PROGRAM 1

OBJECTIVE: Write a program to implement Logistic Regression.

CODE:

 # import libraries and dataset

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read_csv("insurance_data.csv")

#data Preprocessing

df.isnull().sum()

df.shape

df.describe()

#split the data

x=df.iloc[:,:1]

y=df.iloc[:,1:]

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4,random_state=23)

#train the model

from sklearn.linear_model import LogisticRegression

model=LogisticRegression()

model.fit(x_train,y_train)

#model prediction

y_pred=model.predict(x_test)

#finding Evaluation matrics

from sklearn.metrics import confusion_matrix

import matplotlib.pyplot as plt

Aditya Rawat
01290302021

```
import seaborn as sns
cf=confusion_matrix(y_test,y_pred)
plt.figure()
sns.heatmap(cf,annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion matrix')
from sklearn.metrics import accuracy_score
accuracy= accuracy_score(y_test,y_pred)
print("accuracy:{:.2f}%".format(accuracy*100))
from sklearn.metrics import precision_score,recall_score,f1_score
print("Precision score:{:.2f}%".format(precision_score(y_test,y_pred)*100))
print("Recall Score::{:.2f}%".format(recall_score(y_test,y_pred)*100))
print("f1 score:{:.2f}%".format(f1_score(y_test,y_pred)*100))
```


Aditya Rawat
01290302021

OUTPUT:

```
In [2]: df=pd.read_csv("insurance_data.csv")
        df
```

Out[2]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1

```
In [6]: x=df.iloc[:,:1]
        y=df.iloc[:,1:]
```

```
In [7]: x
```

Out[7]:

	age
0	22
1	25
2	47
3	52
4	46
5	56

```
In [8]: y
```

Out[8]:

	bought_insurance
0	0
1	0
2	1
3	0
4	1
5	1

```
In [12]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4,random_state=23)
```

```
In [15]: from sklearn.linear_model import LogisticRegression
         model=LogisticRegression()
```

```
In [16]: model.fit(x_train,y_train)

C:\Users\hp\AppData\Local\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[16]:

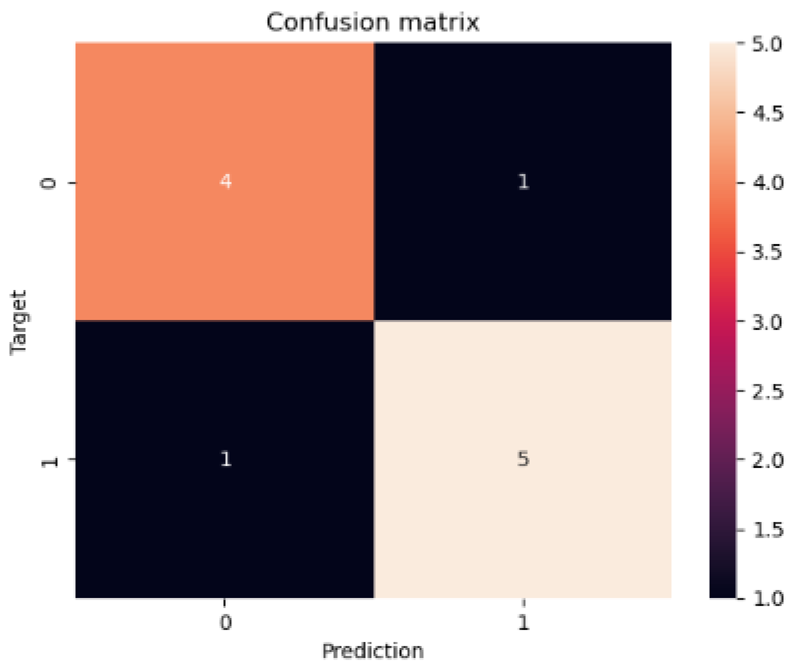
```
▾ LogisticRegression
LogisticRegression()
```

```
In [17]: y_pred=model.predict(x_test)
```


Aditya Rawat
01290302021

In [18]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cf=confusion_matrix(y_test,y_pred)
plt.figure()
sns.heatmap(cf,annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion matrix')
```

Out[18]:
```
Text(0.5, 1.0, 'Confusion matrix')
```



In [20]:
```
from sklearn.metrics import precision_score,recall_score,f1_score
print("Precision score:{:.2f}%".format(precision_score(y_test,y_pred)*100))
print("Recall Score::{:.2f}%".format(recall_score(y_test,y_pred)*100))
print("f1 score:{:.2f}%".format(f1_score(y_test,y_pred)*100))
```

```
Precision score:83.33%
Recall Score::83.33%
f1 score:83.33%
```


Aditya Rawat
01290302021

PROGRAM 2

OBJECTIVE: Write a program to implement Linear Regression with one variables.

CODE:

```
# import libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
# Load the dataset
data = {'Years_of_Experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'Salary':
    [40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000, 80000, 85000]}
df = pd.DataFrame(data)
# Split the data
X = df['Years_of_Experience'].values.reshape(-1, 1)
y = df['Salary'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
# Train the model
model = LinearRegression()
model.fit(X_train,  y_train)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```


Aditya Rawat
01290302021

 # Plotting

```
plt.scatter(X, y, color='blue')

plt.plot(X, model.predict(X), color='red')

plt.title('Years of Experience vs. Salary')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()
```

OUTPUT:

```
In [16]: #inport libraries for signle variable linear regression
         import numpy as np
         import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_squared_error
         import matplotlib.pyplot as plt
```

```
In [17]: # Load the dataset
         data = {'Years_of_Experience': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'Salary':
                 [40000, 45000, 50000, 55000, 60000, 65000, 70000, 75000, 80000, 85000]}
         df = pd.DataFrame(data)
```

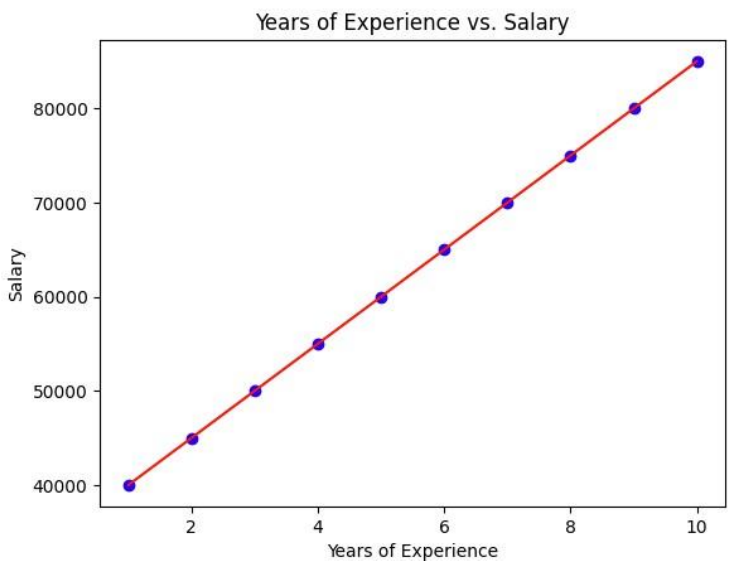
```
In [18]: # Split the data
         X = df['Years_of_Experience'].values.reshape(-1, 1)
         y = df['Salary'].values
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                             random_state=42)
```

```
In [19]: # Train the model
         model = LinearRegression()
         model.fit(X_train, y_train)
```

```
Out[19]: ▾ LinearRegression
         LinearRegression()
```

```
In [20]: # Make predictions
         y_pred = model.predict(X_test)

         # Evaluate the model
         mse = mean_squared_error(y_test, y_pred)
         print(f"Mean Squared Error: {mse}")

Mean Squared Error: 0.0
```

```
In [21]: # Plotting

         plt.scatter(X, y, color='blue')
         plt.plot(X, model.predict(X), color='red')
         plt.title('Years of Experience vs. Salary')
         plt.xlabel('Years of Experience')
         plt.ylabel('Salary')
         plt.show()
```



Aditya Rawat
01290302021

PROGRAM 3

OBJECTIVE: Write a program to implement Linear Regression with two variables.

CODE:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

df=pd.read_csv("boston-housing-dataset.csv")

df.isnull().sum()

df.shape

df.describe()

x= df.iloc[:,0:13]  #diving data into x and y

x

y=df.iloc[:,13]

y

from sklearn.model_selection import train_test_split       #dividing data into train and test based on randomstate method

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.10,random_state=40)

from sklearn.preprocessing import StandardScaler
scaler =StandardScaler()      # to call only standardize function and not to train(just call)
scaler.fit_transform(x_train)  # fit is to give the standard scaler data to transform(standardize) and fit intox train data
X_train =scaler.fit_transform(x_train) # only train

X_test = scaler.transform(x_test) # only change

from sklearn.linear_model import LinearRegression

#cross validation

from sklearn.model_selection import cross_val_score

# estimator

regression = LinearRegression()
```

regression.fit(X_train,y_train)

cross_val_score(regression,X_train,y_train,scoring='neg_mean_squared_error',cv=10) **#these valueswill be different for all systems as values are taken random so score will also vary**

mse=cross_val_score(regression,X_train,y_train,scoring='neg_mean_squared_error',cv=10)

#storing values into mse

np.mean(mse)

#prediction

reg_pred =regression.predict(X_test)

reg_pred

import seaborn as sns

sns.displot(reg_pred-y_test)

from sklearn.metrics import r2_score

score =r2_score(reg_pred,y_test)

score

Aditya Rawat
01290302021

OUTPUT:

```
In [4]: df=pd.read_csv("boston-housing-dataset.csv")
        df
```

Out[4]:

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

```
In [6]: x= df.iloc[:,0:13]  #diving data into x and y
```

```
In [7]: y=df.iloc[:,13]
```

```
In [8]: from sklearn.model_selection import train_test_split       #dividing data into train and test based on random state method
        x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.10,random_state=40)
```

```
In [9]: x_train  # but here the decimal values are discrete(1,4,7 etc)
```

Out[9]:

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
12	12	0.09378	12.5	7.87	0	0.524	5.889	39.0	5.4509	5	311.0	15.2	390.50
310	310	2.63548	0.0	9.90	0	0.544	4.973	37.8	2.5194	4	304.0	18.4	350.45
405	405	67.92080	0.0	18.10	0	0.693	5.683	100.0	1.4254	24	666.0	20.2	384.97
418	418	73.53410	0.0	18.10	0	0.679	5.957	100.0	1.8026	24	666.0	20.2	16.45
163	163	1.51902	0.0	19.58	1	0.605	8.375	93.9	2.1620	5	403.0	14.7	388.45
...	...	...	...	...	...	...	...	...	...	...	...	...	...
440	440	22.05110	0.0	18.10	0	0.740	5.818	92.4	1.8662	24	666.0	20.2	391.45
165	165	2.92400	0.0	19.58	0	0.605	6.101	93.0	2.2834	5	403.0	14.7	240.16
7	7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311.0	15.2	396.90
219	219	0.11425	0.0	13.89	1	0.550	6.373	92.4	3.3633	5	276.0	16.4	393.74
326	326	0.30347	0.0	7.38	0	0.493	6.312	28.9	5.4159	5	287.0	19.6	396.90

455 rows × 13 columns

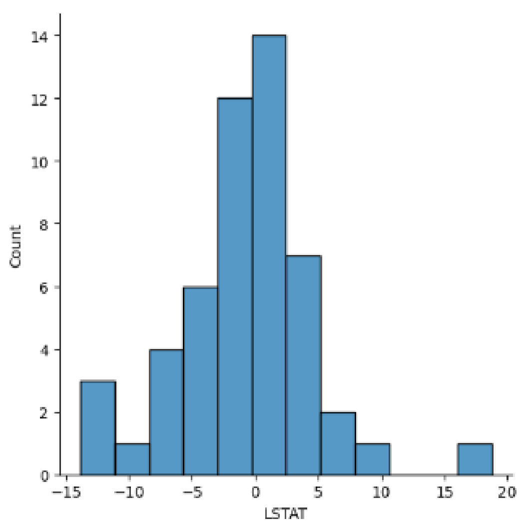
```
In [10]: from sklearn.preprocessing import StandardScaler
         scaler =StandardScaler()      # to call only standardize function and not to train(just call)
```

```
In [11]: scaler.fit_transform(x_train)   # fit is to give the standard scaler data to transform(standardize) and fit into x train data
```

```
Out[11]: array([[-1.66314065, -0.39493589,  0.05881085, ..., -0.56712089,
                 -1.54500291,  0.37941884],
                [ 0.37212357, -0.09175343, -0.48418707, ..., -0.60880171,
                 -0.03697567, -0.05089125],
                [ 1.02094942,  7.69569747, -0.48418707, ...,  1.54669214,
                  0.81128966,  0.32000274],
                ...,
                [-1.69728938, -0.38887987,  0.05881085, ..., -0.56712089,
                 -1.54500291,  0.4481825 ],
                [-0.24938329, -0.39249416, -0.48418707, ..., -0.77552499,
                 -0.9794927 ,  0.41423044],
                [ 0.48139951, -0.36992336, -0.48418707, ..., -0.71002656,
                  0.52853455,  0.4481825 ]])
```

```
In [12]: x_train =scaler.fit_transform(x_train) # only train
         x_test = scaler.transform(x_test) # only change
```

```
In [15]: from sklearn.linear_model import LinearRegression
         #cross validation
         from sklearn.model_selection import cross_val_score

In [16]: # estimator
         regression = LinearRegression()
         regression.fit(X_train,y_train)

Out[16]: ▾ LinearRegression
         LinearRegression()

In [17]: cross_val_score(regression,X_train,y_train,scoring='neg_mean_squared_error',cv=10)  #these values will be differ

Out[17]: array([-28.94179834, -23.0608436 , -15.54632857, -18.0865388 ,
                 -8.95692519, -17.01503615, -11.73314584, -22.88137448,
                -11.60561642, -17.09032995])

In [18]: mse=cross_val_score(regression,X_train,y_train,scoring='neg_mean_squared_error',cv=10) #storing values into mse

In [19]: import numpy as np
         np.mean(mse)

Out[19]: -17.49179373207918

In [20]: #prediction
         reg_pred =regression.predict(X_test)

In [21]: reg_pred

Out[21]: array([16.07879814,  5.00069276, 16.95394955, 18.0443779 , 15.4976683 ,
                14.14550016,  8.63256373, 16.38754087, 16.70900917, 29.4034721 ,
                11.21496112, 15.92697665, 22.65808116, 24.76002238, 10.35786763,
                 9.4243188 , 12.07721013, 16.19729959, 13.148523  , 17.11632489,
                 1.98682757, 20.38472578,  3.60728592,  7.97657611,  2.02481831,
                 3.87492122,  8.60224097, 15.70259083, 15.22821993, 14.39847259,
                 9.48291381, 11.47115589,  9.12462972,  9.23418362, 26.28911803,
                 6.11357086,  6.39458915, 20.14564815,  3.97699318,  8.59140618,
                11.13966363, 21.5752782 , 15.01162204, 11.24311788, 12.30292074,
                21.5319964 , 17.4820148 ,  5.35171674, 22.02567034,  7.38005371,
                15.98160939])

In [22]: import seaborn as sns
         sns.displot(reg_pred-y_test)

Out[22]: <seaborn.axisgrid.FacetGrid at 0x1cae0a01f50>
```




```
In [23]: from sklearn.metrics import r2_score

In [24]: score =r2_score(reg_pred,y_test)

In [25]: score

Out[25]: 0.2287798024913632
```


Aditya Rawat
01290302021

PROGRAM 4

OBJECTIVE: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

CODE:

```
import pandas as pd

from sklearn import tree

from sklearn.preprocessing import LabelEncoder

from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('tennis.csv')

print("The first 5 values of data is:\n",data.head())

#splitting data

X = data.iloc[:,:-1]

print("\nThe First 5 values of train data is\n",X.head())

y = data.iloc[:,-1]

print("\nThe first 5 values of Train outpu tis\n",y.head())

# Convert then in numbers

le_outlook = LabelEncoder()

X.outlook =le_outlook.fit_transform(X.outlook)

le_temp = LabelEncoder()

X.temp =le_temp.fit_transform(X.temp)

le_humidity = LabelEncoder()

X.humidity =le_humidity.fit_transform(X.humidity)

le_windy = LabelEncoder()

X.windy  =le_windy.fit_transform(X.windy)

print("\nNow the Train data is :\n",X.head())

le_play = LabelEncoder()
```


Aditya Rawat
01290302021

```
y = le_play.fit_transform(y)

print("\nNow the Train output is\n",y)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()

classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score

print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```


Aditya Rawat
01290302021

OUTPUT:

```
In [1]: import pandas as pd
        from sklearn import tree
        from sklearn.preprocessing import LabelEncoder
        from sklearn.naive_bayes import GaussianNB

In [3]: data = pd.read_csv('tennis.csv')
        print("THe first 5 values of data is:\n",data.head())

        THe first 5 values of data is:
              outlook  temp humidity  windy play
        0     sunny   hot     high  False   no
        1     sunny   hot     high   True   no
        2  overcast   hot     high  False  yes
        3     rainy  mild     high  False  yes
        4     rainy  cool   normal  False  yes

In [4]: #splitting data
        X = data.iloc[:,:-1]
        print("\nThe First 5 values of train datais\n",X.head())
        y = data.iloc[:,-1]
        print("\nThe first 5 values of Train outputis\n",y.head())

        The First 5 values of train datais
              outlook  temp humidity  windy
        0     sunny   hot     high  False
        1     sunny   hot     high   True
        2  overcast   hot     high  False
        3     rainy  mild     high  False
        4     rainy  cool   normal  False

        The first 5 values of Train outputis
         0     no
        1     no
        2    yes

In [8]: # Convert then in numbers
        le_outlook = LabelEncoder()
        X.outlook =le_outlook.fit_transform(X.outlook)
        le_temp = LabelEncoder()
        X.temp =le_temp.fit_transform(X.temp)
        le_humidity = LabelEncoder()
        X.humidity =le_humidity.fit_transform(X.humidity)
        le_windy = LabelEncoder()
        X.windy =le_windy.fit_transform(X.windy)
        print("\nNow the Train data is :\n",X.head())
        le_play = LabelEncoder()
        y = le_play.fit_transform(y)
        print("\nNow the Train output is\n",y)

        Now the Train data is :
            outlook  temp  humidity  windy
        0        2     1         0      0
        1        2     1         0      1
        2        0     1         0      0
        3        1     2         0      0
        4        1     0         1      0

        Now the Train output is
         [0 0 1 1 1 0 1 0 1 1 1 1 1 0]

In [9]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test =train_test_split(X,y, test_size=0.20)
        classifier = GaussianNB()
        classifier.fit(X_train,y_train)

Out[9]: ▾ GaussianNB
        GaussianNB()

In [10]: from sklearn.metrics import accuracy_score
         print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))

         Accuracy is: 0.6666666666666666
```


Aditya Rawat
01290302021

PROGRAM 5

OBJECTIVE: Implement a K-Nearest Neighbors (KNN) classifier from scratch in Python. Use a sample dataset, such as the Iris dataset, and split it into a training and testing set. Train the KNN classifier on the training set and evaluate its performance on the testing set. Experiment with different values of k and report the accuracy of the classifier.

CODE:

```
#importing libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

#Model creation from scratch

def most_common(lst):

    return max(set(lst), key=lst.count)

def euclidean(point, data):

    # Euclidean distance between points a & data

    return np.sqrt(np.sum((point - data)**2, axis=1))

class KNeighborsClassifier:

    def__init_(self, k=5, dist_metric=euclidean):

        self.k = k

        self.dist_metric = dist_metric

    def fit(self, X_train, y_train):

        self.X_train = X_train

        self.y_train = y_train

    def predict(self, X_test):

        neighbors = []

        for x in X_test:
```


Aditya Rawat
01290302021

```
            distances = self.dist_metric(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(most_common, neighbors))
    def evaluate(self, X_test, y_test):
        y_pred = self.predict(X_test)
        accuracy = sum(y_pred == y_test) / len(y_test)
        return accuracy
# unpack the iris dataset.
iris = datasets.load_iris()
X = iris['data']
y = iris['target']
# Split data into train & test sets
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.20)
# Preprocessing of dataset
ss = StandardScaler().fit(X_train)
X_train, X_test = ss.transform(X_train),ss.transform(X_test)
# Test knn model across varying ks
accuracies = []
ks = range(1, 30)
for k in ks:
    knn = KNeighborsClassifier(k=k)
    knn.fit(X_train, y_train)
    accuracy = knn.evaluate(X_test, y_test)
    accuracies.append(accuracy)
#outputs
```

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

accuracy=accuracy_score(y_test,knn.predict(X_test))

print("Accuracy: {:.2f}%".format(accuracy * 100))

print("Confusion Matrix:\n",confusion_matrix(y_test,knn.predict(X_test)))

print("Classification Report:\n",classification_report(y_test,knn.predict(X_test)))

# Visualize accuracy vs. k

fig, ax = plt.subplots()

ax.plot(ks, accuracies)

ax.set(xlabel="k",

       ylabel="Accuracy",

       title="Performance of knn")

plt.show()
```


Aditya Rawat
01290302021

OUTPUT:

```
#importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
#Model creation from scratch
def most_common(lst):
    return max(set(lst), key=lst.count)
def euclidean(point, data):
    # Euclidean distance between points a & data
    return np.sqrt(np.sum((point - data)**2, axis=1))
class KNeighborsClassifier:
    def __init__(self, k=5, dist_metric=euclidean):
        self.k = k
        self.dist_metric = dist_metric
    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train
    def predict(self, X_test):
        neighbors = []
        for x in X_test:
            distances = self.dist_metric(x, self.X_train)
            y_sorted = [y for _, y in sorted(zip(distances, self.y_train))]
            neighbors.append(y_sorted[:self.k])
        return list(map(most_common, neighbors))
    def evaluate(self, X_test, y_test):
        y_pred = self.predict(X_test)
        accuracy = sum(y_pred == y_test) / len(y_test)
        return accuracy
```

```
# Unpack the iris dataset, from UCI Machine Learning Repository
iris = datasets.load_iris()
X = iris['data']
y = iris['target']
X
```

```
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
# Split data into train & test sets
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.20)
```

```
# Preprocessing of dataset
ss = StandardScaler().fit(X_train)
X_train, X_test = ss.transform(X_train),ss.transform(X_test)
```

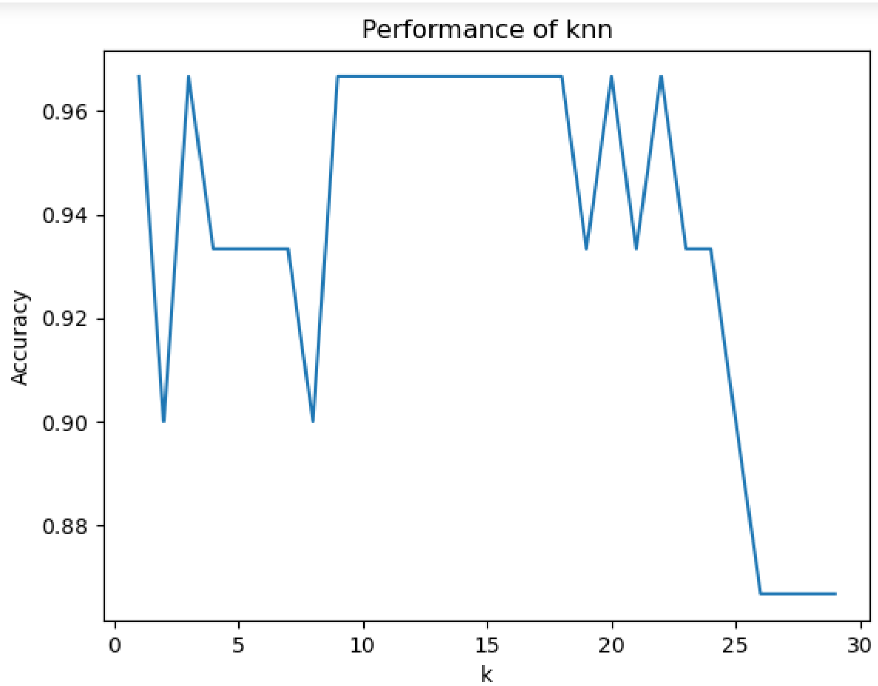

Aditya Rawat
01290302021

```
# Test knn model across varying ks
accuracies = []
ks = range(1, 30)
for k in ks:
    knn = KNeighborsClassifier(k=k)
    knn.fit(X_train, y_train)
    accuracy = knn.evaluate(X_test, y_test)
    accuracies.append(accuracy)
```

```
#outputs
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
accuracy=accuracy_score(y_test,knn.predict(X_test))
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Confusion Matrix:\n",confusion_matrix(y_test,knn.predict(X_test)))
print("Classification Report:\n",classification_report(y_test,knn.predict(X_test)))
```

```
Accuracy: 86.67%
Confusion Matrix:
 [[10  0  0]
 [ 0  8  2]
 [ 0  2  8]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.80      0.80      0.80        10
           2       0.80      0.80      0.80        10

    accuracy                           0.87        30
   macro avg       0.87      0.87      0.87        30
weighted avg       0.87      0.87      0.87        30
```

```
# Visualize accuracy vs. k
fig, ax = plt.subplots()
ax.plot(ks, accuracies)
ax.set(xlabel="k",
       ylabel="Accuracy",
       title="Performance of knn")
plt.show()
```



Aditya Rawat
01290302021

PROGRAM 6

OBJECTIVE: Write a python program to implement support Vector Machine(SVM) classifier using a library like scikit-learn. choose a suitable dataset for binary classification(e.g., the Breast Cancer dataset) and split it into training and testing sets. Train the SVM classifier on the training data and evaluate its performance on the testing data, reporting metrics such as accuracy, precision, recall, and F1-score. Experiment with different kernel functions (e.g., linear, radial basis function) and compare their performance.

CODE:

```
#importing libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_score,confusion_matrix

# Loading the dataset

cancer = datasets.load_breast_cancer()

X = cancer.data

y = cancer.target

# Splitting the dataset into training and testing sets

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2,random_state=42)

#Model creation

def evaluate_classifier(clf, X_test, y_test):

    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test,y_pred)

    precision = precision_score(y_test,y_pred)

    recall = recall_score(y_test, y_pred)

    f1 = f1_score(y_test, y_pred)

    confusion_mat =confusion_matrix(y_test, y_pred)
```

```
    return accuracy, precision, recall, f1,confusion_mat
```

 # Try SVM with different kernel functions

```
kernel_functions = ['linear','rbf']
for kernel in kernel_functions:
```

 # Create and train the SVM classifier

```
    svm_classifier = SVC(kernel=kernel)
    svm_classifier.fit(X_train, y_train)
```

 # Evaluate the classifier

```
  accuracy, precision, recall, f1,confusion_mat =evaluate_classifier(svm_classifier, X_test, y_test)
```

#results

```
print(f'\nResults for SVM with {kernel} kernel:')
```

#Output:

```
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
print(f'Confusion Matrix:\n{confusion_mat}')
```


Aditya Rawat
01290302021

OUTPUT:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_score,confusion_matrix
```

```
# Loading the dataset
cancer = datasets.load_breast_cancer()
X = cancer.data
y = cancer.target
X
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
```

```
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2,random_state=42)
```

```
#Model creation
def evaluate_classifier(clf, X_test, y_test):
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    precision = precision_score(y_test,y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    confusion_mat =confusion_matrix(y_test, y_pred)
    return accuracy, precision, recall, f1,confusion_mat
# Try SVM with different kernel functions
kernel_functions = ['linear','rbf']
for kernel in kernel_functions:
# Create and train the SVM classifier
    svm_classifier = SVC(kernel=kernel)
    svm_classifier.fit(X_train, y_train)
accuracy, precision, recall, f1,confusion_mat =evaluate_classifier(svm_classifier, X_test, y_test)
```

```
#results
print(f'\nResults for SVM with {kernel} kernel:')
#Output:
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
print(f'Confusion Matrix:\n{confusion_mat}')
```

```
Results for SVM with rbf kernel:
Accuracy: 0.9474
Precision: 0.9221
Recall: 1.0000
F1-score: 0.9595
Confusion Matrix:
[[37  6]
 [ 0 71]]
```

```
#results
print(f'\nResults for SVM with {kernel} kernel:')
#Output:
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-score: {f1:.4f}')
print(f'Confusion Matrix:\n{confusion_mat}')
```

```
Results for SVM with linear kernel:
Accuracy: 0.9561
Precision: 0.9459
Recall: 0.9859
F1-score: 0.9655
Confusion Matrix:
[[39  4]
 [ 1 70]]
```

PROGRAM 7

OBJECTIVE: Given a dataset containing features and labels, implement a Random Forest classification model using Python and a library like scikit-learn. Split the dataset into training and testing sets, train the model, and evaluate its performance using metrics like accuracy, precision, and recall.

CODE:

#importing libraries

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

from sklearn.datasets import load_iris
```

#loading dataset

```
iris.data = load_iris()

X = iris.data

y = iris.target

X

y
```

 # Splitting the dataset into training and testing sets

```
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.30,random_state=42)
```

 # Creating a Random Forest Classifier

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

 # Training the model

```
rf_classifier.fit(X_train, y_train)
```

 # Making predictions on test set

```
y_pred = rf_classifier.predict(X_test)
```

 # Evaluating the classifier

Aditya Rawat
01290302021

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
conf_mat = confusion_matrix(y_test, y_pred)
# Printing performance metrics
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print('Confusion Matrix:')
print(conf_mat)
# Plotting the confusion matrix
plt.imshow(conf_mat, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```


Aditya Rawat
01290302021

OUTPUT:

```
#importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
from sklearn.datasets import load_iris
```

```
# Loading the dataset
iris = load_iris()
X = iris.data
y = iris.target
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.30,random_state=42)
```

```
# Creating a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
# Training the model
rf_classifier.fit(X_train, y_train)
```

```
▾        RandomForestClassifier
RandomForestClassifier(random_state=42)
```

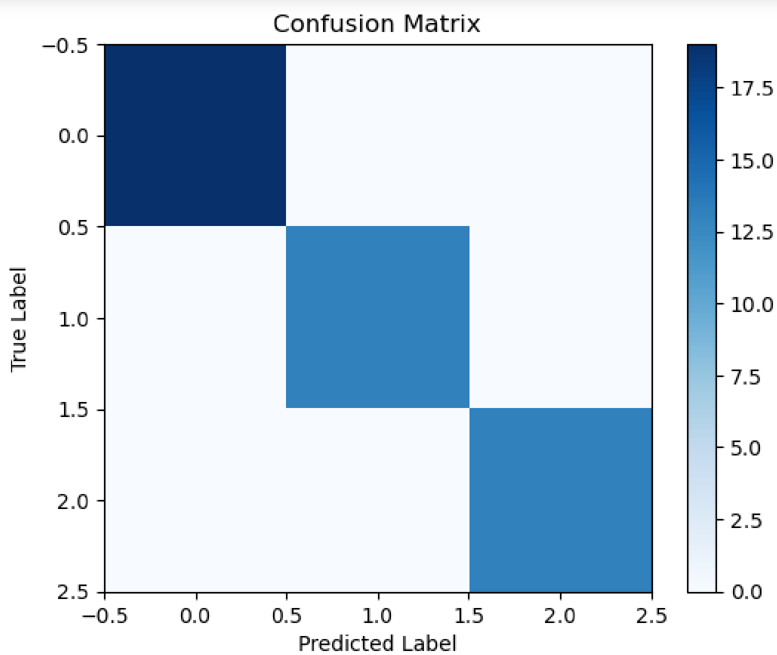
```
# Making predictions on the test set
y_pred = rf_classifier.predict(X_test)
```

```
# Evaluating the classifier
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
conf_mat = confusion_matrix(y_test, y_pred)
```

```
# Printing performance metrics
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print('Confusion Matrix:')
print(conf_mat)
```

```
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

```
# Plotting the confusion matrix
plt.imshow(conf_mat, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



Aditya Rawat
01290302021

PROGRAM 8

OBJECTIVE: Design a Hebb Net to implement logical AND function.

CODE:

```
#ImportingLibrary.

import numpy as np

# Define the input patterns and target for the AND function

input_patterns = np.array([[1, 1], [1, -1], [-1, 1], [-1, -1]])

target_output = np.array([1, -1, -1, -1])

weights = np.zeros(input_patterns.shape[1])

bias = 0

weights

# Train the Hebb Net

for i in range(input_patterns.shape[0]):

    weights += input_patterns[i] * target_output[i]

    bias += target_output[i]

    print(f"weights{i}: ", weights, f"\nbias{i}: ", bias)

# Define function to make predictions with trained Hebb Net

def predict(input_pattern):

    return 1 if np.dot(input_pattern, weights) + bias >= 0 else 0

# Test the trained Hebb Net

for input_pattern in input_patterns:

    print(f"Input: {input_pattern}, Output: {predict(input_pattern)}")
```


Aditya Rawat
01290302021

OUTPUT:

```
In [21]: #importing numpy
         import numpy as np

In [22]: # Define the input patterns and target output for the AND function
         input_patterns = np.array([[1,1], [1,-1], [-1, 1], [-1, -1]])
         target_output = np.array([1, -1, -1, -1])

In [23]:
         weights = np.zeros(input_patterns.shape[1])
         bias = 0
         weights

Out[23]: array([0., 0.])

In [25]: # Train the Hebb Net
         for i in range(input_patterns.shape[0]):
             weights += input_patterns[i] * target_output[i]
             bias += target_output[i]
             print(f"weights{i}: ", weights, f"\nbias{i}: ", bias)

weights0:  [3. 3.]
bias0:  -1
weights1:  [2. 4.]
bias1:  -2
weights2:  [3. 3.]
bias2:  -3
weights3:  [4. 4.]
bias3:  -4
```

```
In [26]: # Define a function to make predictions with the trained Hebb Net
         def predict(input_pattern):
             return 1 if np.dot(input_pattern, weights) + bias >= 0 else 0

In [27]: # Test the trained Hebb Net
         for input_pattern in input_patterns:
             print(f"Input: {input_pattern}, Output: {predict(input_pattern)}")

Input: [1 1], Output: 1
Input: [ 1 -1], Output: 0
Input: [-1  1], Output: 0
Input: [-1 -1], Output: 0
```


Aditya Rawat
01290302021

PROGRAM 9

OBJECTIVE: Apply k-Means algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the Agglomerative Clustering algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes in the program.

CODE:

```
#import libraries

import numpy as np

import pandas as pd

from sklearn.cluster import KMeans, AgglomerativeClustering

import matplotlib.pyplot as plt

from sklearn.metrics import silhouette_score

from sklearn.datasets import load_iris

# Loading dataset

iris = load_iris()

X = iris.data

# we are using the first two features for clustering

X = X[:, :2]

# No need to scale the features in this

#Apply k-Means clustering

kmeans = KMeans(n_clusters=3,random_state=42)

kmeans_labels = kmeans.fit_predict(X)

# Apply hierarchical clustering forcomparison

hierarchical =AgglomerativeClustering(n_clusters=3)

hierarchical_labels =hierarchical.fit_predict(X)

# Visualize the results

plt.figure(figsize=(12, 5))
```


Aditya Rawat
01290302021

```
plt.subplot(1, 2, 1)

plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels,cmap='viridis', edgecolors='k', s=50)

plt.title('k-Means Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.subplot(1, 2, 2)

plt.scatter(X[:, 0], X[:, 1],

c=hierarchical_labels, cmap='viridis',edgecolors='k', s=50)

plt.title('Hierarchical Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.tight_layout()

plt.show()
```

 # Compare clustering quality using silhouette score

```
kmeans_silhouette = silhouette_score(X,kmeans_labels)
```

#Output

```
hierarchical_silhouette = silhouette_score(X,hierarchical_labels)

print(f'Silhouette Score - k-Means:{kmeans_silhouette:.4f}')

print(f'Silhouette Score - Hierarchical:{hierarchical_silhouette:.4f}')
```

OUTPUT:

```
#importing libraries
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans, AgglomerativeClustering
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score
from sklearn.datasets import load_iris
```

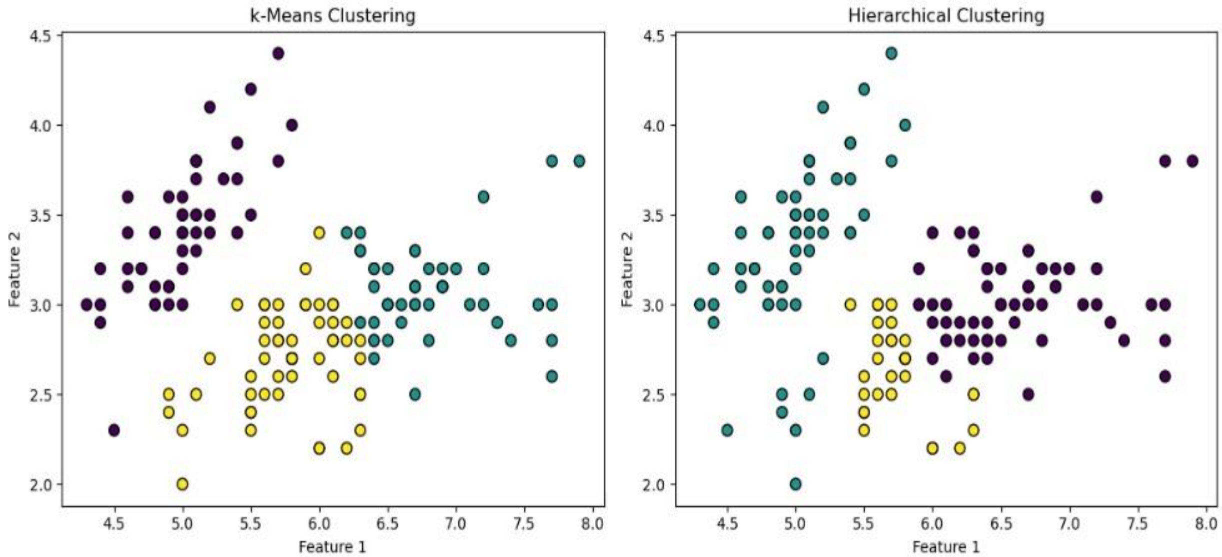
```
# Loading the dataset
iris = load_iris()
X = iris.data
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
```

```
# Assume we are using the first two features for clustering
X = X[:, :2]
# No need to scale the features in this case
# Apply k-Means clustering
kmeans = KMeans(n_clusters=3,random_state=42)
kmeans_labels = kmeans.fit_predict(X)
# Apply hierarchical clustering forcomparison
hierarchical =AgglomerativeClustering(n_clusters=3)
hierarchical_labels =hierarchical.fit_predict(X)
```

```
# Visualize the results
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=kmeans_labels,cmap='viridis', edgecolors='k', s=50)
plt.title('k-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1],
c=hierarchical_labels, cmap='viridis',edgecolors='k', s=50)
plt.title('Hierarchical Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.tight_layout()
plt.show()
```


Aditya Rawat
01290302021




```
# Compare clustering quality using silhouette score
kmeans_silhouette = silhouette_score(X,kmeans_labels)
#Output
hierarchical_silhouette = silhouette_score(X,hierarchical_labels)
print(f'Silhouette Score - k-Means:{kmeans_silhouette:.4f}')
print(f'Silhouette Score - Hierarchical:{hierarchical_silhouette:.4f}')
```

```
Silhouette Score - k-Means:0.4451
Silhouette Score - Hierarchical:0.3653
```


Aditya Rawat
01290302021

PROGRAM 10

OBJECTIVE: Write a Python program to implement a Self-Organizing Map (SOM) and train it on a given dataset, such as a collection of 2D points. Allow the user to specify parameters like the map size, learning rate, and number of training iterations. Visualize the map before and after training to observe how it adapts to the data.

CODE:

```
#numpy based SOM implimentation

!pip install minisom

#importing libraries

import numpy as np

from minisom import MiniSom

import matplotlib.pyplot as plt

#for visualization

def visualize_som(som, data, title):

    plt.figure(figsize=(5, 5))

    plt.pcolor(som.distance_map().T, cmap='bone_r') # plot the distance map as background

    plt.colorbar()

  #for visualization

def visualize_som(som, data, title):

    plt.figure(figsize=(5, 5))

    plt.pcolor(som.distance_map().T, cmap='bone_r') # plot the distance map as background

    plt.colorbar()

    # plot points on the map

    for i (x, _) in enumerate(data):

        w = som.winner(x)

        plt.plot(w[0] + 0.5, w[1] + 0.5, 'o',markerfacecolor='None',  markersize=10,markeredgecolor='r',
        markeredgewidth=2)

        plt.text(w[0] + 0.5, w[1] + 0.5, str(i + 1),color='k', fontweight='bold',ha='center', va='center')
```

```
    plt.title("SOM")
    plt.show()
# Generate synthetic 2D data
np.random.seed(42)
data = np.random.rand(100, 2) # replace this with your own dataset
# User-defined parameters
map_size = (10, 10) # SOM map size
learning_rate = 0.5 # initial learning rate
num_iterations = 1000 # number of training iterations
# Visualize the SOM before training
visualize_som(som, data, title="SOM Before Training")
# Create and train the SOM
som = MiniSom(*map_size, 2, sigma=1.0, learning_rate=learning_rate)
som.random_weights_init(data)
print("Training SOM...")
som.train_random(data, num_iterations)
print("Training complete.")
# Visualize the SOM after training
visualize_som(som, data, title="SOM After Training")
```

OUTPUT:

```
In [5]: #numpy based SOM implimentation
!pip install minisom

Collecting minisom
  Downloading MiniSom-2.3.1.tar.gz (10 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: minisom
  Building wheel for minisom (setup.py): started
  Building wheel for minisom (setup.py): finished with status 'done'
  Created wheel for minisom: filename=MiniSom-2.3.1-py3-none-any.whl size=10601 sha256=36bf28d62436e67a3444b7eec0218017a0b772e0
bbd305fe764135a52ea1517d
  Stored in directory: c:\users\hp\appdata\local\pip\cache\wheels\28\e3\3d\707f393fa9013d5ab7b3ffb914ded8ca3c40dec231fa392528
Successfully built minisom
Installing collected packages: minisom
Successfully installed minisom-2.3.1
```

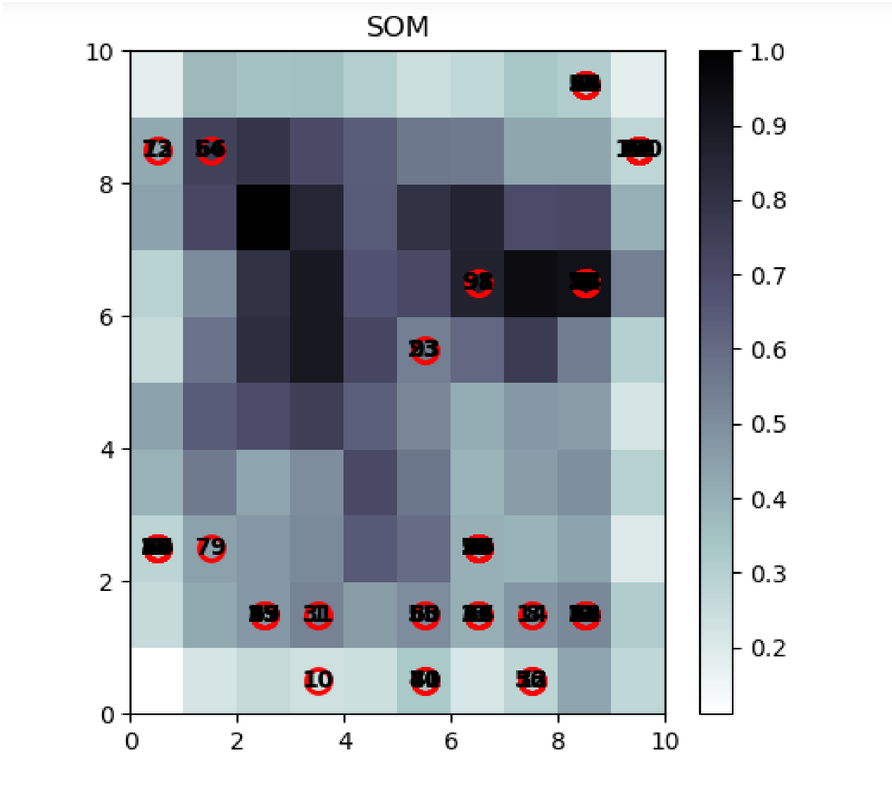
```python
#importing necessary libraries
import numpy as np
from minisom import MiniSom
import matplotlib.pyplot as plt
```

```python
#for visualization
def visualize_som(som, data, title):
    plt.figure(figsize=(5, 5))
    plt.pcolor(som.distance_map().T, cmap='bone_r') # plot the distance map as background
    plt.colorbar()
    # plot data points on the map
    for i, (x, _) in enumerate(data):
        w = som.winner(x)
        plt.plot(w[0] + 0.5, w[1] + 0.5, 'o',markerfacecolor='None', markersize=10,markeredgecolor='r', markeredgewidth=2)
        plt.text(w[0] + 0.5, w[1] + 0.5, str(i + 1),color='k', fontweight='bold',ha='center', va='center')
    plt.title("SOM")
    plt.show()
```

```python
# Generate synthetic 2D data
np.random.seed(42)
data = np.random.rand(100, 2) # replace this with your own dataset
```

```python
# User-defined parameters
map_size = (10, 10) # SOM map size
learning_rate = 0.5 # initial learning rate
num_iterations = 1000 # number of training iterations
```

```python
# Visualize the SOM before training
visualize_som(som, data, title="SOM Before Training")
```

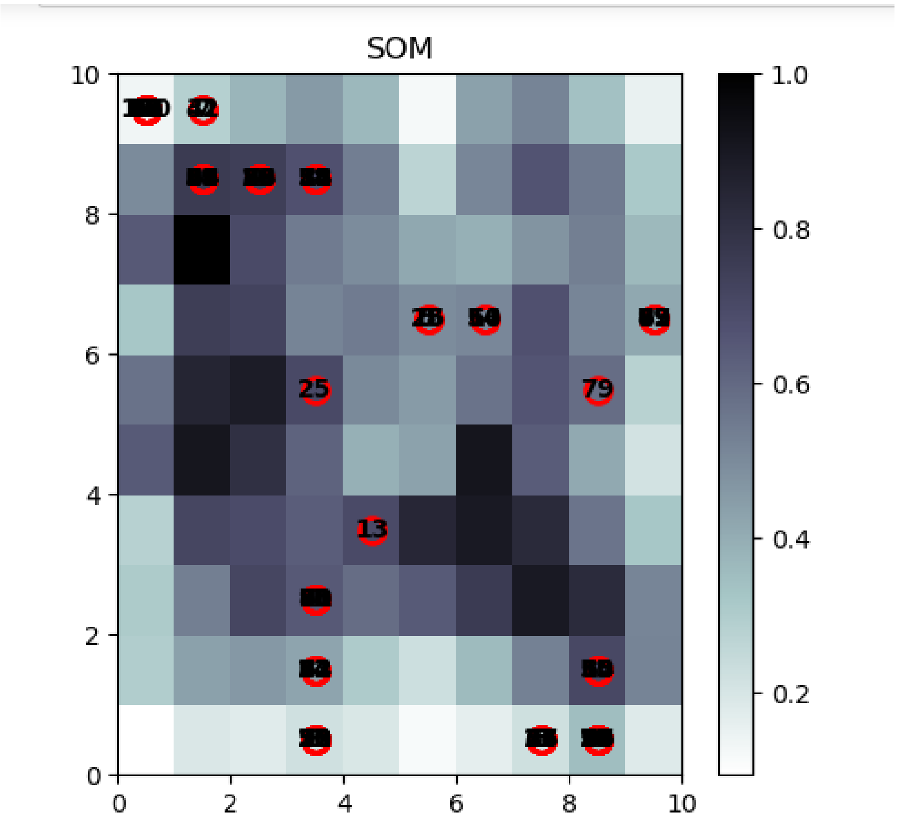



```python
# Create and train the SOM
som = MiniSom(*map_size, 2, sigma=1.0, learning_rate=learning_rate)
som.random_weights_init(data)
print("Training SOM...")
som.train_random(data, num_iterations)
print("Training complete.")
```

```
Training SOM...
Training complete.
```

```python
# Visualize the SOM after training
visualize_som(som, data, title="SOM After Training")
```


Aditya Rawat
01290302021



Aditya Rawat
01290302021

PROGRAM 11

OBJECTIVE: (A) Take a binary classification dataset and implement both the K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) classifiers using Python. Compare the performance of these two algorithms on metrics such as accuracy, precision, recall, and F1-score. Visualize the decision boundaries for both algorithms.

CODE:

```
#installing mlxtend

pip install mlxtend

#importing necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

from mlxtend.plotting import plot_decision_regions

# Loading Iris dataset (for binary classification)

iris = datasets.load_iris()

X = iris.data[:, :2] # Selecting only the first two features for visualization

y = (iris.target != 0).astype(int) # Convert to binary classification

# Splitting the dataset into training and testing sets

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.20,random_state=32)

# K-Nearest Neighbors (KNN) Classifier

knn_classifier = KNeighborsClassifier(n_neighbors=3)

knn_classifier.fit(X_train, y_train)

y_pred_knn = knn_classifier.predict(X_test)
```

 # Support Vector Machine (SVM) Classifier

svm_classifier = SVC(kernel='linear')

svm_classifier.fit(X_train, y_train)

y_pred_svm = svm_classifier.predict(X_test)#

Evaluate classifiers

def evaluate_classifier(y_true, y_pred):

accuracy = accuracy_score(y_true, y_pred)

precision = precision_score(y_true, y_pred)

recall = recall_score(y_true, y_pred)

f1 = f1_score(y_true, y_pred)

confusion_mat = confusion_matrix(y_true, y_pred)

return accuracy, precision, recall, f1,confusion_mat

 # Evaluate KNN classifier

accuracy_knn, precision_knn, recall_knn,f1_knn, confusion_mat_knn = evaluate_classifier(y_test, y_pred_knn)

 # Evaluate SVM classifier

accuracy_svm, precision_svm, recall_svm,f1_svm, confusion_mat_svm = evaluate_classifier(y_test, y_pred_svm)

 # Visualize decision boundaries

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

fig.suptitle('Decision Boundaries of KNN and SVM')

 # Decision boundary for KNN

plot_decision_regions(X, y, clf=knn_classifier, legend=2, ax=axes[0])

axes[0].set_title('K-Nearest Neighbors (KNN)')

 # Decision boundary for SVM

plot_decision_regions(X, y,

clf=svm_classifier, legend=2, ax=axes[1])

Aditya Rawat
01290302021

axes[1].set_title('Support Vector Machine(SVM)')

plt.show()

 # Print performance metrics

print('\nPerformance Metrics for K-Nearest Neighbors (KNN):')

print(f'Accuracy: {accuracy_knn:.4f}')

print(f'Precision: {precision_knn:.4f}')

print(f'Recall: {recall_knn:.4f}')

print(f'F1-score: {f1_knn:.4f}')

print(f'Confusion Matrix:\n{confusion_mat_knn}')

print('\nPerformance Metrics for Support Vector Machine (SVM):')

print(f'Accuracy: {accuracy_svm:.4f}')

print(f'Precision: {precision_svm:.4f}')

print(f'Recall: {recall_svm:.4f}')

print(f'F1-score: {f1_svm:.4f}')

print(f'Confusion Matrix:\n{confusion_mat_svm}')

OUTPUT:

```
#installing mlxtend
pip install mlxtend
```

```
Collecting mlxtend
  Downloading mlxtend-0.23.0-py3-none-any.whl (1.4 MB)
                                             0.0/1.4 MB ? eta -:--:--
     -----                                   0.2/1.4 MB 5.9 MB/s eta 0:00:01
```

```
#importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from mlxtend.plotting import plot_decision_regions
```

```
# Loading  Iris dataset (for binary classification)
iris = datasets.load_iris()
X = iris.data[:, :2] # Selecting only the first two features for visualization
y = (iris.target != 0).astype(int) # Convert to binary classification
```

```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.20,random_state=32)
```

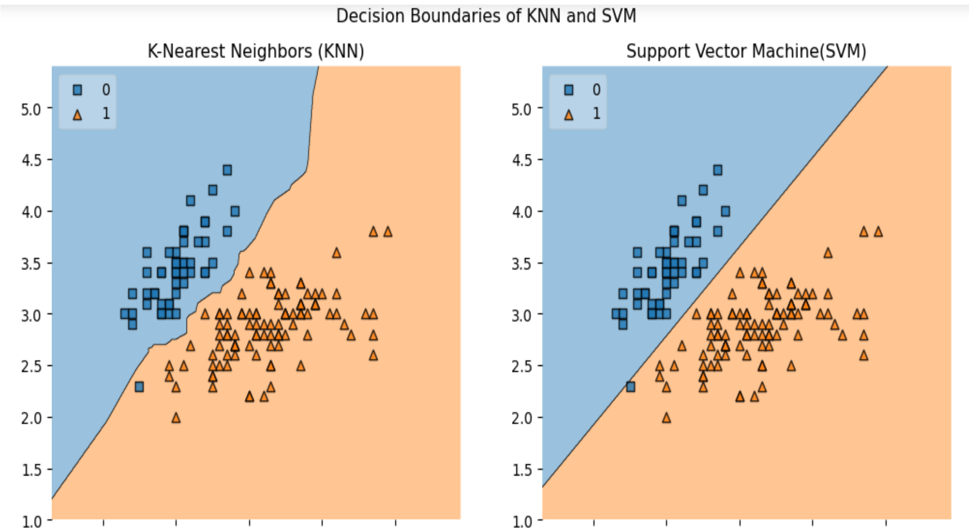
```
# K-Nearest Neighbors (KNN) Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)
y_pred_knn = knn_classifier.predict(X_test)
```

```
# Support Vector Machine (SVM) Classifier
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
y_pred_svm = svm_classifier.predict(X_test)
```

```
# Evaluate classifiers
def evaluate_classifier(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    confusion_mat = confusion_matrix(y_true, y_pred)
    return accuracy, precision, recall, f1,confusion_mat
# Evaluate KNN classifier
accuracy_knn, precision_knn, recall_knn,f1_knn, confusion_mat_knn = evaluate_classifier(y_test, y_pred_knn)
# Evaluate SVM classifier
accuracy_svm, precision_svm, recall_svm,f1_svm, confusion_mat_svm = evaluate_classifier(y_test, y_pred_svm)
```


Aditya Rawat
01290302021

```python
# Visualize decision boundaries
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
fig.suptitle('Decision Boundaries of KNN and SVM')
# Decision boundary for KNN
plot_decision_regions(X, y, clf=knn_classifier, legend=2, ax=axes[0])
axes[0].set_title('K-Nearest Neighbors (KNN)')
# Decision boundary for SVM
plot_decision_regions(X, y,
clf=svm_classifier, legend=2, ax=axes[1])
axes[1].set_title('Support Vector Machine(SVM)')
plt.show()
```




```python
# Print performance metrics
print('\nPerformance Metrics for K-Nearest Neighbors (KNN):')
print(f'Accuracy: {accuracy_knn:.4f}')
print(f'Precision: {precision_knn:.4f}')
print(f'Recall: {recall_knn:.4f}')
print(f'F1-score: {f1_knn:.4f}')
print(f'Confusion Matrix:\n{confusion_mat_knn}')
print('\nPerformance Metrics for Support Vector Machine (SVM):')
print(f'Accuracy: {accuracy_svm:.4f}')
print(f'Precision: {precision_svm:.4f}')
print(f'Recall: {recall_svm:.4f}')
print(f'F1-score: {f1_svm:.4f}')
print(f'Confusion Matrix:\n{confusion_mat_svm}')
```

```
Performance Metrics for K-Nearest Neighbors (KNN):
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-score: 1.0000
Confusion Matrix:
[[12  0]
 [ 0 18]]

Performance Metrics for Support Vector Machine (SVM):
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
F1-score: 1.0000
Confusion Matrix:
[[12  0]
 [ 0 18]]
```


Aditya Rawat
01290302021

B) Given a dataset of customer churn, implement a program that compares the performance of three different supervised learning algorithms (e.g., Logistic Regression, Random Forest, and Support Vector Machine) for binary classification. Split the dataset into training and testing sets, train each algorithm on the training set, and evaluate their performance using metrics like accuracy, precision, recall, and F1-score. Present the results in a clear and informative way, such as through a bar chart or a table.

CODE:

```
#importing necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_score

import matplotlib.pyplot as plt

# Load your dataset

dataset_path = 'telecom_churn.csv'

data = pd.read_csv(dataset_path)

data.head()

# Assuming your dataset has a 'Churn' column indicating binary labels (1 for churn, 0 for nonchurn)

X = data.drop('Churn', axis=1)

y = data['Churn']

X

y

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.20,random_state=32)

# Define the classifiers

classifiers = {'Logistic Regression': LogisticRegression(),  'Random Forest':
RandomForestClassifier(),'Support Vector Machine': SVC()}
```

```
# Train and evaluate each classifier

results = {'Classifier': [], 'Accuracy': [],'Precision': [], 'Recall': [], 'F1-Score': []}

for clf_name, clf in classifiers.items():

    # Train the classifier

    clf.fit(X_train, y_train)#

    Predict on the test set

    y_pred = clf.predict(X_test)

    # Evaluate performance

    accuracy = accuracy_score(y_test, y_pred)

    precision = precision_score(y_test, y_pred)

    recall = recall_score(y_test, y_pred)

    f1 = f1_score(y_test, y_pred)

    # Store results

    results['Classifier'].append(clf_name)

    results['Accuracy'].append(accuracy)

    results['Precision'].append(precision)

    results['Recall'].append(recall)

    results['F1-Score'].append(f1)

# Convert results to a DataFrame for easy visualization

results_df = pd.DataFrame(results)

# Plot the results using a bar chart

plt.figure(figsize=(7, 5))

for metric in ['Accuracy', 'Precision','Recall', 'F1-Score']:

    plt.bar(results_df['Classifier'],results_df[metric], label=metric)

    plt.title('Performance Comparison of Classifiers')

    plt.xlabel('Classifier')
```

```
    plt.ylabel('Score')
    plt.legend()
plt.show()
```

 # Display the results in tabular form

```
print("Results:")
print(results_df)
```


Aditya Rawat
01290302021

OUTPUT:

```
#importing necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
```

```
# Load your dataset
dataset_path = 'telecom_churn.csv'
data = pd.read_csv(dataset_path)
data.head()
```

	Churn	AccountWeeks	ContractRenewal	DataPlan	DataUsage	CustServCalls	DayMins	DayCalls	MonthlyCharge	OverageFee	RoamMins
0	0	128	1	1	2.7	1	265.1	110	89.0	9.87	10.0
1	0	107	1	1	3.7	1	161.6	123	82.0	9.78	13.7
2	0	137	1	0	0.0	0	243.4	114	52.0	6.06	12.2
3	0	84	0	0	0.0	2	299.4	71	57.0	3.10	6.6
4	0	75	0	0	0.0	3	166.7	113	41.0	7.42	10.1

```
# Assuming your dataset has a 'Churn' column indicating binary labels (1 for churn, 0 for nonchurn)
X = data.drop('Churn', axis=1)
y = data['Churn']
X
```

	AccountWeeks	ContractRenewal	DataPlan	DataUsage	CustServCalls	DayMins	DayCalls	MonthlyCharge	OverageFee	RoamMins
0	128	1	1	2.70	1	265.1	110	89.0	9.87	10.0
1	107	1	1	3.70	1	161.6	123	82.0	9.78	13.7
2	137	1	0	0.00	0	243.4	114	52.0	6.06	12.2
3	84	0	0	0.00	2	299.4	71	57.0	3.10	6.6
4	75	0	0	0.00	3	166.7	113	41.0	7.42	10.1
...	...	...	...	...	...	...	...	...	...	...
3328	192	1	1	2.67	2	156.2	77	71.7	10.78	9.9
3329	68	1	0	0.34	3	231.1	57	56.4	7.67	9.6
3330	28	1	0	0.00	2	180.8	109	56.0	14.44	14.1
3331	184	0	0	0.00	2	213.8	105	50.0	7.98	5.0
3332	74	1	1	3.70	0	234.4	113	100.0	13.30	13.7

3333 rows × 10 columns

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.20,random_state=32)
```

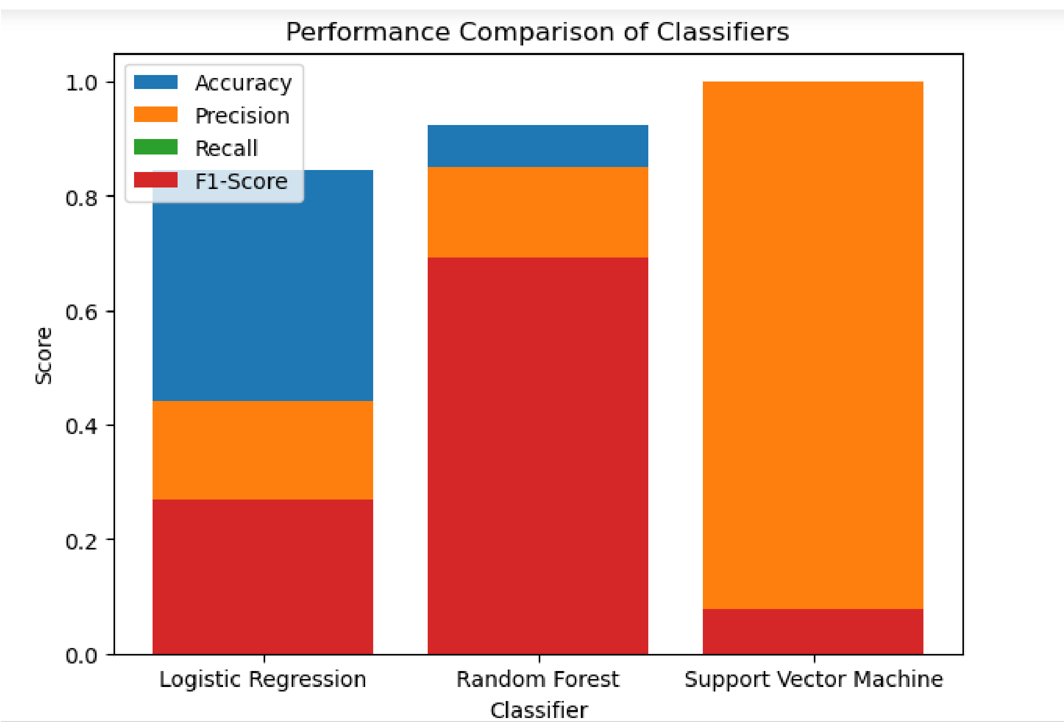
```
# Define the classifiers
classifiers = {'Logistic Regression': LogisticRegression(),
               'Random Forest': RandomForestClassifier(),'Support Vector Machine': SVC()}
```


Aditya Rawat
01290302021

```python
# Train and evaluate each classifier
results = {'Classifier': [], 'Accuracy': [],'Precision': [], 'Recall': [], 'F1-Score': []}
for clf_name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)
    # Predict on the test set
    y_pred = clf.predict(X_test)
    # Evaluate performance
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Store results
    results['Classifier'].append(clf_name)
    results['Accuracy'].append(accuracy)
    results['Precision'].append(precision)
    results['Recall'].append(recall)
    results['F1-Score'].append(f1)
```

```python
# Convert results to a DataFrame for easy visualization
results_df = pd.DataFrame(results)
```

```python
# Plot the results using a bar chart
plt.figure(figsize=(7, 5))
for metric in ['Accuracy', 'Precision','Recall', 'F1-Score']:
    plt.bar(results_df['Classifier'],results_df[metric], label=metric)
    plt.title('Performance Comparison of Classifiers')
    plt.xlabel('Classifier')
    plt.ylabel('Score')
    plt.legend()
plt.show()
```



Aditya Rawat
01290302021

```
# Display the results in tabular form
print("Results:")
print(results_df)
```

```
Results:
               Classifier  Accuracy  Precision    Recall  F1-Score
0     Logistic Regression  0.845577   0.441860  0.193878  0.269504
1           Random Forest  0.923538   0.850746  0.581633  0.690909
2  Support Vector Machine  0.859070   1.000000  0.040816  0.078431
```


Aditya Rawat
01290302021

PROGRAM 12

OBJECTIVE: Write a Python program that loads a dataset and performs an empirical comparison of three different clustering algorithms, such as K-Means, Hierarchical Agglomerative Clustering, and DBSCAN. Evaluate and compare their performance in terms of cluster quality metrics like Silhouette Score or Inertia, and visualize the results.
CODE:

```
#importing libraries

import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans,AgglomerativeClustering, DBSCAN

from sklearn.metrics import silhouette_score

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.pipeline import make_pipeline

# Load the dataset

iris = datasets.load_iris()

X = iris.data

y = iris.target

X

# Standardize the features

scaler =  StandardScaler()

X_std = scaler.fit_transform(X)

# Apply PCA for visualization purposes (2D plot)

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_std)

# Define clustering algorithms

kmeans = KMeans(n_clusters=3, random_state=42)
```

```
hierarchical = AgglomerativeClustering(n_clusters=3)

dbscan = DBSCAN(eps=0.8, min_samples=5)

algorithms = [kmeans, hierarchical, dbscan]

algorithm_names = [ 'K-Means','Hierarchical Agglomerative','DBSCAN']

# Evaluate and visualize each clustering algorithm

for algorithm, algorithm_name in zip(algorithms, algorithm_names):

# Fit the clustering algorithm to the data

    if algorithm_name != 'DBSCAN':

        algorithm.fit(X_std)

        labels=algorithm.labels_

    else:

        labels = algorithm.fit_predict(X_std)

    # Evaluate clustering quality using Silhouette Score

    silhouette_avg = silhouette_score(X_std, labels)

    print(f"{algorithm_name} Silhouette Score: {silhouette_avg:.4f}")

# Visualize the clustering results in a 2D

plotplt.figure(figsize=(8, 5))

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis', edgecolor='k', s=50)

plt.title(f'{algorithm_name} Clustering Results (Silhouette Score: {silhouette_avg:.4f})')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.show()
```


Aditya Rawat
01290302021

OUTPUT:

```
#importing libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans,AgglomerativeClustering, DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
```

```
# Load the dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
X
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]
```

```
# Standardize the features
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

```
# Apply PCA for visualization purposes (2D plot)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_std)
```

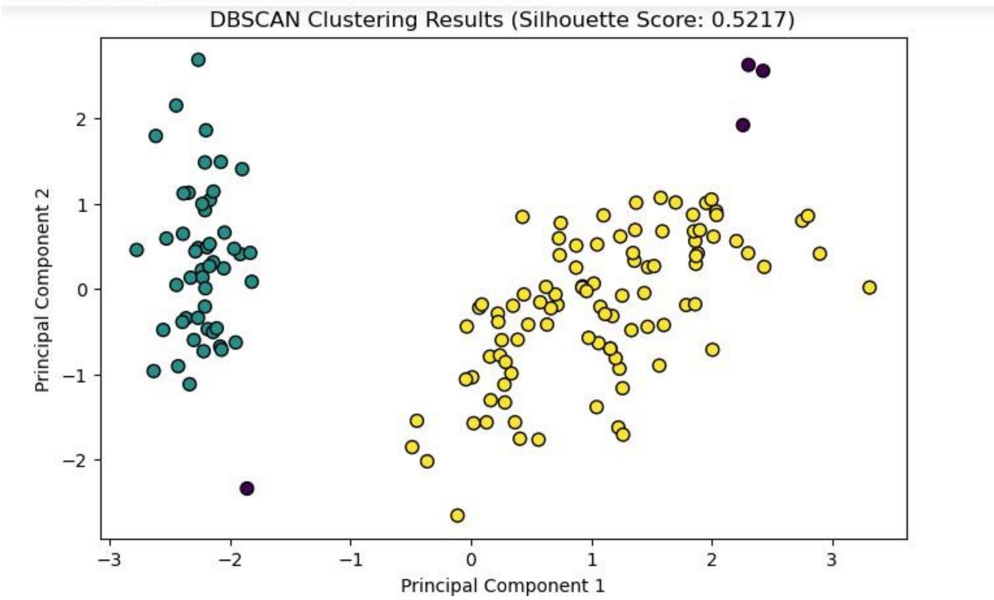
```
# Define clustering algorithms
kmeans = KMeans(n_clusters=3, random_state=42)
hierarchical = AgglomerativeClustering(n_clusters=3)
dbscan = DBSCAN(eps=0.8, min_samples=5)
algorithms = [kmeans, hierarchical, dbscan]
algorithm_names = [ 'K-Means','Hierarchical Agglomerative','DBSCAN']
```

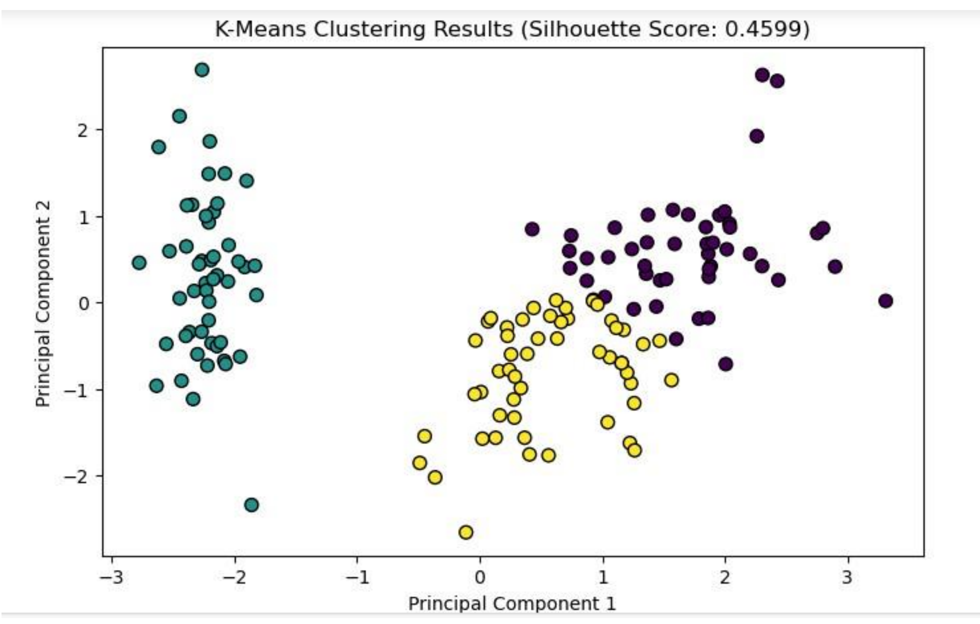
```
# Evaluate and visualize each clustering algorithm
for algorithm, algorithm_name in zip(algorithms, algorithm_names):
# Fit the clustering algorithm to the data
    if algorithm_name != 'DBSCAN':
        algorithm.fit(X_std)
        labels=algorithm.labels_
    else:
        labels = algorithm.fit_predict(X_std)
    # Evaluate clustering quality using Silhouette Score
    silhouette_avg = silhouette_score(X_std, labels)
    print(f"{algorithm_name} Silhouette Score: {silhouette_avg:.4f}")
```

```
K-Means Silhouette Score: 0.4599
Hierarchical Agglomerative Silhouette Score: 0.4467
DBSCAN Silhouette Score: 0.5217
```

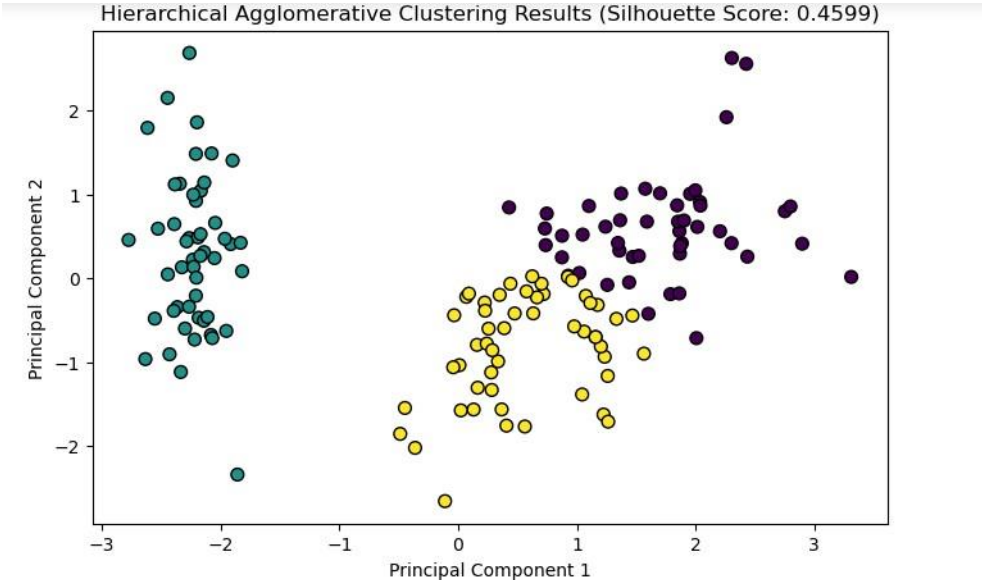

Aditya Rawat
01290302021

```
# Visualize the clustering results in a 2D plot
plt.figure(figsize=(8, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='viridis', edgecolor='k', s=50)
plt.title(f'{algorithm_name} Clustering Results (Silhouette Score: {silhouette_avg:.4f})')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```





Aditya Rawat
01290302021


Hierarchical Agglomerative Clustering Results (Silhouette Score: 0.4599)
Principal Component 2
Principal Component 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3