S NO	LIST OF EXPERIMENT
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.
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.
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.
4	Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier
5	Develop a program for Bias, Variance, Remove duplicates , Cross Validation
6	Write a program to implement Categorical Encoding, One-hot Encoding
7	Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.
8	Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.
9	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
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.
11	Apply EM algorithm to cluster a Heart Disease Data Set. 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.
12	Exploratory Data Analysis for Classification using Pandas or Matplotlib.
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
14	Write a program to Implement Support Vector Machines and Principle Component Analysis
15	Write a program to Implement Principle Component Analysis

Experiment – 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 datafrom a .CSV file.

Aim: Demonstration of FIND-S algorithm for finding the most specific hypothesis

```
Import csv
With open('tennis.csv', 'r') as f:
Reader=csv.reader(f)
Your_list=list(reader)
H=[['0', '0', '0', '0', '0']]
For i in your list
Print(i)
Ifi[-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 hypothesis is")
print(h)
```

Output

'Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same',True

'Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same',True

'Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change',Fals

'Sunny', 'Warm', 'High', 'Strong', 'Cool','Change',True

Maximally Specific set

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

Experiment – 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 allhypotheses consistent with the training examples.

Aim: Demonstration of Candidate-Elimination algorithm

Program code

class Holder:

factors={} #Initialize an empty dictionary

attributes = () #declaration of dictionaries parameters with an arbitrary length

'''

Constructor of class Holder holding two parameters,self refers to the instance of the class

'''

def init (self,attr): # self.attributes = attrfor i in attr:

self.factors[i]=[]

def add_values(self,factor,values):self.factors[factor]=values

class CandidateElimination:

Positive={}

#Initialize positive empty dictionary Negative={}

#Initialize negative empty dictionary

def init (self,data,fact): self.num_factors = len(data[0][0])self.factors = fact.factors

self.attr = fact.attributesself.dataset = data

def run_algorithm(self):'''

Initialize the specific and general boundaries, and loop the dataset against thealgorithm

'''

G = self.initializeG()S = self.initializeS()

'''

Programmatically populate list in the iterating variable trial_set'''

count=0

for trial_set in self.dataset:

if self.is_positive(trial_set): #if trial set/example consists of positive examples

G = self.remove_inconsistent_G(G,trial_set[0]) #remove inconsitent data fromthe general boundary

S_new = S[:] #initialize the dictionary with no key-value pairprint (S_new)

for s in S:

if not self.consistent(s,trial_set[0]):S_new.remove(s)

generalization = self.generalize_inconsistent_S(s,trial_set[0])generalization =

self.get_general(generalization,G)

if generalization: S_new.append(generalization)

S = S_new[:]

S = self.remove_more_general(S)print(S)

else:#if it is negative

S = self.remove_inconsistent_S(S,trial_set[0]) #remove inconsitent data fromthe specific boundary

G_new = G[:] #initialize the dictionary with no key-value pair (dataset cantake any value)

print (G_new)for g in G:

ifself.consistent(g,trial_set[0]):G_new.remove(g)

specializations = self.specialize_inconsistent_G(g,trial_set[0])specializationss =

self.get_specific(specializations,S)

if specializations != []: G_new += specializationss

G = G_new[:]

G = self.remove_more_specific(G)print(G)

print (S)print (G)

```
def initializeS(self):
```

''' Initialize the specific boundary '''

S = tuple(['-' for factor in range(self.num_factors)]) #6 constraints in the vectorreturn [S]

def initializeG(self):

''' Initialize the general boundary '''

G = tuple(['?' for factor in range(self.num_factors)]) # 6 constraints in the vectorreturn [G]

def is_positive(self,trial_set):

''' Check if a given training trial_set is positive '''if trial_set[1] == 'Y':

return True

elif trial_set[1] == 'N':return False

else:

raise TypeError("invalid target value")

def match_factor(self,value1,value2):

''' Check for the factors values match, necessary while checking the consistency oftraining trial_set with

the hypothesis '''

if value1 == '?' or value2 == '?':return True

elif value1 == value2 :return True

return False

def consistent(self,hypothesis,instance):

''' Check whether the instance is part of the hypothesis '''for i,factor in enumerate(hypothesis):

if not self.match_factor(factor,instance[i]):return False

return True

def remove_inconsistent_G(self,hypotheses,instance):''' For a positive trial_set, the hypotheses in G

inconsistent with it should be removed '''G_new = hypotheses[:]

for g in hypotheses:

if not self.consistent(g,instance):G_new.remove(g)

return G_new

def remove_inconsistent_S(self,hypotheses,instance):''' For a negative trial_set, the hypotheses in S inconsistent with it should be removed '''S_new = hypotheses[:]

for s in hypotheses:

if self.consistent(s,instance):S_new.remove(s)

return S_new

def remove_more_general(self,hypotheses):

''' After generalizing S for a positive trial_set, the hypothesis in Sgeneral than others in S should be

removed '''

S_new = hypotheses[:]for old in hypotheses:

for new in S_new:

if old!=new and self.more_general(new,old):S_new.remove[new]

return S_new

def remove_more_specific(self,hypotheses):

''' After specializing G for a negative trial_set, the hypothesis in Gspecific than others in G should be

removed '''

G_new = hypotheses[:]for old in hypotheses: for new in G_new:

if old!=new and self.more_specific(new,old):G_new.remove[new]

return G_new

def generalize_inconsistent_S(self,hypothesis,instance):

''' When a inconsistent hypothesis for positive trial_set is seen in the specificboundary S,

itshould be generalized to be consistent with the trial_set ... we will get onehypothesis'''

hypo = list(hypothesis) # convert tuple to list for mutabilityfor i,factor in enumerate(hypo):

if factor == '-':

hypo[i] = instance[i]

elif not self.match_factor(factor,instance[i]):hypo[i] = '?'

generalization = tuple(hypo) # convert list back to tuple for immutabilityreturn generalization

def specialize_inconsistent_G(self,hypothesis,instance):

''' When a inconsistent hypothesis for negative trial_set is seen in the generalboundary G should be specialized to be consistent with the trial_set.. we will get a set ofhypotheses '''

specializations = []

hypo = list(hypothesis) # convert tuple to list for mutabilityfor i,factor in enumerate(hypo):

if factor == '?':

values = self.factors[self.attr[i]]for j in values:

if instance[i] != j:hyp=hypo[:] hyp[i]=j

hyp=tuple(hyp) # convert list back to tuple for immutabilityspecializations.append(hyp)

return specializations

def get_general(self,generalization,G):

''' Checks if there is more general hypothesis in G

for a generalization of inconsistent hypothesis in S

in case of positive trial_set and returns valid generalization '''

for g in G:

if self.more_general(g,generalization):return generalization

return None

def get_specific(self,specializations,S):

''' Checks if there is more specific hypothesis in Sfor each of hypothesis in specializations of an

inconsistent hypothesis in G in case of negative trial_setand return the valid specializations'''

valid_specializations = [] for hypo in specializations:

for s in S:

if self.more_specific(s,hypo) or s==self.initializeS()[0]:valid_specializations.append(hypo)

return valid_specializations

def exists_general(self,hypothesis,G):

'''Used to check if there exists a more general hypothesis ingeneral boundary for version space'''

for g in G:

if self.more_general(g,hypothesis):return True

```
return False
```

```
def exists_specific(self,hypothesis,S):

'''Used to check if there exists a more specific hypothesis ingeneral boundary for version space'''

for s in S:

if self.more_specific(s,hypothesis):return True

return False

def more_general(self,hyp1,hyp2):

''' Check whether hyp1 is more general than hyp2 '''hyp = zip(hyp1,hyp2)

for i,j in hyp:if i == '?':

continue


elif j == '?':

if i != '?':

return False

elif i != j:

return False

else:

continue

return True

def more_specific(self,hyp1,hyp2): ''' hyp1 more specific than hyp2 is

equivalent to hyp2 being more general than hyp1 '''return self.more_general(hyp2,hyp1)

dataset=[(('sunny','warm','normal','strong','warm','same'),'Y'),(('sunny','warm','high','stron

g','warm','same'),'Y'),(('rainy','cold','high','strong','warm','change'),'N'),(('sunny','warm','hi

gh','strong','cool','change'),'Y')]

attributes =('Sky','Temp','Humidity','Wind','Water','Forecast')f = Holder(attributes)

f.add_values('Sky',('sunny','rainy','cloudy')) #sky can be sunny rainy or cloudy

f.add_values('Temp',('cold','warm')) #Temp can be sunny cold or warm

f.add_values('Humidity',('normal','high')) #Humidity can be normal or high
```

f.add_values('Wind',('weak','strong')) #wind can be weak or strong f.add_values('Water',('warm','cold'))

#water can be warm or cold f.add_values('Forecast',('same','change')) #Forecast can be same or change

a = CandidateElimination(dataset,f) #pass the dataset to the algorithm class and call therun algoritm method

a.run_algorithm()

Output

[('sunny', 'warm', 'normal', 'strong', 'warm', 'same')]

[('sunny', 'warm', 'normal', 'strong', 'warm','same')]

[('sunny', 'warm', '?', 'strong', 'warm', 'same')]

[('?', '?', '?', '?', '?', '?')]

[('sunny', '?', '?', '?', '?', '?'), ('?', 'warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'same')]

[('sunny', 'warm', '?', 'strong', 'warm', 'same')]

[('sunny', 'warm', '?', 'strong', '?', '?')]

[('sunny', 'warm', '?', 'strong', '?', '?')]

[('sunny']

Experiment-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.

Aim: Demonstration of ID3 algorithm

Dataset: Tennis dataset

Program code:

import numpy as npimport math

from data_loader import read_data

class Node:

def init (self, attribute): self.attribute = attributeself.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("tennis.csv")data = np.array(traindata)

node = create_node(data, metadata)print_tree(node, 0)
```

Data_loader.py

```
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)
```

Input:

Tennis.csv

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

Output

outlook

overcastb'yes'

rain

wind

b'strong'b'no' b'weak' b'yes'

sunny

humidityb'high'b'no'

b'normal'b'yes

Experiment – 4:

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

Aim:
To solve the real-world problems using the machine learning methods. Linear Regression and Logistic Regression

Dataset: std_marks.csv-constructed on own by using students lab internal and external marks.

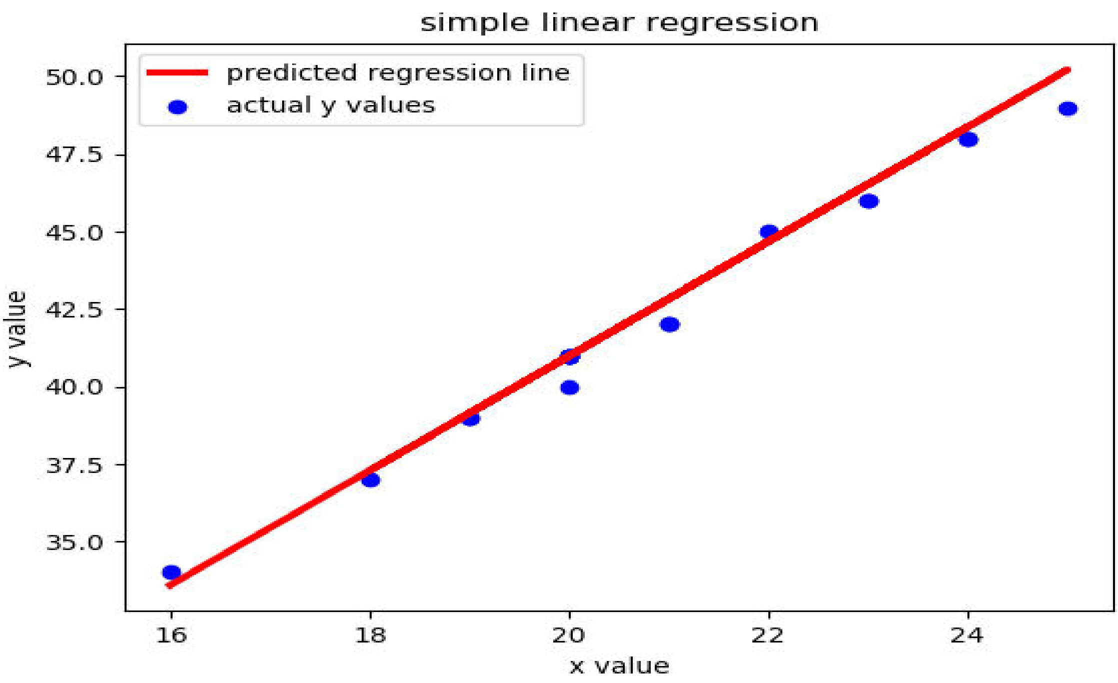
Program code:

```
import pandas as pd
from sklearn import linear_model
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
data=pd.read_csv(r"E:\sudhakar\std_marks.csv")
print('First 5 rows of the data set are:')
print(data.head())
dim=data.shape
print('Dimensions of the data set are',dim)
print('Statistics of the data are:')
```

```
print(data.describe())
print('Correlation matrix of the data set is:')
print(data.corr())
x_set=data[['internal']]
print('First 5 rows of features set are:')
print(x_set.head())
y_set=data[['external']]
print('First 5 rows of features set are:')
print(y_set.head())
x_train,x_test, y_train, y_test = train_test_split(x_set,y_set, test_size = 0.3)
model=linear_model.LinearRegression()
model.fit(x_train,y_train)
print('Regression coefficient is',float(model.coef_))
print('Regression intercept is',float(model.intercept_))
y_pred=model.predict(x_test)
y_preds=[]
for i in y_pred:
7 y_preds.append(float(i))
print('Predicted values for test data are:')
print(y_preds)
print('mean squared error is ',mean_squared_error(y_test,y_pred))
plt.scatter(x_test,y_test,color='blue',label='actual y values')
plt.plot(x_test,y_pred,color='red',linewidth=3,label='predicted regression line')
plt.ylabel('y value')
plt.xlabel('x value')
plt.title('simple linear regression')
plt.legend(loc='best')
plt.show()
```

Output screen shots:

```
C:\Users\harsini>python linearregression.py
First 5 rows of the data set are:
   internal  external
0        23        47
1        18        37
2        20        41
3        25        50
4        24        49
Dimensions of the data set are (60, 2)
Statistics of the data are:
        internal   external
count  60.000000  60.000000
mean   21.033333  42.800000
std     2.449259   4.505364
min    16.000000  34.000000
25%    19.000000  39.000000
50%    21.000000  42.500000
75%    23.000000  46.250000
max    25.000000  50.000000
Correlation matrix of the data set is:
          internal  external
internal  1.000000  0.991316
external  0.991316  1.000000
First 5 rows of features set are:
   internal
0        23
1        18
2        20
3        25
4        24
First 5 rows of features set are:
   external
0        47
1        37
2        41
3        50
4        49
Regression coefficient is 1.847382270211416
Regression intercept is 4.032664912439856
Predicted values for test data are:
[46.522457127302424, 50.217221667725255, 40.980310316668174, 39.13292804645676, 48.36983939751384, 40.980310316668174, 4
0.980310316668174, 37.28554577624534, 44.675074857091005, 39.13292804645676, 46.522457127302424, 42.82769258687959, 42.8
2769258687959, 48.36983939751384, 40.980310316668174, 40.980310316668174, 40.980310316668174, 33.59078123582251]
mean squared error is  0.2791179492633819
```



Exercise 1b:

Program code:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.preprocessing import StandardScaler
data=pd.read_csv(r"E:\sudhakar\heart.csv")
print('The first 5 rows of the data set are:')
print(data.head())
dim=data.shape
print('Dimensions of the data set are',dim)
print('Statistics of the data are:')
print(data.describe())
print('Correlation matrix of the data set is:')
print(data.corr())
class_lbls=data['target'].unique()
class_labels=[]
for x in class_lbls:
   class_labels.append(str(x))
print('Class labels are:')
print(class_labels)
sns.countplot(data['target'])
col_names=data.columns
```

```
feature_names=col_names[:-1]
feature_names=list(feature_names)
print('Feature names are:')
print(feature_names)
x_set = data.drop(['target'], axis=1)
print('First 5 rows of features set are:')
print(x_set.head())
y_set=data[['target']]
print('First 5 rows of features set are:')
print(y_set.head())
scaler=StandardScaler()
x_train,x_test, y_train, y_test = train_test_split(x_set,y_set, test_size = 0.3)
scaler.fit(x_train)
x_train=scaler.transform(x_train)
model = LogisticRegression()
model.fit(x_train, y_train)
x_test=scaler.transform(x_test)
y_pred=model.predict(x_test)
print('Predicted class labels for test data are:')
print(y_pred)
print("Accuracy:",accuracy_score(y_test, y_pred))
print("Precision:",precision_score(y_test, y_pred))
print("Recall:",recall_score(y_test, y_pred))
```

```
print(classification_report(y_test,y_pred,target_names=class_labels))
cm=confusion_matrix(y_test,y_pred)
df_cm = pd.DataFrame(cm, columns=class_labels, index = class_labels)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
sns.set(font_scale=1.5)
sns.heatmap(df_cm, annot=True,cmap="Blues",fmt='d')
```

Output screen shots:

```
(base) C:\Users\harsini>python logisticregression.py
The first 5 rows of the data set are:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  ca  thal  target
0   63    1   3       145   233    1        0      150      0      2.3      0   0     1       1
1   37    1   2       130   250    0        1      187      0      3.5      0   0     2       1
2   41    0   1       130   204    0        0      172      0      1.4      2   0     2       1
3   56    1   1       120   236    0        1      178      0      0.8      2   0     2       1
4   57    0   0       120   354    0        1      163      1      0.6      2   0     2       1
Dimensions of the data set are (303, 14)
```

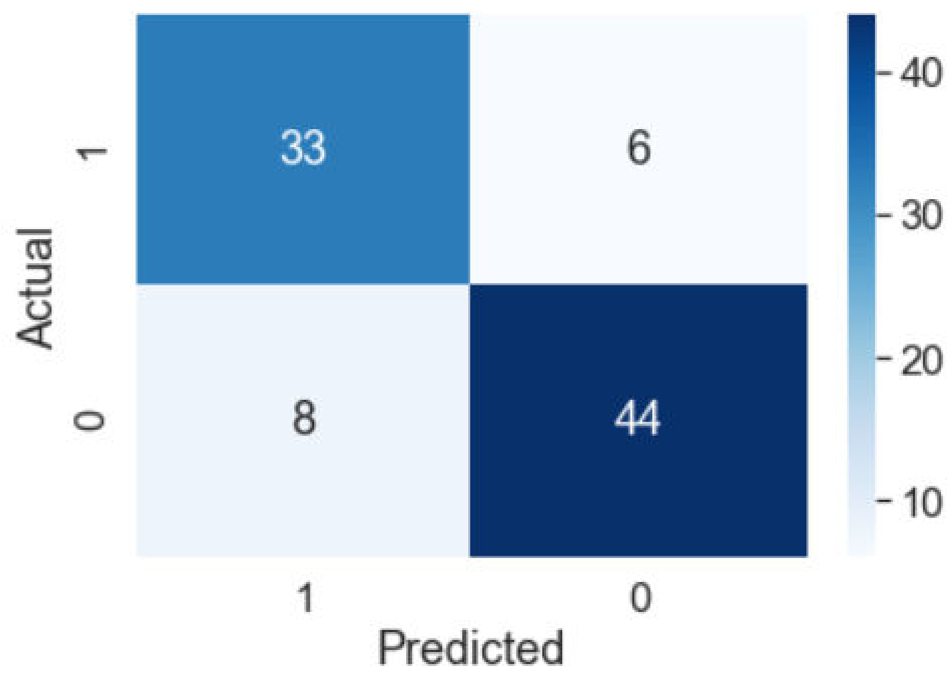
```
Statistics of the data are:
              age         sex          cp    trestbps  ...       slope          ca        thal      target
count  303.000000  303.000000  303.000000  303.000000  ...  303.000000  303.000000  303.000000  303.000000
mean    54.366337    0.683168    0.966997  131.623762  ...    1.399340    0.729373    2.313531    0.544554
std      9.082101    0.466011    1.032052   17.538143  ...    0.616226    1.022606    0.612277    0.498835
min     29.000000    0.000000    0.000000   94.000000  ...    0.000000    0.000000    0.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  ...    1.000000    0.000000    2.000000    0.000000
50%     55.000000    1.000000    1.000000  130.000000  ...    1.000000    0.000000    2.000000    1.000000
75%     61.000000    1.000000    2.000000  140.000000  ...    2.000000    1.000000    3.000000    1.000000
max     77.000000    1.000000    3.000000  200.000000  ...    2.000000    4.000000    3.000000    1.000000

[8 rows x 14 columns]
```

```
Correlation matrix of the data set is:
               age       sex        cp  trestbps      chol  ...   oldpeak     slope        ca      thal    target
age       1.000000 -0.098447 -0.068653  0.279351  0.213678  ...  0.210013 -0.168814  0.276326  0.068001 -0.225439
sex      -0.098447  1.000000 -0.049353 -0.056769 -0.197912  ...  0.096093 -0.030711  0.118261  0.210041 -0.280937
cp       -0.068653 -0.049353  1.000000  0.047608 -0.076904  ... -0.149230  0.119717 -0.181053 -0.161736  0.433798
trestbps  0.279351 -0.056769  0.047608  1.000000  0.123174  ...  0.193216 -0.121475  0.101389  0.062210 -0.144931
chol      0.213678 -0.197912 -0.076904  0.123174  1.000000  ...  0.053952 -0.004038  0.070511  0.098803 -0.085239
fbs       0.121308  0.045032  0.094444  0.177531  0.013294  ...  0.005747 -0.059894  0.137979 -0.032019 -0.028046
restecg  -0.116211 -0.058196  0.044421 -0.114103 -0.151040  ... -0.058770  0.093045 -0.072042 -0.011981  0.137230
thalach  -0.398522 -0.044020  0.295762 -0.046698 -0.009940  ... -0.344187  0.386784 -0.213177 -0.096439  0.421741
exang     0.096801  0.141664 -0.394280  0.067616  0.067023  ...  0.288223 -0.257748  0.115739  0.206754 -0.436757
oldpeak   0.210013  0.096093 -0.149230  0.193216  0.053952  ...  1.000000 -0.577537  0.222682  0.210244 -0.430696
slope    -0.168814 -0.030711  0.119717 -0.121475 -0.004038  ... -0.577537  1.000000 -0.080155 -0.104764  0.345877
ca        0.276326  0.118261 -0.181053  0.101389  0.070511  ...  0.222682 -0.080155  1.000000  0.151832 -0.391724
thal      0.068001  0.210041 -0.161736  0.062210  0.098803  ...  0.210244 -0.104764  0.151832  1.000000 -0.344029
target   -0.225439 -0.280937  0.433798 -0.144931 -0.085239  ... -0.430696  0.345877 -0.391724 -0.344029  1.000000

[14 rows x 14 columns]
```

```
Class labels are:
['1', '0']
Feature names are:
['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']
First 5 rows of features set are:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  ca  thal
0   63    1   3       145   233    1        0      150      0      2.3      0   0     1
1   37    1   2       130   250    0        1      187      0      3.5      0   0     2
2   41    0   1       130   204    0        0      172      0      1.4      2   0     2
3   56    1   1       120   236    0        1      178      0      0.8      2   0     2
4   57    0   0       120   354    0        1      163      1      0.6      2   0     2
First 5 rows of features set are:
   target
0       1
1       1
2       1
3       1
4       1
```

```
Predicted class labels for test data are:
[1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1
 1 1 0 1 1 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1
 0 0 0 1 0 0 0 0 1 1 0 0 1 1 0 1 1]
Accuracy: 0.8571428571428571
Precision: 0.8076923076923077
Recall: 0.9333333333333333
              precision    recall  f1-score   support

           1       0.92      0.78      0.85        46
           0       0.81      0.93      0.87        45

    accuracy                           0.86        91
   macro avg       0.87      0.86      0.86        91
weighted avg       0.87      0.86      0.86        91
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fc5a116b48>
```



Experiment – 5:

Aim: **Implement a program for Bias, Variance and cross-validation**

Dataset: winequality.csv- The data set is related to white variant of the Portuguese "Vinho Verde" wine. The data set is collected from https://archive.ics.uci.edu/ml/datasets/wine+quality.

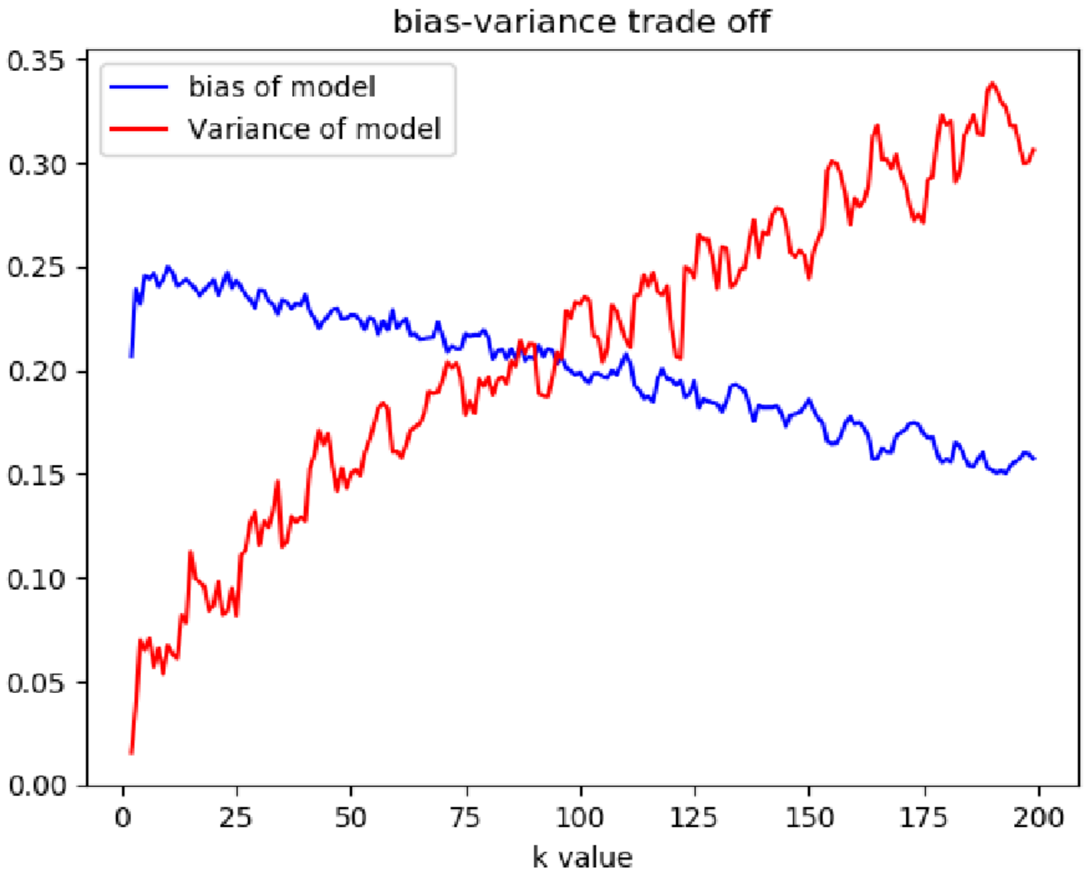
Program code:

```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
import matplotlib.pyplot as plt
from statistics import mean,stdev
data=pd.read_csv(r"E:\machine learning\datasets\winequality.csv")
dim=data.shape
print('Dimensions of the data set are',dim)
```

```
print('First 5 rows of the data set are:')
print(data.head())
col_names=data.columns
col_names=list(col_names)
print('Attrubte names are:')
print(col_names)
feature_names=col_names[:-1]
print('Feature names are:',feature_names)
x_set=data.drop('quality',axis=1)
y_set=data['quality']
model=linear_model.LinearRegression()
scores=cross_val_score(model, x_set, y_set, cv=10)
k_list=range(2,200)
bias=[]
variance=[]
for k in k_list:
   model=linear_model.LinearRegression()
   scores=cross_val_score(model, x_set, y_set, cv=k)
   bias.append(mean(scores))
   variance.append(stdev(scores))
plt.plot(k_list, bias, 'b', label='bias of model')
plt.plot(k_list, variance, 'r', label='Variance of model')
plt.xlabel('k value')
plt.title('bias-variance trade off')
plt.legend(loc='best')
plt.show()
#From, graph , best value is about 85
model=linear_model.LinearRegression()
scores=cross_val_score(model, x_set, y_set, cv=85)
bias=mean(scores)
variance=stdev(scores)
print('Bias of the model is',bias)
print('Variance of the model is',variance)
```

Output screen shots:

```
(base) C:\Users\harsini>python ex4.py
Dimensions of the data set are (4898, 12)
First 5 rows of the data set are:
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  ...  density    pH  sulphates  alcohol  qual
ity
0            7.0              0.27         0.36            20.7      0.045  ...   1.0010  3.00       0.45      8.8
  6
1            6.3              0.30         0.34             1.6      0.049  ...   0.9940  3.30       0.49      9.5
  6
2            8.1              0.28         0.40             6.9      0.050  ...   0.9951  3.26       0.44     10.1
  6
3            7.2              0.23         0.32             8.5      0.058  ...   0.9956  3.19       0.40      9.9
  6
4            7.2              0.23         0.32             8.5      0.058  ...   0.9956  3.19       0.40      9.9
  6

[5 rows x 12 columns]
Attrubte names are:
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur
 dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality']
Feature names are: ['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur diox
ide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol']
```




```
Bias of the model is 0.21036992445788824
Variance of the model is 0.20439342232832935
```

Experiment-7

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

Aim: Demonstration of Artificial neural network using back propagation algorithm

Program 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 longitudinallyy = y/100

#Sigmoid Functiondef sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Functiondef derivatives_sigmoid(x):

return x * (1 - x)
```

#Variable initialization

```
epoch=7000 #Setting training iterationslr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set hiddenlayer_neurons = 3 #number of hidden

layers neuronsoutput_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*yfor 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 wtscontributed to error

d_hiddenlayer = EH * hiddengrad
```

```
wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror andcurrentlayerop
```

bout += np.sum(d_output, axis=0,keepdims=True) *lrwh += X.T.dot(d_hiddenlayer) *lr

#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lrprint("Input: \n" + str(X))

print("Actual Output: \n" + str(y)) print("Predicted Output: \n" ,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]]

Experiment-8:

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

Aim: To implement k-Nearest Neighbor algorithm

Program Code:

import csv import random

import math import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[]):with open(filename, 'rb') as csvfile:

lines = csv.reader(csvfile)dataset = list(lines)

for x in range(len(dataset)-1):for y in range(4):

dataset[x][y] = float(dataset[x][y])if random.random() < split:

trainingSet.append(dataset[x])else:

testSet.append(dataset[x])

def euclideanDistance(instance1, instance2, length):distance = 0

```
for x in range(length):
```

```
distance += pow((instance1[x] - instance2[x]), 2)return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):distances = []

length =  len(testInstance)-1

for x in range(len(trainingSet)):

dist = euclideanDistance(testInstance, trainingSet[x], length)distances.append((trainingSet[x], dist))

distances.sort(key=operator.itemgetter(1))neighbors = []

for x in range(k):

neighbors.append(distances[x][0])return neighbors

def getResponse(neighbors):classVotes = {}

for x in range(len(neighbors)): response = neighbors[x][-1]if response in classVotes:

classVotes[response] += 1

else:

classVotes[response] = 1

sortedVotes = sorted(classVotes.iteritems(),reverse=True)

return sortedVotes[0][0]

def getAccuracy(testSet, predictions): correct = 0 for x in range(len(testSet)):

key=operator.itemgetter(1

),

if testSet[x][-1] == predictions[x]:correct += 1

return (correct/float(len(testSet))) * 100.0

def main():

# prepare data trainingSet=[] testSet=[]split = 0.67

loadDataset('knndat.data', split, trainingSet, testSet) print('Train set: ' + repr(len(trainingSet)))

print('Test set: ' + repr(len(testSet)))

# generate predictions predictions=[]k=3

for x in range(len(testSet)):

neighbors = getNeighbors(trainingSet, testSet[x],k) result = getResponse(neighbors)
```

```
predictions.append(result)
```

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1])) accuracy = getAccuracy(testSet,

predictions)

print('Accuracy: ' + repr(accuracy) +'%') main()

OUTPUT

Confusion matrix is as follows

[[11 0 0]

[0 9 1]

[0 1 8]]

Accuracy metrics0 1.00 1.00 1.00 11

1 0.90 0.90 0.90 10

2 0.89 0.89 0,89 9

Avg/Total 0.93 0.93 0.93 30

Experiment – 9:

Implement the non-parametric Locally Weighted Regression algorithm in orderto fit data points.

Select appropriate data set for your experiment and drawgraphs.

Aim: Demonstration of -parametric Locally Weighted Regression algorithm

Program Code

from numpy import *import operator

from os import listdirimport matplotlib

import matplotlib.pyplot as pltimport pandas as pd

import numpy as np1 import numpy.linalg as np

from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):m,n = np1.shape(xmat)

weights = np1.mat(np1.eye((m)))for j in range(m):

diff = point - X[j]

weights[j,j] = np1.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 = np1.shape(xmat)

ypred = np1.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('data10.csv')bill = np1.array(data.total_bill) tip = np1.array(data.tip)

#preparing and add 1 in billmbill = np1.mat(bill)

mtip = np1.mat(tip)

m= np1.shape(mbill)[1]

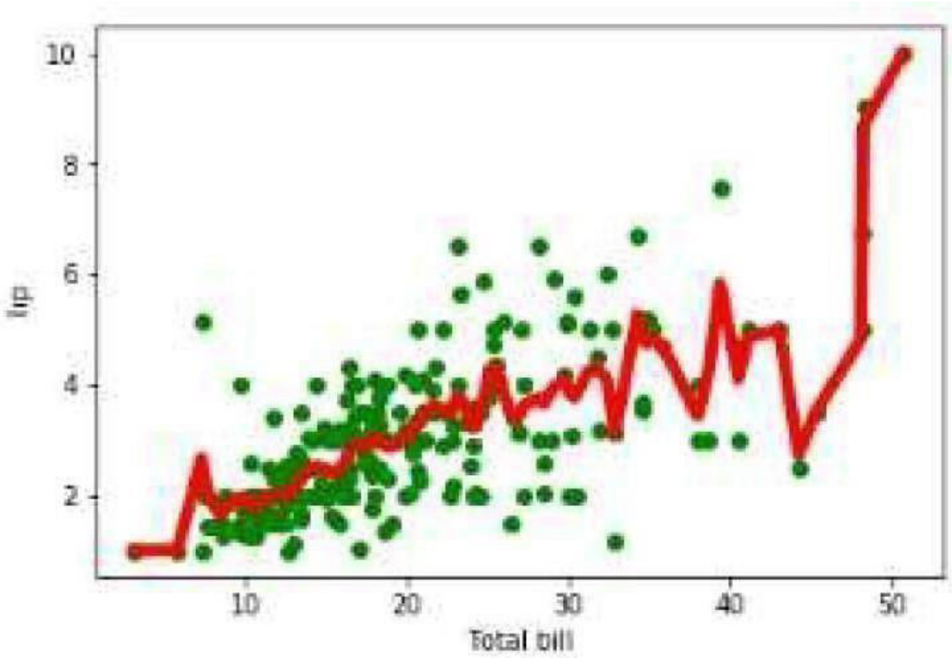
one = np1.mat(np1.ones(m)) X= np1.hstack((one.T,mbill.T))

#set k here

ypred = localWeightRegression(X,mtip,2)

SortIndex = X[:,1].argsort(0)xsort = X[SortIndex][:,0]

Output



Experiment-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
Aim: classification of set of documents using Naive Bayesian classification
Program code

```
import pandas as pd
msg=pd.read_csv('naivetext1.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.messagey=msg.labelnumprint(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 CountVectorizercount_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 metricsprint('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))
'''docs_new = ['I like this place', 'My boss is not my saviour']

X_new_counts = count_vect.transform(docs_new)predictednew = clf.predict(X_new_counts)
for doc, category in zip(docs_new, predictednew):
print('%s->%s' % (doc, msg.labelnum[category]))'''
I love this sandwich,pos This is an amazing place,pos
I feel very good about these beers,posThis is my best work,pos
What an awesome view,pos
I do not like this restaurant,negI am tired of this stuff,neg
I can't deal with this,neg He is my sworn enemy,negMy boss is horrible,neg
This is an awesome place,pos
I do not like the taste of this juice,negI love to dance,pos

I am sick and tired of this place,negWhat a great holiday,pos
That is a bad locality to stay,neg
We will have good fun tomorrow,posI went to my enemy's house today,neg

OUTPUT

```
['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'boss', 'can', 'deal',
'do', 'enemy', 'feel', 'fun', 'good', 'have', 'horrible', 'house', 'is', 'like', 'love', 'my',
'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'stuff', 'these', 'this', 'tired', 'to',
'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']about am amazing an and
awesome beers best boss can ... today \
0   1 0   0 0 0 0 1   0 0 0 ... 0
1   0 0   0 0 0 0 0   1 0 0 ... 0
2   0 0   1 1 0 0     0 0 0 0 ... 0
3   0 0   0 0 0 0 0   0 0 0 ... 1
4   0 0   0 0 0 0 0   0 0 0 ... 0
5   0 1   0 0 1       0 0 0 0 0 ... 0
6   0 0   0 0 0 0 0   0 0 1 ... 0
7   0 0   0 0 0 0 0   0 0 0 ... 0
8   0 1   0 0 0 0 0   0 0 0 ... 0
9   0 0   0 1 0 1 0   0 0 0 ... 0
10 0 0   0 0 0 0 0 0 0 0 ... 0
11 0 0   0 0 0   0 0 0 1 0 ... 0
12 0 0   0 1 0 1 0 0 0 0 ... 0
```

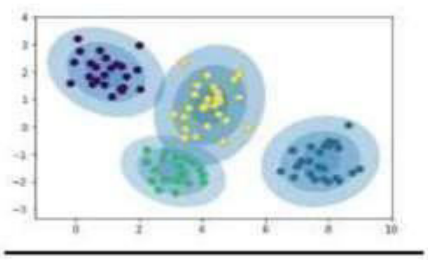
Experiment-11:
Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering usingkMeans algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
Aim: Implementation of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

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator
import make_blobsX, y_true = make_blobs(n_samples=100, centers =
4,Cluster_std=0.60,random_state=0)
X = X[:, ::-1]
#flip axes for better plotting
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture (n_components = 4).fit(X)lables = gmm.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap="viridis");probs = gmm.predict_proba(X)
print(probs[:5].round(3))
size = 50 * probs.max(1) ** 2
# square emphasizes differences
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap="viridis", s=size);
from matplotlib.patches import Ellipse
```

```
def draw_ellipse(position, covariance, ax=None, **kwargs);
“””Draw an ellipse with a given position and covariance”””Ax
= ax or plt.gca()

# Convert covariance to principal axes
if covariance.shape ==(2,2):
U, s, Vt = np.linalg.svd(covariance)
Angle = np.degrees(np.arctan2(U[1, 0], U[0,0]))Width, height = 2 * np.sqrt(s)
else:
angle = 0
width, height = 2 * np.sqrt(covariance)
#Draw the Ellipse
for nsig in range(1,4):
ax.add_patch(Ellipse(position, nsig * width, nsig *height,angle, **kwargs))
def plot_gmm(gmm, X, label=True, ax=None):ax = ax or plt.gca()
labels = gmm.fit(X).predict(X)if label:
ax.scatter(X[:, 0], x[:, 1], c=labels, s=40, cmap=‟viridis‟, zorder=2)else:
ax.scatter(X[:, 0], x[:, 1], s=40, zorder=2)ax.axis(„equal‟)
w_factor = 0.2 / gmm.weights_.max()
for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):draw_ellipse(pos, covar,
alpha=w * w_factor)
gmm = GaussianMixture(n_components=4, random_state=42)plot_gmm(gmm, X)
gmm = GaussianMixture(n_components=4, covariance_type=‟full‟,random_state=42)
plot_gmm(gmm, X)
```

Output

[[1 ,0, 0, 0]
[0 ,0, 1, 0]
[1 ,0, 0, 0]
[1 ,0, 0, 0]
[1 ,0, 0, 0]]



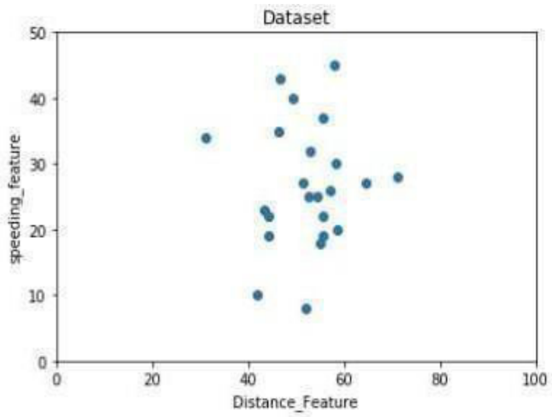
K MEANS :

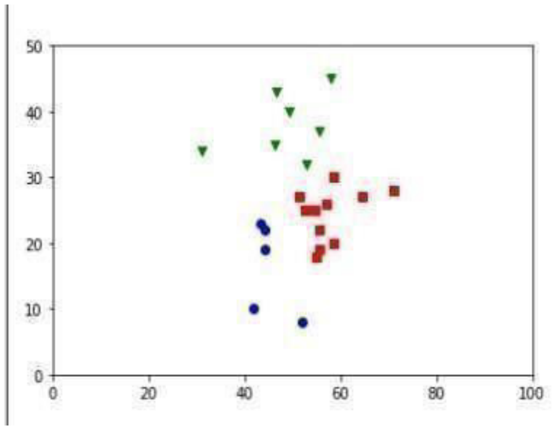
```
from sklearn.cluster import KMeans
#from sklearn import metricsimport numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data=pd.read_csv("kmeansdata.csv")
df1=pd.DataFrame(data)
print(df1)
f1 = df1['Distance_Feature'].valuesf2 = df1['Speeding_Feature'].values
X=np.matrix(list(zip(f1,f2)))plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50]) plt.title('Dataset') plt.ylabel('speeding_feature')plt.xlabel('Distance_Feature')
plt.scatter(f1,f2)
plt.show()
```

 # create new plot and data

```
plt.plot()
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
# KMeans algorithm#K = 3
kmeans_model = KMeans(n_clusters=3).fit(X)
plt.plot()
for i, l in enumerate(kmeans_model.labels_):
plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')plt.xlim([0, 100])
plt.ylim([0, 50])plt.show()
```

```
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
```


Dataset
speeding_feature
Distance_Feature
0
10
20
30
40
50
0
20
40
60
80
100


0
10
20
30
40
50
0
20
40
60
80
100

Experiment -12

Aim: **Exploratory data analysis for classification using pandas and Matplotlib**
Dataset: tae.csv- The data consist of evaluations of teaching performance over three regular semesters and two summer semesters of 151 teaching assistant (TA) assignments at the Statistics Department of the University of Wisconsin-Madison. The scores were divided into 3 roughly equal-sized categories ("low", "medium", and "high") to form the class variable. The data set is collected from https://archive.ics.uci.edu/ml/datasets/Teaching+Assistant+Evaluation
Program code:

```
import pandas as pd
import matplotlib.pyplot as plt
print('pandas version is', pd._version_)
data = pd.read_csv(r"E:\sudhakar\tae.csv",header=None)
col_names=['native_speaker','instructor','course','semester','class_size','score']
data.columns=col_names
print('Data type of target variable is:',data['score'].dtype)
print('Converting target variable data type to categorical')
data['score']=data['score'].astype('category')
print('Afrer conversion, data type of target variable is:',data['score'].dtype)
print('Dimesnions of the data set:')
print(data.shape)
print('The first 5 rows of the data set are:')
print(data.head())
print('The last 5 rows of the data set are:')
print(data.tail())
print('Randomly selected 5 rows of the data set are:')
print(data.sample(5))
print('The columns of the data set are:')
print(data.columns.tolist())
print('Names and data types of attributes are:')
print(data.dtypes)
print('Converting native_speaker data type to categorical')
data['native_speaker']=data['native_speaker'].astype('category')
print('After conversion,Names and data types of attributes are:')
print(data.dtypes)
print('Information of the data set attributes:')
print(data.info())
print('Statistics of the numerical attributes of the data set are:')
print(data.describe())
print('Statistics of the all attributes of the data set are:')
print(data.describe(include='all'))
print('Corelation matrix of the numerical attributes of the data set is:')
corr=data.corr()
print(corr)
print('Distribution of the target variable is:')
print(data['score'].value_counts())
print('Target class distrubtion w.r.t \'native_speaker\' attribute')
print(pd.crosstab(data.native_speaker,data.score))
```

```
print('Target class distrubtion w.r.t \'native_speaker\' attribute')
```

```
print(pd.crosstab(data.native_speaker,data.score,normalize='index'))
print('Target class distrubtion w.r.t \'native_speaker\' attribute using groupby')
data.groupby('native_speaker').score.value_counts()
print('Checking for null values:')
print(data.isnull().sum())
data.dropna(subset=['instructor'],axis=0,inplace=True)
```

```
print('After removal rows with null values in column \'instructor\'')
print(data.isnull().sum())
print('Unique values in the column named \'score\'')
print(data['score'].unique())
data.plot(kind='scatter',x='semester',y='class_size',color='red')
print('Number of distinct courses semester wise')
data.groupby('semester')['course'].nunique().plot(kind='bar')
print('Frequency of values in column \'semester\'')
data[['semester']].plot(kind='hist')
data.plot(kind='bar',x='semester',y='course',color='red')
ax = plt.gca()#gca means get current axes
data.plot(kind='line',x='semester',y='class_size',ax=ax)
```

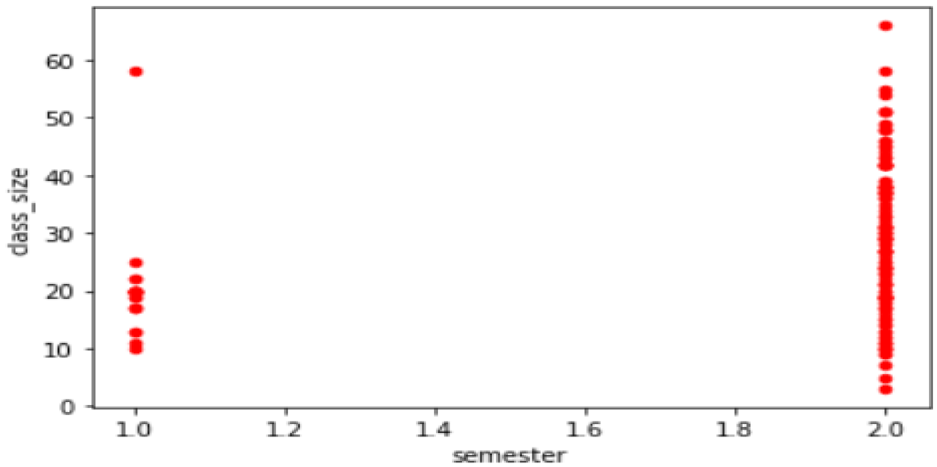
Output screen shots:

```
(base) C:\Users\harsini>python mtech_ml_ex3.py
pandas version is 1.0.1
Data type of target variable is: int64
Converting target variable data type to categorical
Afrer conversion, data type of target variable is: category
Dimesnions of the data set:
(151, 6)
The first 5 rows of the data set are:
   native_speaker  instructor  course  semester  class_size score
0               1        23.0     3.0         1        19.0     3
1               2        15.0     3.0         1        17.0     3
2               1        23.0     3.0         2        49.0     3
3               1         5.0     2.0         2        33.0     3
4               2         7.0    11.0         2         NaN     3
The last 5 rows of the data set are:
     native_speaker  instructor  course  semester  class_size score
146               2         3.0     2.0         2        26.0     1
147               2        10.0     3.0         2        12.0     1
148               1        18.0     7.0         2        48.0     1
149               2        22.0     1.0         2        51.0     1
150               2         2.0    10.0         2        27.0     1
Randomly selected 5 rows of the data set are:
     native_speaker  instructor  course  semester  class_size score
0                 1        23.0     3.0         1        19.0     3
2                 1        23.0     3.0         2        49.0     3
33                1        13.0     3.0         1        13.0     1
146               2         3.0     2.0         2        26.0     1
137               2        22.0     1.0         2        42.0     2
The columns of the data set are:
['native_speaker', 'instructor', 'course', 'semester', 'class_size', 'score']
Names and data types of attributes are:
native_speaker       int64
instructor         float64
course             float64
semester             int64
class_size         float64
score             category
dtype: object
```

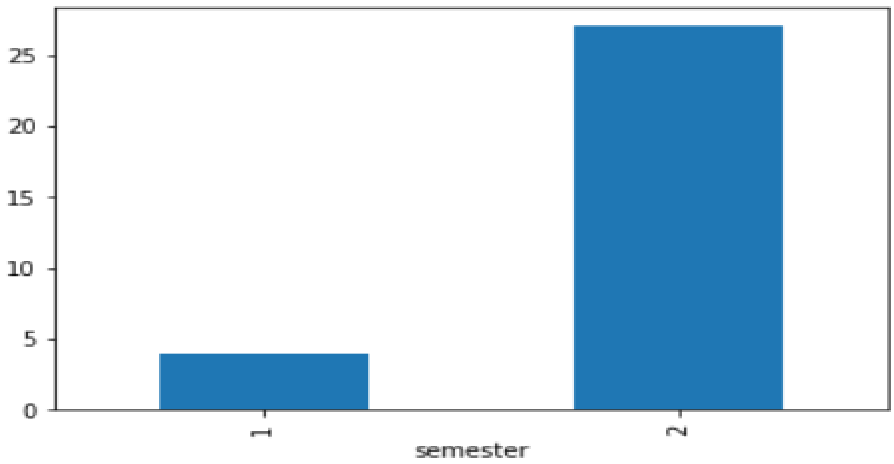
```
Converting native_speaker data type to categorical
After conversion,Names and data types of attributes are:
native_speaker    category
instructor         float64
course             float64
semester             int64
class_size         float64
score             category
dtype: object
Information of the data set attributes:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 151 entries, 0 to 150
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   native_speaker  151 non-null    category
 1   instructor      150 non-null    float64
 2   course          148 non-null    float64
 3   semester        151 non-null    int64
 4   class_size      149 non-null    float64
 5   score           151 non-null    category
dtypes: category(2), float64(3), int64(1)
memory usage: 5.3 KB
Statistics of the numerical attributes of the data set are:
       instructor      course    semester  class_size
count  150.000000  148.000000  151.000000  149.000000
mean    13.646667    8.155405    1.847682   27.610738
std      6.848442    7.077523    0.360525   12.752165
min      1.000000    1.000000    1.000000    3.000000
25%      8.000000    3.000000    2.000000   19.000000
50%     13.000000    3.500000    2.000000   26.000000
75%     20.000000   15.000000    2.000000   37.000000
max     25.000000   26.000000    2.000000   66.000000
Statistics of the all attributes of the data set are:
        native_speaker  instructor      course    semester  class_size  score
count            151.0  150.000000  148.000000  151.000000  149.000000  151.0
unique             2.0         NaN         NaN         NaN         NaN    3.0
top                2.0         NaN         NaN         NaN         NaN    3.0
freq             122.0         NaN         NaN         NaN         NaN   52.0
mean               NaN   13.646667    8.155405    1.847682   27.610738    NaN
std                NaN    6.848442    7.077523    0.360525   12.752165    NaN
min                NaN    1.000000    1.000000    1.000000    3.000000    NaN
25%                NaN    8.000000    3.000000    2.000000   19.000000    NaN
50%                NaN   13.000000    3.500000    2.000000   26.000000    NaN
75%                NaN   20.000000   15.000000    2.000000   37.000000    NaN
max                NaN   25.000000   26.000000    2.000000   66.000000    NaN
Corelation matrix of the numerical attributes of the data set is:
            instructor    course  semester  class_size
instructor    1.000000 -0.231942 -0.173308   -0.016912
course       -0.231942  1.000000  0.219240   -0.039441
semester     -0.173308  0.219240  1.000000    0.266080
class_size   -0.016912 -0.039441  0.266080    1.000000
Distribution of the target variable is:
3    52
2    50
1    49
Name: score, dtype: int64
```

```
Target class distrubtion w.r.t 'native_speaker' attribute
score              1   2   3
native_speaker
1                  5   6  18
2                 44  44  34
Target class distrubtion w.r.t 'native_speaker' attribute
score                    1         2         3
native_speaker
1                 0.172414  0.206897  0.620690
2                 0.360656  0.360656  0.278689
Checking for null values:
native_speaker     0
instructor         1
course             3
semester           0
class_size         2
score              0
dtype: int64
After removal rows with null values in column 'instructor'
native_speaker     0
instructor         0
course             3
semester           0
class_size         2
score              0
dtype: int64
Unique values in the column named 'score'
[3, 2, 1]
Categories (3, int64): [3, 2, 1]
```

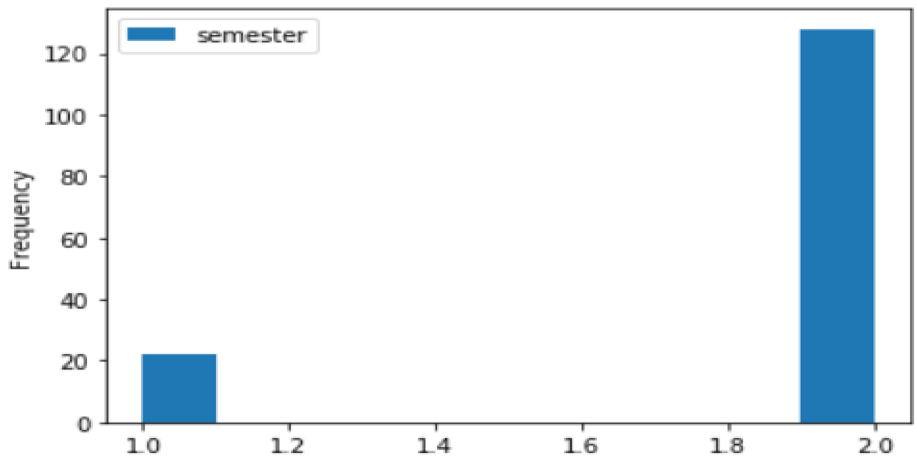
```
<matplotlib.axes._subplots.AxesSubplot at 0x29e16780e48>
```



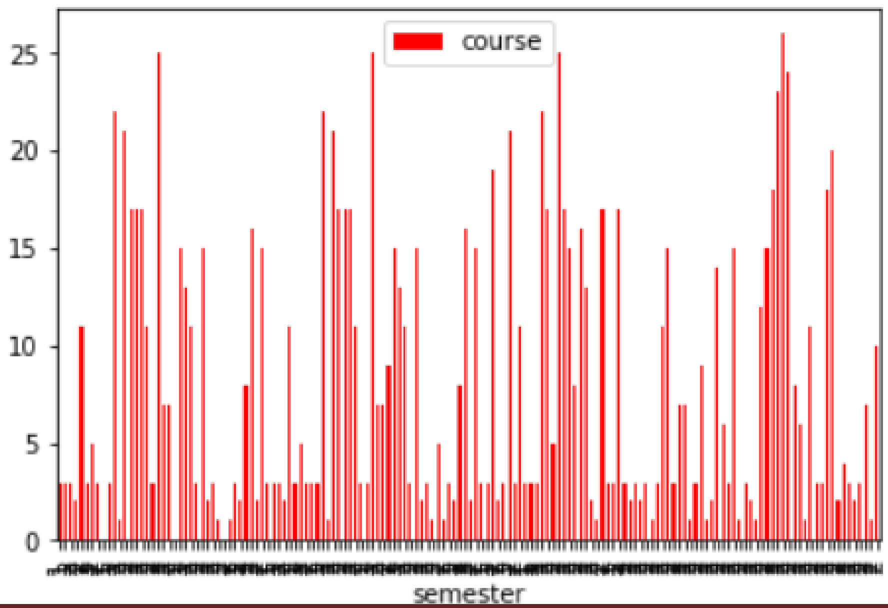
```
Number of distinct courses semester wise
<matplotlib.axes._subplots.AxesSubplot at 0x29e17ee8a08>
```



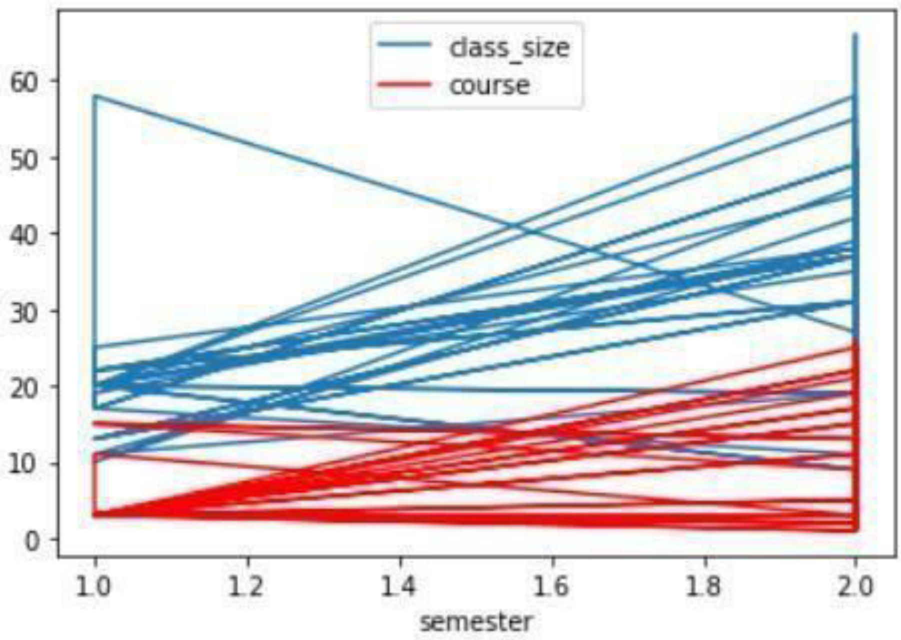

```
Frequency of values in column 'semester'
<matplotlib.axes._subplots.AxesSubplot at 0x29e18100f08>
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x29e16a79c88>
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x29e17e46408>
```



Experiment -13:

Write a 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.

```
import bayespy as bp
import numpy as np
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

# Define Parameter Enum values
#Age
ageEnum = {'SuperSeniorCitizen':0, 'SeniorCitizen':1, 'MiddleAged':2, 'Youth':3, 'Teen':4}
# Gender
genderEnum = {'Male':0, 'Female':1}
# FamilyHistory
familyHistoryEnum = {'Yes':0, 'No':1}
# Diet(Calorie Intake)
dietEnum = {'High':0, 'Medium':1, 'Low':2}
# LifeStyle
lifeStyleEnum = {'Athlete':0, 'Active':1, 'Moderate':2, 'Sedetary':3}
# Cholesterol
cholesterolEnum = {'High':0, 'BorderLine':1, 'Normal':2}
# HeartDisease
heartDiseaseEnum = {'Yes':0, 'No':1}
#heart_disease_data.csv
```

```
with open('heart_disease_data.csv') as csvfile:
    lines = csv.reader(csvfile)
    dataset = list(lines)
    data = []
    for x in dataset:

data.append([ageEnum[x[0]],genderEnum[x[1]],familyHistoryEnum[x[2]],dietEnum[x[3]],lifeStyleEn
um[x[4]],cholesterolEnum[x[5]],heartDiseaseEnum[x[6]]])
# Training data for machine learning todo: should import from csv
data = np.array(data)
N = len(data)

# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:,0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:,1])

p_familyhistory =  bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:,2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:,3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:,4])

p_cholesterol =  bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:,5])
```

```
C:\Anaconda3\lib\site-packages\bayespy\inference\vmp\nodes\categorical.py:107: FutureWarning:
Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead
of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will
result either in an error or a different result.
  u0[[np.arange(np.size(x)), np.ravel(x)]] = 1
```

```
# Prepare nodes and establish edges
# np.ones(2) ->  HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3)  ->  corresponds to options present for domain values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture([age, gender, familyhistory, diet, lifestyle, cholesterol],
bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:,6])
```

```
p_heartdisease.update()

# Sample Test with hardcoded values
#print("Sample Probability")
#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, FamilyHistory=Yes, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")
#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Female'], familyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedetary'], cholesterolEnum['High']], bp.nodes.Categorical, p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']])

# Interactive Test
m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categorical, p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]
    print("Probability(HeartDisease) = " + str(res))
    #print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1  "))
```

OUTPUT

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}1
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}2
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}1
C:\Anaconda3\lib\site-packages\bayespy\inference\vmp\nodes\categorical.py:43: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  u0[[np.arange(np.size(x)), np.ravel(x)]] = 1
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1  1
```

Experiment -14:

Write a program to implement Support Vector Machines
Aim:
To implement Support Vector Machines
Dataset: haberman.csv- The dataset contains cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer. The goal is to predict the Survival status (class attribute) of the patient(1 = the patient survived 5 years or longer,2 = the patient died within 5 years). The data set is

collected from https://archive.ics.uci.edu/ml/datasets/Haberman's+Survival.

Program code:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
data = pd.read_csv(r"E:\sudhakar\haberman.csv", header=None)
#age=age of the patient
#year=Patient's year of operation (year - 1900)
#pos_axil_nodes=Number of positive axillary nodes detected
#survival_status:1 -the patient survived 5 years or longer
#             :2 -the patient died within 5 year
col_names=['age','year','pos_axil_nodes','survival_status']
data.columns=col_names
#we removed the attribute year of operation
data=data.drop(['year'], axis=1)
print('The first 5 rows of the data set are:')
print(data.head())
dim=data.shape
print('Dimensions of the data set are',dim)
print('Statistics of the data are:')
print(data.describe())
print('Correlation matrix of the data set is:')
print(data.corr())

class_lbls=data['survival_status'].unique()
class_labels=[]
for x in class_lbls:
    class_labels.append(str(x))
print('Class labels are:')
print(class_labels)
sns.countplot(data['survival_status'])
col_names=data.columns
feature_names=col_names[:-1]
feature_names=list(feature_names)
print('Feature names are:')
print(feature_names)
x_set = data.drop(['survival_status'], axis=1)
print('First 5 rows of features set are:')
print(x_set.head())
y_set=data['survival_status']
print('First 5 rows of target variable are:')
print(y_set.head())
```

```
print('Distribution of Target variable is:')
print(y_set.value_counts())
scaler=StandardScaler()
x_train,x_test, y_train, y_test = train_test_split(x_set,y_set, test_size = 0.3)
scaler.fit(x_train)
x_train=scaler.transform(x_train)
model =SVC()
print("Traning the model with train data set")model.fit(x_train, y_train)
```

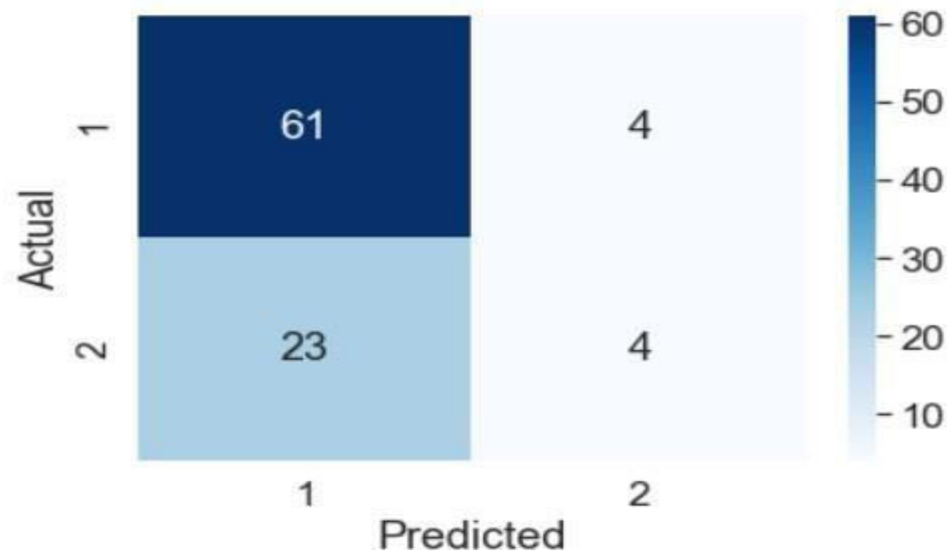
```
x_test=scaler.transform(x_test)
y_pred=model.predict(x_test)
print('Predicted class labels for test data are:')
print(y_pred)
print("Accuracy:",accuracy_score(y_test, y_pred))
print("Precision:",precision_score(y_test, y_pred))
print("Recall:",recall_score(y_test, y_pred))
print(classification_report(y_test,y_pred,target_names=class_labels))
cm=confusion_matrix(y_test,y_pred)
df_cm = pd.DataFrame(cm, columns=class_labels, index = class_labels)
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
sns.set(font_scale=1.5)
sns.heatmap(df_cm, annot=True,cmap="Blues",fmt='d')
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=30, cmap=plt.cm.Paired)
plt.xlabel('age')
plt.ylabel('pos_axil_nodes')
plt.title('Data points in traning data set')
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=30, cmap=plt.cm.Paired)
plt.xlabel('age')
plt.ylabel('pos_axil_nodes')
plt.title('support vectors and decision boundary')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z          =          model.decision_function(xy).reshape(XX.shape)
ax.contour(XX, YY, Z, colors='red', levels=[-1, 0, 1], alpha=0.5,
        linestyles=['--', '-', '--'])
# plot support vectors
ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=30,
        facecolors='green')
plt.show()
```

Output screen shots:

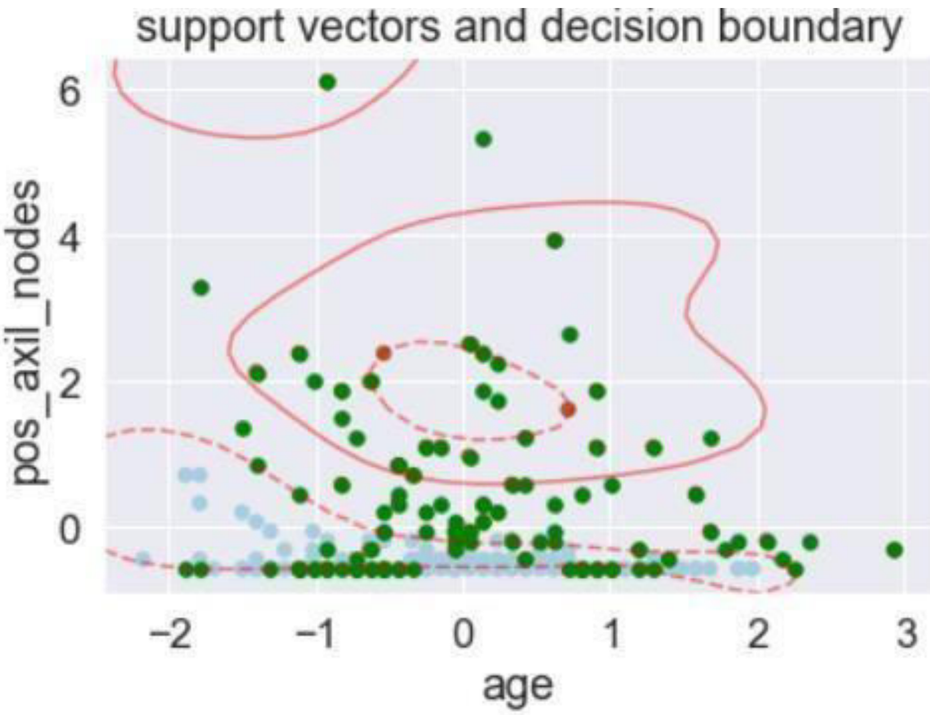
```
(base) C:\Users\harsini>python svm.py
The first 5 rows of the data set are:
   age  pos_axil_nodes  survival_status
0   30               1                1
1   30               3                1
2   30               0                1
3   31               2                1
4   31               4                1
Dimensions of the data set are (306, 3)
Statistics of the data are:
              age  pos_axil_nodes  survival_status
count  306.000000      306.000000       306.000000
mean    52.457516        4.026144         1.264706
std     10.803452        7.189654         0.441899
min     30.000000        0.000000         1.000000
25%     44.000000        0.000000         1.000000
50%     52.000000        1.000000         1.000000
75%     60.750000        4.000000         2.000000
max     83.000000       52.000000         2.000000
Correlation matrix of the data set is:
                      age  pos_axil_nodes  survival_status
age              1.000000       -0.063176         0.067950
pos_axil_nodes  -0.063176        1.000000         0.286768
survival_status  0.067950        0.286768         1.000000
Class labels are:
['1', '2']
Feature names are:
['age', 'pos_axil_nodes']
First 5 rows of features set are:
   age  pos_axil_nodes
0   30               1
1   30               3
2   30               0
3   31               2
4   31               4
First 5 rows of target variable are:
0    1
1    1
2    1
3    1
4    1
Name: survival_status, dtype: int64
Distribution of Target variable is:
1    225
2     81
Traning the model with train data set
Predicted class labels for test data are:
[1 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1
 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1]
Accuracy: 0.7282608695652174
Precision: 0.7948717948717948
Recall: 0.8732394366197183
```

```
              precision    recall  f1-score   support

           1       0.79      0.87      0.83        71
           2       0.36      0.24      0.29        21

    accuracy                           0.73        92
   macro avg       0.58      0.56      0.56        92
weighted avg       0.69      0.73      0.71        92
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d67a7ef608>
```




```
Text(0.5, 1.0, 'Data points in traning data set')
```



Experiment -14:

Write a program to implement principle component analysis

import numpy as nmp

import matplotlib.pyplot as mpltl

import pandas as pnd

DS = pnd.read_csv('Wine.csv')

 # Now, we will distribute the dataset into two components "X" and "Y"

X = DS.iloc[: , 0:13].values

Y = DS.iloc[: , 13].values

from sklearn.model_selection import train_test_split as tts

X_train, X_test, Y_train, Y_test = tts(X, Y, test_size = 0.2, random_state = 0)

```
from sklearn.preprocessing import StandardScaler as SS
SC = SS()

X_train = SC.fit_transform(X_train)
X_test = SC.transform(X_test)
from sklearn.decomposition import PCA

PCa = PCA (n_components = 1)

X_train = PCa.fit_transform(X_train)
X_test = PCa.transform(X_test)

explained_variance = PCa.explained_variance_ratio_
from sklearn.linear_model import LogisticRegression as LR

classifier_1 = LR (random_state = 0)
classifier_1.fit(X_train, Y_train)
```

Output:

```
LogisticRegression(random_state=0)
```