QMP7.1 D/F

ChannabasaveshwaraInstituteofTechnology
(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)
(NAAC Accredited &ISO9001:2015CertifiedInstitution)
NH206(B.H.Road),Gubbi,Tumkur–572216.Karnataka.

Department of Artificial Intelligence and Data Science

MACHINE LEARNING LAB
BCSL606

(CBCS SCHEME)

B.E- VI Semester

Lab Manual 2025-26

Name:_____

USN:_____

Batch:_____Section:_____

.

QMP7.1 D/F

ChannabasaveshwaraInstituteofTechnology
(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)
(NAAC Accredited &ISO9001:2015CertifiedInstitution)
NH206(B.H.Road),Gubbi,Tumkur–572216.Karnataka.

Department of Artificial Intelligence and Data Science

MACHINE LEARNING LAB
BCSL606
(PRACTICAL COMPONENT)

Course Coordinator:

Mrs. Tejaswini S
Assistant Professor
AD Department

HOD:

Dr. Gavisiddappa
Prof. & Head
AD Department

.

Department of Artificial Intelligence & Data Science

SYLLABUS

MACHINE LEARNING LAB
PRACTICAL COMPONENT
[As per Choice Based Credit System (CBCS) scheme]
SEMESTER– VI (AD)

SubjectCode: BCSL606 **CIE Marks: 50**
Hours/Week : 02(02HoursLaboratory) **Test Hours:03**

Using suitable simulation software, demonstrate the operation of the following programs:

Sl.No	Experiments
1.	Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.
2.	Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.
3.	Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.
4.	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.
5.	Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated. a. Label the first 50 points {x1,……,x50} as follows: if (xi ≤ 0.5), then xi ε Class1, else xi ε Class1 b. Classify the remaining points, x51,……,x100 using KNN. Perform this for k=1,2,3,4,5,20,30
6.	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
7.	Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.
8.	Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.
9.	Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.
10.	Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.

Course outcomes (Course Skill Set):

- At the end of the course the student will be able to:

1. Illustrate the principles of multivariate data and apply dimensionality reduction techniques.
2. Demonstrate similarity-based learning methods and perform regression analysis.
3. Develop decision trees for classification and regression problems, and Bayesian models for probabilistic learning.
4. Implement the clustering algorithms to share computing resources..

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are 50 Marks.

The split-up of CIE marks for record/ journal and test are in the ratio 60:40.

● Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.

● Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.

● Total marks scored by the students are scaled down to 30 marks (60% of maximum marks).

● Weightage to be given for neatness and submission of record/write-up on time.

● Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.

● In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.

● The suitable rubrics can be designed to evaluate each student's performance and learning ability.

● The marks scored shall be scaled down to 20 marks (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

● SEE marks for the practical course are 50 Marks.

● SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.

● The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.

● All laboratory experiments are to be included for practical examination.

● (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.

● Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
● Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.
● General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in - 60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero. The minimum duration of SEE is 02 hours

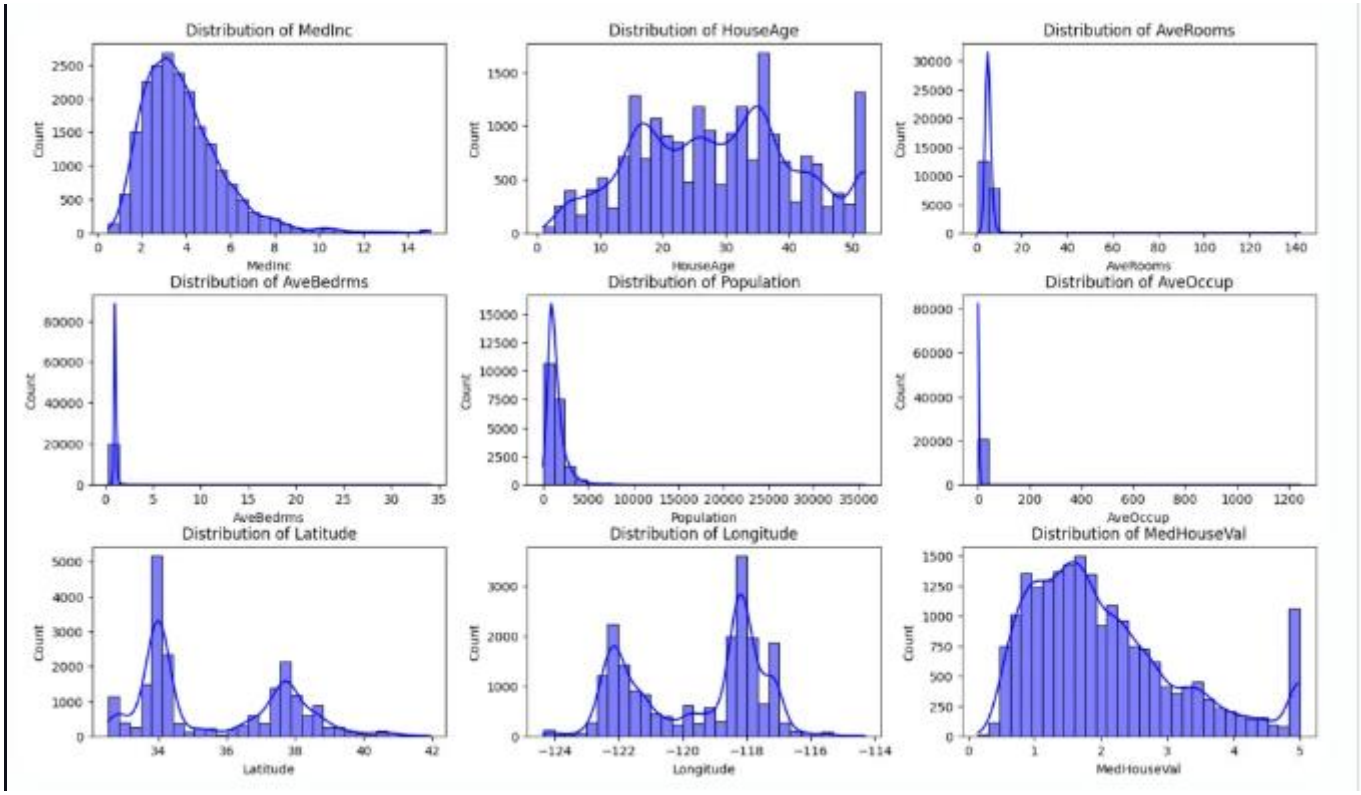
Suggested Learning Resources:
Books:
1. S Sridhar and M Vijayalakshmi, "Machine Learning", Oxford University Press, 2021.
2. M N Murty and Ananthanarayana V S, "Machine Learning: Theory and Practice", Universities Press (India) Pvt. Limited, 2024.
Web links and Video Lectures (e-Resources):
- https://www.drssridhar.com/?page_id=1053
- https://www.universitiespress.com/resources?id=9789393330697
- https://onlinecourses.nptel.ac.in/noc23_cs18/preview

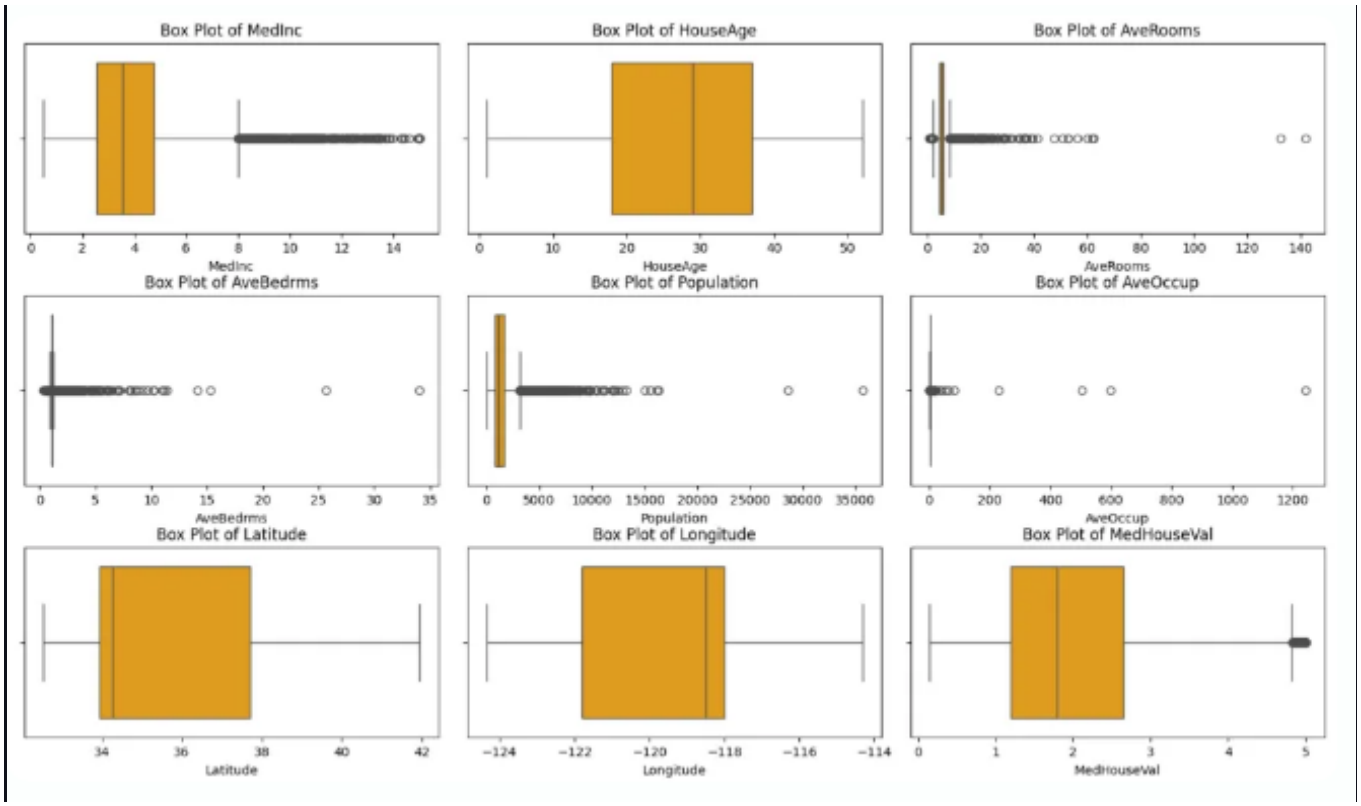
Program 1: Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.

Program:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

# Step 1: Load the California Housing dataset
data = fetch_california_housing(as_frame=True)
housing_df = data.frame

# Step 2: Create histograms for numerical features
numerical_features = housing_df.select_dtypes(include=[np.number]).columns

# Plot histograms
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.histplot(housing_df[feature], kde=True, bins=30, color='blue')
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()

# Step 3: Generate box plots for numerical features
plt.figure(figsize=(15, 10))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(x=housing_df[feature], color='orange')
    plt.title(f'Box Plot of {feature}')
plt.tight_layout()
plt.show()

# Step 4: Identify outliers using the IQR method
print("Outliers Detection:")
outliers_summary = {}
for feature in numerical_features:
    Q1 = housing_df[feature].quantile(0.25)
    Q3 = housing_df[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = housing_df[(housing_df[feature] < lower_bound) | (housing_df[feature] > upper_bound)]
    outliers_summary[feature] = len(outliers)
    print(f"{feature}: {len(outliers)} outliers")
```

Output:


Distribution of MedInc
Distribution of HouseAge
Distribution of AveRooms
Distribution of AveBedrms
Distribution of Population
Distribution of AveOccup
Distribution of Latitude
Distribution of Longitude
Distribution of MedHouseVal
Count
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal

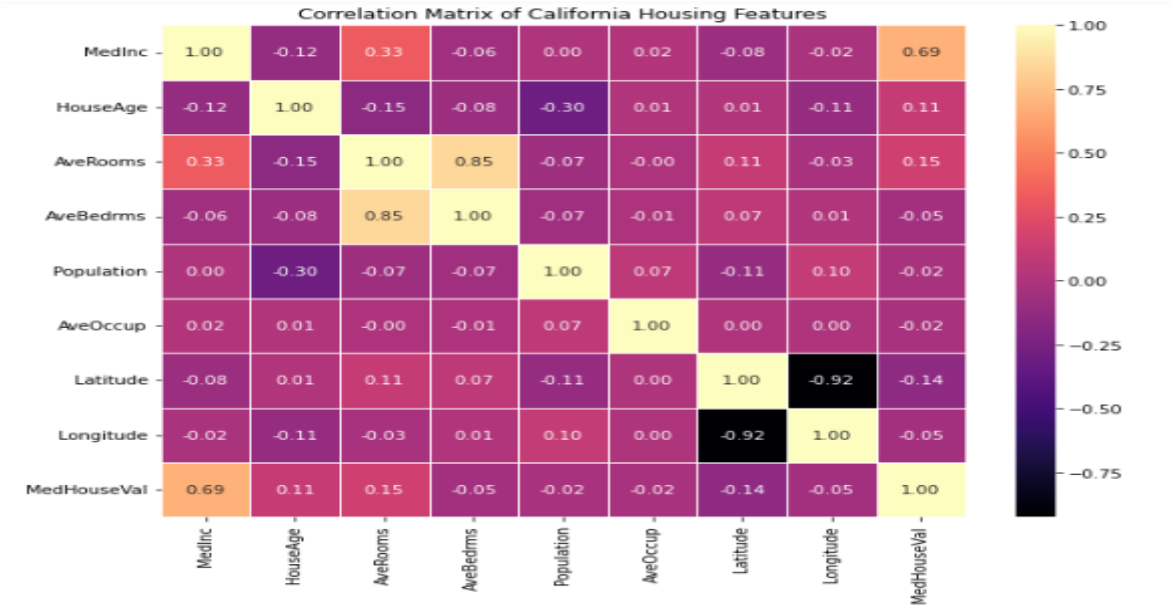

Box Plot of MedInc
Box Plot of HouseAge
Box Plot of AveRooms
Box Plot of AveBedrms
Box Plot of Population
Box Plot of AveOccup
Box Plot of Latitude
Box Plot of Longitude
Box Plot of MedHouseVal
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal

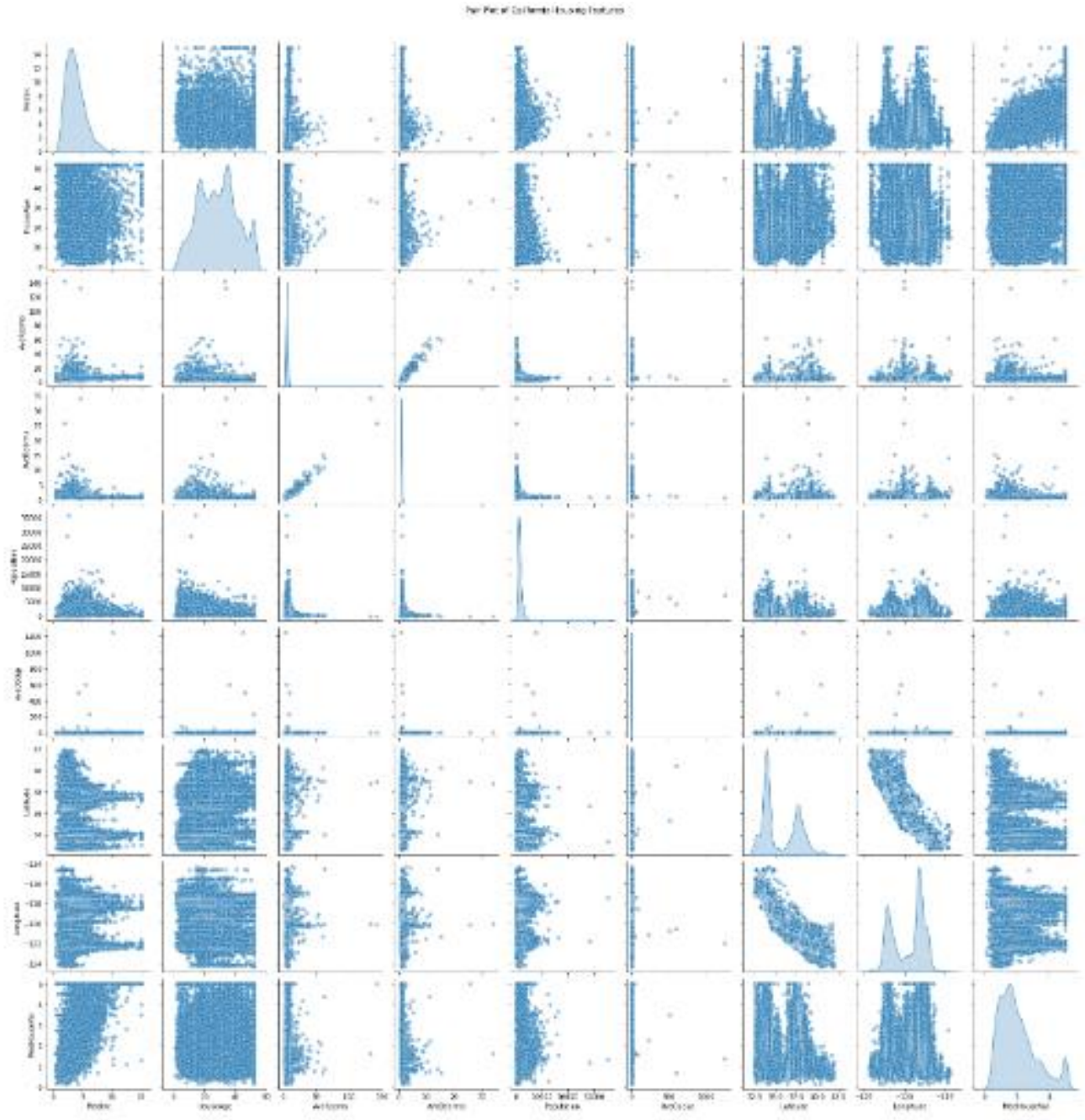
Program 2: Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features. Use California Housing dataset.

Program:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing

# Step 1: Load the California Housing Dataset
california_data = fetch_california_housing(as_frame=True)
data = california_data.frame

# Step 2: Compute the correlation matrix
correlation_matrix = data.corr()

# Step 3: Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of California Housing Features')
plt.show()

# Step 4: Create a pair plot to visualize pairwise relationships
sns.pairplot(data, diag_kind='kde', plot_kws={'alpha': 0.5})
plt.suptitle('Pair Plot of California Housing Features', y=1.02)
plt.show()
```

Output:



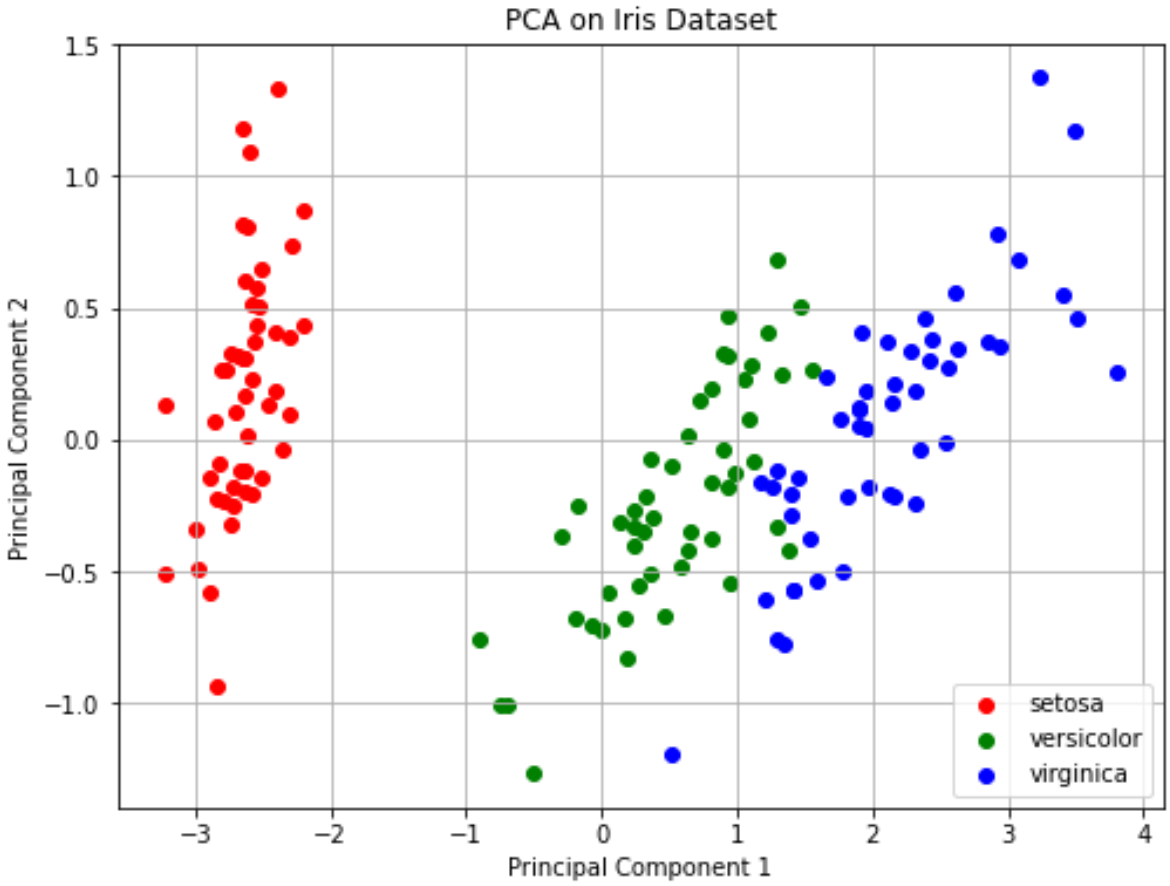


Program 2: Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.

Program:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()          #Loads the Iris dataset into the iris variable.
data = iris.data            #Contains the feature data(measurements of sepal & petal length/width).
labels = iris.target        #Contains the labels (species of Iris flower: setosa,versicolor,virginica).
label_names = iris.target_names     #Contains the names of the labels.

# Convert to a DataFrame for better visualization
iris_df = pd.DataFrame(data, columns=iris.feature_names)

# Perform PCA to reduce dimensionality to 2
pca = PCA(n_components=2)
data_reduced = pca.fit_transform(data)

# Create a DataFrame for the reduced data
reduced_df = pd.DataFrame(data_reduced, columns=['Principal Component 1', 'Principal Component 2'])
reduced_df['Label'] = labels

# Plot the reduced data
plt.figure(figsize=(8, 6))
colors = ['r', 'g', 'b']
for i, label in enumerate(np.unique(labels)):                #Gets the unique labels (0, 1, 2).
    plt.scatter(
        reduced_df[reduced_df['Label'] == label]['Principal Component 1'],
        reduced_df[reduced_df['Label'] == label]['Principal Component 2'],
        label=label_names[label],
        color=colors[i]
    )

plt.title('PCA on Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid()
plt.show()
```

Output:


PCA on Iris Dataset
Principal Component 2
Principal Component 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4
setosa
versicolor
virginica

Program 4: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples.

Program:

```
import csv

def load_data(filename):
    """Load training data from a CSV file."""
    data = []
    with open(filename, 'r') as csvfile:
        reader = csv.reader(csvfile)
        data = list(reader)  # Convert reader to list
    return data

def find_s_algorithm(training_data):
    """Implements the Find-S algorithm to find the maximally specific hypothesis."""
    num_attributes = len(training_data[0]) - 1  # Last column is the target class

    # Find the first positive example to initialize the hypothesis
    hypothesis = None
    for row in training_data:
        if row[-1].lower() == 'yes':  # Ensure case-insensitive match
            hypothesis = row[:num_attributes]
            break

    if hypothesis is None:
        print("No positive examples found in the dataset.")
        return None

    # Iterate through training examples and update hypothesis
    for row in training_data:
        if row[-1].lower() == 'yes':  # Process only positive examples
            for j in range(num_attributes):
                if row[j] != hypothesis[j]:  # Generalize when values differ
                    hypothesis[j] = '?'

    return hypothesis
# Load dataset (Make sure the CSV file exists in the same directory)
filename = "enjoysport.csv"
data = load_data(filename)

print("\n The Given Training Data Set:\n")
for row in data:
    print(row)

# Remove header row if present
header = data[0]
training_data = data[1:]  # Exclude header
```

```
# Run Find-S algorithm
hypothesis = find_s_algorithm(training_data)

if hypothesis:
    print("\n The Maximally Specific Hypothesis for the given Training Examples:\n")
print(hypothesis)
```

Output1:

```
 The Given Training Data Set:

['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport']
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

 The Maximally Specific Hypothesis for the given Training Examples:

['sunny', 'warm', '?', 'strong', '?', '?']
```

Output2:

```
 The Given Training Data Set:

['Outlook', 'Temperature', 'Humidity', 'Windy', 'PlayTennis']
['Sunny', 'Hot', 'High', 'FALSE', 'No']
['Sunny', 'Hot', 'High', 'TRUE', 'No']
['Overcast', 'Hot', 'High', 'FALSE', 'Yes']
['Rain', 'Cold', 'High', 'FALSE', 'Yes']
['Rain', 'Cold', 'High', 'TRUE', 'No']
['Overcast', 'Hot', 'High', 'TRUE', 'Yes']
['Sunny', 'Hot', 'High', 'FALSE', 'No']

 The Maximally Specific Hypothesis for the given Training Examples:

['?', '?', 'High', '?']
```

Program 5: Develop a program to implement k-Nearest Neighbour algorithm to classify the randomly generated 100 values of x in the range of [0,1]. Perform the following based on dataset generated.

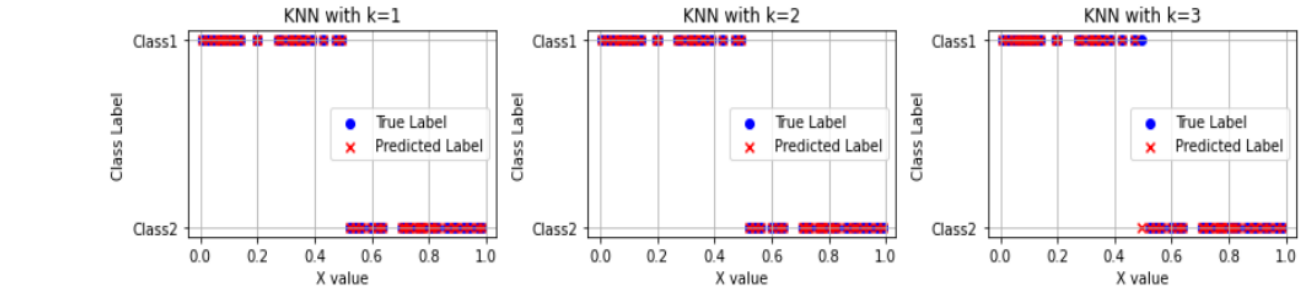
a) Label the first 50 points {x1,……,x50} as follows: if (xi ≤ 0.5), then xi ∈ Class1, else xi ∈ Class2
b) Classify the remaining points, x51,……,x100 using KNN. Perform this for k=1,2,3,4,5,20,30

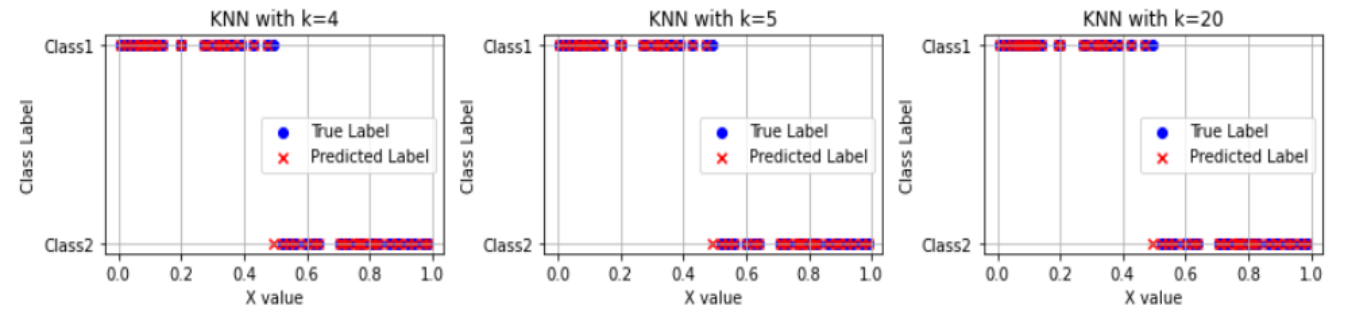
Program:

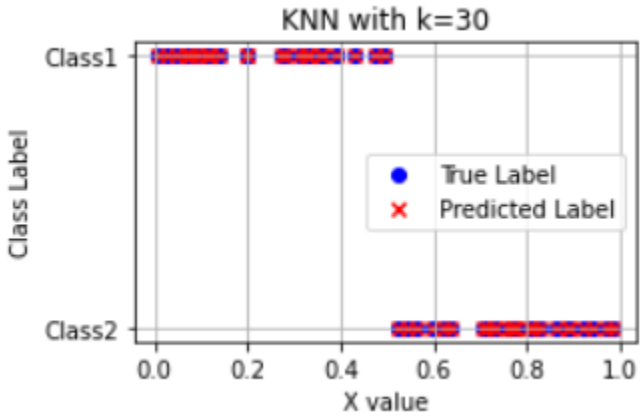
```
import matplotlib
matplotlib.use('TkAgg')  # Use the TkAgg backend for stable display
                #Ensures stable visualization when running in certain environments.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier

# Step 1: Generate 100 random values of x in the range [0, 1]
np.random.seed(42)  #For reproducibility random seed(42)ensures the results are the same each time you run the script.
x_values = np.random.rand(100, 1)  # 100 random values in the range [0,1]

# Step 2: Label the first 50 points as Class1 and the rest as Class2
y_labels = np.array(['Class1' if x <= 0.5 else 'Class2' for x in x_values.flatten()])

# Split into training and testing sets
X_train = x_values[:50]  # First 50 points are used for training.
y_train = y_labels[:50]  # First 50 labels
X_test = x_values[50:]   # Remaining 50 points are used for testing.
y_test = y_labels[50:]   # Remaining 50 labels

# Step 3: Classify using KNN for different k values
k_values = [1, 2, 3, 4, 5, 20, 30]
plt.figure(figsize=(12, 8))
for i, k in enumerate(k_values, 1):    # Initialize the k-NN classifier with the current k value
   knn = KNeighborsClassifier(n_neighbors=k)
   # Fit the model on the training data
   knn.fit(X_train, y_train)
   # Predict the labels for the test set
   y_pred = knn.predict(X_test)
   # Plot the decision boundary and the points
   plt.subplot(3, 3, i)
   plt.scatter(X_test, y_test, color='blue', label='True Label')
   plt.scatter(X_test, y_pred, color='red', marker='x', label='Predicted Label')
   plt.title(f"KNN with k={k}")
```

```
plt.xlabel("X value")
plt.ylabel("Class Label")
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.show()
# Step 4: Evaluate classification accuracy for each k value
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    accuracy = knn.score(X_test, y_test)   # Computes the accuracy of the model.
    print(f"Accuracy for k={k}: {accuracy:.2f}") #The accuracy is printed for each k.
```

Output:







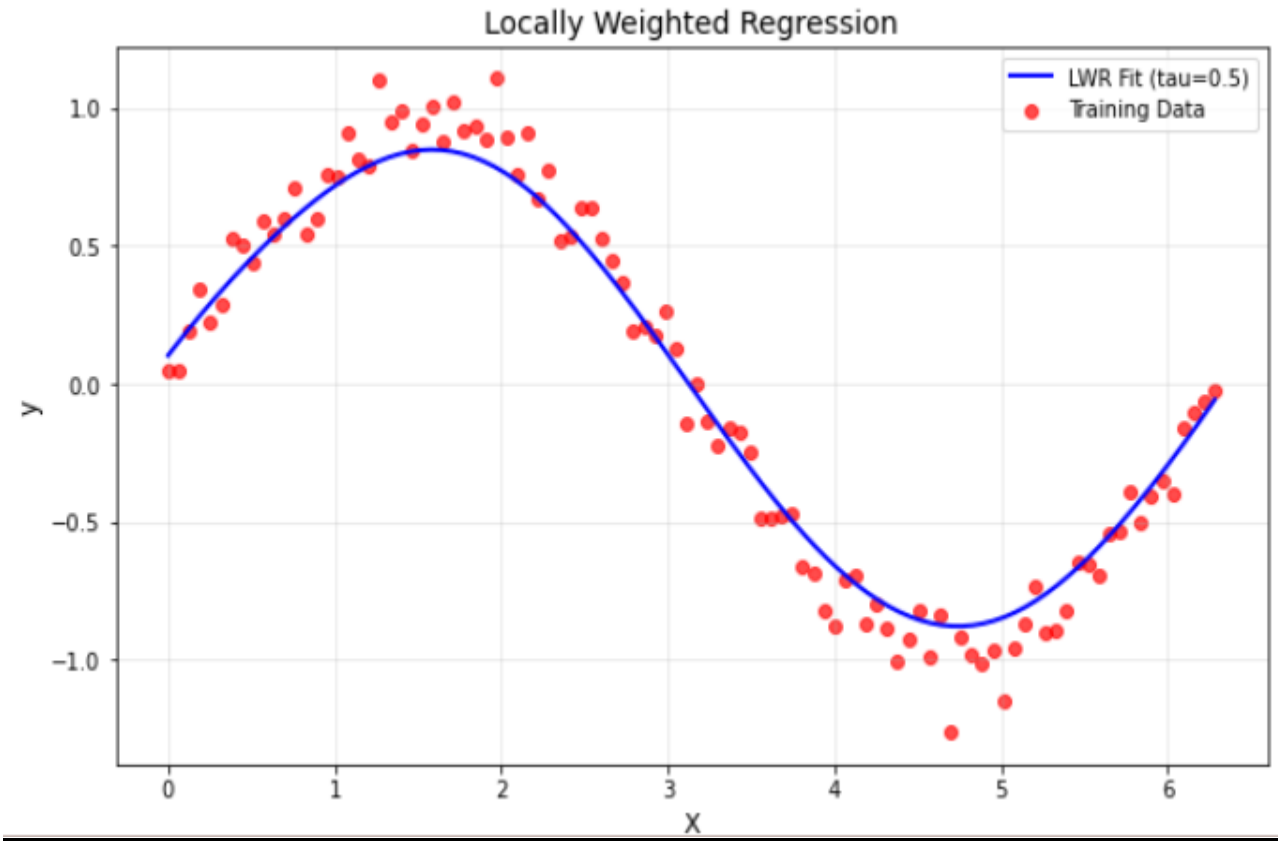
Program 6: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Program:

```
import numpy as np
import matplotlib.pyplot as plt

def gaussian_kernel(x, xi, tau):
    return np.exp(-np.sum((x - xi) ** 2) / (2 * tau ** 2))

def locally_weighted_regression(x, X, y, tau):
    m = X.shape[0]
    weights = np.array([gaussian_kernel(x, X[i], tau) for i in range(m)])
    W = np.diag(weights)
    X_transpose_W = X.T @ W
    theta = np.linalg.inv(X_transpose_W @ X) @ X_transpose_W @ y
    return x @ theta

np.random.seed(42)
X = np.linspace(0, 2 * np.pi, 100)
y = np.sin(X) + 0.1 * np.random.randn(100)
X_bias = np.c_[np.ones(X.shape), X]

x_test = np.linspace(0, 2 * np.pi, 200)
x_test_bias = np.c_[np.ones(x_test.shape), x_test]
tau = 0.5
y_pred = np.array([locally_weighted_regression(xi, X_bias, y, tau) for xi in x_test_bias])

plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='red', label='Training Data', alpha=0.7)
plt.plot(x_test, y_pred, color='blue', label=f'LWR Fit (tau={tau})', linewidth=2)
plt.xlabel('X', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title('Locally Weighted Regression', fontsize=14)
```

```
plt.legend(fontsize=10)
plt.grid(alpha=0.3)
plt.show()
```

Output:



Locally Weighted Regression Algorithm

Regression:

 Regression is a technique from statistics that is used to predict values of a desired target quantity when the target quantity is continuous.

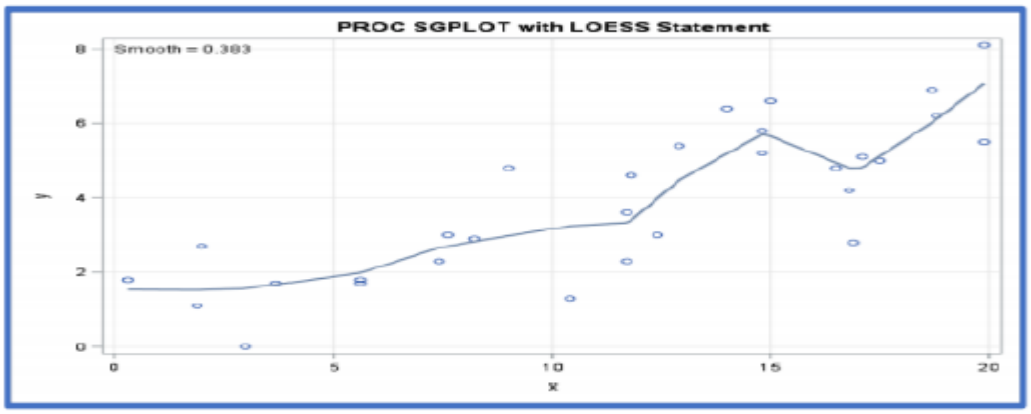
 In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.

 y is called the dependent variable.

 x is called the independent variable.

Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



Lowess Algorithm:

 Locally weighted regression is a very powerful nonparametric model used in statistical learning.

 Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.

 The weights are given by a kernel function (k or w) which can be chosen arbitrarily

Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

6. Prediction = x0*β:

Program 7: Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.

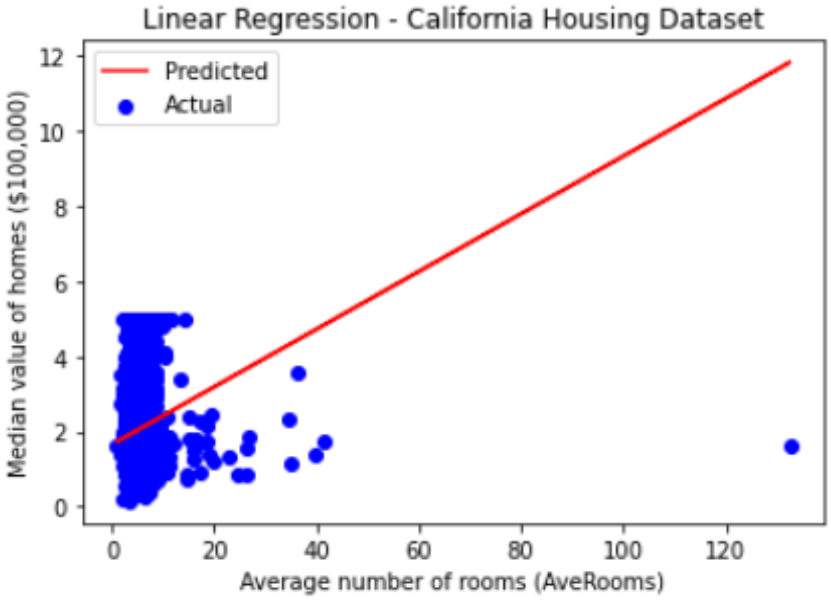
Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score

def linear_regression_california():
    housing = fetch_california_housing(as_frame=True)
    X = housing.data[["AveRooms"]]
    y = housing.target

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

    model = LinearRegression()
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    plt.scatter(X_test, y_test, color="blue", label="Actual")
    plt.plot(X_test, y_pred, color="red", label="Predicted")
    plt.xlabel("Average number of rooms (AveRooms)")
    plt.ylabel("Median value of homes ($100,000)")
    plt.title("Linear Regression - California Housing Dataset")
    plt.legend()
    plt.show()

    print("Linear Regression - California Housing Dataset")
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
    print("R^2 Score:", r2_score(y_test, y_pred))


def polynomial_regression_auto_mpg():
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
    column_names = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration",
```

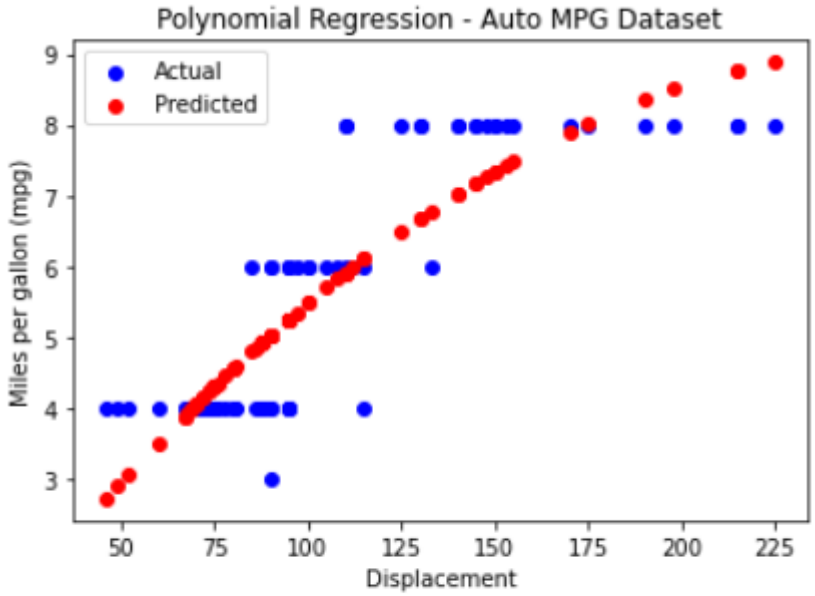
```
"model_year", "origin"]
    data = pd.read_csv(url, sep='\s+', names=column_names, na_values="?")
    data = data.dropna()

    X = data["displacement"].values.reshape(-1, 1)
    y = data["mpg"].values

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

    poly_model = make_pipeline(PolynomialFeatures(degree=2), StandardScaler(),
LinearRegression())
    poly_model.fit(X_train, y_train)

    y_pred = poly_model.predict(X_test)

    plt.scatter(X_test, y_test, color="blue", label="Actual")
    plt.scatter(X_test, y_pred, color="red", label="Predicted")
    plt.xlabel("Displacement")
    plt.ylabel("Miles per gallon (mpg)")
    plt.title("Polynomial Regression - Auto MPG Dataset")
    plt.legend()
    plt.show()

    print("Polynomial Regression - Auto MPG Dataset")
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
    print("R^2 Score:", r2_score(y_test, y_pred))
if __name__ == "__main__":
    print("Demonstrating Linear Regression and Polynomial Regression\n")
    linear_regression_california()
    polynomial_regression_auto_mpg()
```

Output:

```
Demonstrating Linear Regression and Polynomial Regression
```




```
Linear Regression - California Housing Dataset
Mean Squared Error: 1.2923314440807299
R^2 Score: 0.013795337532284901
```




```
Polynomial Regression - Auto MPG Dataset
Mean Squared Error: 0.7431490557205861
R^2 Score: 0.7505650609469626
```

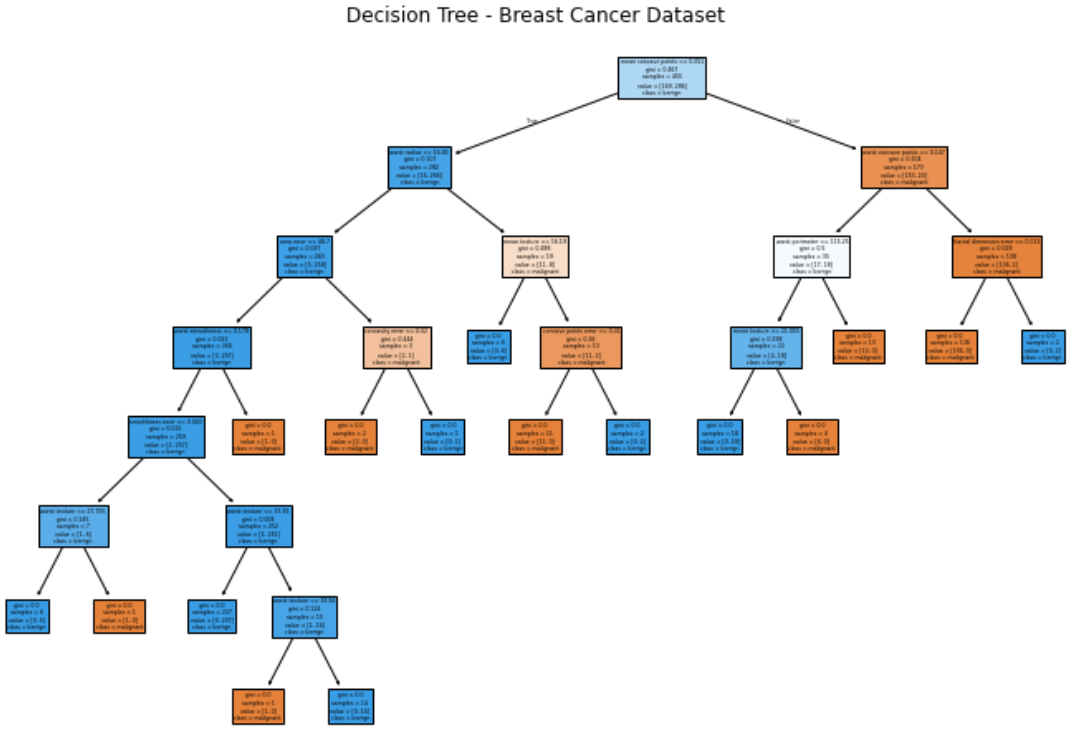
Program 8: Develop a program to demonstrate the working of the decision tree algorithm. Use Breast Cancer Data set for building the decision tree and apply this knowledge to classify a new sample.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree

data = load_breast_cancer()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
new_sample = np.array([X_test[0]])
prediction = clf.predict(new_sample)

prediction_class = "Benign" if prediction == 1 else "Malignant"
print(f"Predicted Class for the new sample: {prediction_class}")

plt.figure(figsize=(12,8))
tree.plot_tree(clf, filled=True, feature_names=data.feature_names, class_names=data.target_names)
plt.title("Decision Tree - Breast Cancer Dataset")
plt.show()
```

Output:

```
Model Accuracy: 94.74%
Predicted Class for the new sample: Benign
```


Decision Tree - Breast Cancer Dataset

Program 9: Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.

Program:

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

import numpy as np
from sklearn.datasets import fetch_olivetti_faces
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt

data = fetch_olivetti_faces(shuffle=True, random_state=42)
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=1))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

cross_val_accuracy = cross_val_score(gnb, X, y, cv=5, scoring='accuracy')
print(f'\nCross-validation accuracy: {cross_val_accuracy.mean() * 100:.2f}%')

fig, axes = plt.subplots(3, 5, figsize=(12, 8))
for ax, image, label, prediction in zip(axes.ravel(), X_test, y_test, y_pred):
    ax.imshow(image.reshape(64, 64), cmap=plt.cm.gray)
    ax.set_title(f"True: {label}, Pred: {prediction}")
    ax.axis('off')

plt.show()
```

Output:

```
Accuracy: 80.83%

Classification Report:
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      1.00      1.00         2
           2       0.33      0.67      0.44         3
           3       1.00      0.00      0.00         5
           4       1.00      0.50      0.67         4
           5       1.00      1.00      1.00         2
           7       1.00      0.75      0.86         4
           8       1.00      0.67      0.80         3
           9       1.00      0.75      0.86         4
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         1
          12       0.40      1.00      0.57         4
          13       1.00      0.80      0.89         5
          14       1.00      0.40      0.57         5
          15       0.67      1.00      0.80         2
          16       1.00      0.67      0.80         3
          17       1.00      1.00      1.00         3
          18       1.00      1.00      1.00         3
          19       0.67      1.00      0.80         2
          20       1.00      1.00      1.00         3
          21       1.00      0.67      0.80         3
          22       1.00      0.60      0.75         5
          23       1.00      0.75      0.86         4
          24       1.00      1.00      1.00         3
          25       1.00      0.75      0.86         4
          26       1.00      1.00      1.00         2
          27       1.00      1.00      1.00         5
          28       0.50      1.00      0.67         2
          29       1.00      1.00      1.00         2
          30       1.00      1.00      1.00         2
          31       1.00      0.75      0.86         4
          32       1.00      1.00      1.00         2
          34       0.25      1.00      0.40         1
          35       1.00      1.00      1.00         5
          36       1.00      1.00      1.00         3
          37       1.00      1.00      1.00         1
          38       1.00      0.75      0.86         4
          39       0.50      1.00      0.67         5

    accuracy                           0.81       120
   macro avg       0.89      0.85      0.83       120
```

```
weighted avg       0.91      0.81      0.81       120


Confusion Matrix:
[[2 0 0 ... 0 0 0]
 [0 2 0 ... 0 0 0]
 [0 0 2 ... 0 0 1]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 3 0]
 [0 0 0 ... 0 0 5]]

Cross-validation accuracy: 87.25%
```

Program 10: Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.

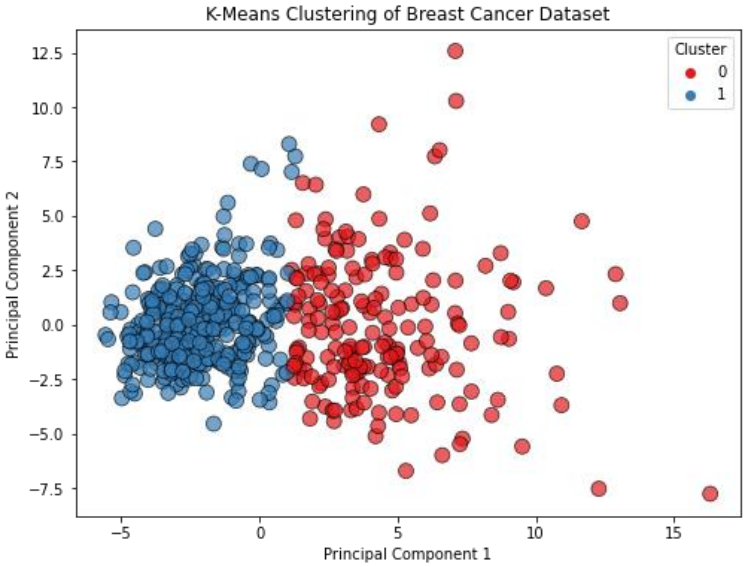
Program:

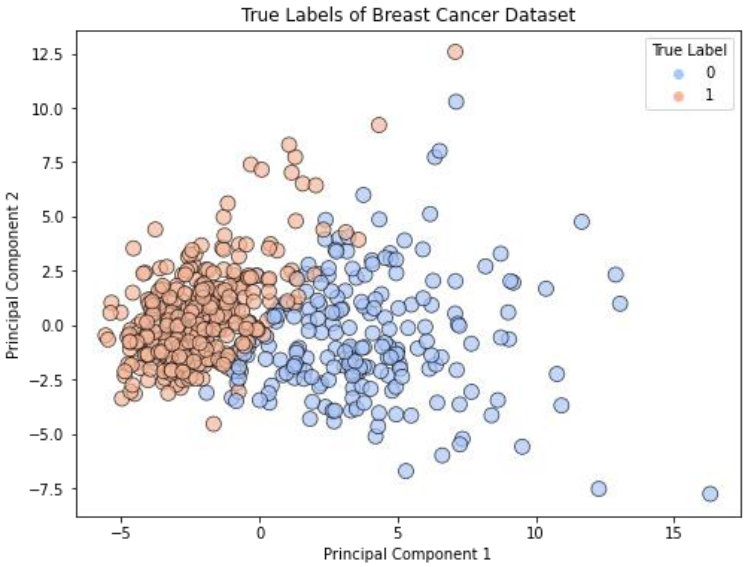
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, classification_report

data = load_breast_cancer()
X = data.data
y = data.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=2, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)

print("Confusion Matrix:")
print(confusion_matrix(y, y_kmeans))
print("\nClassification Report:")
print(classification_report(y, y_kmeans))

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
df['Cluster'] = y_kmeans
df['True Label'] = y

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=100, edgecolor='black',
alpha=0.7)
plt.title('K-Means Clustering of Breast Cancer Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="Cluster")
```

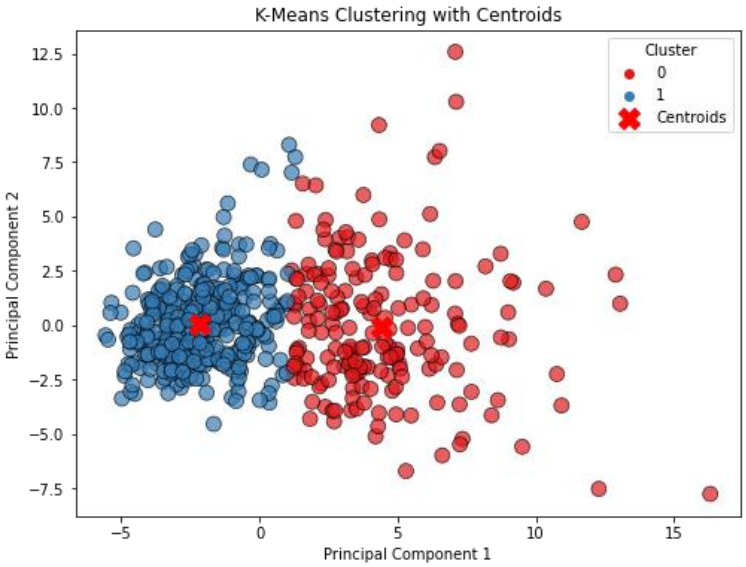
```
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='True Label', palette='coolwarm', s=100,
edgecolor='black', alpha=0.7)
plt.title('True Labels of Breast Cancer Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="True Label")
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=100, edgecolor='black',
alpha=0.7)
centers = pca.transform(kmeans.cluster_centers_)
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X', label='Centroids')
plt.title('K-Means Clustering with Centroids')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="Cluster")
plt.show()
```

Output:

```
Confusion Matrix:
[[175  37]
 [ 13 344]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.83      0.88       212
           1       0.90      0.96      0.93       357

    accuracy                           0.91       569
   macro avg       0.92      0.89      0.90       569
weighted avg       0.91      0.91      0.91       569
```


K-Means Clustering of Breast Cancer Dataset
Cluster
0
1
Principal Component 2
Principal Component 1
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−5
0
5
10
15


True Labels of Breast Cancer Dataset
True Label
0
1
Principal Component 2
Principal Component 1
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−5
0
5
10
15


K-Means Clustering with Centroids
Cluster
0
1
Centroids
Principal Component 2
Principal Component 1
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−5
0
5
10
15