C BYREGOWDA INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Affiliated to Visvesvaraya Technological University

“Jnana Sangama”, Belgaum – 560018.

LABORATORY MANUAL

“MACHINE LEARNING (BAIL606)”

Semester:VI **Scheme:CBCS**

Ashok Babu

and

Vijetha K

Assistant Professor

Dept. of AIML

Scrutinizedby

Dr.DEEPIKA LOKESH

Professor & HOD

Artificial Intelligence and

Machine Learning

C BYREGOWDA INSTITUTE OF TECHNOLOGY

Department of Artificial Intelligence & Machine Learning

An ISO 9001:2015 Certified Institute

Kolar– SrinivaspurRoad

Sl.NO	Experiments
1	Develop a program to Load a dataset and select one numerical column. Compute mean, median, mode, standard deviation, variance, and range for a given numerical column in a dataset. Generate a histogram and box plot to understand the distribution of the data.Identifyanyoutliers in the data using IQR. Select a categorical variable from a dataset. Compute the frequency of each category and display it as a bar chart or pie chart.
2	Develop a program to Load a dataset with at least two numerical columns (e.g., Iris, Titanic). Plot a scatter plot of two variables and calculate their Pearson correlation coefficient. Write a programtocomputethecovarianceandcorrelationmatrixforadataset.Visualizethecorrelation matrix using a heatmap to know which variables have strong positive/negative correlations.
3	DevelopaprogramtoimplementPrincipalComponentAnalysis(PCA)forreducingthe dimensionality of the Iris dataset from 4 features to 2.
4	Develop a program to load the Iris dataset. Implement the k-Nearest Neighbors (k-NN) algorithm for classifying flowers based on their features. Split the dataset into training and testingsetsandevaluatethemodelusingmetricslikeaccuracyandF1-score.Testitfordifferent values of k (e.g., k=1,3,5) and evaluate the accuracy. Extend the k-NN algorithm to assign weights based on the distance of neighbors (e.g., $weight$=1/d2).Comparetheperformanceofweightedk-NNandregulark-NNonasyntheticor real-world dataset.
5	Implementthenon-parametricLocallyWeightedRegressionalgorithminordertofitdata points. Select appropriate data set for your experiment and draw graphs.
6	Develop a program to demonstrate the working of Linear Regression and Polynomial Regression.UseBostonHousingDatasetforLinearRegressionandAutoMPGDataset(for vehicle fuel efficiency prediction) for Polynomial Regression.
7	Develop a program to load the Titanic dataset. Split the data into training and test sets. Train a decisiontreeclassifier.Visualizethetreestructure.Evaluateaccuracy,precision,recall,andF1- score.
8	DevelopaprogramtoimplementtheNaiveBayesianclassifierconsideringIrisdatasetfor training. Compute the accuracy of the classifier, considering the test data.
9	Developaprogramtoimplementk-meansclusteringusingWisconsinBreastCancerdataset and visualize the clustering result.

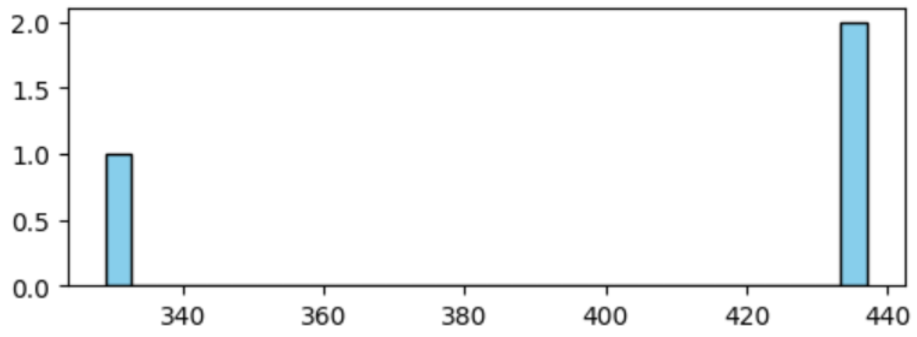
Program1: Develop a program to Load a dataset and select one numerical column. Compute mean, median, mode, standard deviation, variance, and range for a given numerical column in a dataset. Generate a histogram and boxplot to understand the distribution of the data. Identify any outliers in the data using IQR. Select a categorical variable from a dataset. Compute the frequency of each category and display it as a bar chart or pie chart.

```
importpandasaspd
import numpy as np
importmatplotlib.pyplotasplt
import seaborn as sns
fromscipyimport stats
data=pd.read_csv('your_dataset.csv')
print(data.head())
column=‘sal'
data[column]=pd.to_numeric(data[column],errors='coerce')
data_clean = data[column].dropna()
mean_value = data_clean.mean()
median_value=data_clean.median()
mode_value = data_clean.mode()[0]
std_dev = data_clean.std()
variance_value = data_clean.var()
range_value=data_clean.max()-data_clean.min()
print(f"\nMean: {mean_value}")
print(f"Median: {median_value}")
print(f"Mode: {mode_value}")
print(f"StandardDeviation:{std_dev}")
print(f"Variance: {variance_value}")
print(f"Range: {range_value}")
plt.figure(figsize=(10, 6))
plt.hist(data_clean,bins=30,color='skyblue',edgecolor='black')
plt.show()
plt.figure(figsize=(8, 6))
```

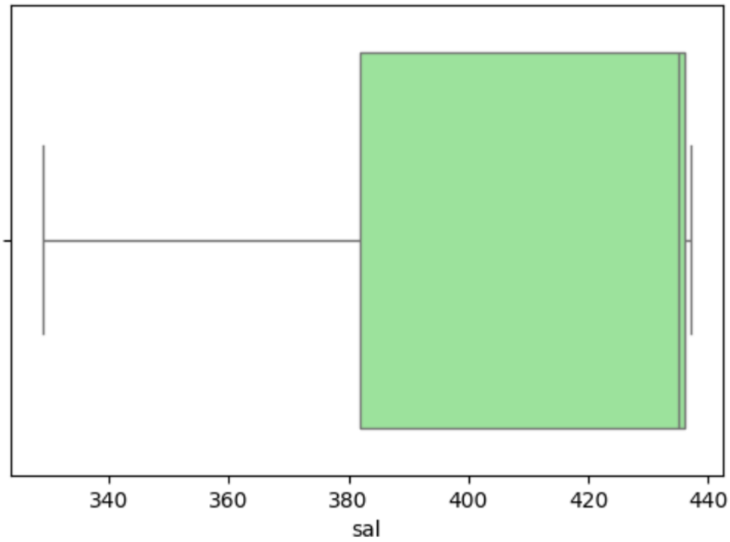
```
sns.boxplot(x=data_clean,color='lightgreen')
plt.show()
Q1=data_clean.quantile(0.25) Q3
= data_clean.quantile(0.75)
IQR=Q3-Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound=Q3+1.5*IQR
outliers=data_clean[(data_clean<lower_bound)|(data_clean>upper_bound)]
print(f"Number of outliers detected: {len(outliers)}")
print(outliers)
column1='age’
category_counts=data[categorical_column].value_counts()
print(f"\nFrequency of each category in {column1}:")
print(category_counts)
plt.figure(figsize=(6, 3))
category_counts.plot(kind='pie',color='lightblue')
plt.show()
```

Output:

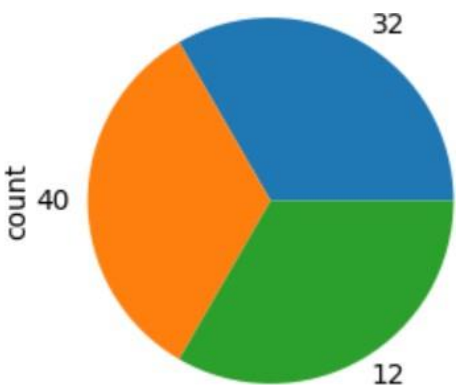
Histogram



Boxplot



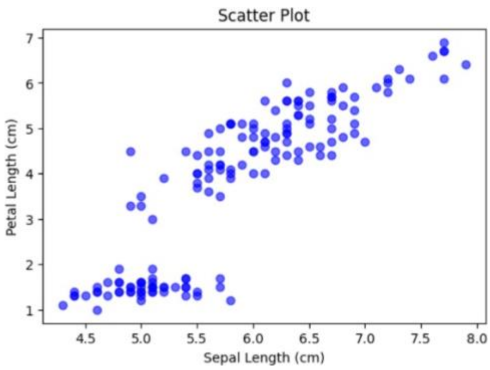
Piechart



Program 2 : Develop a program to Load a dataset with at least two numerical columns (e.g., Iris, Titanic). Plot a scatter plot of two variables and calculate their Pearson correlation coefficient. Write a program to compute the covariance and correlation matrix for a dataset. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations.

```
import numpy as np
import pandas as pd
importseabornassns
importmatplotlib.pyplotasplt
fromsklearn.datasetsimport load_iris

iris= load_iris()
df=pd.DataFrame(data=iris.data,columns=iris.feature_names) x
= df['sepal length (cm)']
y=df['petallength (cm)']

plt.figure(figsize=(8, 6))
plt.scatter(x,y,color='blue',alpha=0.6)
plt.title('Scatter Plot')
plt.xlabel('SepalLength(cm)')
plt.ylabel('Petal Length (cm)')
plt.show()

corr=np.corrcoef(x,y)[0, 1]
print(f"Pearsoncorrelationcoefficientis: {corr:.2f}")

cov_mat = df.cov()
print("\nCovarianceMatrix:")
print(cov_mat)

corr_mat = df.corr()
print("\nCorrelationMatrix:")
print(corr_mat)

plt.figure(figsize=(8, 6))
sns.heatmap(corr_mat,annot=True,cmap='coolwarm',fmt='.2f',cbar=True,linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

Output



Pearsoncorrelationcoefficientis:0.87

CovarianceMatrix:

	sepallength(cm)	sepal width(cm)	petallength (cm)\
sepallength(cm)	0.685694	-0.042434	1.274315
sepalwidth(cm)	-0.042434	0.189979	-0.329656
petallength (cm)	1.274315	-0.329656	3.116278
petalwidth (cm)	0.516271	-0.121639	1.295609

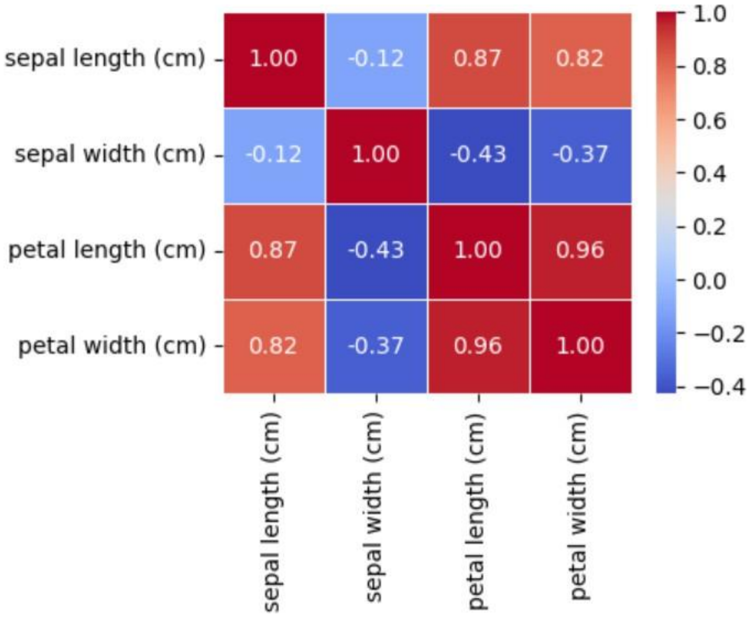
	petalwidth (cm)
sepallength(cm)	0.516271
sepalwidth(cm)	-0.121639
petallength (cm)	1.295609
petalwidth (cm)	0.581006

CorrelationMatrix:

	sepallength(cm)	sepal width(cm)	petallength (cm)\
sepallength(cm)	1.000000	-0.117570	0.871754
sepalwidth(cm)	-0.117570	1.000000	-0.428440
petallength (cm)	0.871754	-0.428440	1.000000
petalwidth (cm)	0.817941	-0.366126	0.962865

	petalwidth (cm)
sepallength(cm)	0.817941
sepalwidth(cm)	-0.366126
petallength (cm)	0.962865
petalwidth (cm)	1.000000

Heapmap

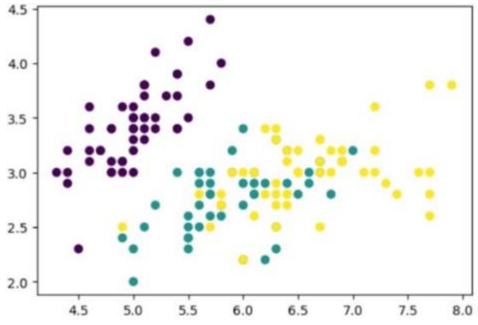


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
1.00
-0.12
0.87
0.82
-0.12
1.00
-0.43
-0.37
0.87
-0.43
1.00
0.96
0.82
-0.37
0.96
1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

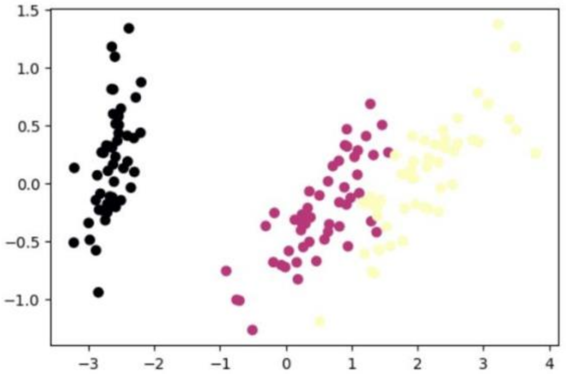
Program3:DevelopaprogramtoimplementPrincipalComponentAnalysis(PCA)for reducing the dimensionality of the Iris dataset from 4 features to 2.

```
importpandasaspd
importmatplotlib.pyplotasplt
fromsklearn.decompositionimportPCA
from sklearn.datasets import load_iris

iris= load_iris()
iris_data=pd.DataFrame(iris.data,columns=iris.feature_names)
print(iris_data.describe())
print(iris_data.head())
X = iris.data
y=iris.target
plt.figure(figsize=(8, 6))
plt.scatter(X[:,0],X[:,1],c=y,cmap='viridis') pca
= PCA(n_components=2)
X_pca=pca.fit_transform(X)
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:,0],X_pca[:,1],c=y,cmap='magma')
plt.show()
```

Output:
BeforeapplyingPrincipalcomponentanalysis



AfterapplyingPrincipalcomponentanalysis



Program4: Develop a program to load the Iris dataset. Implement the k-Nearest Neighbors (k-NN) algorithm forvclassifying flowers based on their features. Split the dataset into training and testing sets and evaluate the model using metrics like accuracy and F1-score. Test it for different values of k (e.g., k=1,3,5) and evaluate the accuracy. Extendthek-NNalgorithmtoassignweightsbasedonthedistanceofneighbors(e.g., $weight$=1/d2). Compare the performance of weighted k-NN and regular k-NN on a synthetic or real-world dataset.

```
importnumpyasnp
importpandasaspd
from sklearn.model_selection import train_test_split
fromsklearn.metricsimportaccuracy_score,f1_score
from sklearn.neighbors import KNeighborsClassifier

iris=pd.read_csv(r'C:\Users\SmartUser\Documents\iris.csv')
print(iris.head())
print(iris.columns)
X=iris.drop('species',axis=1).values y
= iris['species'].values

print(X[:5])
print(y [:5])
X_train,X_test,y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=42)

defknn_model(X_train,X_test,y_train,y_test,k,weighted=False): if
   weighted:
      model=KNeighborsClassifier(n_neighbors=k,weights='distance')
   else:
      model=KNeighborsClassifier(n_neighbors=k,weights='uniform')
   model.fit(X_train, y_train)
   y_pred= model.predict(X_test)
   accuracy=accuracy_score(y_test, y_pred)
   f1=f1_score(y_test,y_pred,average='weighted')
   return accuracy, f1

k_values=[1,3, 5]
results={'k':[],'Regulark-NNAccuracy':[],'Regulark-NNF1Score':[],'Weightedk-NN Accuracy': [],
'Weighted k-NN F1 Score': []}
fork in k_values:
   reg_accuracy,reg_f1=knn_model(X_train,X_test,y_train,y_test,k,weighted=False)
   weighted_accuracy, weighted_f1 = knn_model(X_train, X_test, y_train, y_test, k,
weighted=True)
   results['k'].append(k)
   results['Regular k-NN Accuracy'].append(reg_accuracy)
   results['Regular k-NN F1 Score'].append(reg_f1)
   results['Weightedk-NNAccuracy'].append(weighted_accuracy)
   results['Weighted k-NN F1 Score'].append(weighted_f1)
results_df=pd.DataFrame(results)
print(results_df)
```

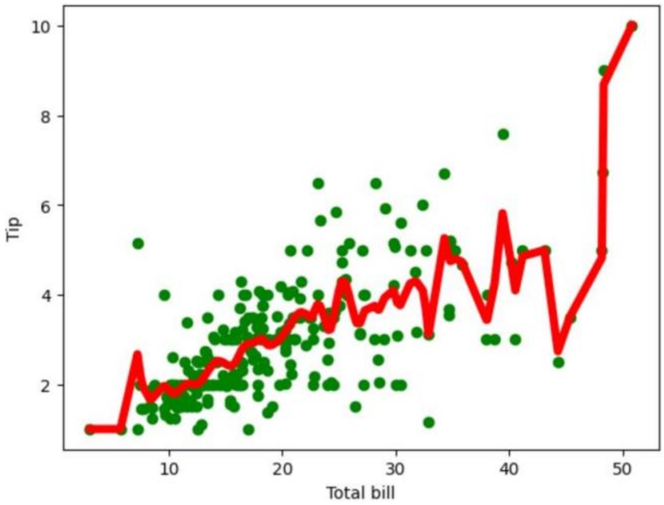
Output:

```
   k  Regular k-NN Accuracy  Regular k-NN F1 Score  Weighted k-NN Accuracy  \
0  1                    1.0                    1.0                     1.0   
1  3                    1.0                    1.0                     1.0   
2  5                    1.0                    1.0                     1.0   

   Weighted k-NN F1 Score  
0                     1.0  
1                     1.0  
2                     1.0  
```

Program5:Implementthenon-parametricLocallyWeightedRegressionalgorithmin ordertofitdatapoints.Selectappropriatedatasetforyourexperimentanddrawgraphs.

```
importmatplotlib.pyplotasplt
import pandas as pd
import numpy as np

defkernel(point,xmat,k):
    m,n = np.shape(xmat)
    weights=np.mat(np.eye((m)))
    for j in range(m):
        diff=point-X[j]
        weights[j,j]=np.exp(diff*diff.T/(-2.0*k**2))
    return weights

deflocalWeight(point,xmat,ymat,k): wei
    = kernel(point,xmat,k)
    W=(X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

deflocalWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred=np.zeros(m)
    for i in range(m):
        ypred[i]=xmat[i]*localWeight(xmat[i],xmat,ymat,k) return
    ypred


data=pd.read_csv(r'C:\Users\SmartUser\Documents\Restuarant.csv') bill
= np.array(data.total_bill)
tip=np.array(data.tip)
mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one=np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
ypred=localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort=X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='green')
ax.plot(xsort[:,1],ypred[SortIndex],color='red',linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

Output



6.Develop a program to demonstrate the working of Linear Regression and Polynomial Regression.UseBostonHousingDatasetforLinearRegressionandAutoMPGDataset(for vehicle fuel efficiency prediction) for Polynomial Regression.

```
importnumpyasnp

importpandasaspd

importmatplotlib.pyplotasplt

fromsklearn.model_selectionimporttrain_test_split

from sklearn.linear_model import LinearRegression

fromsklearn.preprocessingimportPolynomialFeatures,StandardScaler

from sklearn.pipeline import Pipeline

fromsklearn.metricsimportmean_squared_error,r2_score from

sklearn.datasets import fetch_openml


#Part1:LinearRegressionwithBostonHousingDataset #

Load Boston Housing Dataset

boston=fetch_openml(name='boston', version=1)

X_boston=pd.DataFrame(boston.data,columns=boston.feature_names) y_boston

= boston.target.astype(float)


# Split data

X_train_b,X_test_b,y_train_b,y_test_b=train_test_split( X_boston,

    y_boston, test_size=0.2, random_state=42

)


#Trainand evaluateLinear Regression
```

```
lr = LinearRegression()

lr.fit(X_train_b,y_train_b)


#ConvertallcolumnsinX_test_btonumeric,coercingerrors

#Thisstepensuresallfeaturesareinaformatsuitablefornumericaloperations for col in

X_test_b.columns:

    X_test_b[col]=pd.to_numeric(X_test_b[col],errors='coerce')


#Optional:HandlepotentialNaNsintroducedbycoercionifanynon-numericvaluesexisted # If the

dataset is clean from fetch_openml, this might not be strictly necessary,

# but it's a safeguard.

#X_test_b= X_test_b.fillna(X_test_b.mean())# Example:fillNaNs withcolumn mean


#Debugging:CheckdatatypesofX_test_bafterconversion print("Data

types of X_test_b after numeric conversion:")

print(X_test_b.dtypes)



y_pred_b= lr.predict(X_test_b)


print("Boston Housing - Linear Regression Results:")

print(f"MSE:{mean_squared_error(y_test_b,y_pred_b):.2f}")

print(f"R²: {r2_score(y_test_b, y_pred_b):.2f}\n")


# Plot results
```

```
plt.figure(figsize=(8, 6))

plt.scatter(y_test_b,y_pred_b,alpha=0.5)

plt.plot([y_test_b.min(), y_test_b.max()],

        [y_test_b.min(),y_test_b.max()],'k--',lw=2)

plt.xlabel('Actual Prices')

plt.ylabel('PredictedPrices')

plt.title('BostonHousing:ActualvsPredictedPrices')

plt.show()


#Part2:PolynomialRegressionwithAutoMPGDataset #

Load and preprocess Auto MPG data

url="https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"

columns = ['mpg', 'cylinders', 'displacement', 'horsepower',

          'weight','acceleration','model_year','origin','name']

auto_df=pd.read_csv(url,delim_whitespace=True,header=None,names=columns)


#Cleandata

auto_df['horsepower']=pd.to_numeric(auto_df['horsepower'],errors='coerce')

auto_df = auto_df.dropna().reset_index(drop=True)

auto_df=auto_df.drop('name', axis=1)


X_auto=auto_df.drop('mpg',axis=1)

y_auto = auto_df['mpg']


# Split data
```

```
X_train_a,X_test_a,y_train_a,y_test_a=train_test_split(
    X_auto, y_auto, test_size=0.2, random_state=42
)

#CreatePolynomialRegressionpipeline
degree = 2
poly_reg= Pipeline([
    ('poly',PolynomialFeatures(degree=degree)),
    ('scaler', StandardScaler()),
    ('regressor',LinearRegression())
])

# Train and evaluate
poly_reg.fit(X_train_a, y_train_a)
y_pred_a=poly_reg.predict(X_test_a)

print("AutoMPG-PolynomialRegressionResults:") print(f"Degree:
{degree}")
print(f"MSE:{mean_squared_error(y_test_a,y_pred_a):.2f}")
print(f"R²: {r2_score(y_test_a, y_pred_a):.2f}")

# Plot results
plt.figure(figsize=(8,6))
plt.scatter(y_test_a,y_pred_a,alpha=0.5)
plt.plot([y_test_a.min(), y_test_a.max()],
```

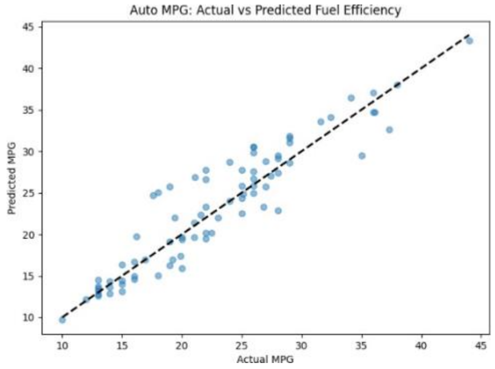
```
        [y_test_a.min(),y_test_a.max()],'k--',lw=2)
plt.xlabel('Actual MPG')
plt.ylabel('PredictedMPG')
plt.title('AutoMPG:ActualvsPredictedFuelEfficiency')
plt.show()
```

OUTPUT:

BostonHousing-LinearRegressionResults:

MSE: 24.29

R^2: 0.67



PolynomialRegressionResults:

Degree: 2

MSE: 7.16

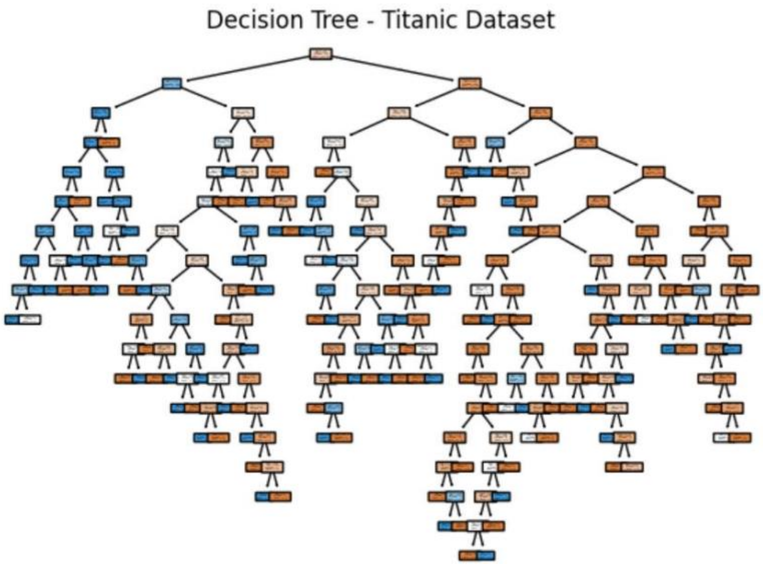
R^2: 0.86

Program7:DevelopaprogramtoloadtheTitanicdataset.Splitthedataintotraining and test sets. Train a decision tree classifier. Visualize the tree structure. Evaluate accuracy, precision, recall, and F1-score.

```
importpandasaspd
import numpy as np
importmatplotlib.pyplotasplt
fromsklearn.model_selectionimporttrain_test_split
fromsklearn.treeimportDecisionTreeClassifier, plot_tree
fromsklearn.metricsimportaccuracy_score,precision_score,recall_score,f1_score from
sklearn.preprocessing import LabelEncoder

titanic_data=pd.read_csv(r'C:\Users\Smart User\Documents\titanic.csv')
titanic_data=titanic_data.dropna(subset=['Survived','Pclass','Sex','Age','Embarked'])

label_encoder=LabelEncoder()
titanic_data['Sex'] = label_encoder.fit_transform(titanic_data['Sex'])
titanic_data['Embarked']=label_encoder.fit_transform(titanic_data['Embarked'])
X = titanic_data[['Pclass', 'Sex', 'Age', 'Fare', 'Embarked']]
y =titanic_data['Survived']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
y_pred=dt_classifier.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)
print(f"Accuracy:{accuracy*100:.2f}%")
print(f"Precision:{precision*100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")
print(f"F1-Score: {f1 * 100:.2f}%")
plt.figure(figsize=(15, 10))
plot_tree(dt_classifier,feature_names=X.columns,class_names=['NotSurvived','Survived'],
filled=True, rounded=True)
plt.title("DecisionTree-TitanicDataset")
plt.show()
```

Output

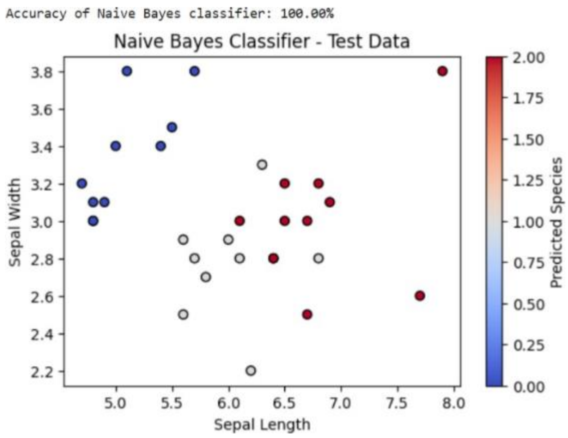
```
Accuracy: 66.43%
Precision: 64.71%
Recall: 52.38%
F1-Score: 57.89%
```



Program8:DevelopaprogramtoimplementtheNaiveBayesianclassifierconsidering Iris dataset for training. Compute the accuracy of the classifier, considering the test data.

```
fromsklearnimportdatasets
fromsklearn.model_selectionimporttrain_test_split
from sklearn.naive_bayes import GaussianNB
fromsklearn.metricsimportaccuracy_score
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris.data
y=iris.target
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred=nb_classifier.predict(X_test)
accuracy=accuracy_score(y_test, y_pred)
print(f'AccuracyofNaiveBayesclassifier:{accuracy*100:.2f}%')
plt.figure(figsize=(8, 6))
plt.scatter(X_test[:,0],X_test[:,1],c=y_pred,cmap='coolwarm',marker='o',edgecolor='k')
plt.title('Naive Bayes Classifier - Test Data')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.colorbar(label='PredictedSpecies')
plt.show()
```

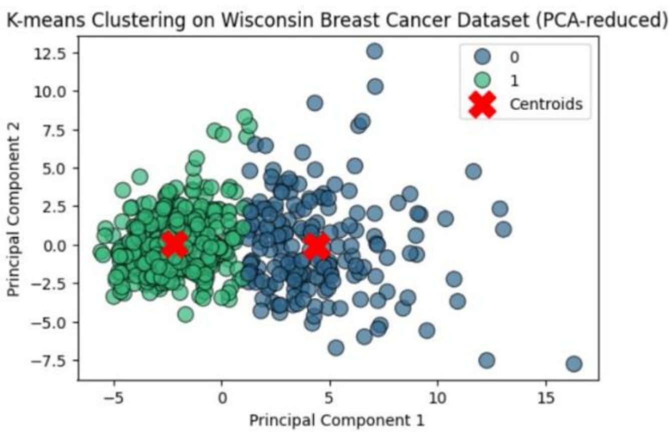
Output



Program9:Developaprogramtoimplementk-meansclusteringusingWisconsinBreast Cancer data set and visualize the clustering result.

```
Importnumpyasnp
importpandasaspd
importmatplotlib.pyplotasplt
fromsklearn.datasetsimportload_breast_cancer
from sklearn.cluster import KMeans
fromsklearn.preprocessingimportStandardScaler from
sklearn.decomposition import PCA
importseabornassns

data= load_breast_cancer()
X = data.data
y=data.target
scaler =
StandardScaler()X_scaled=scaler.f
it_transform(X)
kmeans=KMeans(n_clusters=2,random_state=42)
kmeans.fit(X_scaled)
y_kmeans=kmeans.predict(X_scaled)
pca = PCA(n_components=2)
X_pca=pca.fit_transform(X_scaled)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:,0],y=X_pca[:,1],hue=y_kmeans,palette='viridis',s=100, alpha=0.7,
edgecolor='k')
centroids=pca.transform(kmeans.cluster_centers_)
plt.scatter(centroids[:,0],centroids[:,1],s=300,c='red',marker='X',label='Centroids')
plt.title('K-means Clustering on Wisconsin Breast Cancer Dataset (PCA-reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('PrincipalComponent2')
plt.legend()
plt.show()
```

Output:



Implementthenon-parametricLocallyWeightedRegressionalgorithminordertofitdata points. Select appropriate data set for your experiment and draw graphs.

```
import numpy as np
importmatplotlib.pyplotasplt

deflocally_weighted_regression(X,y,tau=0.1):
    m, n = X.shape
    predictions=np.zeros(m)

    fori in range(m):
        weights=np.exp(-np.sum((X-X[i])**2,axis=1)/(2*tau**2)) W =
        np.diag(weights)# Diagonal matrix of weights X_transpose =
        X.T
        theta=np.linalg.inv(X_transpose@W@X) @ X_transpose@ W@y

        #Predictthevalueforthetestpoint
        predictions[i] = X[i] @ theta

    return predictions

#Generateanon-lineardatasetfordemonstration np.random.seed(42)
X=np.linspace(-3,3,100).reshape(-1,1)#100pointsbetween-3and3 y =
np.sin(X) + 0.3 * np.random.randn(100, 1)# y = sin(x) + noise

#FittheLocallyWeightedRegressionmodel
y_pred=locally_weighted_regression(X,y.flatten(), tau=0.5)

# Plot the results
plt.figure(figsize=(10,6))
plt.scatter(X,y,color='blue',label='Datapoints',s=50)
plt.plot(X,y_pred,color='red',label='LocallyWeightedRegression',linewidth=2)
plt.title("Locally Weighted Regression (LWR) Fit")
plt.xlabel('Feature(X)')
plt.ylabel('Target (y)')
plt.legend()
plt.show()
```