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

Program

```
import csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
  print("\n The given dataset is")
  for row in csv.reader(csvfile):
    a.append(row)
  print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
  if a[i][num_attribute] == 'yes':
    for j in range(0, num_attribute):
      if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
          hypothesis[j] = a[i][j]
      else:
          hypothesis[j] = '?'
    print("\n The hypothesis for the training instance {} is : \n"
.format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is
")
print(hypothesis)
```

Dataset →"enjoysport.csv"

sky	airtemp	humidity	wind	water	forcast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Output:

```
The given dataset is
[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport'],
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny',
'warm', 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold',
'high', 'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high',
'strong', 'cool', 'change', 'yes']]

 The total number of training instances are :  5

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 1 is :
 ['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 2 is :
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

 The hypothesis for the training instance 3 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

 The hypothesis for the training instance 4 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

 The hypothesis for the training instance 5 is :
 ['sunny', 'warm', '?', 'strong', '?', '?']

 The Maximally specific hypothesis for the training instance is
['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.

```python
import numpy as np
import pandas as pd

data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)
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):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("If instance is Negative ")
            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)
        print(general_h)
        print("\n")
```

```
        print("\n")

     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")
```

Dataset →"enjoysport.csv"

sky	airtemp	humidity	wind	water	forcast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

Output:

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?']]

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.

```
import math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""

def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic
```

```python
def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy

def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
```

```python
    node=Node(features[split])
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node

def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)

def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

'''Main program'''
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test.csv")
for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="   ")
    classify(node1,xtest,features)
```

<u>Dataset →</u>"<u>id3.csv</u>"

Outlook	Temperature	Humidity	Wind	Answer
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

<u>Dataset →</u>"id3_test.csv"

Outlook	Temperature	Humidity	Wind
rain	cool	normal	strong
sunny	mild	normal	strong

<u>Output:</u>

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   rain
     Wind
       weak
         yes
       strong
         no
```

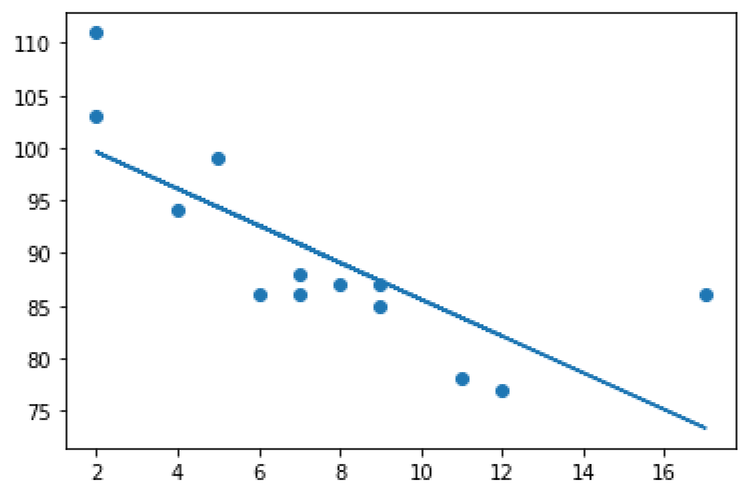
```
   sunny
     Humidity
       high
         no
       normal
         yes
   overcast
     yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:   yes
```

4. Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

a) Linear Regression

```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Output:



b) Logistic Regression

```python
import numpy
from sklearn import linear_model

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96,
4.52, 3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

def logit2prob(logr, X):
  log_odds = logr.coef_ * X + logr.intercept_
  odds = numpy.exp(log_odds)
  probability = odds / (1 + odds)
  return(probability)

print(logit2prob(logr, X))
```

__Output:__

```
[[0.60749955]
 [0.19268876]
 [0.12775886]
 [0.00955221]
 [0.08038616]
 [0.07345637]
 [0.88362743]
 [0.77901378]
 [0.88924409]
 [0.81293497]
 [0.57719129]
 [0.96664243]]
```

c) Binary Classifier

```
import numpy as np

class Perceptron(object):
  """ Perceptron Classifier
  Parameters
  ------------
  rate : float
    Learning rate (ranging from 0.0 to 1.0)
  number_of_iteration : int
    Number of iterations over the input dataset.

  Attributes:
  ------------

  weight_matrix : 1d-array
    Weights after fitting.

  error_matrix : list
     Number of misclassification in every epoch(one full training cycle on
the training set)

  """

  def __init__(self, rate = 0.01, number_of_iterations = 100):
    self.rate = rate
    self.number_of_iterations = number_of_iterations

  def fit(self, X, y):
    """ Fit training data

    Parameters:
    ------------
    X : array-like, shape = [number_of_samples, number_of_features]
      Training vectors.
    y : array-like, shape = [number_of_samples]
      Target values.
    Returns
    ------------
    self : object
    """
```

```
    self.weight_matrix = np.zeros(1 + X.shape[1])
    self.errors_list = []

    for _ in range(self.number_of_iterations):
      errors = 0
      for xi, target in zip(X, y):
        update = self.rate * (target - self.predict(xi))
        self.weight_matrix[1:] += update * xi
        self.weight_matrix[0] += update
        errors += int(update != 0.0)
      self.errors_list.append(errors)
    return self

  def dot_product(self, X):
    """ Calculate the dot product """
    return (np.dot(X, self.weight_matrix[1:]) + self.weight_matrix[0])

  def predict(self, X):
    """ Predicting the label for the input data """
    return np.where(self.dot_product(X) >= 0.0, 1, 0)


if __name__ == '__main__':
  X = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1,
0, 1], [1, 1, 0]])
  y = np.array([0, 1, 1, 1, 1, 1, 1])
  p = Perceptron()
  p.fit(X, y)
  print("Predicting the output of [1, 1, 1] = {}".format(p.predict([1, 1,
1])))
```

Output:

```
Predicting the output of [1, 1, 1] = 1
```

Experiment-5: Develop a program for Bias, Variance, Remove duplicates , Cross Validation

First, you must install the mlxtend library;

```
sudo pip install mlxtend
```

```
# estimate the bias and variance for a regression model
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
dataframe = read_csv(url, header=None)
# separate into inputs and outputs
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# define the model
model = LinearRegression()
# estimate bias and variance
mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test, loss='mse',
num_rounds=200, random_seed=1)
# summarize results
print('MSE: %.3f' % mse)
print('Bias: %.3f' % bias)
print('Variance: %.3f' % var)
```

Output:

MSE: 22.487

Bias: 20.726

Variance: 1.761

Experiment-6: Write a program to implement Categorical Encoding, One-hot EncodingEx

6a.py

```
# Program for demonstration of one hot encoding
# import libraries
import numpy as np
import pandas as pd
# import the data required
data = pd.read_csv("fruit_data.csv")
print(data.head())
```

Dataset→"fruit_data.csv"

Fruit	Categorical value of fruit	Price
apple	1	5
mango	2	10
apple	1	15
orange	3	20

Output:

```
Fruit  Categorical value of fruit  Price
0   apple                          1      5
1   mango                          2     10
2   apple                          1     15
3  orange                          3     20
```

6b.py

```
#importing libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
#Retrieving data
data = pd.read_csv('employee_data.csv')
# Converting type of columns to category
data['Gender']=data['Gender'].astype('category')
data['Remarks']=data['Remarks'].astype('category')
#Assigning numerical values and storing it in another columns
data['Gen_new']=data['Gender'].cat.codes
data['Rem_new']=data['Remarks'].cat.codes
```

```
#Create an instance of One-hot-encoder
enc=OneHotEncoder()
#Passing encoded columns
enc_data=pd.DataFrame(enc.fit_transform(data[['Gen_new','Rem_new']]).toarr
ay())

#Merge with main
New_df=data.join(enc_data)
print(New_df)
```

Dataset→ "employee_data.csv"

	employee_id	Gender	Remarks
0	45	Male	Nice
1	78	Female	Good
2	56	Female	Great
3	12	Male	Great
4	7	Female	Nice

Output:

```
   Unnamed: 0  employee_id  Gender Remarks  Gen_new  Rem_new    0    1    2  \
0           0           45    Male    Nice        1        2  0.0  1.0  0.0
1           1           78  Female    Good        0        0  1.0  0.0  1.0
2           2           56  Female   Great        0        1  1.0  0.0  0.0
3           3           12    Male   Great        1        1  0.0  1.0  0.0
4           4            7  Female    Nice        0        2  1.0  0.0  0.0

     3    4
0  0.0  1.0
1  0.0  0.0
2  1.0  0.0
3  1.0  0.0
4  0.0  1.0
```

7. 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) # two inputs
[sleep,study]
y = np.array(([92], [86], [89]), dtype=float) # one output [Expected % in
Exams]
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=5000  #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))
#weight of the link from input node to hidden node
bh=np.random.uniform(size=(1,hiddenlayer_neurons)) # bias of the link from
input node to hidden node
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))  #weight
of the link from hidden node to output node
bout=np.random.uniform(size=(1,output_neurons))  #bias of the link from
hidden node to output node


#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
```

```python
#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)

#how much hidden layer weights contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

# dotproduct of nextlayererror and currentlayerop
    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *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:
 [[0.89407756]
 [0.88158983]
 [0.89441448]]
```

8. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

```
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)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

Output:

```
[[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-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-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 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 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 2]
Confusion Matrix
[[12  0  0]
 [ 0 16  1]
 [ 0  1 15]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.94      0.94      0.94        17
           2       0.94      0.94      0.94        16

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

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

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):# add bias term
 x0 = np.r_[1, x0] # Add one to avoid the loss in information
 X = np.c_[np.ones(len(X)), X]

 # fit model: normal equations with kernel
 xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

 beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot
Product


 # predict value
 return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
 return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
 # prediction through regression
 prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
 plot = figure(plot_width=400, plot_height=400)
```

```
 plot.title.text='tau=%g' % tau
 plot.scatter(X, Y, alpha=.3)
 plot.line(domain, prediction, line_width=2, color='red')
 return plot

show(gridplot([
 [plot_lwr(10.), plot_lwr(1.)],
 [plot_lwr(0.1), plot_lwr(0.01)]]))
```

Output:

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-3.05946315 -2.90064801 -2.78706527 -2.88962629 -3.06183363 -3.16309513
 -2.94624656 -2.91043628 -2.97360818]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```

10. 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
msg=pd.read_csv('naivetext.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 ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.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('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names(
))
# 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, Confusion matrix, Precision and Recall
from sklearn import metrics
print('\n Accuracy of the classifer is',
metrics.accuracy_score(ytest,predicted))
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n The value of Precision' ,
metrics.precision_score(ytest,predicted))
print('\n The value of Recall' , metrics.recall_score(ytest,predicted))
```

Dataset→"naivetext.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

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

 The total number of Training Data : (13,)

 The total number of Test Data : (5,)

 The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'boss', 'dance', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have', 'holiday', 'horrible', 'house', 'is', 'juice', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'stuff', 'taste', 'the', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will']

 Accuracy of the classifer is 0.6
 Confusion matrix
[[2 1]
 [1 1]]
 The value of Precision 0.5
 The value of Recall 0.5
```

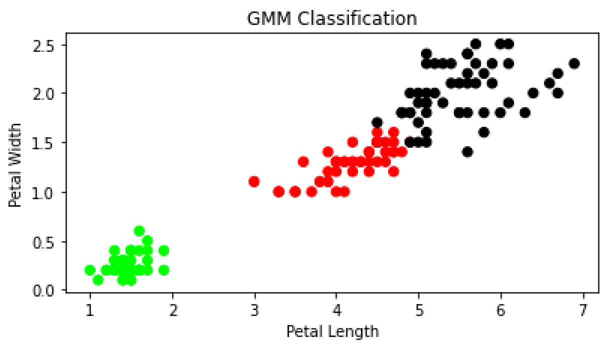
11. Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using the k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

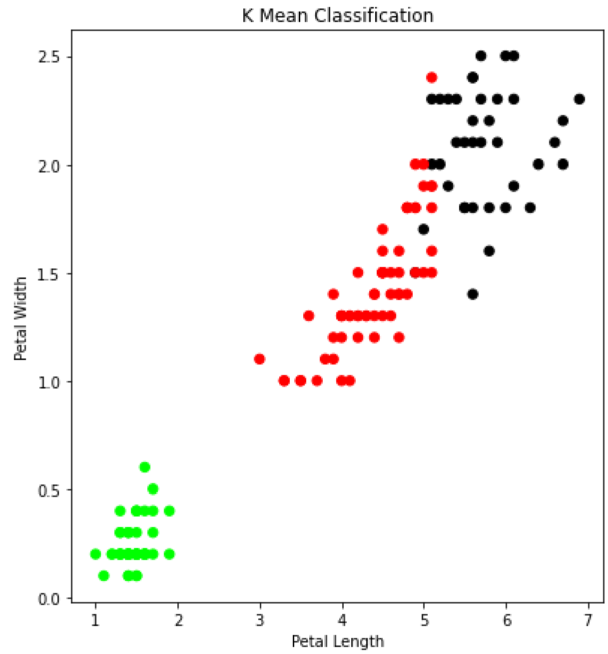
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

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']

model = KMeans(n_clusters=3)
model.fit(X)


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

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

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length,    X.Petal_Width,    c=colormap[model.labels_],
s=40)
plt.title('K Mean Classification')
```

```python
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The    accuracy    score    of    K-Mean:    ',sm.accuracy_score(y,
model.labels_))
print('The    Confusion    matrixof    K-Mean:    ',sm.confusion_matrix(y,
model.labels_))


from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_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:  0.3333333333333333
The Confusion matrix of EM:  [[ 0 50  0]
 [45  0  5]
 [ 0  0 50]]
```


GMM Classification
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7


K Mean Classification
Petal Width
Petal Length
0.0
0.5
1.0
1.5
2.0
2.5
1
2
3
4
5
6
7

12. Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Goto Website https://www.kaggle.com/datasets/sukhmanibedi/cars4u

Download `used_cars.csv` and Upload to the environment

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#to ignore warnings
import warnings
warnings.filterwarnings('ignore')
data = pd.read_csv("used_cars.csv")

data.head()

data.tail()

data.info()

data.nunique()

data.isnull().sum()

# Remove S.No. column from data
data = data.drop(['S.No.'], axis = 1)
data.info()

from datetime import date
date.today().year
data['Car_Age']=date.today().year-data['Year']
data.head()

print(data.Brand.unique())
print(data.Brand.nunique())
```

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.60	998.0	58.16	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67	1582.0	126.20	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.20	1199.0	88.70	5.0	8.61	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77	1248.0	88.76	7.0	NaN	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.20	1968.0	140.80	5.0	NaN	17.74

Experiment-13:

Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

Experiment-14:

Write a program to Implement Support Vector Machines and Principle Component Analysis

Experiment-15:

Write a program to Implement Principle Component Analysis