Program 1:-

```
import csv
with open('p1.csv', 'r') as f:
        reader = csv.reader(f)
        your_list = list(reader)
h = [['0', '0', '0', '0', '0', '0']]
for i in your_list:
        print(i)
        if i[-1]=="yes":
                j=0
                    for x in i:
                                if x!="yes":
                                            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)
```

Program 2:-

```
import numpy as np
import pandas as pd
data=pd.DataFrame(pd.read_csv('p2.csv'))
concepts=np.array(data.iloc[:,0:-1])
target=np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h=concepts[0].copy()
    print("initalization of specific_h and general_h")
    print(specific_h)
    general_h=[["?" for i in range(len(specific_h))] for i in range (len(specific_h))]
    print(general_h)
    for i,h in enumerate(concepts):
            if target[i]=="Yes":
                    for x in range(len(specific_h)):
                                if h[x]!=specific_h[x]:

                                        specific_h[x]='#'

                                        general_h[x][x]='?'
            if target[i]=="No":
                    for x in range(len(specific_h)):
                                if h[x]!=specific_h[x]:
                                            general_h[x][x]=specific_h[x]
                                else:
                                        general_h[x][x]='?'
            print("steps of candidate elimination algorithm", i+1)
            print(specific_h)
            print(general_h)
```

```
    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)
print("Final General_h:",g_final)
```

Program 3:-

```
import csv
from math import log

def  decision_tree(data,labels):
        results=[row[-1] for row in data]
        if results.count(results[0])==len(results):
                return results[0]

        max_gain_attribute=select_attribute(data)
        tree={max_gain_attribute:{} }
        del (labels[max_gain_attribute])
        nodes=[row[max_gain_attribute] for row in data]
        unique_nodes=set(nodes)

        for node in unique_nodes:
                sublabels=labels[:]
        tree[max_gain_attribute][node]=decision_tree(split_data(data,max_gain_attribute,node),subla
Sbels)
        return tree

def split_data(data,attribute,value):
        new_data=[]
        for row in data:
                if row[attribute]==value:
                        new_row=row[:attribute]
                        new_row.extend(row[attribute+1:])
                        new_data.append(new_row)
         return new_data

def select_attribute(data):
        base_entropy=entropy(data)
        attributes=len(data[0])-1
        best_attribute=-1
        max_info_gain=-1

        for attribute in range(attributes):
                values=[rec[attribute] for rec in data]
                unique_values=set(values)
                attr_entropy=0

                for value in unique_values:
                        new_data=split_data(data,attribute,value)
                        prob=len(new_data)/len(data)
```

```
                        new_entropy=prob*entropy(new_data)
                        attr_entropy+=new_entropy
                info_gain=base_entropy-attr_entropy

                if info_gain>max_info_gain:
                        max_info_gain=info_gain
                        best_attribute=attribute

        return best_attribute

def entropy(data):
        total_rows=len(data)
        dict_outcomes={}
        for row in data:
                outcome=row[-1]
                if outcome not in dict_outcomes.keys():
                        dict_outcomes[outcome]=0
                dict_outcomes[outcome]+=1
 entropy=0
 for key in dict_outcomes:
        prob=dict_outcomes[key]/total_rows
        entropy-=prob*log(prob,2)
 return entropy


def getData(file):
         csv_file=csv.reader(open(file))
         data=[]
         for row in csv_file:
                  data.append(row)
                  print(row)
        return data

def main():
         file="p3.csv"
         data=getData(file)
         labels=data[0]
         tree=decision_tree(data[1:],labels)
         print(".....DECISION-TREE.....")
         print(tree)

main()
```

Program 4:-

```
import numpy as np # numpy is commonly used to process number array

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # Features ( Hrs Slept, Hrs Studied)
y = np.array(([92], [86], [89]), dtype=float) # Labels(Marks obtained)

X = X/np.amax(X,axis=0) # Normalize
y = y/100
```

```
def sigmoid(x):
        return 1/(1 + np.exp(-x))
def sigmoid_grad(x):
        return x * (1 - x)

 # Variable initialization
epoch=1000 #Setting training iterations

eta =0.2 #Setting learning rate (eta)
input_neurons = 2 #number of features in data set
hidden_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

# Weight and bias - Random initialization
wh=np.random.uniform(size=(input_neurons,hidden_neurons)) # 2x3
bh=np.random.uniform(size=(1,hidden_neurons)) # 1x3
wout=np.random.uniform(size=(hidden_neurons,output_neurons)) # 1x1
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
        #Forward Propogation
        h_ip=np.dot(X,wh) + bh # Dot product + bias
        h_act = sigmoid(h_ip) # Activation function
        o_ip=np.dot(h_act,wout) + bout
        output = sigmoid(o_ip)

        #Backpropagation
        # Error at Output layer
        Eo = y-output # Error at o/p
        outgrad = sigmoid_grad(output)
        d_output = Eo* outgrad # Errj=Oj(1-Oj)(Tj-Oj)
        # Error at Hidden later
        Eh = d_output.dot(wout.T) # .T means transpose
        hiddengrad = sigmoid_grad(h_act) # How much hidden layer wts contributed to error
        d_hidden = Eh * hiddengrad
        wout += h_act.T.dot(d_output) *eta # Dotproduct of nextlayererror and currentlayerop
        wh += X.T.dot(d_hidden) *eta
        # Error at Hidden later
        Eh = d_output.dot(wout.T) # .T means transpose
        hiddengrad = sigmoid_grad(h_act) # How much hidden layer wts contributed to error
        d_hidden = Eh * hiddengrad
        wout += h_act.T.dot(d_output) *eta # Dotproduct of nextlayererror and currentlayerop
        wh += X.T.dot(d_hidden) *eta
print("Normalized Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Program 5:-

```
import csv
pyes=0
pno=0

def Bayes(data,item,col):
```

```
        N=len(data)
        noc=0
        yesc=0
        for row in data:
                if row[-1]=='yes':

                        yesc+=1
                else:
                        noc+=1

        global pyes,pno
        pyes=yesc/N
        pno=noc/N
        itemy=0
        itemn=0

        for row in data:

              if row[col]==item:
                if row[-1]=='yes':
                        itemy+=1
                else:
                        itemn+=1
        py=itemy/yesc
        pn=itemn/noc
        return py,pn

def getData(file):
   csv_file = csv.reader(open(file))
   data = []
   print(".....Traine data.....")
   for row in csv_file:
        data.append(row)
        print(row)
   return data

def main():
        file = "p5.csv"
        data = getData(file)
        print("enter ur car specifaction?")
        x,y,z=input().split()
        p1y,p1n=Bayes(data[1:],x,0)
        p2y,p2n=Bayes(data[1:],y,1)
        p3y,p3n=Bayes(data[1:],z,2)
        resy=p1y*p2y*p3y*pyes
        resn=p1n*p2n*p3n*pno
        print(".......probality of ur car stolen =", resy)
        print(".......probality of ur car not stolen =", resn)
        if(resy>resn):
                print(" UR CAR STOLEN CHANCE MORE>>>>>>>>>>>>>>>")
        else:
                print(" UR CAR  STOLEN CHANCE LESS>>>>>>>>>>>>>>>")
main()
```

Program 6:-

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True)
twenty_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True)
print(len(twenty_train.data))
print(len(twenty_test.data))
print(twenty_train.target_names)
print("\n".join(twenty_train.data[0].split("\n")))
print(twenty_train.target[0])
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_tf = count_vect.fit_transform(twenty_train.data)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_tf)
X_train_tfidf.shape
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
mod = MultinomialNB()
mod.fit(X_train_tfidf, twenty_train.target)
X_test_tf = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_tf)
predicted = mod.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target,predicted,target_names=twenty_test.target_names))
print("confusion matrix is \n",metrics.confusion_matrix(twenty_test.target, predicted))
```

Program 7:-

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
#Read the attributes
lines = list(csv.reader(open('data7_names.csv', 'r')));
attributes = lines[0]
#Read Cleveland Heart dicease data
heartDisease = pd.read_csv('data7_heart.csv', names = attributes)
heartDisease = heartDisease.replace('?', np.nan)
# Model Baysian Network

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex',
'trestbps'),('exang','trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),('heartdisease','restecg'),('hea
rtdisease','thalach'),('heartdisease','chol')])
# Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPDs using Maximum Likelihood Estimators...');
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
```

```
# Inferencing with Bayesian Network
print('\nInferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
print('\n1.Probability of HeartDisease given Age=20')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])

print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 100})
print(q['heartdisease'])
```

Program 8:-

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

# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']

# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to

# # Visualise the clustering results
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
```

```
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 4)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

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

plt.show()
```

Program 9:-

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

iris=datasets.load_iris()
print("Iris Data set loaded...")

x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
#random_state=0
print("Dataset is split into training and testing samples...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))

classifier = KNeighborsClassifier(n_neighbors=1)

classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)

print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
        print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r])," Predicted-label:",
str(y_pred[r]))

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

Program 10:-

```
from math import ceil
import numpy as np
from scipy import linalg


def lowess(x, y, f=2./3., iter=3):
	n = len(x)
	r = int(ceil(f*n))
	h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
	w = np.clip(np.abs((x[:,None] - x[None,:]) / h), 0.0, 1.0)
	w = (1 - w**3)**3
	yest = np.zeros(n)
	delta = np.ones(n)
	for iteration in range(iter):

		for i in range(n):

			weights = delta * w[:,i]
			b = np.array([np.sum(weights*y), np.sum(weights*y*x)])
			A = np.array([[np.sum(weights),
np.sum(weights*x)],[np.sum(weights*x),np.sum(weights*x*x)]])

			beta = linalg.solve(A, b)
			yest[i] = beta[0] + beta[1]*x[i]

		residuals = y - yest
		s = np.median(np.abs(residuals))

		delta = np.clip(residuals / (6.0 * s), -1, 1)
		delta = (1 - delta**2)**2

	return yest

if __name__ == '__main__':
	import math
	n = 100
	x = np.linspace(0, 2 * math.pi, n)
	print("==========================values of x====================")
	print(x)
	y = np.sin(x) + 0.3*np.random.randn(n)
	print("================================Values of y=================")
	print(y)
	f = 0.25
	yest = lowess(x, y, f=f, iter=3)
	import pylab as pl
	pl.clf()
	pl.plot(x, y, label='y noisy')
	pl.plot(x, yest, label='y pred')
	pl.legend()
	pl.show()
```