
  for i in range(len(g)):
    for j in range(len(s)):
      l=g[i].copy()
      if g[i][j]=='?' and s[j]!='?':
          l[j]=s[j]
          if accept(l,df) and l not in g and l not in lvs:
            lvs.append(l)
  return lvs
def candidate(df):
  s=['pi']*(len(df.columns)-1)
  g=[]
  for i in range(len(df)):
    if df.iloc[i,-1]=='YES':
      s=specified(s,df.iloc[i])
    else:
      g.extend(generalised(df,i))
  if len(g)!=0:
    lvs=learned(s,df,g)
    print(s,'\n',g,'\n',lvs)
  else:
    print(s)
candidate(df)
```

Output:

```
     Sky  AirTemp Humidity   Wind  Water   Forecast EnjoySport
0 Sunny     Warm    Normal  Strong   Warm     Same        YES
1 Sunny     Warm      High  Strong   Warm     Same        YES
2 Rainy     Cold      High  Strong   Warm   Change         NO
3 Sunny     Warm      High  Strong   Cool   Change        YES
['Sunny', 'Warm', '?', 'Strong', '?', '?'] [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?',
'?']] [['Sunny', 'Warm', '?', '?', '?', '?'], ['Sunny', '?', '?', 'Strong', '?', '?'], ['?', 'Warm', '?',
'Strong', '?', '?']]
```

3. **Aim**: 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.

Source Code:

```
from sklearn.metrics import accuracy_score
df=pd.read_csv('tabletb.csv')
df=df.iloc[:,1:-1]
print(df.head())
x_train=df[:9]
x_test=df.iloc[9:]
yes_train=x_train['Play Tennis'][x_train['Play Tennis']=="Yes"].count()
no_train=x_train['Play Tennis'][x_train['Play Tennis']=="No"].count()
yes=yes_train/len(x_train)
no=no_train/len(x_train)
def probabilties(x_train):
  p_train={}
  for x in x_train.columns[:-1]:
    l=x_train[x].unique()
    y=n=0
    for z in l:
      k=x_train[x_train[x]==z][x_train.columns[-1]]
      y=k[k=="Yes"].count()/yes_train
      n=k[k=="No"].count()/no_train
      p_train.update({z:[y,n]})
  return p_train
p_train=probabilties(x_train)
def test(x_test,p_train):
  classify=[]
  for i in range(len(x_test)):
    y=yes
    n=no
    for j in (x_test.columns[:-1]):
      y*=p_train[x_test.iloc[i][j]][0]
      n*=p_train[x_test.iloc[i][j]][1]
    if y>n:
      classify.append("Yes")
    else:
      classify.append("No")
  return classify
actual=list(x_test.iloc[:,-1])
predicted=test(x_test,p_train)
print('Actual :',actual,',','Predicted :',predicted)
z=accuracy_score(actual, predicted)
print("Accuracy = ",100*z)
```

Output:

```
Actual : ['Yes', 'Yes', 'Yes', 'Yes', 'No'] , Predicted : ['Yes', 'No', 'Yes', 'Yes', 'No']
Accuracy = 80.0
```

4. **Aim:**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.

Source Code:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import pandas as pd
import numpy as np
df=pd.read_csv("text_classification.csv")
df['Text']=df['Text'].str.lower()
text_train=df[:10]
text_test=df[10:]
def train(text):
  vocab=[]
  pos=[]
  neg=[]
  for i in range(len(text)):
    s=text.iloc[i,:-1].tolist()
    l=s[0].split(" ")
    vocab.extend(l)
    if(text.iloc[i,-1]=="pos"):
      pos.extend(l)
    else:
      neg.extend(l)
  vocab=list(set(vocab))
  return pos,neg,vocab
def classify(test,tp,tn,p,n,vocab):
  classify=[]
  for i in range(len(test)):
    pos=tp
    neg=tn
    s=test[i].split(" ")
    for j in s:
      if j in vocab:
        pos*=(p.count(j)+1)/(len(p)+len(vocab))
        neg*=(n.count(j)+1)/(len(n)+len(vocab))
      else:
        pos*=1/(len(p)+len(vocab))
        neg*=1/(len(n)+len(vocab))
    if pos>=neg:
      classify.append("pos")
    else:
      classify.append("neg")
  return classify
p=text_train.iloc[:,-1]
tp=p[p=="pos"].count()/len(p)
tn=p[p=="neg"].count()/len(p)
pos,neg,vocab=train(text_train)
```

```
actual=text_test['Label'].tolist()
predicted=classify(text_test['Text'].tolist(),tp,tn,pos,neg,vocab)
print("Actual :",actual,",","Predicted :",predicted)
z=accuracy_score(actual, predicted)
c=confusion_matrix(actual, predicted)
precision=c[1][1]/(c[1][1]+c[0][1])
recall=c[1][1]/(c[1][1]+c[1][0])
print("Accuracy = ",100*z)
print("Precision =",precision)
print("Recall =",recall)
```

Output:

```
Actual: ['pos', 'neg', 'pos', 'neg', 'pos', 'neg', 'pos', 'neg']
Predicted: ['pos', 'neg', 'pos', 'neg', 'pos', 'pos', 'pos', 'neg']
Accuracy = 87.5
Precision : 0.8
Recall = 1.0
```

5. **Aim:**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.

Source Code:

```
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
import pandas as pd
data=pd.read_csv("heart.csv")
heart_disease=pd.DataFrame(data)
model = BayesianNetwork([('age','target'),('sex','target'),('exang','target'),
('cp','target'),('target','restecg'),('target','chol')])
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
HeartDisease_infer = VariableElimination(model)
q1=HeartDisease_infer.query(variables=['target'],evidence={'restecg':1})
print(q1)
q2=HeartDisease_infer.query(variables=['target'],evidence={'cp':2})
print(q2)
```

Output:

```
+-----------+---------------+
| target    |  phi(target) |
+===========+===============+
| target(0) |        0.4242 |
+-----------+---------------+
| target(1) |        0.5758 |
+-----------+---------------+

+-----------+---------------+
| target    |  phi(target) |
+===========+===============+
| target(0) |        0.3755 |
+-----------+---------------+
| target(1) |        0.6245 |
+-----------+---------------+
```

6. **Aim**: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.

Source Code:

```
import numpy as np
import math
import csv
def read_data(filename):
    with open(filename, 'r') as csvfile:
        datareader = csv.reader(csvfile, delimiter=',')
        headers = next(datareader)
        metadata = []
        traindata = []
        for name in headers:
            metadata.append(name)
        for row in datareader:
            traindata.append(row)
    return (metadata, traindata)
class Node:
    def___init_(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""
    def___str_(self):
        return self.attribute
def subtables(data, col, delete):
    dict = {}
    items = np.unique(data[:, col])
    count = np.zeros((items.shape[0], 1), dtype=np.int32)

    for x in range(items.shape[0]):
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                count[x] += 1
    for x in range(items.shape[0]):
        dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")
        pos = 0
        for y in range(data.shape[0]):
            if data[y, col] == items[x]:
                dict[items[x]][pos] = data[y]
                pos += 1
        if delete:
            dict[items[x]] = np.delete(dict[items[x]], col, 1)
    return items, dict
def entropy(S):
    items = np.unique(S)
    if items.size == 1:
        return 0
    counts = np.zeros((items.shape[0], 1))
    sums = 0
    for x in range(items.shape[0]):
        counts[x] = sum(S == items[x]) / (S.size * 1.0)
    for count in counts:
        sums += -1 * count * math.log(count, 2)
    return sums
def gain_ratio(data, col):
    items, dict = subtables(data, col, delete=False)
    total_size = data.shape[0]
    entropies = np.zeros((items.shape[0], 1))
    intrinsic = np.zeros((items.shape[0], 1))
    for x in range(items.shape[0]):
        ratio = dict[items[x]].shape[0]/(total_size * 1.0)
        entropies[x] = ratio * entropy(dict[items[x]][:, -1])
        intrinsic[x] = ratio * math.log(ratio, 2)
    total_entropy = entropy(data[:, -1])
    iv = -1 * sum(intrinsic)
    for x in range(entropies.shape[0]):
        total_entropy -= entropies[x]
    return total_entropy / iv
def create_node(data, metadata):
    if (np.unique(data[:, -1])).shape[0] == 1:
        node = Node("")
        node.answer = np.unique(data[:, -1])[0]
        return node
    gains = np.zeros((data.shape[1] - 1, 1))
    for col in range(data.shape[1] - 1):
        gains[col] = gain_ratio(data, col)
    split = np.argmax(gains)
    node = Node(metadata[split])
    metadata = np.delete(metadata, split, 0)
    items, dict = subtables(data, split, delete=True)
    for x in range(items.shape[0]):
        child = create_node(dict[items[x]], metadata)
        node.children.append((items[x], child))
    return node
def empty(size):
    s = ""
    for x in range(size):
        s += "  "
    return s
def print_tree(node, level):
    if node.answer != "":
        print(empty(level), node.answer)
        return
    print(empty(level), node.attribute)
    for value, n in node.children:
        print(empty(level + 1), value)
        print_tree(n, level + 2)
metadata, traindata = read_data("tennisdata.csv")
data = np.array(traindata)
node = create_node(data, metadata)
print_tree(node, 0)
```

Output:

```
Outlook
      Overcast
            b'Yes'
      Rainy
            Windy
                  b'False'
                        b'Yes'
                  b'True'
                        b'No'
      Sunny
            Humidity
                  b'High'
                        b'No'
                  b'Normal'
                        b'Yes'
```

7. **Aim:**Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Source Code:

```
 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))
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)
    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)
```

<u>Output:</u>

```
-----------Epoch- 1 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.85298387]
 [0.84199654]
 [0.85658346]]
-----------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:
 [[0.85335328]
 [0.84236671]
 [0.85695042]]
-----------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:
 [[0.85371778]
 [0.84273201]
 [0.85731248]]
-----------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:
 [[0.85407747]
 [0.84309253]
 [0.85766973]]
..............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:
 [[0.85443244]
 [0.84344836]
 [0.85802226]]
..............Epoch- 5 Ends..............
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.85443244]
 [0.84344836]
 [0.85802226]]
```

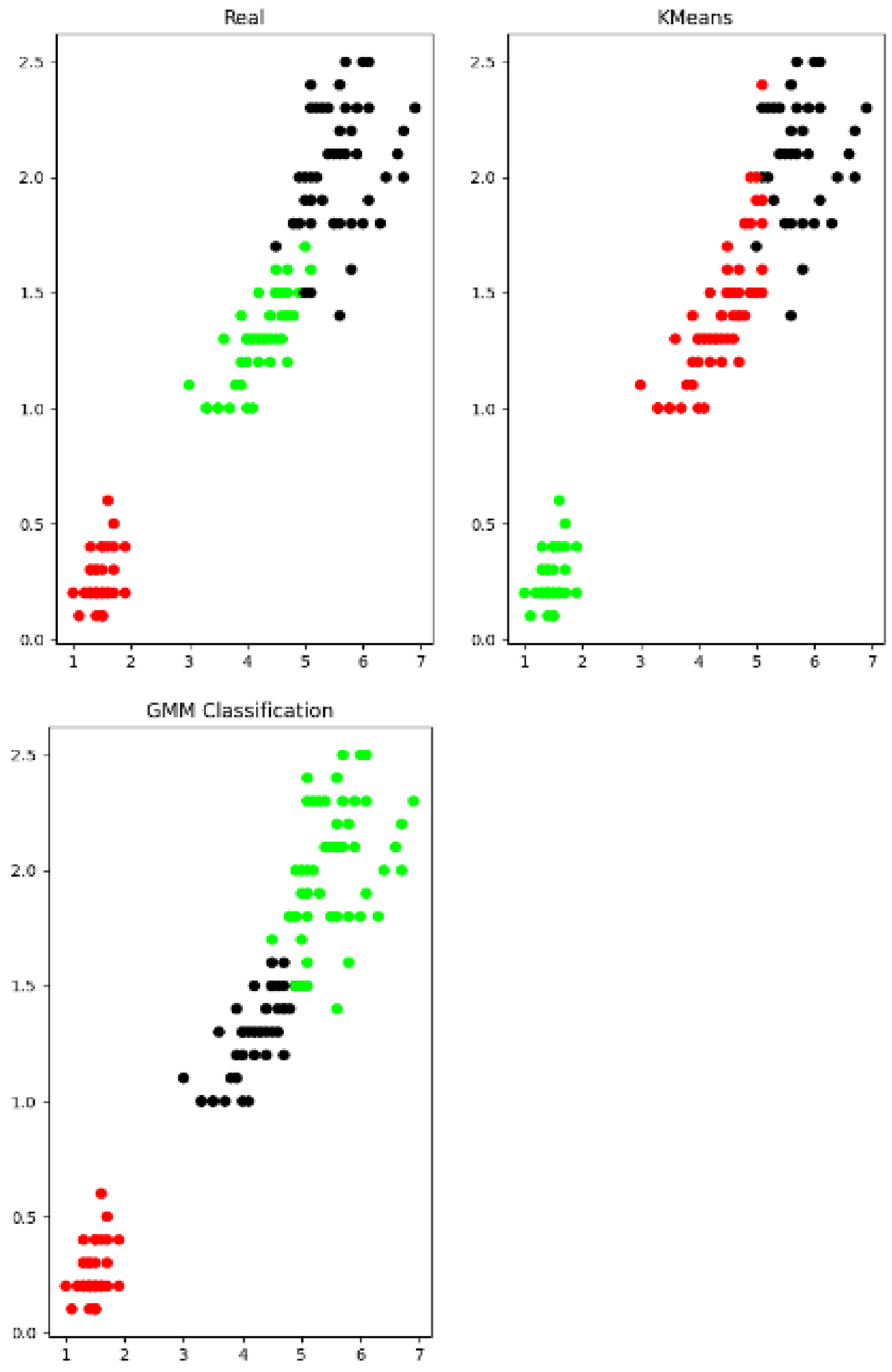
8. **Aim:**Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using K-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java / Python ML library classes/API in the program.

Source Code:

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']
dataset = pd.read_csv("8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}
y = [label[c] for c in dataset.iloc[:, -1]]
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])
# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])
print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y,
model.labels_))
# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

Output:

```
The accuracy score of K-Mean: 0.24
The Confusion matrixof K-Mean: [[ 0 50 0] [48 0 2] [14 0 36]]
The accuracy score of EM: 0.36666666666666664
The Confusion matrix of EM: [[50 0 0] [ 0 5 45] [ 0 50 0]]
```


Real
KMeans
GMM Classification
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

9. **Aim:**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 this problem.

Source Code:

```
import pandas as pd
import numpy as np
df=pd.read_csv("Iris.csv")
df=df.iloc[:,1:]
x_train=df[:130]
x_test=df[130:]
from math import sqrt
def euclideanDistance(iris,l):
  p=[]
  for i in range(len(iris)):
    dist=0
    for j in range(len(iris.columns)):
      dist+=(iris.iloc[i,j]-l[j])**2
    p.append(sqrt(dist))
  return p
def Rank(dist):
  k=sorted(dist)
  return [dist.index(k[0]),dist.index(k[1]),dist.index(k[2])]
def knn(x_train,l):
  dist=euclideanDistance(x_train.iloc[:,:-1],l)
  rank=Rank(dist)
  species=pd.unique(x_train.iloc[:,-1])
  max=0
  for x in species:
    delta=0
    for y in rank:
      if df.iloc[y,-1]==x:
        delta+=1
    if delta>max:
      max=delta
      c=x
  return c
classify=[]
test=x_test.iloc[:,:-1].to_numpy()
for i in range(len(x_test)):
  classify.append(knn(x_train,test[i]))
classifier=pd.DataFrame({"Actual":x_test.iloc[:,-1].tolist(),"Predicted":classify})
print(classifier)
```

Output:

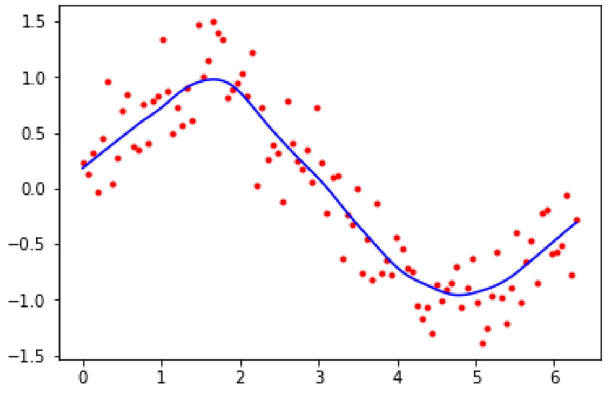
index	Actual	Predicted
0	Iris-virginica	Iris-virginica
1	Iris-virginica	Iris-virginica
2	Iris-virginica	Iris-virginica
3	Iris-virginica	Iris-versicolor
4	Iris-virginica	Iris-virginica
5	Iris-virginica	Iris-virginica
6	Iris-virginica	Iris-virginica
7	Iris-virginica	Iris-virginica
8	Iris-virginica	Iris-virginica
9	Iris-virginica	Iris-virginica
10	Iris-virginica	Iris-virginica
11	Iris-virginica	Iris-virginica
12	Iris-virginica	Iris-virginica
13	Iris-virginica	Iris-virginica
14	Iris-virginica	Iris-virginica
15	Iris-virginica	Iris-virginica
16	Iris-virginica	Iris-virginica
17	Iris-virginica	Iris-virginica
18	Iris-virginica	Iris-virginica
19	Iris-virginica	Iris-virginica

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

Source Code:

```
from math import ceil
import numpy as np
from scipy import linalg
def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
            A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x),
np.sum(weights * x * x)]])
            beta = linalg.solve(A, b)
            yest[i] = beta[0] + beta[1] * x[i]
        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2
    return yest
import math
n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f =0.25
iterations=3
yest = lowess(x, y, f, iterations)
import matplotlib.pyplot as plt
plt.plot(x,y,"r.")
plt.plot(x,yest,"b-")]
```

Output:



1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5
6