Program – 1

Implement simple linear regression using a python program and estimate statistical quantities from training data.

Python Code:

```
import matplotlib.pyplot as plt
import numpy as np
from math import sqrt

# Calculate root mean squared error
def rmse_metric(actual, predicted):
    sum_error = 0.0
    for i in range(len(actual)):
        prediction_error = predicted[i] - actual[i]
        sum_error += (prediction_error**2)
    mean_error = sum_error / float(len(actual))
    return sqrt(mean_error)

# Evaluate regression algorithm on training dataset'
def evaluate_algorithm(dataset, algorithm):
    test_set=[]
    for row in dataset:
        row_copy = list(row)
        row_copy[-1] = None
        test_set.append(row_copy)
    predicted = algorithm(dataset, test_set)
    print(predicted)
    actual = [row[-1] for row in dataset]
    rmse = rmse_metric(actual, predicted)
    return rmse

# Calculate the mean value of a list of numbers
def mean(values):
    return sum(values) / float(len(values))
```

```
# Calculate covariance between x and y
def covariance(x, mean_x, y, mean_y):
    covar = 0.0
    for i in range(len(x)):
        covar += (x[i] - mean_x) * (y[i] - mean_y)
    return covar/float(len(x))

# Calculate the variance of a list of numbers
def variance(values, mean):
    return (sum([(x-mean)**2 for x in values]))/float(len(values))

# Calculate coefficients
def coefficients(dataset):
    x = [row[0] for row in dataset]
    y = [row[1] for row in dataset]
    x_mean, y_mean = mean (x), mean(y)
    b1 = covariance(x, x_mean, y, y_mean) / variance(x, x_mean)
    b0 = y_mean - b1 * x_mean
    return [b0, b1]

# Simple linear regression algorithm
def simple_linear_regression(train, test):
    predictions = []
    b0, b1 = coefficients(train)
    for row in test:
        yhat = b0 + b1 * row[0]
        predictions.append(yhat)
    return predictions

# Test simple linear regression
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
mean_x, mean_y = mean(x), mean (y)
var_x, var_y = variance(x, mean_x), variance(y, mean_y)
print('x stats: Mean = %.3f Variance = %.3f' % (mean_x, var_x))
print('y stats: Mean = %.3f Variance = %.3f' % (mean_y, var_y))
covar = covariance(x, mean_x, y, mean_y)
print('Covariance : %.3f' % (covar))
rmse = evaluate_algorithm(dataset, simple_linear_regression)
```

```
print('RMSE: %.3f' % (rmse))

# calculate coefficients
b0, b1 = coefficients(dataset)
print('Coefficients: B0 = %.3f, B1 = %.3f' % (b0, b1))
```

Output:

```
x stats: Mean = 3.000 Variance = 2.000

y stats: Mean = 2.800 Variance = 1.760

Covariance : 1.600

[1.1999999999999995, 1.9999999999999996, 3.5999999999999996, 2.8, 4.3999999999999995]

RMSE: 0.693

Coefficients: B0 = 0.400, B1 = 0.800
```

Program – 2

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.

<u>The CSV file:</u>

Time	Weather	Temperature	Company	Humidity	Wind	Goes
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

<u>Python Code:</u>

```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("C:/Users/ISE14/Documents/CSV_AIML/P2Data.csv")
print(data)

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("The attributes are: \n",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
```

```
            if val[x] != specific_hypothesis[x]:
                specific_hypothesis[x] = "?"
            else:
                pass
    return specific_hypothesis

#obtaining the final hypothesis
print("The final hypothesis is:",train(d,target))
```

Output:

```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes
The attributes are:
 [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]
The target is:  ['Yes' 'No' 'Yes' 'Yes']
The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

Program – 3

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.

The CSV File:

Sky	AirTemp	Humidity	Wind	Water	Forecast	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	Cold	Change	Yes

Python Code:

```
#Importing Important Libraries
import numpy as np
import pandas as pd

data=pd.read_csv('C:/Users/ISE-LAB7/Documents/P3Data.csv')
print(data)
concepts=np.array(data.iloc[:,0:-1])
print(concepts)
target= np.array(data.iloc[:,-1])
print(target)

#Candidate Elimination algorithm
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific hypothesis: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print("\nGeneric hypothesis: ",general_h)
    for i, h in enumerate(concepts):
        print("\nInstance", i+1, "is ", h)
        if target[i] == "Yes":
            print("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("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("Specific hypothesis after ", i+1, "Instance is ", specific_h)
        print("Generic hypothesis after ", i+1, "Instance is ", general_h)
        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")
```

Output:

```
     Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cold   Change        Yes
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cold' 'Change']]
['Yes' 'Yes' 'No' 'Yes']

Initialization of specific_h and genearal_h

Specific hypothesis:  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Generic hypothesis:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

Instance 1 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Instance is Positive
Specific hypothesis after 1 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Generic hypothesis after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Instance is Positive
Specific hypothesis after 2 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Generic hypothesis after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
Instance is Negative
Specific hypothesis after 3 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Generic hypothesis after 3 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Instance 4 is ['Sunny' 'Warm' 'High' 'Strong' 'Cold' 'Change']
Instance is Positive
Specific hypothesis after 4 Instance is ['Sunny' 'Warm' '?' 'Strong' '?' '?']
Generic hypothesis after 4 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

Program – 4

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.

The CSV file:

Outlook	Temperature	Humidity	Windy	PT
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rainy	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

Python Code:

```
# Importing important libraries
import pandas as pd
from pandas import DataFrame
# Reading Dataset
df_tennis = pd.read_csv('C:/Users/Lochan/OneDrive/Documents/P4Data.csv')
print(df_tennis)

# Function to calculate final Entropy
def entropy(probs):
    import math
    return sum([-prob*math.log(prob,2) for prob in probs])

# Function to calculate Probabilities of positive and negative examples
def entropy_of_list(a_list):
    from collections import Counter
    cnt = Counter(x for x in a_list)
```

```
    #Count the positive and negative ex
    num_instances = len(a_list)
    #Calculate the probabilities that we required for our entropy formula
    probs = [x / num_instances for x in cnt.values()]
    #Calling entropy function for final entropy
    return entropy(probs)

total_entropy = entropy_of_list(df_tennis['PT'])
print("\nTotal Entropy of PlayTennis Data Set:",total_entropy)

# Defining Information Gain Function
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
    print("\nInformation Gain Calculation of ",split_attribute_name)
    print("target_attribute_name:",target_attribute_name)

    # Grouping features of Current Attribute
    df_split = df.groupby(split_attribute_name)
    for name,group in df_split:
        print("Name: ",name)
        print("Group: ",group)
    nobs = len(df.index) * 1.0
    print("NOBS",nobs)

    # Calculating Entropy of the Attribute and probability part of formula
    df_agg_ent = df_split.agg(
        Entropy=(target_attribute_name, entropy_of_list),
        Prob1=(target_attribute_name, lambda x: len(x) / nobs)
    )
    print("df_agg_ent",df_agg_ent)

    # Calculate Information Gain
    avg_info = sum(df_agg_ent['Entropy'] * df_agg_ent['Prob1'])
    old_entropy = entropy_of_list(df[target_attribute_name])
    return old_entropy - avg_info

print('Info-gain for Outlook is : '+str(information_gain(df_tennis, 'Outlook',
'PT')),"\n")

#Defining ID3  Algorithm Function
def id3(df, target_attribute_name, attribute_names, default_class=None):
```

```
    # Counting Total number of yes and no classes (Positive and negative Ex)
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])
    if len(cnt) == 1:
        return next(iter(cnt))
    # Return None for Empty Data Set
    elif df.empty or (not attribute_names):
        return default_class
    else:
        default_class = max(cnt.keys())
        print("attribute_names:",attribute_names)
        gainz = [information_gain(df, attr, target_attribute_name) for attr in
attribute_names]
        # Separating the maximum information gain attribute after calculating the
information gain
        index_of_max = gainz.index(max(gainz)) #Index of Best Attribute
        best_attr = attribute_names[index_of_max] #choosing best attribute
        # The tree is initially an empty dictionary
        tree = {best_attr:{}} # Initiate the tree with best attribute as a node
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset, target_attribute_name,
remaining_attribute_names, default_class)
            tree[best_attr][attr_val] = subtree
        return tree

# Get Predictor Names (all but 'class')
attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PT')
# Remove the class attribute
print("Predicting Attributes:", attribute_names)
# Run Algorithm (Calling ID3 function)
from pprint import pprint
tree = id3(df_tennis,'PT',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```

```
# Defining a function to calculate accuracy
def classify(instance, tree, default=None):
    attribute = next(iter(tree))
    print("Key:",tree.keys())
    print("Attribute:",attribute)
    print("Instance of Attribute :",instance[attribute],attribute)
    if instance[attribute] in tree[attribute].keys():
        result = tree[attribute][instance[attribute]]
        print("Instance Attribute:",instance[attribute],"TreeKeys
:",tree[attribute].keys())
        if isinstance(result, dict):
            return classify(instance, result)
        else:
            return result
    else:
        return default

df_tennis['predicted'] = df_tennis.apply(classify, axis=1, args=(tree,'No') )
print(df_tennis['predicted'])
print('\n Accuracy is:\n' + str( sum(df_tennis['PT']==df_tennis['predicted'] ) /
(1.0*len(df_tennis.index)) ))
df_tennis[['PT', 'predicted']]

training_data = df_tennis.iloc[1:-4]
test_data  = df_tennis.iloc[-4:]
train_tree = id3(training_data, 'PT', attribute_names)
test_data['predicted2'] = test_data.apply(
classify, axis=1, args=(train_tree,'Yes') )
print ('\n\n Accuracy is : ' + str( sum(test_data['PT']==test_data['predicted2'] ) /
(1.0*len(test_data.index)) ))
```

Output:

```
    Outlook Temperature Humidity   Windy   PT
0     Sunny        Hot     High    Weak   No
1     Sunny        Hot     High  Strong   No
2  Overcast        Hot     High    Weak  Yes
3     Rainy       Mild     High    Weak  Yes
4     Rainy       Cool   Normal    Weak  Yes
5     Rainy       Cool   Normal  Strong   No
```

```
6   Overcast       Cool   Normal  Strong  Yes
7     Sunny       Mild     High    Weak   No
8     Sunny       Cool   Normal    Weak  Yes
9     Rainy       Mild   Normal    Weak  Yes
10     Sunny       Mild   Normal  Strong  Yes
11  Overcast       Mild     High  Strong  Yes
12  Overcast        Hot   Normal    Weak  Yes
13     Rainy       Mild     High  Strong   No

Total Entropy of PlayTennis Data Set: 0.9402859586706309

Information Gain Calculation of  Outlook
target_attribute_name: PT
Name:  Overcast
Group:      Outlook Temperature Humidity   Windy   PT
2   Overcast        Hot     High    Weak  Yes
6   Overcast       Cool   Normal  Strong  Yes
11  Overcast       Mild     High  Strong  Yes
12  Overcast        Hot   Normal    Weak  Yes
Name:  Rainy
Group:    Outlook Temperature Humidity   Windy   PT
3    Rainy       Mild     High    Weak  Yes
4    Rainy       Cool   Normal    Weak  Yes
5    Rainy       Cool   Normal  Strong   No
9    Rainy       Mild   Normal    Weak  Yes
13   Rainy       Mild     High  Strong   No
Name:  Sunny
Group:    Outlook Temperature Humidity   Windy   PT
0    Sunny        Hot     High    Weak   No
1    Sunny        Hot     High  Strong   No
7    Sunny       Mild     High    Weak   No
8    Sunny       Cool   Normal    Weak  Yes
10   Sunny       Mild   Normal  Strong  Yes
NOBS 14.0
df_agg_ent          Entropy     Prob1
Outlook
Overcast  0.000000  0.285714
Rainy     0.970951  0.357143
Sunny     0.970951  0.357143
Info-gain for Outlook is : 0.2467498197744391
```

```
List of Attributes: ['Outlook', 'Temperature', 'Humidity', 'Windy', 'PT']
Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Windy']

The Resultant Decision Tree is :

{'Outlook': {'Overcast': 'Yes',
            'Rainy': {'Windy': {'Strong': 'No', 'Weak': 'Yes'}},
            'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
Best Attribute :
 Outlook
Tree Keys:
 dict_keys(['Overcast', 'Rainy', 'Sunny'])

Accuracy is : 0.75
```

Program – 5

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

Python Code:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
# Read dataset to pandas dataframe
dataset = pd.read_csv("C:/Users/65/Documents/P5Data.csv")
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)
classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)
ypred = classifier.predict(Xtest)
i = 0
print ("\n-----------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label',
'Correct/Wrong'))
print ("-----------------------------------------------------------------------")
for label in ytest:
   print ('%-25s %-25s' % (label, ypred[i]), end="")
   if (label == ypred[i]):
      print (' %-25s' % ('Correct'))
   else:
      print (' %-25s' % ('Wrong'))
   i = i + 1
print ("-----------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-----------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-----------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-----------------------------------------------------------------------")
```

Output:

```
   sepal_length  sepal_width  petal_length  petal_width
0          5.1          3.5           1.4          0.2
1          4.9          3.0           1.4          0.2
2          4.7          3.2           1.3          0.2
3          4.6          3.1           1.5          0.2
4          5.0          3.6           1.4          0.2
-----------------------------------------------------------------------
Original Label          Predicted Label          Correct/Wrong
-----------------------------------------------------------------------
Setosa                  Setosa                   Correct
Virginica               Virginica                Correct
Virginica               Virginica                Correct
Setosa                  Setosa                   Correct
Setosa                  Setosa                   Correct
Virginica               Virginica                Correct
Versicolor              Versicolor               Correct
Virginica               Versicolor               Wrong
Setosa                  Setosa                   Correct
Virginica               Virginica                Correct
Virginica               Virginica                Correct
Versicolor              Versicolor               Correct
Setosa                  Setosa                   Correct
Virginica               Virginica                Correct
Setosa                  Setosa                   Correct
-----------------------------------------------------------------------

Confusion Matrix:
 [[6 0 0]
 [0 2 0]
 [0 1 6]]
-----------------------------------------------------------------------

Classification Report:
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00         6
  Versicolor       0.67      1.00      0.80         2
   Virginica       1.00      0.86      0.92         7
```

accuracy			0.93	15
macro avg	0.89	0.95	0.91	15
weighted avg	0.96	0.93	0.94	15

```
-----------------------------------------------------------------------
Accuracy of the classifer is 0.93
-----------------------------------------------------------------------
```

Program – 6

Develop 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.

Python Code:

```
import numpy as np
import csv
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
heartDisease=pd.read_csv
("C:/Users/Lochan/OneDrive/Documents/P6Data.csv")
heartDisease = heartDisease.replace('?',np.nan)
print('Few examples from the dataset are given below')
print(heartDisease.head())

model=BayesianModel([('age','Heartdisease'),('gender','Heartdisease'),('exang','
Heartdisease'),
('cp','Heartdisease'),('Heartdisease','restecg'),('Heartdisease','chol')])
print('\n Learning CPD using Maximum likelihood estimators')
model.fit (heartDisease,estimator=MaximumLikelihoodEstimator)
print ('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
print ('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['Heartdisease'],evidence={'age':35}
)
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['Heartdisease'],evidence={'chol':25
0})
print(q2)
```

Output:

```
Few examples from the dataset are given below
   age  gender  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0   60       1   3       145   233    1        0      150      0      2.3
```

1	35	1	2	130	250	0	1	187	0	3.5
2	41	0	1	130	204	0	0	172	0	1.4
3	55	1	1	120	236	0	1	178	0	0.8
4	56	0	0	120	354	0	1	163	1	0.6

	slope	ca	thal	Heartdisease
0	0	0	1	1
1	0	0	2	1
2	2	0	2	1
3	2	0	2	1
4	2	0	2	1

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

1. Probability of HeartDisease given evidence= restecg

Heartdisease	phi(Heartdisease)
Heartdisease(0)	0.3873
Heartdisease(1)	0.6127

2. Probability of HeartDisease given evidence= cp

Heartdisease	phi(Heartdisease)
Heartdisease(0)	0.0000
Heartdisease(1)	1.0000

Program – 7

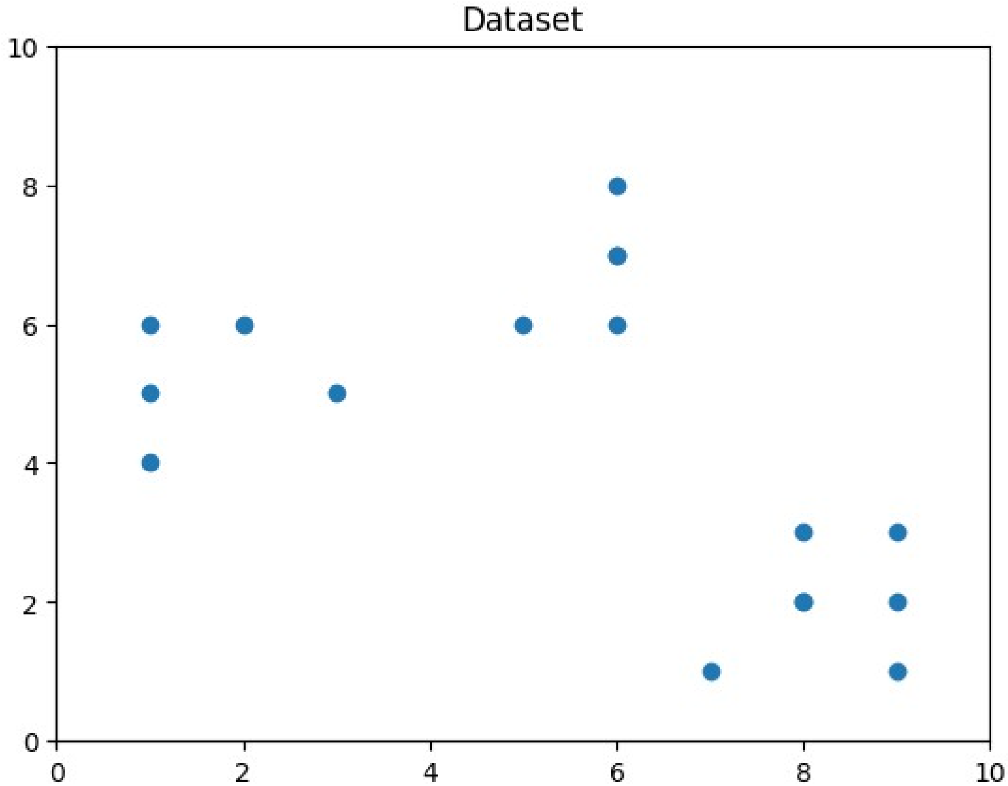
For the given table, write a python program to perform K-Means Clustering.

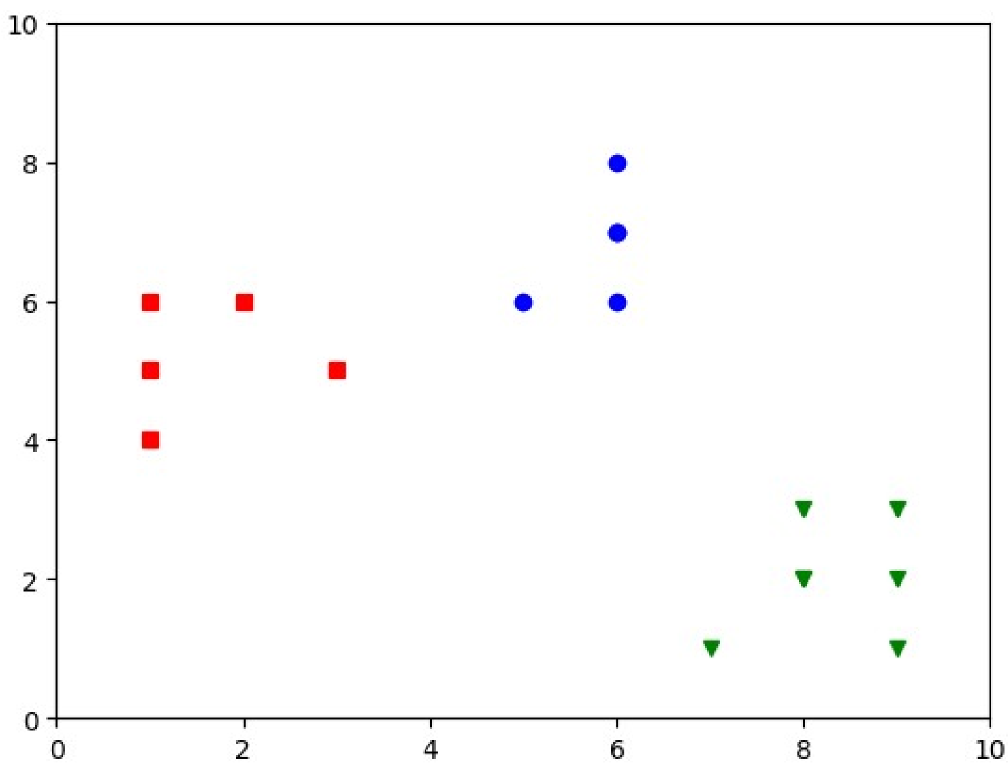
X1	3	1	1	2	1	6	6	6	5	6	7	8	9	8	9	9	8
X2	5	4	6	6	5	8	6	7	6	7	1	2	1	2	3	2	3

Python Code:

```
# clustering dataset
from sklearn.cluster import KMeans
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 6, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])
plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show() #first output

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

# KMeans algorithm
K = 3
kmeans_model = KMeans(n_clusters=K).fit(X)
plt.plot()
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(x1[i], x2[i], color=colors[l], marker=markers[l],ls='None')
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.show()
```

Output:





Program – 8

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same dataset for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

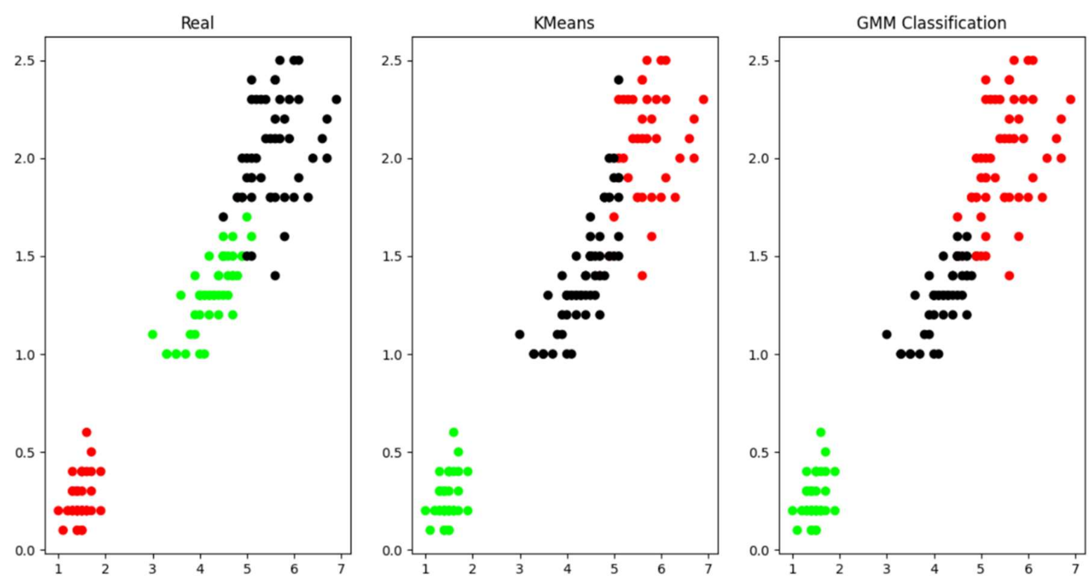
Python Code:

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

dataset = pd.read_csv("C:/Users/Lochan/OneDrive/Documents/P8Data.csv")
X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}
y = [label[c] for c in dataset.iloc[:, -1]]
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=3425).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])
print('The accuracy score of K-Mean: ',metrics.accuracy_score(y,
model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y,
model.labels_))

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=3425).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
```

```
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y,
y_cluster_gmm))
```

Output:

```
The accuracy score of K-Mean:  0.32666666666666666
The Confusion matrixof K-Mean:
 [[ 0  1 49]
 [ 1 49  0]
 [50  0  0]]
The accuracy score of EM:  0.3333333333333333
The Confusion matrix of EM:
  [[ 0  0 50]
 [ 0 50  0]
 [50  0  0]]
```



Program – 9

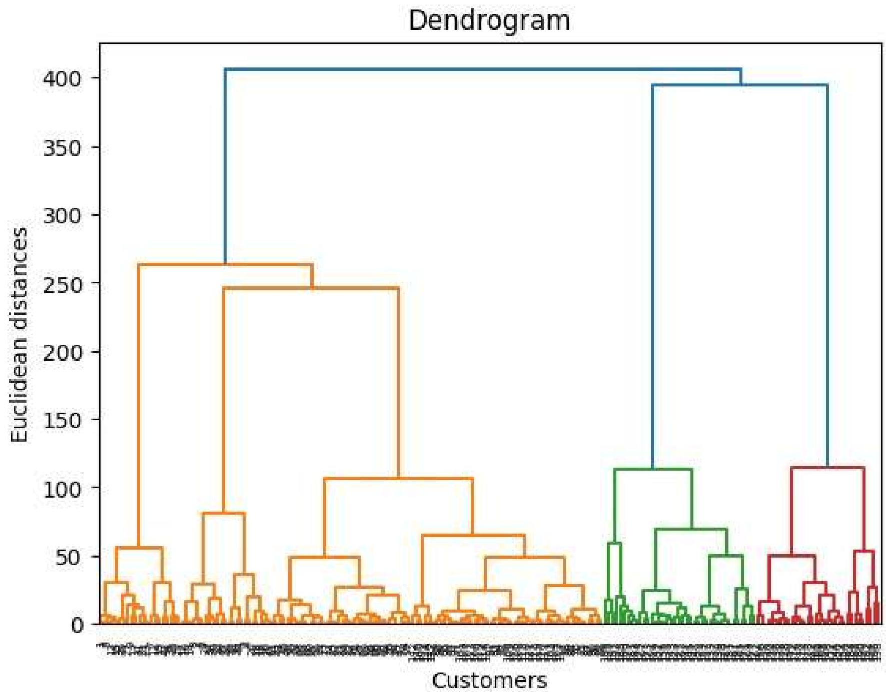
For the given customer dataset, using the dendrogram to find the optimal number of clusters and finding Hierarchical Clustering to the dataset.

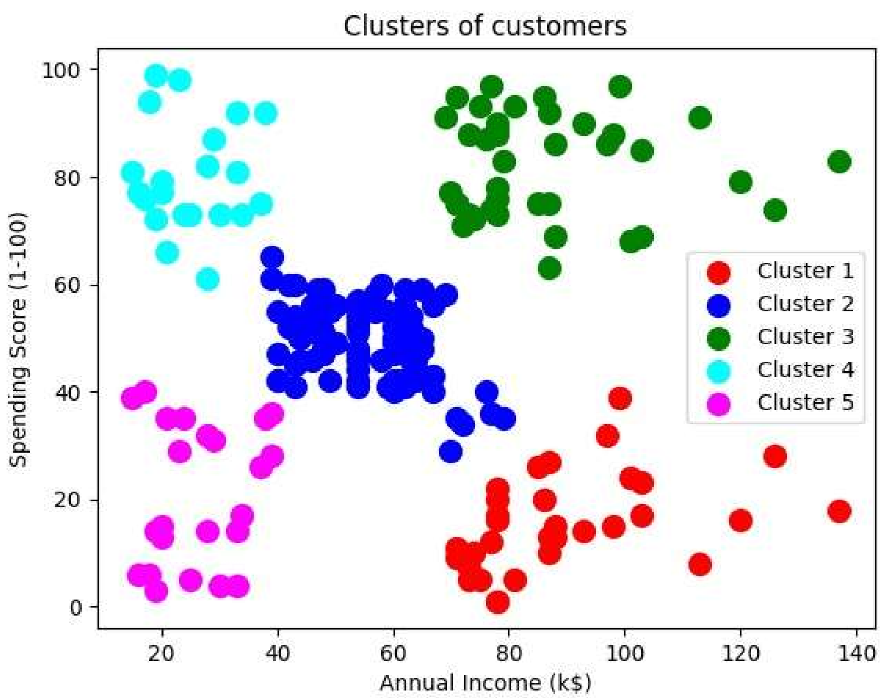
Python Code:

```
# Hierarchical Clustering
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('C:/Users/Lochan/OneDrive/Documents/P9Data.csv')
X = dataset.iloc[:, [3, 4]].values
# y = dataset.iloc[:, 3].values
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, metric = 'euclidean', linkage =
'ward')
y_hc = hc.fit_predict(X)
# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster
1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster
2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label =
'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster
4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label =
'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
```

```
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

<u>Output:</u>





Program – 10

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

Python Code:

```
import numpy as np
x = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
print("small x",x)
#original output
y = np.array(([92], [86], [89]), dtype=float)
X = x/np.amax(x,axis=0) #maximum along the first axis
print("Capital X",X)
#Defining Sigmoid Function for output
def sigmoid (x):
    return (1/(1 + np.exp(-x)))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
#Variables initialization
epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of input layer neurons
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#Defining weight and biases for hidden and output layer
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))
#Forward Propagation
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp = outinp1+ bout
    output = sigmoid(outinp)
```

```
#Backpropagation Algorithm
EO = y-output
outgrad = derivatives_sigmoid(output)
d_output = EO* outgrad
EH = d_output.dot(wout.T)
hiddengrad = derivatives_sigmoid(hlayer_act)
#how much hidden layer wts contributed to error
d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output) *lr
# dotproduct of nextlayererror and currentlayerop
bout += np.sum(d_output, axis=0,keepdims=True) *lr
#Updating Weights
wh += X.T.dot(d_hiddenlayer) *lr
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

```
small x [[2. 9.]
 [1. 5.]
 [3. 6.]]
Capital X [[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[92.]
 [86.]
 [89.]]
Predicted Output:
 [[0.86796822]
 [0.85764499]
 [0.86684427]]
```