LABORATORY MANUAL

Semester : VI

Subject : Machine Learning Lab

Subject Code : BCM601

MACHINE LEARNING		Semester	**VI**
Course Code	**BCM601**	CIE Marks	50
Teaching Hours/Week (L: T:P: S)	3:0:2:0	SEE Marks	50
Total Hours of Pedagogy	40 hours Theory + 8-10 Lab slots	Total Marks	100
Credits	04	Exam Hours	100
Examination type (SEE)	Theory/Practical		

Course objectives:

- To understand the basic theory underlying machine learning, types, and the process.
- To become familiar with data and visualize univariate, bivariate, and multivariate data using statistical techniques and dimensionality reduction.
- To understand various machine learning algorithms such as similarity-based learning, regression, decision trees, and clustering.
- To familiarize with learning theories, probability-based models, and reinforcement learning, developing the skills required for decision-making in dynamic environments.

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 Neighbor 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 $\{x1, \dots, x50\}$ as follows: if $(x_i \leq 0.5)$, then $x_i \in Class_1$, else $x_i \in Class_1$ b. Classify the remaining points, $x51, \dots, x100$ 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:

- Demonstrate the need for machine learning, its relationship to other fields, and different types of machine learning.
- Illustrate the fundamental principles of multivariate data and apply dimensionality reduction techniques.
- Apply similarity-based learning methods and perform linear, polynomial regression analysis.
- Apply decision trees for classification and regression problems, and Bayesian models for probabilistic learning.
- Analyze the clustering algorithms and reinforce their understanding by applying Q-learning for decision making tasks.

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.

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks**.
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

Suggested Learning Resources:

1. Textbooks 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.

Reference Books:

1. Tom M. Mitchell, "Machine Learning", McGraw-Hill Education, 2013.
2. Miroslav Kubat, "An Introduction to Machine Learning", Springer, 2017.

Datasets:

1. https://violetto-rose.github.io/ML/

Installation (Optional; you can use the online version at jupyter.org):

Note: Run terminal as Admin while installation.

1. **JupyterLab:**

   Install JupyterLab with pip:

   ```
   pip install jupyterlab
   ```

   Once installed, launch JupyterLab in your desired folder using with:

   ```
   jupyter lab
   ```

2. **Or if you prefer to use Jupyter Notebook:**

   Install the classic Jupyter Notebook with:

   ```
   pip install notebook
   ```

   To run the notebook:

   ```
   jupyter notebook
   ```

3. **Homebrew (MacOS and Linux):**

   Homebrew is a package manager for macOS and Linux. You can use it to install Jupyter by running:

   ```
   brew install jupyterlab
   ```

4. **Troubleshooting:**

   Use Virtual Environment

   a) Create a new virtual environment:

   ```
   python -m venv jupyter_env
   ```

   b) Activate it:

   ```
   jupyter_env\Scripts\activate # For Windows
   ```

   c) Install Jupyter inside the environment:

   ```
   pip install jupyter
   ```

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.

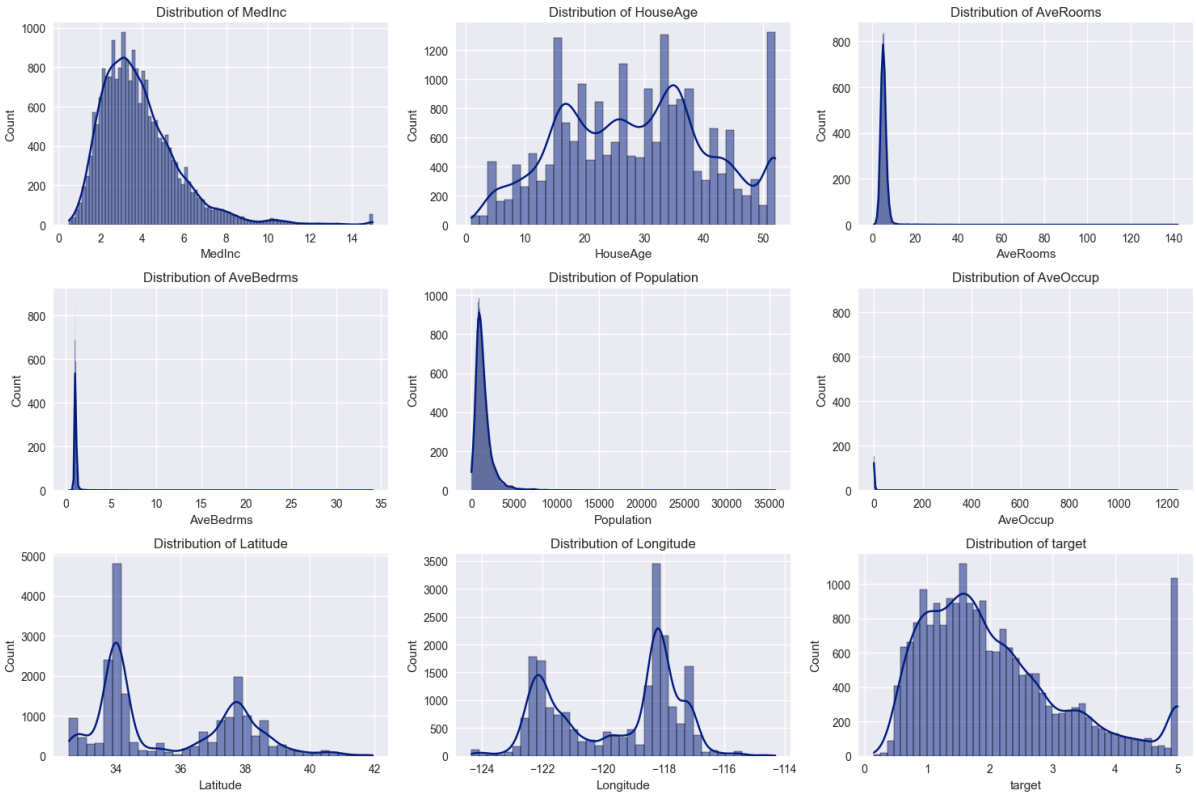
Import necessary libraries and load the dataset:

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


# Load California Housing dataset
california = fetch_california_housing()
df = pd.DataFrame(california.data, columns=california.feature_names)
```

Create histograms for all numerical features:

```
# Set up the figure size and style
plt.style.use('seaborn-v0_8-darkgrid) # Can also use 'default'

fig = plt.figure(figsize=(15, 10))

# Create histograms for each feature
for idx, column in enumerate(df.columns, 1):
    plt.subplot(3, 3, idx)
    sns.histplot(data=df, x=column, kde=True)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')

plt.tight_layout()
plt.show()
```

Output:

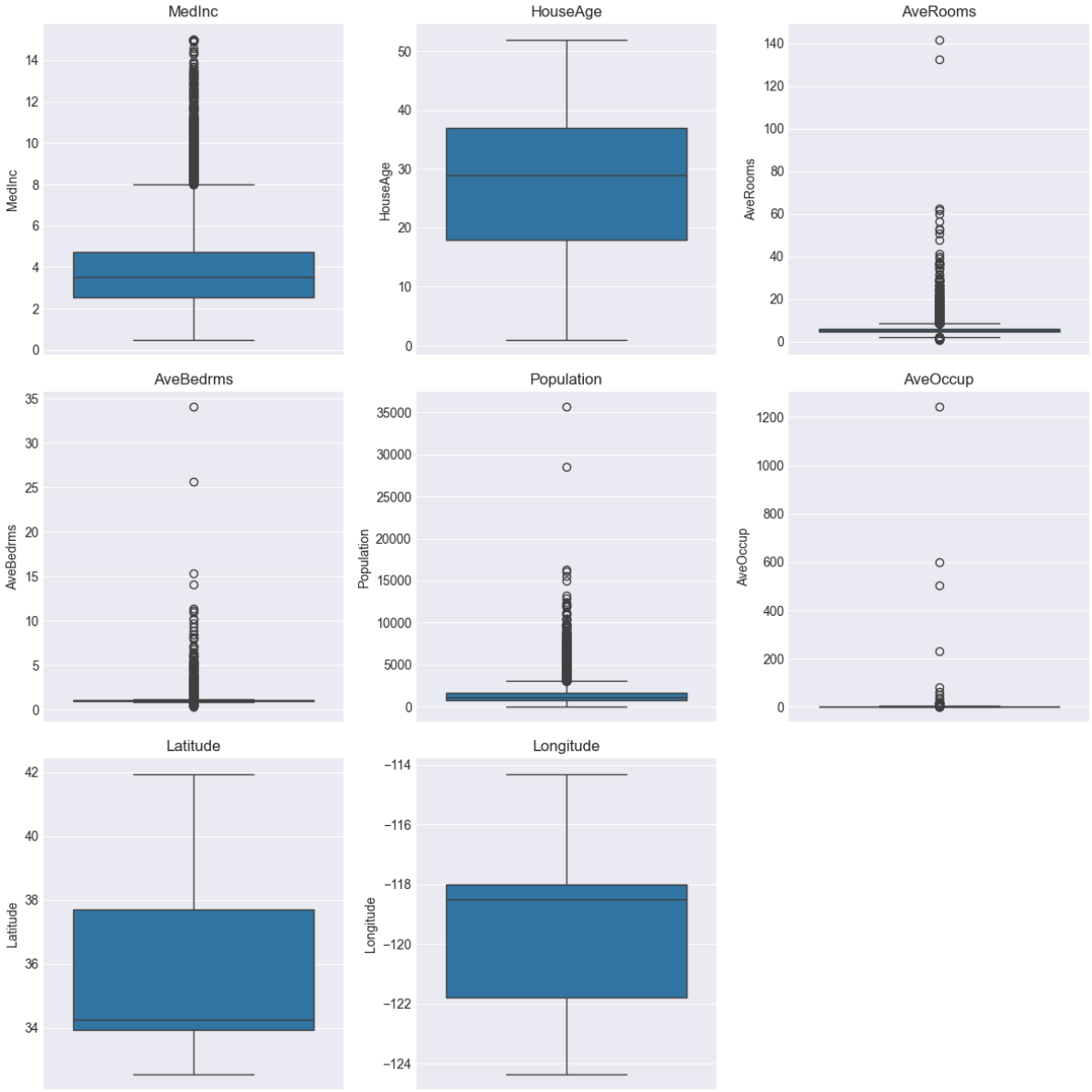


Create box plots for all numerical features:

```
# Function to identify outliers using the Interquartile Range (IQR)
method
def find_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1  # Interquartile range (IQR)
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return column[(column < lower_bound) | (column > upper_bound)]

# Create box plots for all numerical features
plt.figure(figsize=(12, 12))
for i, col in enumerate(df.columns):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(y=df[col])
    plt.title(col)

    # Identify and count outliers in the column
    outliers = find_outliers(df[col])
    print(f"Feature: {col}, Outliers: {len(outliers)}")

plt.tight_layout()
plt.show()
```

Output:

```
Feature: MedInc, Outliers: 681
Feature: HouseAge, Outliers: 0
Feature: AveRooms, Outliers: 511
Feature: AveBedrms, Outliers: 1424
Feature: Population, Outliers: 1196
Feature: AveOccup, Outliers: 711
Feature: Latitude, Outliers: 0
Feature: Longitude, Outliers: 0
```



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.

Import necessary libraries and load the dataset:

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

# Load the dataset
california = fetch_california_housing()
df = pd.DataFrame(california.data, columns=california.feature_names)
df['PRICE'] = california.target
```

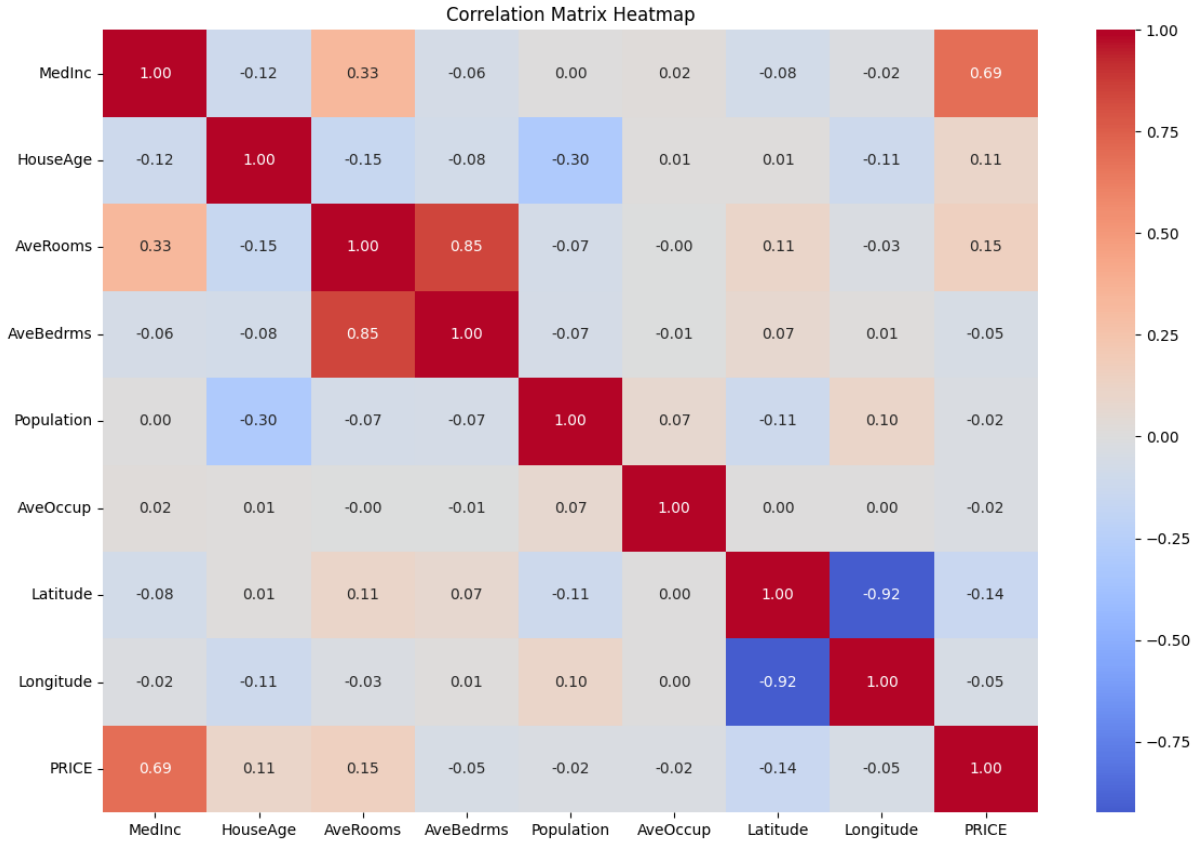
Correlation Matrix Calculation:

```
# Compute the correlation matrix
correlation_matrix = df.corr()
```

Heatmap Visualization:

```
# Create a heatmap of correlations
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix,
            annot=True,          # Show correlation values
            cmap='coolwarm',     # Color scheme
            center=0,            # Center the colormap at 0
            fmt='.2f')          # Round to 2 decimal places
plt.title('Correlation Matrix Heatmap')
plt.tight_layout()
plt.show()
```

Output:



Strong Correlations:

```
# Print strong correlations (absolute value > 0.5)
print("\nStrong Correlations (|correlation| > 0.5):")
for i in range(len(correlation_matrix.columns)):
    for j in range(i+1, len(correlation_matrix.columns)):
        if abs(correlation_matrix.iloc[i,j]) > 0.5:
            print(f"{correlation_matrix.index[i]} -
{correlation_matrix.columns[j]}:
{correlation_matrix.iloc[i,j]:.3f}")
```

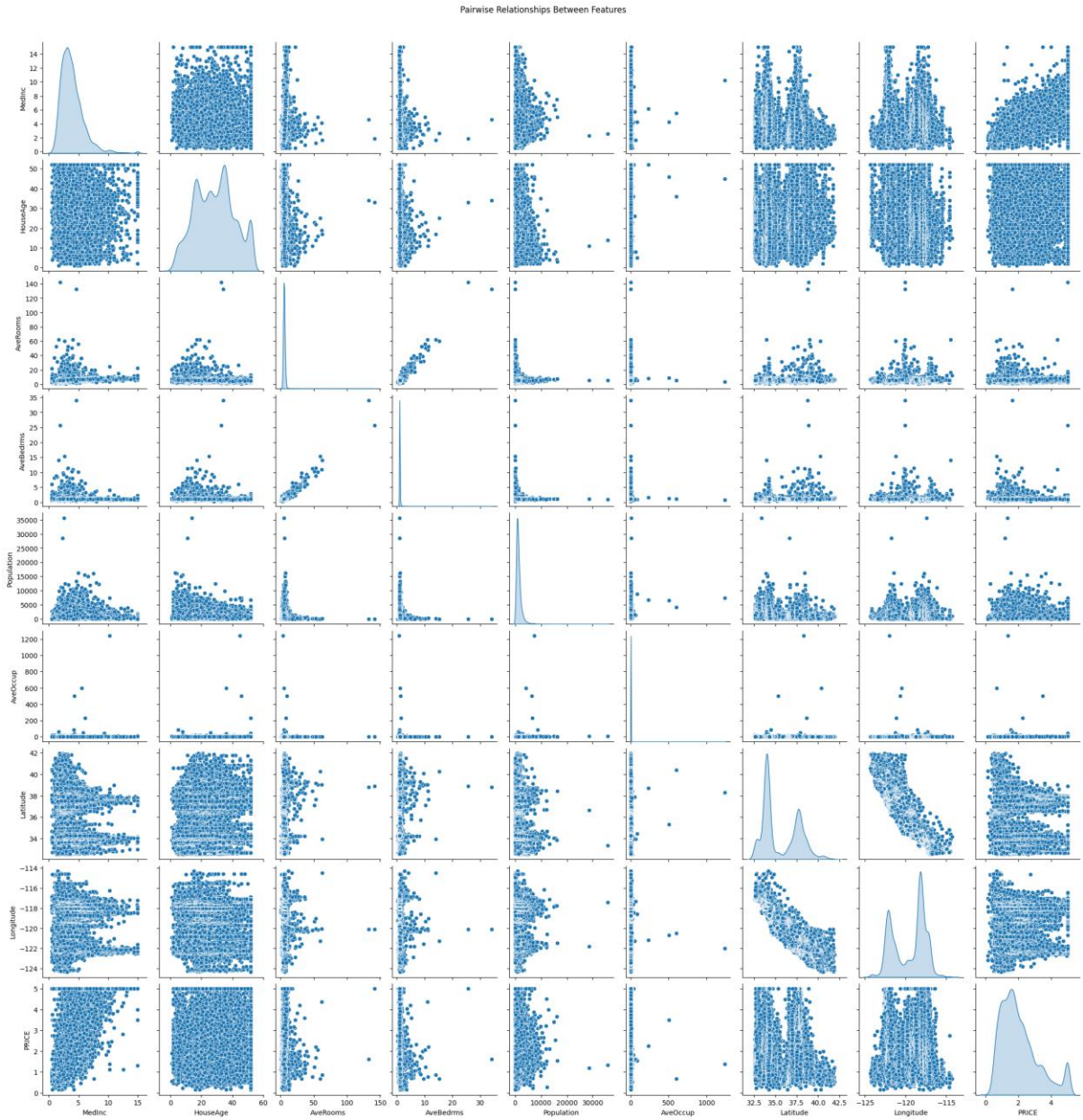
Output:

Strong Correlations (|correlation| > 0.5):
MedInc - PRICE: 0.688
AveRooms - AveBedrms: 0.848
Latitude - Longitude: -0.925

Pair Plot:

```
# Create a pair plot
sns.pairplot(df, diag_kind='kde')
plt.suptitle('Pairwise Relationships Between Features', y=1.02)
plt.show()
```

Output:



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

Import necessary libraries and load the dataset:

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

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

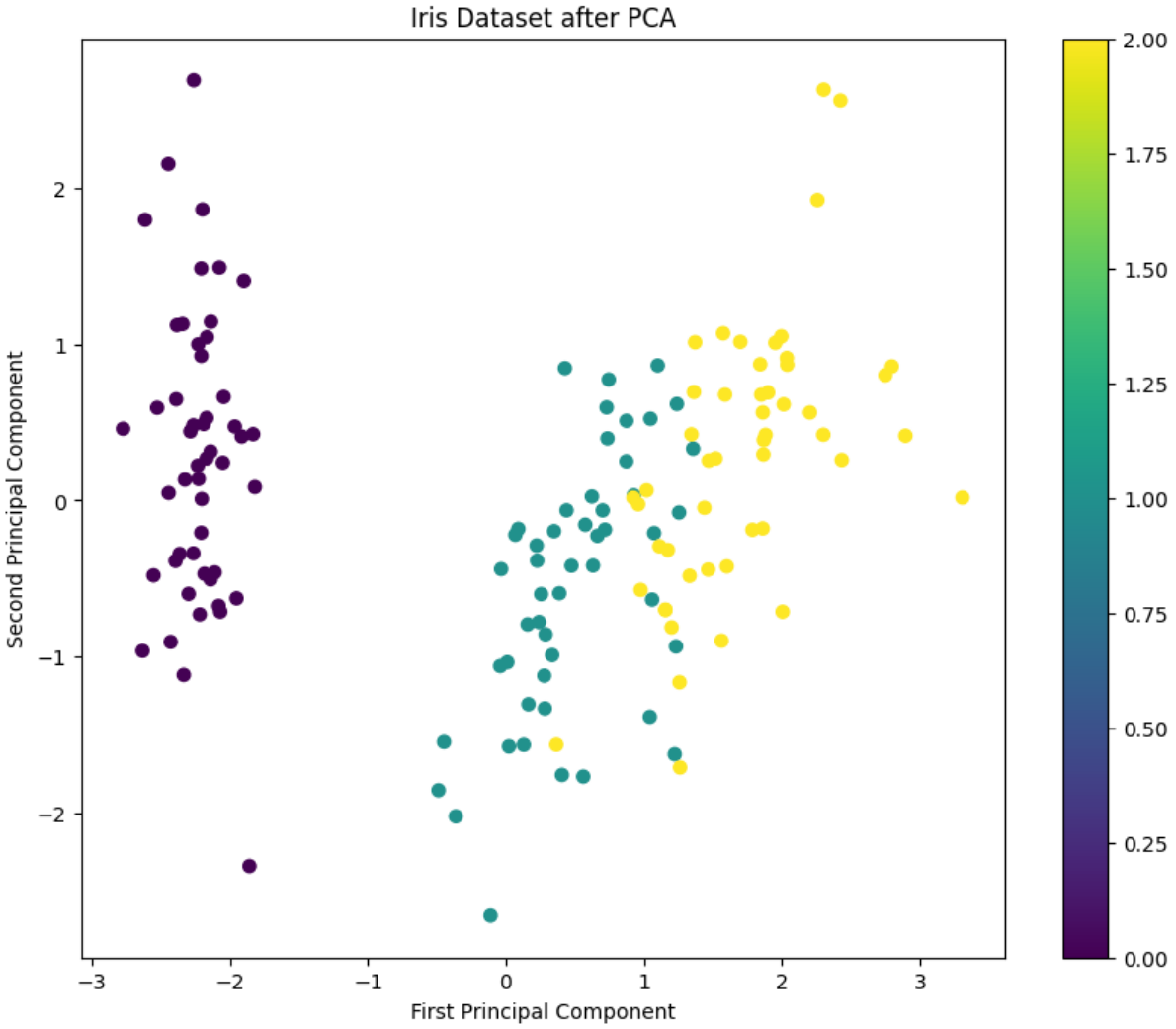
Standardization and PCA Implementation:

```
# Standardize the features (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

Results Visualization:

```
# Create a DataFrame with the transformed data
pca_df = pd.DataFrame(
    data=X_pca,
    columns=['PC1', 'PC2']
)
pca_df['Target'] = y

# Plot the results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(pca_df['PC1'],
                      pca_df['PC2'],
                      c=y,
                      cmap='viridis')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('Iris Dataset after PCA')
plt.colorbar(scatter)
```

Output:


Iris Dataset after PCA
Second Principal Component
First Principal Component
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

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.

```
import csv
import pandas as pd

def find_s_algorithm(data):
    # Initialize hypothesis with the first positive example
    hypothesis = None

    # Process each training example
    for index, row in data.iterrows():
        # Consider only positive examples
        if row['EnjoysMarket'] == 'Yes':  # Assuming 'EnjoysMarket'
is our target
            # For first positive example, initialize hypothesis
            if hypothesis is None:
                hypothesis = row[:-1].copy()  # Copy all attributes
except the last (target)
            # For subsequent positive examples
            else:
                for attr in hypothesis.index:
                    # If attribute values are different, make it
more general
                    if hypothesis[attr] != row[attr]:
                        hypothesis[attr] = '?'

    return hypothesis

df = pd.read_csv('weather_data.csv')
print("Training Data:")
print(df)
print("\nApplying Find-S Algorithm...")

# Apply Find-S algorithm
hypothesis = find_s_algorithm(df)
print("\nFinal Hypothesis:")
print(hypothesis)
```

Output:

```
Training Data:
  Weather Temperature Humidity    Wind EnjoysMarket
0   Sunny        Warm   Normal  Strong          Yes
1   Rainy        Cold     High  Strong           No
2   Sunny        Warm     High    Weak          Yes
3  Cloudy        Warm     High    Weak          Yes
4   Sunny        Cold   Normal    Weak           No

Applying Find-S Algorithm...

Final Hypothesis:
Weather           ?
Temperature    Warm
Humidity          ?
Wind              ?
Name: 0, dtype: object
```

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 $\{x1, \dots, x50\}$ as follows: if $(x\_i \leq 0.5)$, then $x_i \in Class_1$, else $x_i \in Class_1$

b. Classify the remaining points, $x51, \dots, x100$ using KNN. Perform this for k=1,2,3,4,5,20,30

Import necessary libraries:

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
```

Data Generation and Labelling:

```
# Generate and label the data
np.random.seed(42)  # for reproducibility
X = np.random.rand(100, 1)  # Generate 100 random points between 0
and 1

# Label first 50 points
y = np.zeros(100)  # Initialize labels array
for i in range(50):
    if X[i] <= 0.5:
        y[i] = 0  # Class 1
    else:
        y[i] = 1  # Class 2

# Split data into training (first 50) and testing (last 50)
X_train = X[:50]
y_train = y[:50]
X_test = X[50:]
y_test = y[50:]
```

KNN Classification Function:

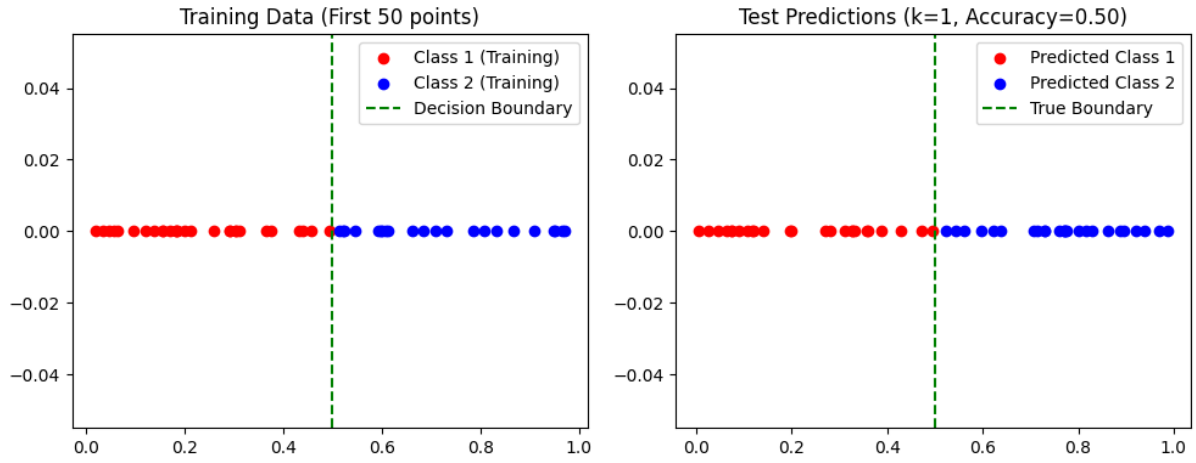
```
# Function to perform kNN classification and visualization
def knn_classify_and_plot(k):
    # Create and train the kNN classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Get predictions
    y_pred = knn.predict(X_test)

    # Calculate accuracy
    accuracy = np.mean(y_pred == y_test)

    # Plotting
    plt.figure(figsize=(10, 4))

    # Plot training data
    plt.subplot(1, 2, 1)
    plt.scatter(X_train[y_train == 0], np.zeros_like(X_train[y_train
== 0]),
                c='red', label='Class 1 (Training)')
    plt.scatter(X_train[y_train == 1], np.zeros_like(X_train[y_train
== 1]),
                c='blue', label='Class 2 (Training)')
    plt.axvline(x=0.5, color='green', linestyle='--',
label='Decision Boundary')
    plt.title(f'Training Data (First 50 points)')
    plt.legend()

    # Plot test data with predictions
    plt.subplot(1, 2, 2)
    plt.scatter(X_test[y_pred == 0], np.zeros_like(X_test[y_pred ==
0]),
                c='red', label='Predicted Class 1')
    plt.scatter(X_test[y_pred == 1], np.zeros_like(X_test[y_pred ==
1]),
                c='blue', label='Predicted Class 2')
    plt.axvline(x=0.5, color='green', linestyle='--', label='True
Boundary')
    plt.title(f'Test Predictions (k={k}, Accuracy={accuracy:.2f})')
    plt.legend()

    plt.tight_layout()
    plt.show()

    return accuracy
```

Visualization for Each k:

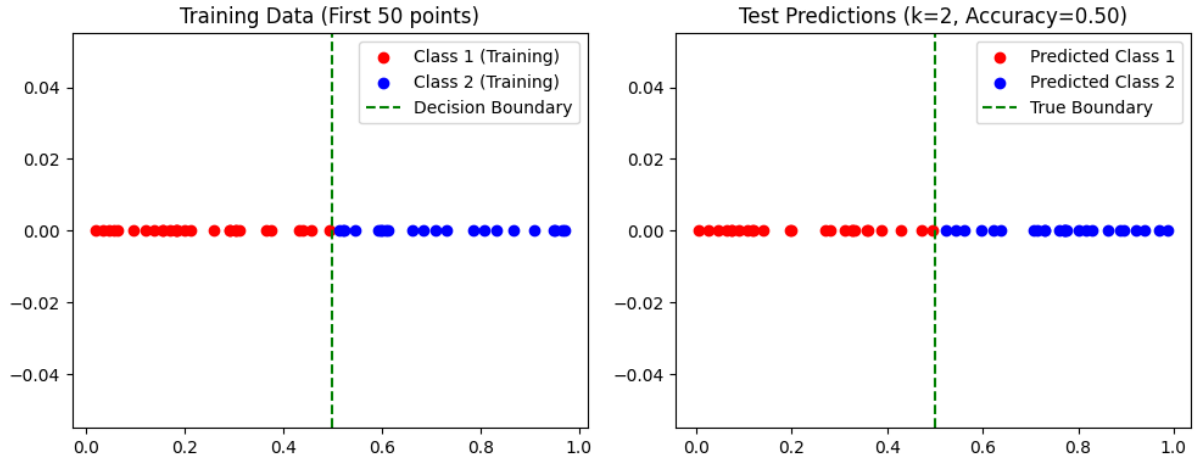
```
# Run kNN for different k values
k_values = [1, 2, 3, 4, 5, 20, 30]
accuracies = []

for k in k_values:
    print(f"\nClassification with k = {k}")
    accuracy = knn_classify_and_plot(k)
    accuracies.append(accuracy)
```

Output:

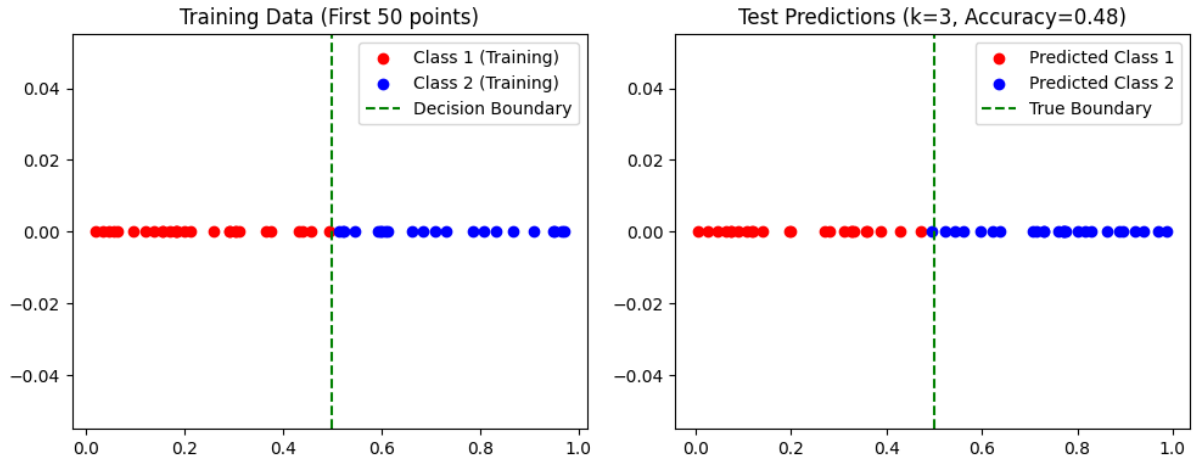
```
Classification with k = 1
```



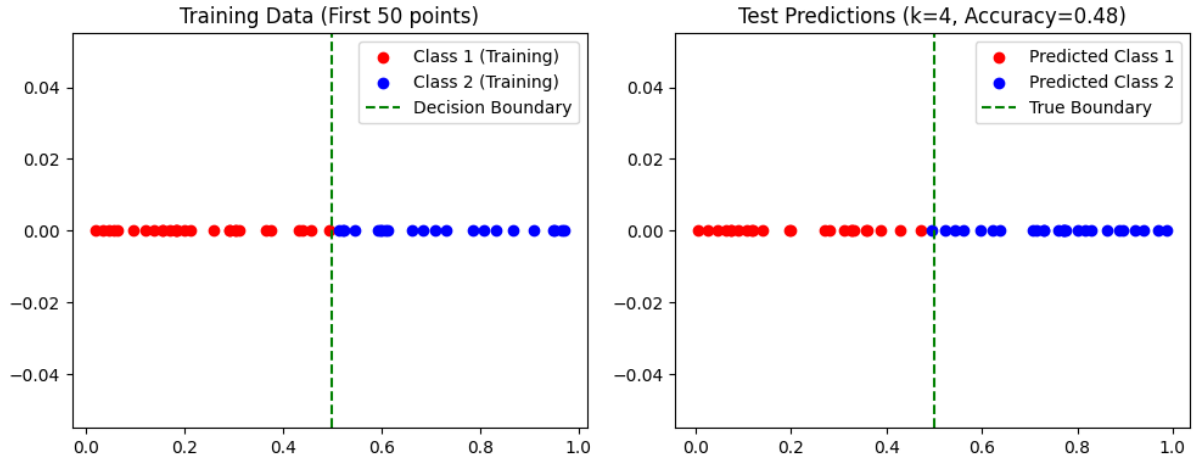

```
Classification with k = 2
```



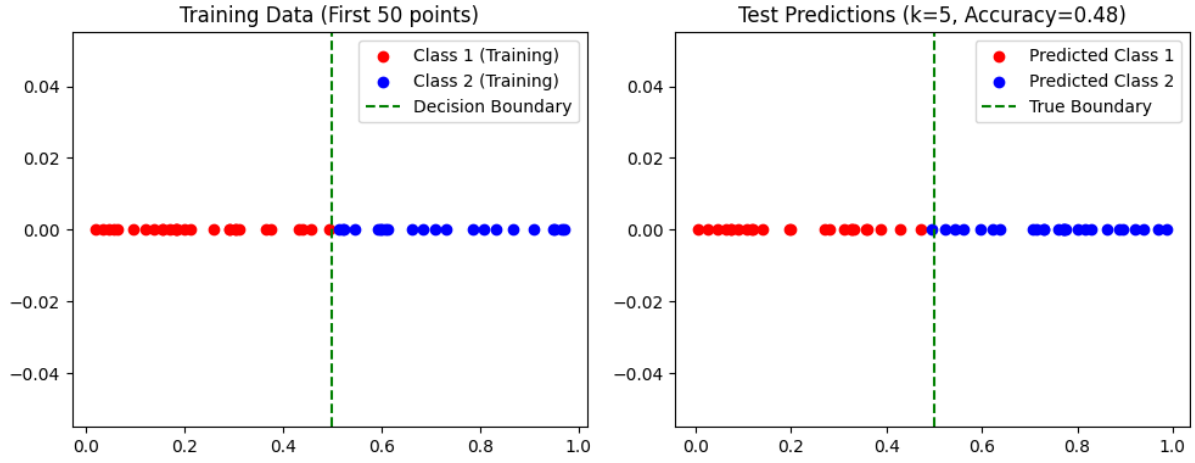
```
Classification with k = 3
```



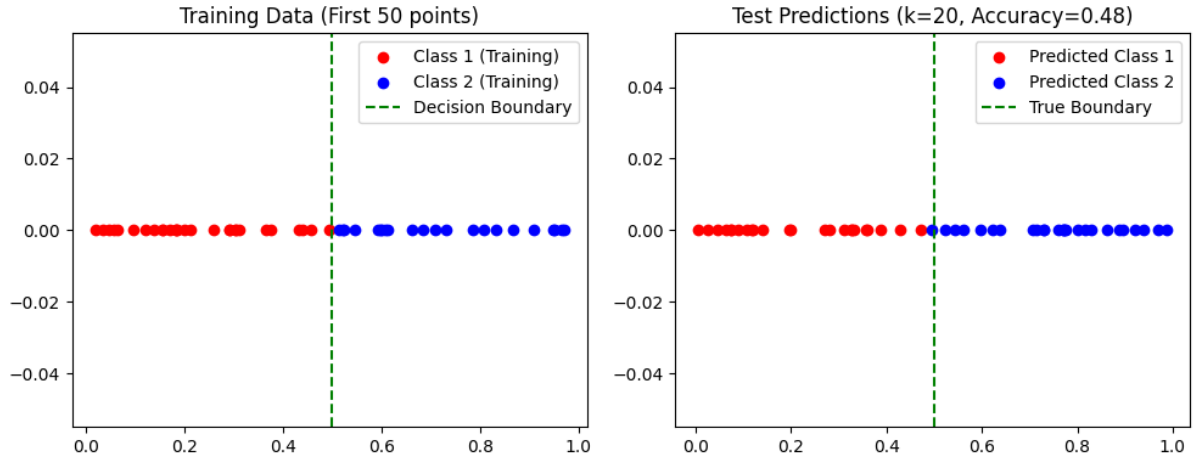

```
Classification with k = 4
```



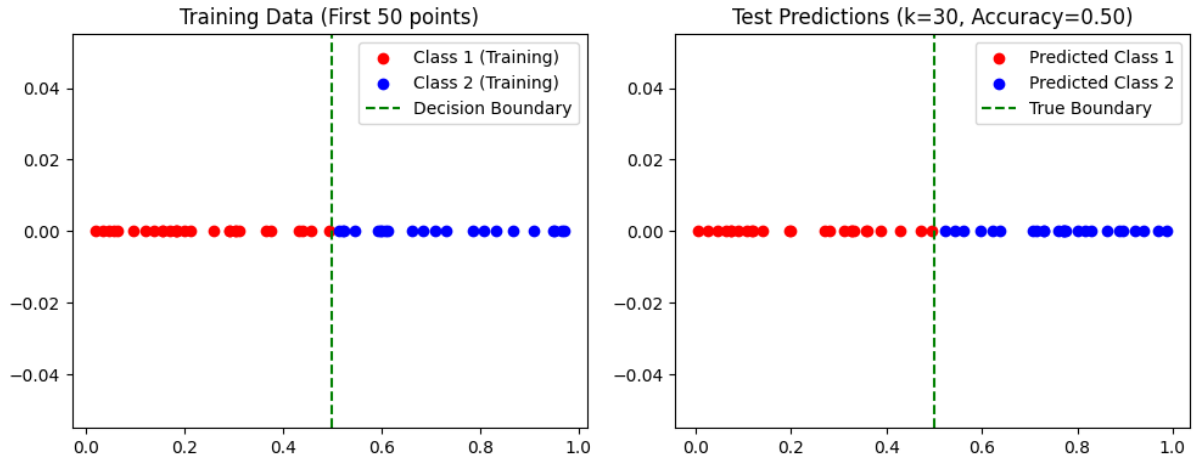

```
Classification with k = 5
```



```
Classification with k = 20
```



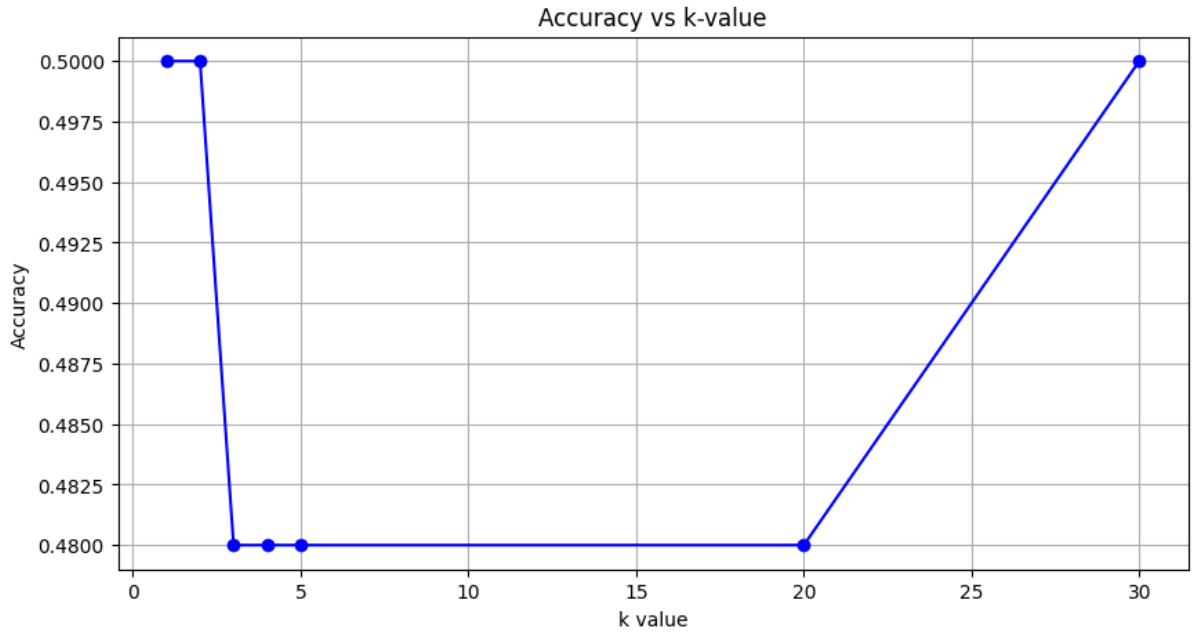

```
Classification with k = 30
```



Performance Analysis:

```
# Plot accuracies for different k values
plt.figure(figsize=(10, 5))
plt.plot(k_values, accuracies, 'bo-')
plt.xlabel('k value')
plt.ylabel('Accuracy')
plt.title('Accuracy vs k-value')
plt.grid(True)
plt.show()
```

Output:


Accuracy vs k-value
Accuracy
k value
0.5000
0.4975
0.4950
0.4925
0.4900
0.4875
0.4850
0.4825
0.4800
0
5
10
15
20
25
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

Import necessary libraries:

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

Data Generation:

```
# Function to generate sample data with some noise
def generate_data(n_samples=50):
    np.random.seed(42)
    X = np.linspace(0, 10, n_samples)
    y = np.sin(X) + 0.3 * np.random.randn(n_samples)
    return X, y
```

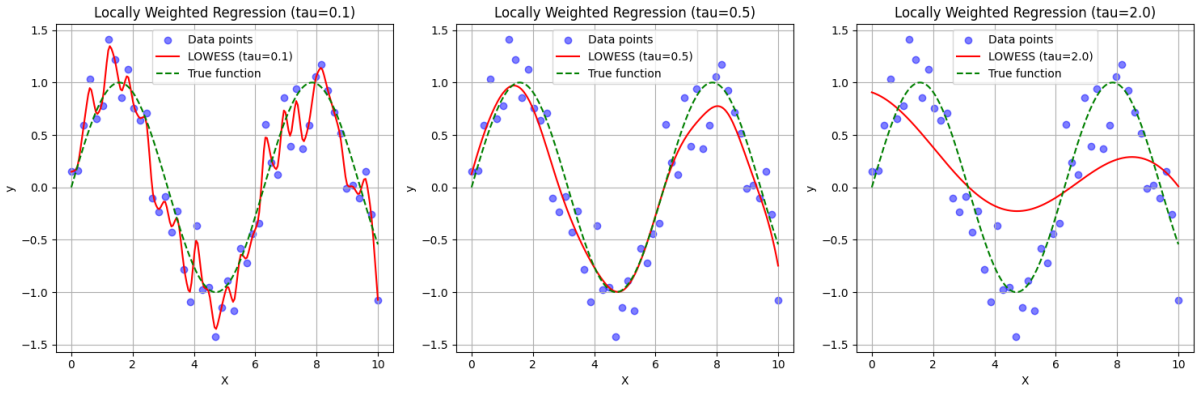
LOWESS Implementation:

```
# Locally Weighted Regression implementation
def lowess(x, y, xtest, tau=0.5):
    # Initialize predictions array
    ytest = np.zeros(len(xtest))

    # For each point we want to predict
    for i, x0 in enumerate(xtest):
        # Compute weights for all training points
        weights = np.exp(-(x - x0)**2 / (2 * tau**2))

        # Weighted least squares
        W = np.diag(weights.ravel())
        X = np.column_stack([np.ones_like(x), x])  # Add bias term

        # Compute weighted least squares solution
        theta = np.linalg.inv(X.T @ W @ X) @ X.T @ W @ y

        # Make prediction
        ytest[i] = np.array([1, x0]) @ theta

    return ytest
```

Visualization:

```
# Generate sample data
X, y = generate_data(50)

# Create test points for smooth curve
X_test = np.linspace(0, 10, 200)

# Fit with different bandwidth parameters
taus = [0.1, 0.5, 2.0]


plt.figure(figsize=(15, 5))

for i, tau in enumerate(taus):
    # Apply LOWESS
    y_pred = lowess(X, y, X_test, tau)

    # Plot
    plt.subplot(1, 3, i+1)
    plt.scatter(X, y, color='blue', alpha=0.5, label='Data points')
    plt.plot(X_test, y_pred, 'r-', label=f'LOWESS (tau={tau})')
    plt.plot(X_test, np.sin(X_test), 'g--', label='True function')
    plt.title(f'Locally Weighted Regression (tau={tau})')
    plt.xlabel('X')
    plt.ylabel('y')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```

Output:



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.

Import necessary libraries:

```
import numpy as np
import pandas as pd
from pandas import read_csv
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
```

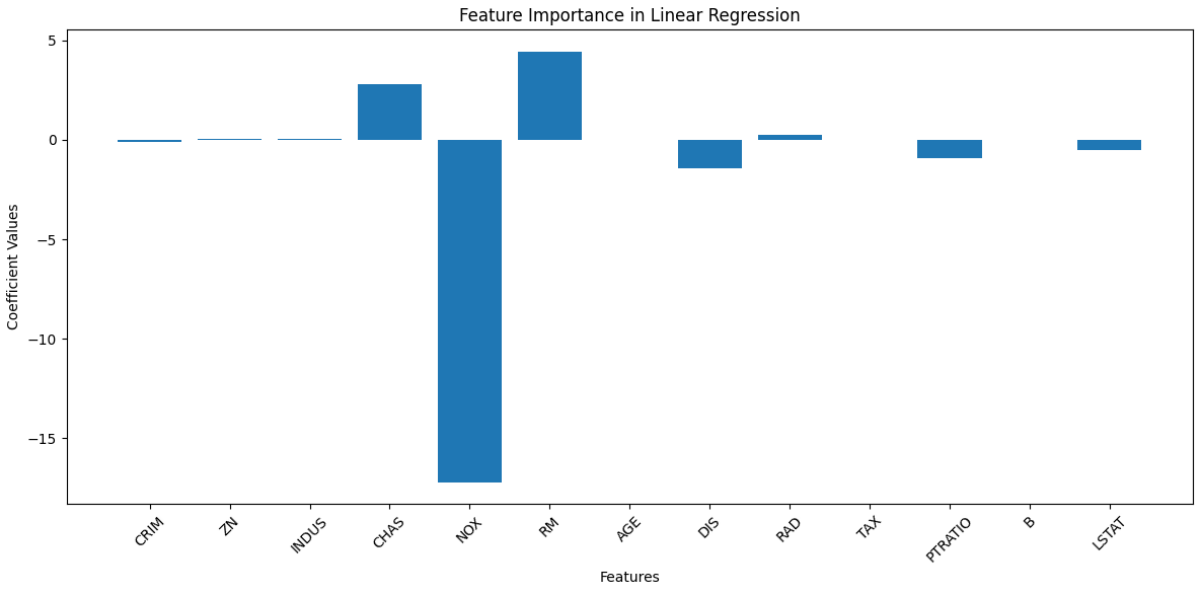
Part 1: Linear Regression (Boston Housing)

```
# Part 1: Linear Regression with Boston Housing Dataset
print("Part 1: Linear Regression - Boston Housing Dataset")
print("-" * 50)

# Load Boston Housing dataset
housing = pd.read_csv('housing.csv')
X_boston = housing.drop('MEDV', axis=1)
y_boston = housing['MEDV']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_boston,
y_boston, test_size=0.2, random_state=42)

# Create and train the linear regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Make predictions
y_pred_train = lr_model.predict(X_train)
y_pred_test = lr_model.predict(X_test)

# Calculate metrics
train_mse = mean_squared_error(y_train, y_pred_train)
test_mse = mean_squared_error(y_test, y_pred_test)
train_r2 = r2_score(y_train, y_pred_train)
test_r2 = r2_score(y_test, y_pred_test)
```

```
# Print results
print("Linear Regression Results:")
print(f"Training MSE: {train_mse:.2f}")
print(f"Test MSE: {test_mse:.2f}")
print(f"Training R²: {train_r2:.2f}")
print(f"Test R²: {test_r2:.2f}")

# Visualize feature importance
plt.figure(figsize=(12, 6))
plt.bar(X_boston.columns, lr_model.coef_)
plt.xticks(rotation=45)
plt.title('Feature Importance in Linear Regression')
plt.xlabel('Features')
plt.ylabel('Coefficient Values')
plt.tight_layout()
plt.show()
```

Output:

```
Part 1: Linear Regression - Boston Housing Dataset
--------------------------------------------------
Linear Regression Results:
Training MSE: 21.64
Test MSE: 24.29
Training R²: 0.75
Test R²: 0.67
```



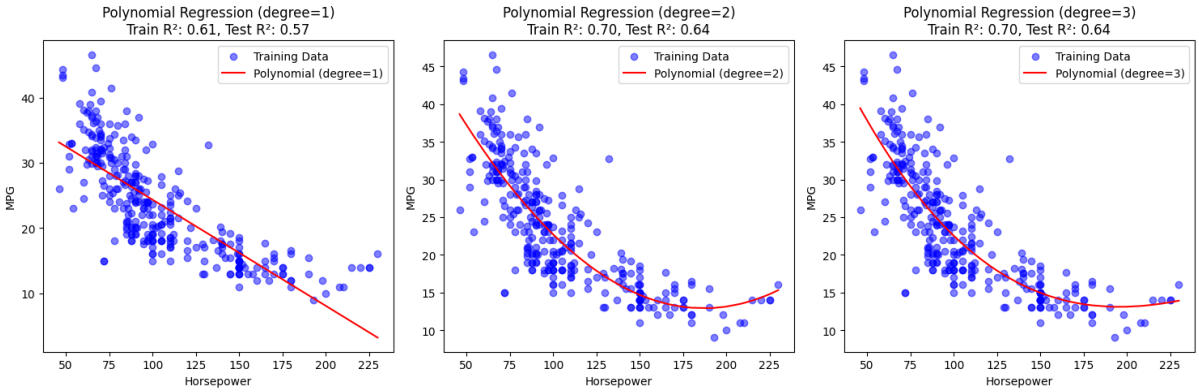
Part 2: Polynomial Regression (Auto MPG)

```
# Part 2: Polynomial Regression with Auto MPG Dataset
print("\nPart 2: Polynomial Regression - Auto MPG Dataset")
print("-" * 50)

# Load Auto MPG dataset (using horsepower to predict mpg)
auto_mpg = 'auto-mpg.data'
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower',
'Weight', 'Acceleration', 'Model Year', 'Origin']
df = pd.read_csv(auto_mpg, names=column_names, na_values='?',
comment='\t', sep=' ', skipinitialspace=True)

# Clean the data
df = df.dropna()
X_mpg = df['Horsepower'].values.reshape(-1, 1)
y_mpg = df['MPG'].values

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_mpg, y_mpg,
test_size=0.2, random_state=42)

# Create polynomial features
degrees = [1, 2, 3]  # Test different polynomial degrees
plt.figure(figsize=(15, 5))

for i, degree in enumerate(degrees, 1):
    # Create polynomial features
    poly_features = PolynomialFeatures(degree=degree)
    X_train_poly = poly_features.fit_transform(X_train)
    X_test_poly = poly_features.transform(X_test)

    # Train model
    poly_model = LinearRegression()
    poly_model.fit(X_train_poly, y_train)

    # Make predictions
    y_pred_train = poly_model.predict(X_train_poly)
    y_pred_test = poly_model.predict(X_test_poly)

    # Calculate metrics
    train_r2 = r2_score(y_train, y_pred_train)
    test_r2 = r2_score(y_test, y_pred_test)
```

```
    # Plot results
    plt.subplot(1, 3, i)
    plt.scatter(X_train, y_train, color='blue', alpha=0.5,
label='Training Data')

    # Sort X values for smooth curve
    X_sort = np.sort(X_train, axis=0)
    X_sort_poly = poly_features.transform(X_sort)
    y_smooth = poly_model.predict(X_sort_poly)

    plt.plot(X_sort, y_smooth, color='red', label=f'Polynomial
(degree={degree})')
    plt.xlabel('Horsepower')
    plt.ylabel('MPG')
    plt.title(f'Polynomial Regression (degree={degree})\nTrain R²:
{train_r2:.2f}, Test R²: {test_r2:.2f}')
    plt.legend()

plt.tight_layout()
plt.show()
```

Output:

```
Part 2: Polynomial Regression - Auto MPG Dataset
--------------------------------------------------
```



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.

Import necessary libraries:

```
import pandas as pd
import numpy as np
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, classification_report
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
```

Load and Prepare Data:

```
# Load the breast cancer dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

# Split the data 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)
```

Create and Train Model:

```
# Initialize the decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42, max_depth=5)

# Train the model
dt_classifier.fit(X_train, y_train)
```

Model Evaluation:

```
# Make predictions on test data
y_pred = dt_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

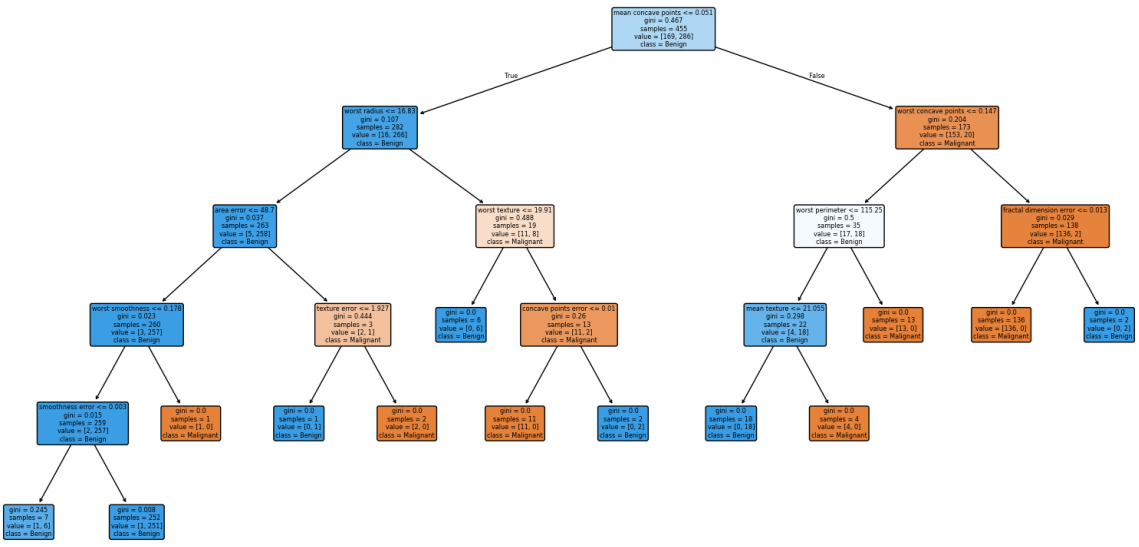
```
# Print detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=['Malignant', 'Benign']))
```

Output:

```
Model Accuracy: 0.95

Classification Report:
              precision    recall  f1-score   support

   Malignant       0.93      0.93      0.93        43
      Benign       0.96      0.96      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

Decision Tree Visualization:

```
# Visualize the decision tree
plt.figure(figsize=(20,10))
plot_tree(dt_classifier, feature_names=data.feature_names,
          class_names=['Malignant', 'Benign'], filled=True,
rounded=True)
plt.show()
```



New Sample Classification:

```
# Create a sample case (using mean values of first 5 most important
features)
new_sample = X.iloc[0:1].copy()  # Take first row as template
print("\nClassifying a new sample:")
prediction = dt_classifier.predict(new_sample)
prediction_proba = dt_classifier.predict_proba(new_sample)

print(f"Prediction: {'Benign' if prediction[0] == 1 else
'Malignant'}")
print(f"Probability of being Malignant:
{prediction_proba[0][0]:.2f}")
print(f"Probability of being Benign: {prediction_proba[0][1]:.2f}")
```

Output:

```
Classifying a new sample:
Prediction: Malignant
Probability of being Malignant: 1.00
Probability of being Benign: 0.00
```

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.

Import necessary libraries:

```
from sklearn.datasets import fetch_olivetti_faces
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import numpy as np
```

Load data:

```
# Load dataset
faces = fetch_olivetti_faces(shuffle=True, random_state=42)
X = faces.data
y = faces.target

# Print dataset information
print("Dataset Information:")
print(f"Number of samples: {X.shape[0]}")
print(f"Number of features: {X.shape[1]}")
print(f"Number of classes: {len(np.unique(y))}")
```

Output:

```
Dataset Information:
Number of samples: 400
Number of features: 4096
Number of classes: 40
```

Data Splitting:

```
# Split 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, stratify=y
)

print("\nSplit Information:")
print(f"Training samples: {X_train.shape[0]}")
print(f"Testing samples: {X_test.shape[0]}")
```

Model Training:

```
# Initialize and train the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
```

Evaluation:

```
# Make predictions on test set
y_pred = nb_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("\nModel Evaluation:")
print(f"Accuracy: {accuracy:.4f}")

# Print detailed classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Output:

```
Model Evaluation:
Accuracy: 0.9125

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       1.00      0.50      0.67         2
           2       0.50      0.50      0.50         2
           3       0.50      1.00      0.67         2
           4       1.00      1.00      1.00         2
           5       1.00      1.00      1.00         2
           6       1.00      1.00      1.00         2
           7       0.67      1.00      0.80         2
           8       1.00      1.00      1.00         2
           9       1.00      0.50      0.67         2
          10       1.00      1.00      1.00         2
          11       1.00      1.00      1.00         2
          12       1.00      0.50      0.67         2
          13       1.00      1.00      1.00         2
          14       1.00      1.00      1.00         2
          15       1.00      1.00      1.00         2
          16       0.50      1.00      0.67         2
          17       1.00      1.00      1.00         2
...
    accuracy                           0.91        80
   macro avg       0.95      0.91      0.91        80
weighted avg       0.95      0.91      0.91        80
```

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

Import necessary libraries:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

Data Preparation:

```
# Load the Wisconsin Breast Cancer dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

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

K-means Implementation:

```
# Initialize and fit k-means (k=2 since we know there are 2 classes)
kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
```

Evaluation:

```
# Compare clusters with actual labels
matches = sum(np.abs(clusters - y)) if sum(clusters == y) < len(y)/2
else sum(clusters == y)
accuracy = matches / len(y)
print(f"Clustering accuracy: {accuracy:.2f}")
```

Output:

Clustering accuracy: 0.91

Visualization:

```
# Reduce dimensions to 2D for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Create scatter plot
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters,
cmap='viridis')
plt.title('K-means Clustering of Breast Cancer Data\n(Visualized
using PCA)')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.colorbar(scatter, label='Cluster')
plt.show()
```

Output: