1. **Bayes' Theorem**

Program:

```
def bayes_theorem(p_f,p_f_and_a):
        p_a_given_f = (p_f_and_a / p_f)
        return p_a_given_f
p_f = float(input('Enter value of P(F): '))
p_f_and_a = float(input('Enter value of P(F^A): '))
result = bayes_theorem(p_f,p_f_and_a)
print('Result P(A|F) = %.2f%%' % (result*100))
```

Output:

```
Enter value of P(F): 20
Enter value of P(F^A): 3
Result P(A|F) = 15.00%
```

2. **Database connection with python**

Program:

```
import sqlite3
conn = sqlite3.connect('StudentInfo',timeout=2.0)
cur = conn.cursor()
cur.execute('DROP TABLE IF EXISTS Student')
cur.execute('CREATE TABLE Student (name TEXT,rno INTEGER)')
cur.execute('INSERT INTO Student VALUES(?,?)',('Mahesh',45))
cur.execute('INSERT INTO Student VALUES(?,?)',('Rahul',30))
data = cur.execute('SELECT * FROM Student')
print('Date in database is : ')
for row in data:
        print(row)
cur.execute('select * from Student')
print("data in databse using fetchone(): ")
print(cur.fetchone())
cur.execute('select * from Student')
print("data in databse using fetchall(): ")
print(cur.fetchall())
conn.close()
```

Output:

```
Date in database is:
('Mahesh', 45) ('Rahul', 30)
data in database using fetchone():  ('Mahesh', 45)
data in database using fetchall(): [('Mahesh', 45), ('Rahul', 30)]
```

3. **k-nearest neighbours classification using python**

Program:

```
from sklearn.datasets import load_iris
iris = load_iris()
print("Feature name: ", iris.feature_names, " Iris Data: ", iris.data,"Target names: ",
Iris.target_names, "Target: ",iris.target)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(iris.data,iris.target,test_size = .25)

from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
clf.fit(x_train,y_train)

print("Accuracy: ",clf.score(x_train,y_train))
print("Accuracy: ",clf.score(x_test,y_test))

print("Predicted Data")
print(clf.predict(x_test))
prediction = clf.predict(x_test)
print("Test data: ")
print(y_test)

diff = prediction-y_test
print("Result is: ")
print(diff)
print("Total no of samples misclassified: ",sum(abs(diff)))
```

Output:

```
Accuracy= 1.0
Predicted Data: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1 0]
Test data: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1 0]
Result is [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Total no of samples misclassified = 0
```

4. **kmeans clustering**

Program:

```
import numpy as np
from sklearn.cluster import KMeans
x1=np.array([[1.713,1.586],[0.180,1.786],[0.353,1.240],[0.940,1.566],[1.486,0.759],[1.26
6,1.106],[1.540,0.459],[0.459,1.799],[0.773,0.186],[0.906,0.606]])
k_means = KMeans(n_clusters=3,random_state=15)
k_means.fit(x1)
centroid = k_means.cluster_centers_
labels = k_means.labels_
print(centroid)
print(labels)
for i in range(len(x1)):
        print("Cordinate: ",x1[i],"label: ",labels[i])
```

Output:

```
centroids:
[[0.33066667 1.60833333][1.17625    0.5025][1.30633333 1.41933333]]
Labels: [2 0 0 2 1 2 1 0 1 1]
Cordinate: [1.713 1.586] label:  2
Cordinate: [0.18  1.786] label:  0
Cordinate: [0.353 1.24 ] label:  0
Cordinate: [0.94  1.566] label:  2
Cordinate: [1.486 0.759] label:  1
Cordinate: [1.266 1.106] label:  2
Cordinate: [1.54  0.459] label:  1
Cordinate: [0.459 1.799] label:  0
Cordinate: [0.773 0.186] label:  1
Cordinate: [0.906 0.606] label:  1
```

5. **unconditional probability of `golf' and the conditional probability of `single' given `medRisk'**

Program:

```
import pandas as pd
df = pd.read_csv("individual (1).csv")
print(df.head(10))
print(" P(recreation=golf) = ",df[(df.recreation=='golf')].shape[0] / df.shape[0])
res = df[(df.status=="single") & (df.cw=="medRisk")].shape[0] / df[(df.cw=="medRisk")].shape[0]
print("P(status=single/creditworthiness=medRisk) = ",res)
```

Output:

```
P(recreation=golf) = 0.4
P(status=single/creditworthiness=medRisk)= 0.6666666666666666
```

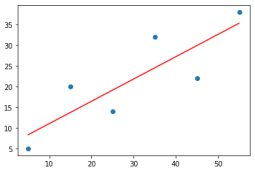
6. **Implement linear regression using python.**

Program:

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])
model = LinearRegression().fit(x, y)
r_sq = model.score(x,y)
print('coefficient of determination:', r_sq)
print('intercept:', model.intercept_)
print('slope:', model.coef_)
y_pred = model.predict(x)
print('predicted response:', y_pred, sep='\n')
```

OUTPUT:

```
coefficient of determination: 0.715875613747954
intercept: 5.633333333333333
slope: [0.54]
predicted response: [ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333
35.33333333]
```



7. **Implement Naïve Bayes theorem to classify the English tex**

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
msg = pd.read_csv('NaiveText.csv',names=['message','label'])
print("The dimensions : ",msg.shape)
msg['labelnum'] = msg.label.map({'pos':1,'neg':0})
x = msg.message
y = msg.labelnum
xtrain,xtest,ytrain,ytest = train_test_split(x,y,random_state=42)
print("Total no if training data: ",ytrain.shape)
print("Total no of test data: ",ytest.shape)
cv = CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm = cv.transform(xtest)
df = pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names_out())
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
print('Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\nThe value of Precision', metrics.precision_score(ytest,predicted))
print('The value of Recall', metrics.recall_score(ytest,predicted))
```

OUTPUT:

```
The dimensions of the dataset (18, 2)
The total number of Training Data : (13,)
The total number of Test Data : (5,)
Accuracy of the classifier is 0.6
Confusion matrix
[[2 0]
 [2 1]]
The value of Precision 1.0 The value of Recall 0.3333333333333333
```

8. **Build an ANN by implementing the Back-propagation algorithm.**

Program:

```
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)
y = y/100

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

def derivatives_sigmoid(x):
    return x*(1-x)

epoch = 5
lr = 0.1
np.random.seed(1)
inputlayer_neurons = 2
hiddenlayer_neurons = 3
outputlayer_neurons = 1

wh = np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh = np.random.uniform(size=(1,hiddenlayer_neurons))
wout = np.random.uniform(size=(hiddenlayer_neurons,outputlayer_neurons))
bout = np.random.uniform(size=(1,outputlayer_neurons))

for i in range(epoch):
    #Forward Propagation
    hinp1=np.dot(x,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)
    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
```

```
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)

d_hiddenlayer = EH * hiddengrad
wout += hlayer_act.T.dot(d_output)*lr
wh += x.T.dot(d_hiddenlayer)*lr
print("Input: \n" + str(x))
print("\nActual Output: \n" + str(y))
print("\nPredicted Output: \n" ,output)
```

```
Output
   Input:
   [[0.66666667 1.00000000]
    [0.33333333 0.55555556]
    [1.00000000 0.66666667]]

   Actual Output:
   [[0.92]
    [0.86]
    [0.89]]

   Predicted Output:
    [[0.78605432]
    [0.77454138]
    [0.78851621]]
```