

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



**MACHINE LEARNING LABORATORY MANUAL**

**COURSE CODE: BCSL606**

**SEMESTER: VI**

**Prepared by**  
**Dr. Pooja. K. Revankar**

**(Academic Year: 2024-25)**

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### DEPARTMENT'S VISION

To become a prominent department of computer science and engineering whose graduates are globally recognized as innovative and well-prepared computing professionals.

### DEPARTMENT'S MISSION

- To provide a superior, student-centered learning environment this emphasizes close faculty-student interaction, experiential education, and distinctive research opportunities.
- Graduates will be prepared to excel as professionals, pursue advanced degrees, and possess the technical knowledge, critical thinking skills and creativity.
- Graduates will be prepared with ethical values needed to lead the development and application of technology for betterment of society and sustaining the world environment.
- Graduates will be prepared to explore in various avenues with entrepreneurship skills.

### PROGRAM EDUCATIONAL OBJECTIVES(PEOs)

- PEO-1: Educate and train students to possess versatile knowledge in computational science, enabling them to excel in various domains.
- PEO-2: Educate students to effectively leverage current and emerging technologies in solving societal problems and engaging in research activities.
- PEO-3: Foster the development of entrepreneurial qualities in students, empowering them to contribute towards achieving organizational goals.

## PROGRAMME OUTCOMES (POs)

- PO1: Engineering knowledge: Apply the knowledge of mathematics science engineering fundamentals and an mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems engineering problems.
- PO2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
- PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- PO6: The engineer and society: Apply reasoning informed by Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development and need for sustainable development.
- PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- PO9: Individual and team work: Function effectively as an individual and as a member or leader in diverse teams and individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and knowledge and understanding of the engineering and management principles

and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: long learning: Recognize the need for and have the Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

### **PROGRAMME SPECIFIC OUTCOMS (PSOs )**

- **PSO-1:** Apply mathematics and computational science knowledge with an interdisciplinary approach to develop applications that meet the requirements of diverse domains.
- **PSO-2:** Demonstrate proficient programming language skills to develop efficient and cost-effective applications.
- **PSO-3:** Cultivate students' ability to learn and utilize the latest computer languages and platforms. Foster an environment of innovation, entrepreneurship, and lifelong learning, instilling moral values and ethics to produce responsible citizens.

### **GENERAL INSTRUCTIONS (Do's And Don'ts)**

1. Wearing ID-Card is compulsory.
2. Keep your bag at the specified place.
3. Shut down the system after use.
4. Place the chairs in proper position before leaving the laboratory.
5. Report failure/Non-working of equipment to Faculty In-charge /Technical Support staff immediately.
6. Know the location of the fire extinguisher and the FIRST-AID Box and how to use then in case of an emergency.
7. Do not eat or drink in the laboratory.
8. Do not litter in the laboratory.
9. Avoid stepping on electrical wires or any other computer cables.
10. Do not open the system unit casing or monitor casing particularly when the power is turned ON.
11. Do not insert metal objects such as clips, pins and needles into the computer casing. They may cause fire.
12. Do not remove anything from laboratory without permissions.
13. Do not touch, connect or disconnect any plug or cable without permission.

**LAB OUTCOMES (LO)**

LO1: Illustrate the principles of multivariate data and apply dimensionality reduction techniques.

LO2: Demonstrate similarity-based learning methods and perform regression analysis.

LO3: Develop decision trees for classification and regression problems, and Bayesian models for probabilistic learning.

LO4: Implement the clustering algorithms to share computing resources.

**LAB ARTICULATION MATRIX (MAPPING WITH PO & PSO)**

<b>Course Outcomes (COs) / Program Outcomes (POs)</b>		1	2	3	4	5	6	7	8	9	10	11	12	PSO1	PSO2	PSO3
CO1	Illustrate the principles of multivariate data and apply dimensionality reduction techniques.	3	3	3	2	2							2	2	2	1
CO2	Demonstrate similarity-based learning methods and perform regression analysis.	3	3	3	2	2							2	2	2	1
CO3	Develop decision trees for classification and regression problems, and Bayesian models for probabilistic learning.	3	3	3	2	2							2	2	2	1
CO4	Implement the clustering algorithms to share computing resources.	3	3	3	2	2							2	2	2	1

## LIST OF EXPERIMENTS

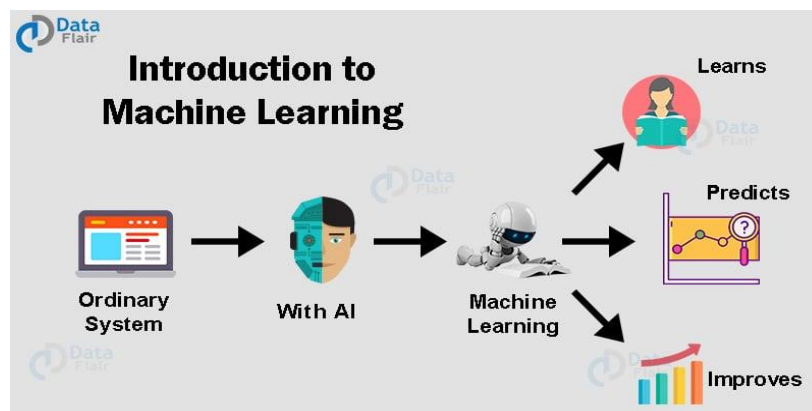
Expt. No.	Experiment	CO Mapped
1	Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.	CO1
2	Develop a program to Compute the correlation 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.	CO1
3	Develop a program to implement Principal Component Analysis (PCA) for reducing the dimensionality of the Iris dataset from 4 features to 2.	CO1
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.	CO2
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 $\{x_1, \dots, x_{50}\}$ as follows: if $(x_i \leq 0.5)$ , then $x_i \in \text{Class1}$ , else $x_i \in \text{Class2}$ b. Classify the remaining points, $x_{51}, \dots, x_{100}$ using KNN. Perform this for $k=1,2,3,4,5,20,30$	CO2
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	CO2
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.	CO2
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.	CO3
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.	CO3
10	Develop a program to implement k-means clustering using Wisconsin Breast Cancer data set and visualize the clustering result.	CO4

## Experiment No. 1

**Aim:** 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.

**Theory:** Machine Learning (ML) enables systems to learn from data and make decisions or predictions. ML tasks are broadly categorized into:

- **Supervised Learning:** Uses labeled data to train models to predict outputs (e.g., predicting house prices).
- **Unsupervised Learning:** Identifies patterns in data without labels (e.g., clustering).
- **Reinforcement Learning:** Models learn from interactions with an environment to maximize rewards.



In Machine Learning (ML), the essential steps to develop effective models are:

1. **Define the Problem:** Clearly articulate the objective, such as classification, regression, or clustering.
2. **Collect Data:** Gather relevant data from various sources, ensuring it aligns with your problem's requirements.
3. **Prepare the Data:** Clean and preprocess the data by handling missing values, removing duplicates, and converting data types as needed.
4. **Select a Model:** Choose an appropriate ML algorithm based on the problem type and data characteristics.

5. **Train the Model:** Use the prepared data to train the selected model, allowing it to learn patterns and relationships.
6. **Evaluate the Model:** Assess the model's performance using evaluation metrics like accuracy, precision, recall, or mean squared error, depending on the problem.
7. **Make Predictions:** Apply the trained and evaluated model to new, unseen data to make predictions.

**Example: Features in the California Housing Dataset**

The California Housing dataset (from `sklearn.datasets.fetch_california_housing()`) has 8 features:

Feature	Description
MedInc	Median income of households in the area.
HouseAge	Median age of houses in the area.
AveRooms	Average number of rooms per house.
AveBedrms	Average number of bedrooms per house.
Population	Total population in the area.
AveOccup	Average number of people per household.
Latitude	Geographic latitude of the house.
Longitude	Geographic longitude of the house.

These features help in predicting house prices (target variable).

### **Program:**

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

# Load the California Housing dataset from sklearn
data = fetch_california_housing()

# Convert the dataset into a DataFrame for easier manipulation
df = pd.DataFrame(data.data, columns=data.feature_names)
```



```
# Add the target variable (Median House Value)
df['MedHouseVal'] = data.target

# Display basic information about the dataset
print("\nDataset Information:")
df.info()

# Display summary statistics
print("\nSummary Statistics of the Dataset:")
print(df.describe())

# Plot histograms for all numerical features
plt.figure(figsize=(12, 8))
df.hist(bins=30, figsize=(12, 8), edgecolor='black')
plt.suptitle("Histograms of Numerical Features", fontsize=16)
plt.show()

# Box plots for each numerical feature
plt.figure(figsize=(12, 8))
for i, column in enumerate(df.columns, 1):
    plt.subplot(3, 3, i) # Adjusted for 9 columns (features + target)
    sns.boxplot(y=df[column], color='skyblue')
    plt.title(column)
plt.tight_layout()
plt.suptitle("Box Plots of Numerical Features", fontsize=16, y=1.02)
plt.show()

# Function to identify outliers using IQR
def identify_outliers(df):
    outliers = { }
    for column in df.columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
```

```
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers[column]=df[(df[column]<lower_bound)|(df[column]>upper_bound)][column].count()
return outliers

# Call the outlier identification function and print the results
outliers = identify_outliers(df)
outlier_df = pd.DataFrame.from_dict(outliers, orient='index', columns=['Outlier Count'])
print("\nOutlier Counts per Feature:")
print(outlier_df)

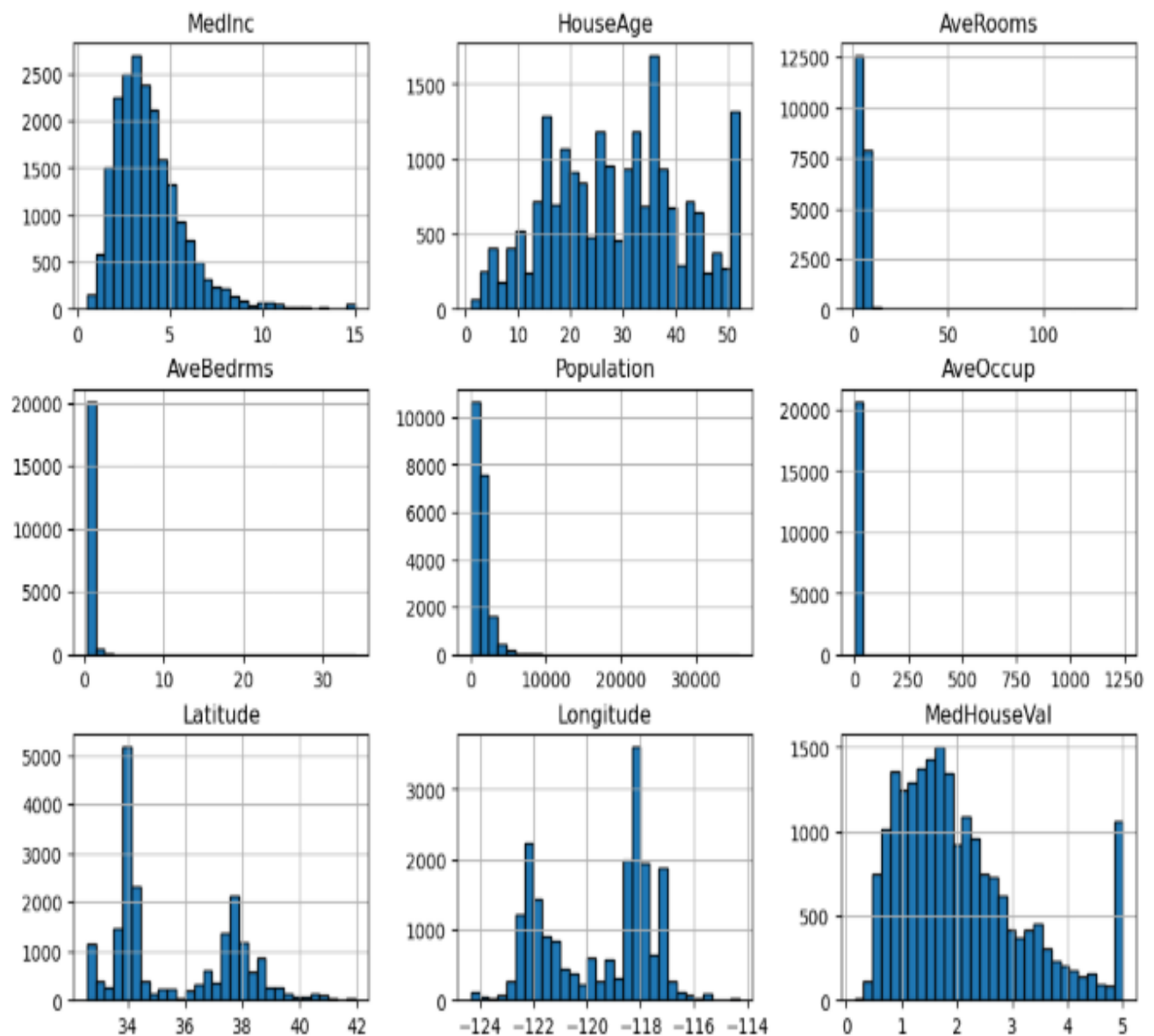
# Pairplot for feature relationships (sampling to reduce computation time)
sns.pairplot(df.sample(500))
plt.show()
```

### Steps in the Code:

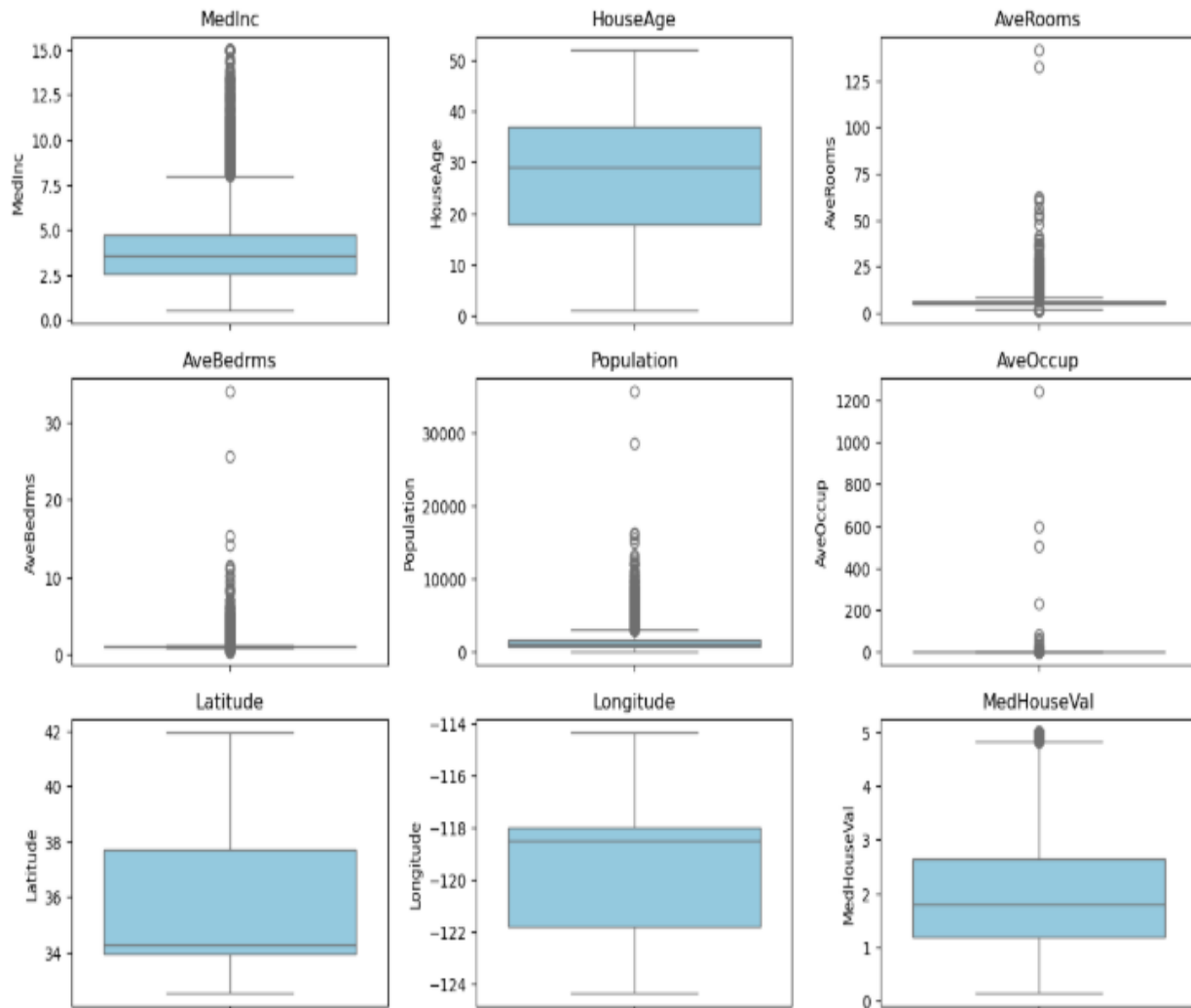
1. **Import Libraries:** The required libraries (numpy, pandas, matplotlib, seaborn, and sklearn) are imported.
2. **Load Dataset:** The California housing dataset is fetched from sklearn and stored in a pandas DataFrame (df).
3. **Basic Info:** The .info() method provides a summary of the DataFrame (e.g., number of entries, data types, missing values).
4. **Histograms:** df.hist() generates histograms for all numerical features to show their distributions.
5. **Box Plots:** Using sns.boxplot(), box plots for each numerical feature are created to visually check for potential outliers.
6. **Outlier Detection:** The identify\_outliers() function uses the Interquartile Range (IQR) method to detect outliers. The IQR is calculated as the difference between the 75th and 25th percentiles (Q3 - Q1), and outliers are considered as values falling outside the range:
  - lower\_bound = Q1 - 1.5 \* IQR
  - upper\_bound = Q3 + 1.5 \* IQR
7. **Print Outliers:** Finally, the code prints the number of outliers for each feature.

**Output:**

### Histograms of Numerical Features



Box Plots of Numerical Features



### Outlier Counts per Feature:

	Outlier Count
MedInc	681
HouseAge	0
AveRooms	511
AveBedrms	1424
Population	1196
AveOccup	711
Latitude	0
Longitude	0
MedHouseVal	1071

## Experiment No. 2

**Aim:** 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.

**Theory:** The California Housing dataset to understand how different features in houses are related to each other.

The main purpose is to find correlations between different housing features. A correlation shows how strongly two features are related. For example, it can tell us if house prices tend to go up when the number of rooms increases.

Correlation values range from -1 to +1:

- +1 means perfect positive correlation (when one goes up, the other goes up)
- 0 means no correlation (no relationship)
- -1 means perfect negative correlation (when one goes up, the other goes down)

The code creates two main visualizations:

1. A Correlation Heatmap
2. A Pair Plot

This analysis helps understand patterns in the housing market, like:

- Which features most strongly affect house prices
- Which features tend to occur together
- Whether features have expected or surprising relationships

### **Program:**

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

# Load the California Housing dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['MedHouseVal'] = data.target # Add target column to the DataFrame

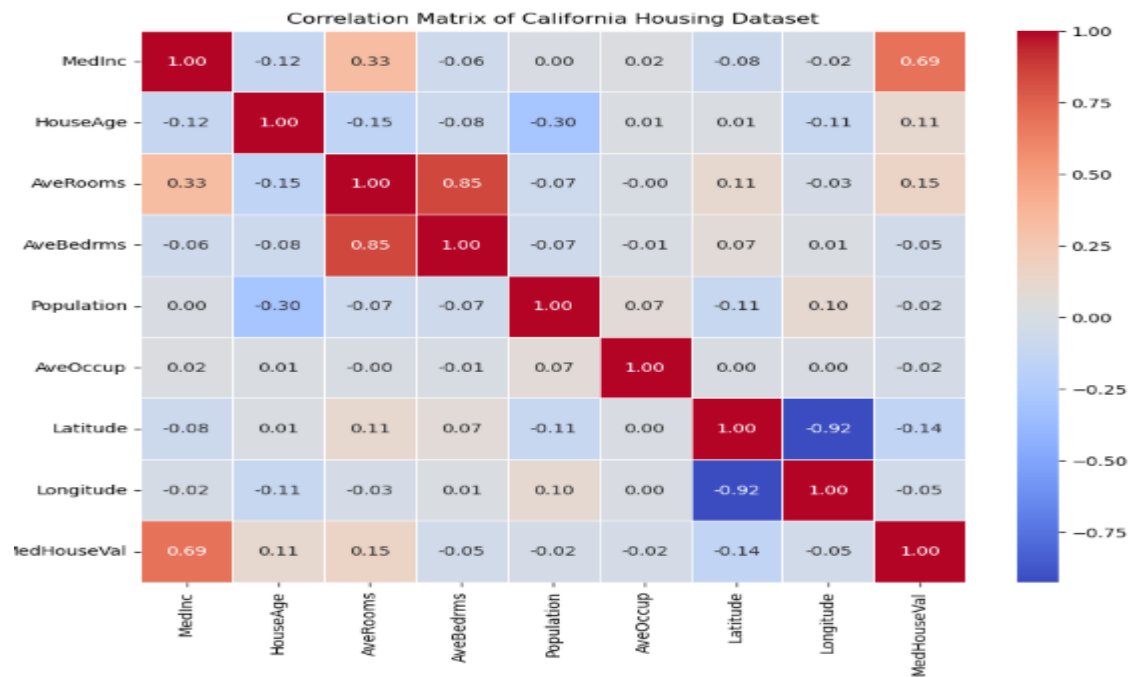
# Compute the correlation matrix
corr_matrix = df.corr()

# Plot the heatmap of the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Matrix of California Housing Dataset')
plt.show()

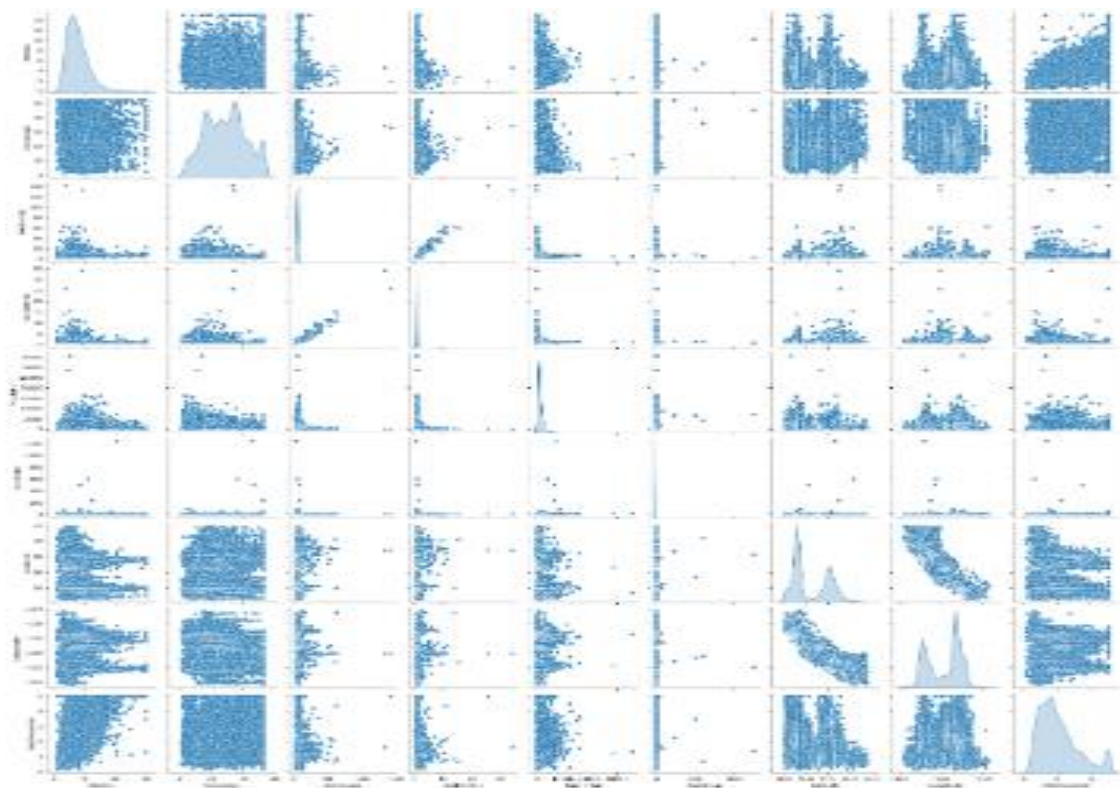
# Pair plot to visualize pairwise relationships
sns.pairplot(df, diag_kind='kde', markers='o')
plt.show()
```

**Output:**

**Heatmap:**



**KDE:**



## Experiment No. 3

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

**Theory:** PCA is a technique that reduces the dimensionality of data while preserving as much important information as possible. It transforms high-dimensional data into a new set of features called principal components.

**The Code Does:**

1. Takes 4-dimensional iris flower measurements
2. Reduces them to 2 dimensions while keeping most important patterns
3. Shows how well different iris species can be distinguished
4. Tells us which original measurements are most important

**This is Useful:**

- Helps visualize high-dimensional data
- Finds most important patterns in the data
- Shows which original features matter most
- Can help classify different types of iris flowers using fewer measurements

**Program:**

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

# Load the Iris dataset
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
```



```
# Standardizing the features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.iloc[:, :-1])

# Applying PCA to reduce dimensions from 4 to 2
pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_data)

# Creating a DataFrame with principal components
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['target'] = df['target']

# Visualizing the results
plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
labels = iris.target_names

for i in range(3):
    plt.scatter(pca_df.loc[pca_df['target'] == i, 'PC1'],
                pca_df.loc[pca_df['target'] == i, 'PC2'],
                label=labels[i], color=colors[i], alpha=0.7)

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

**Output:**



## Experiment No. 4

**Aim:** 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.

### Theory:

#### Find-S Algorithm for Hypothesis Learning

- The Find-S algorithm is used to identify the most specific hypothesis that is consistent with the given training examples.
- It focuses only on positive examples to generalize the hypothesis.
- The algorithm helps in concept learning by determining necessary conditions for classification.

#### Working Principle

- The algorithm considers only positive examples from the training dataset.
- For each positive example:
  - It compares each attribute with the current hypothesis.
  - If the attribute matches the hypothesis, it remains unchanged.
  - If the attribute differs, the hypothesis is generalized using '?'.
- The hypothesis never becomes more specific once generalized.

#### Generalization Rules

- If the attribute matches the current hypothesis, it remains unchanged.
- If the current hypothesis is null ( $\phi$ ), the example's value is used.
- If a mismatch occurs, the hypothesis generalizes to '?' to accommodate variations.

**Dataset:** Consider a dataset where we classify whether to **play tennis** based on weather conditions. The attributes include:

- Outlook (Sunny, Overcast, Rainy)
- Temperature (Hot, Mild, Cool)

- Humidity (High, Normal)
- Wind (Weak, Strong)

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	Yes
Sunny	Hot	High	Strong	Yes
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No

**Final Hypothesis:** After applying the Find-S algorithm, the resulting hypothesis is:

{Outlook: 'Overcast', Temperature: 'Hot', Humidity: 'High', Wind: '?'}

This means that whenever the outlook is **Overcast**, the temperature is **Hot**, and the humidity is **High**, we can predict that **tennis will be played**, regardless of wind conditions.

### Program:

```
#Import the necessary libraries
import pandas as pd
import numpy as np

# Load CSV dataset
df = pd.read_csv('training_data.csv')
print("Training Data:")
df

def find_s_algorithm(data):
    print("Training data:")
    print(data)

    attributes = data.columns[:-1]
    class_label = data.columns[-1]
```

```
hypothesis = ['?' for _ in attributes]

for index, row in data.iterrows():
    if row[class_label] == 'Yes':
        for i, value in enumerate(row[attributes]):
            if hypothesis[i] == '?' or hypothesis[i] == value:
                hypothesis[i] = value
            else:
                hypothesis[i] = '?'

return {attr: val for attr, val in zip(attributes, hypothesis)}
```

# Apply Find-S Algorithm

```
hypothesis = find_s_algorithm(df)
print("\nFinal Hypothesis:")
print(hypothesis)
```

### **Output:**

Training data:

	Outlook	Temperature	Humidity	Windy	PlayTennis
0	Sunny	Hot	High	False	No
1	Sunny	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Rain	Cold	High	False	Yes
4	Rain	Cold	High	True	No
5	Overcast	Hot	High	True	Yes
6	Sunny	Hot	High	False	No

Final Hypothesis:

```
{'Outlook': 'Overcast', 'Temperature': 'Hot', 'Humidity': 'High', 'Windy': '?'}
```

## Experiment No. 5

**Aim:** 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  $\{x_1, \dots, x_{50}\}$  as follows: if  $(x_i \leq 0.5)$ , then  $x_i \in \text{Class1}$ , else  $x_i \in \text{Class2}$ .
- b. Classify the remaining points,  $x_{51}, \dots, x_{100}$  using KNN. Perform this for  $k=1,2,3,4,5,20,30$ .

### Theory:

The **k-Nearest Neighbors (k-NN)** algorithm is a **supervised learning** method used for **classification and regression**. It is based on the principle that **similar data points exist in close proximity**.

### Working of k-NN:

1. **Training Phase:**
  - k-NN does **not** explicitly learn a model; it simply stores the training data.
2. **Prediction Phase:**
  - For a given test point, the algorithm:
    - Computes the **distance** (e.g., Euclidean, Manhattan) between the test point and all training points.
    - Selects the **k nearest neighbors** based on the smallest distances.
    - **Majority voting** (for classification) or **average value** (for regression) determines the output.

### Key Features of k-NN:

- **Lazy Learning:** No explicit training phase, making it a **memory-based** approach.
- **Distance-based Classification:** Works well when data has a meaningful distance metric.
- **Sensitive to k:** A small **k** may lead to **overfitting**, while a large **k** may lead to **underfitting**.

**Program:**

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter

data = np.random.rand(100)
labels = ["Class1" if x <= 0.5 else "Class2" for x in data[:50]]

def knn_classifier(train_data, train_labels, test_point, k):
    distances = sorted([(abs(test_point - train_data[i]), train_labels[i]) for i in range(len(train_data))],
key=lambda x: x[0])
    k_nearest_labels = [label for _, label in distances[:k]]
    return Counter(k_nearest_labels).most_common(1)[0][0]

train_data, train_labels = data[:50], labels
test_data = data[50:]
k_values = [1, 2, 3, 4, 5, 20, 30]
print("--- k-Nearest Neighbors Classification ---")
print("Training dataset: First 50 points labeled based on (x <= 0.5 -> Class1, x > 0.5 -> Class2)")
print("Testing dataset: Remaining 50 points to be classified\n")

results = {k: [knn_classifier(train_data, train_labels, test_point, k) for test_point in test_data] for k in
k_values}

for k, classified_labels in results.items():
    print(f"Results for k = {k}:")
    for i, label in enumerate(classified_labels, start=51):
        print(f"Point x{i} (value: {test_data[i - 51]:.4f}) is classified as {label}")
    print()

print("Classification complete.\n")

for k, classified_labels in results.items():
    class1_points = [test_data[i] for i in range(len(test_data)) if classified_labels[i] == "Class1"]
```

```
class2_points = [test_data[i] for i in range(len(test_data)) if classified_labels[i] == "Class2"]

plt.figure(figsize=(10, 6))

plt.scatter(train_data, [0] * len(train_data), c=["blue" if label == "Class1" else "red" for label in
train_labels], label="Training Data", marker="o")

plt.scatter(class1_points, [1] * len(class1_points), c="blue", label="Class1 (Test)", marker="x")
plt.scatter(class2_points, [1] * len(class2_points), c="red", label="Class2 (Test)", marker="x")

plt.title(f"k-NN Classification Results for k = {k}")

plt.xlabel("Data Points")

plt.ylabel("Classification Level")

plt.legend()

plt.grid(True)

plt.show()
```

### Output:

```
--- k-Nearest Neighbors Classification ---
Training dataset: First 50 points labeled based on (x <= 0.5 -> Class1, x > 0.5 -> Class2)
Testing dataset: Remaining 50 points to be classified

Results for k = 1:
Point x51 (value: 0.2944) is classified as Class1
Point x52 (value: 0.3851) is classified as Class1
Point x53 (value: 0.8511) is classified as Class2
Point x54 (value: 0.3169) is classified as Class1
Point x55 (value: 0.1695) is classified as Class1
Point x56 (value: 0.5568) is classified as Class2
Point x57 (value: 0.9362) is classified as Class2
Point x58 (value: 0.6960) is classified as Class2
Point x59 (value: 0.5701) is classified as Class2
Point x60 (value: 0.0972) is classified as Class1
Point x61 (value: 0.6150) is classified as Class2
Point x62 (value: 0.9901) is classified as Class2
Point x63 (value: 0.1401) is classified as Class1
Point x64 (value: 0.5183) is classified as Class2
Point x65 (value: 0.8774) is classified as Class2
Point x66 (value: 0.7408) is classified as Class2
Point x67 (value: 0.6970) is classified as Class2
Point x68 (value: 0.7025) is classified as Class2
Point x69 (value: 0.3595) is classified as Class1
Point x70 (value: 0.2936) is classified as Class1
Point x71 (value: 0.8094) is classified as Class2
Point x72 (value: 0.8101) is classified as Class2
Point x73 (value: 0.8671) is classified as Class2
Point x74 (value: 0.9132) is classified as Class2
Point x75 (value: 0.5113) is classified as Class2
Point x76 (value: 0.5015) is classified as Class2
Point x77 (value: 0.7983) is classified as Class2
Point x78 (value: 0.6500) is classified as Class2
Point x79 (value: 0.7020) is classified as Class2
Point x80 (value: 0.7958) is classified as Class2
Point x81 (value: 0.8900) is classified as Class2
Point x82 (value: 0.3380) is classified as Class1
Point x83 (value: 0.3756) is classified as Class1
Point x84 (value: 0.0940) is classified as Class1
```



```
Results for k = 2:
Point x51 (value: 0.2944) is classified as Class1
Point x52 (value: 0.3851) is classified as Class1
Point x53 (value: 0.8511) is classified as Class2
Point x54 (value: 0.3169) is classified as Class1
Point x55 (value: 0.1695) is classified as Class1
Point x56 (value: 0.5568) is classified as Class2
Point x57 (value: 0.9362) is classified as Class2
Point x58 (value: 0.6960) is classified as Class2
Point x59 (value: 0.5701) is classified as Class2
Point x60 (value: 0.0972) is classified as Class1
Point x61 (value: 0.6150) is classified as Class2
Point x62 (value: 0.9901) is classified as Class2
Point x63 (value: 0.1401) is classified as Class1
Point x64 (value: 0.5183) is classified as Class2
Point x65 (value: 0.8774) is classified as Class2
Point x66 (value: 0.7408) is classified as Class2
Point x67 (value: 0.6970) is classified as Class2
Point x68 (value: 0.7025) is classified as Class2
Point x69 (value: 0.3595) is classified as Class1
Point x70 (value: 0.2936) is classified as Class1
Point x71 (value: 0.8094) is classified as Class2
Point x72 (value: 0.8101) is classified as Class2
Point x73 (value: 0.8671) is classified as Class2
Point x74 (value: 0.9132) is classified as Class2
Point x75 (value: 0.5113) is classified as Class2
Point x76 (value: 0.5015) is classified as Class2
Point x77 (value: 0.7983) is classified as Class2
Point x78 (value: 0.6500) is classified as Class2
Point x79 (value: 0.7020) is classified as Class2
Point x80 (value: 0.7958) is classified as Class2
Point x81 (value: 0.8900) is classified as Class2
Point x82 (value: 0.3380) is classified as Class1
Point x83 (value: 0.3756) is classified as Class1

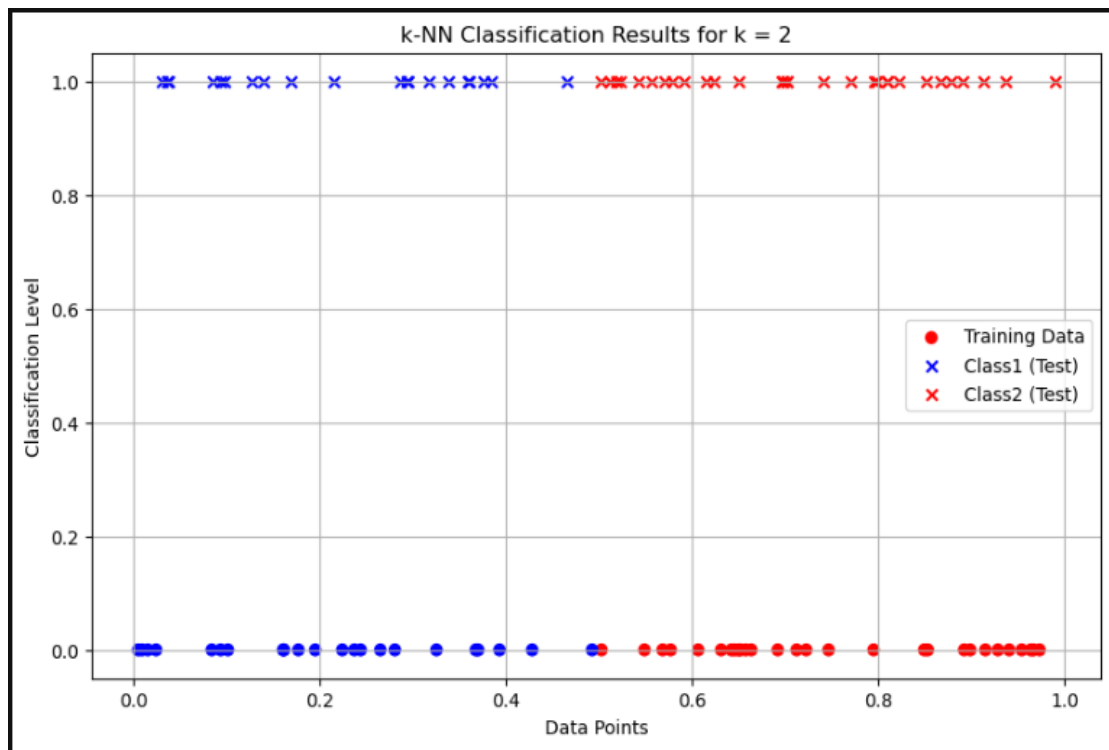
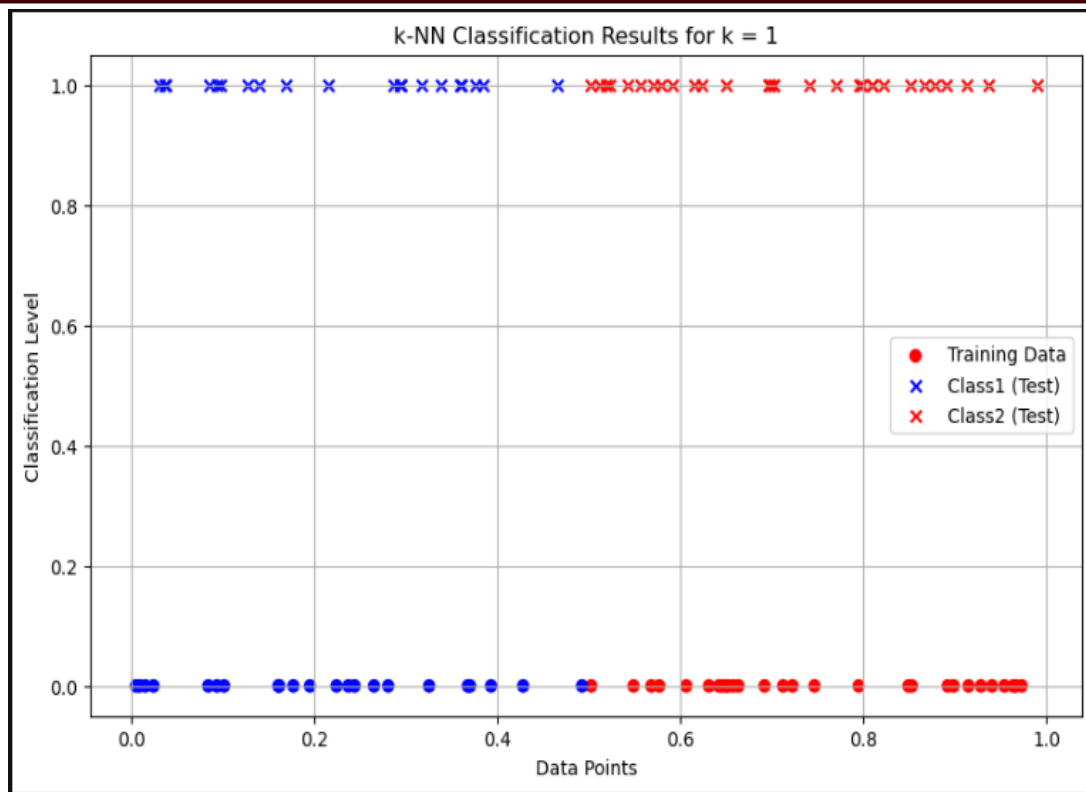
Results for k = 4:
Point x51 (value: 0.2944) is classified as Class1
Point x52 (value: 0.3851) is classified as Class1
Point x53 (value: 0.8511) is classified as Class2
Point x54 (value: 0.3169) is classified as Class1
Point x55 (value: 0.1695) is classified as Class1
Point x56 (value: 0.5568) is classified as Class2
Point x57 (value: 0.9362) is classified as Class2
Point x58 (value: 0.6960) is classified as Class2
Point x59 (value: 0.5701) is classified as Class2
Point x60 (value: 0.0972) is classified as Class1
Point x61 (value: 0.6150) is classified as Class2
Point x62 (value: 0.9901) is classified as Class2
Point x63 (value: 0.1401) is classified as Class1
Point x64 (value: 0.5183) is classified as Class2
Point x65 (value: 0.8774) is classified as Class2
Point x66 (value: 0.7408) is classified as Class2
Point x67 (value: 0.6970) is classified as Class2
Point x68 (value: 0.7025) is classified as Class2
Point x69 (value: 0.3595) is classified as Class1
Point x70 (value: 0.2936) is classified as Class1
Point x71 (value: 0.8094) is classified as Class2
Point x72 (value: 0.8101) is classified as Class2
Point x73 (value: 0.8671) is classified as Class2
Point x74 (value: 0.9132) is classified as Class2
Point x75 (value: 0.5113) is classified as Class2
Point x76 (value: 0.5015) is classified as Class2
Point x77 (value: 0.7983) is classified as Class2
Point x78 (value: 0.6500) is classified as Class2
Point x79 (value: 0.7020) is classified as Class2
Point x80 (value: 0.7958) is classified as Class2
Point x81 (value: 0.8900) is classified as Class2
Point x82 (value: 0.3380) is classified as Class1
```

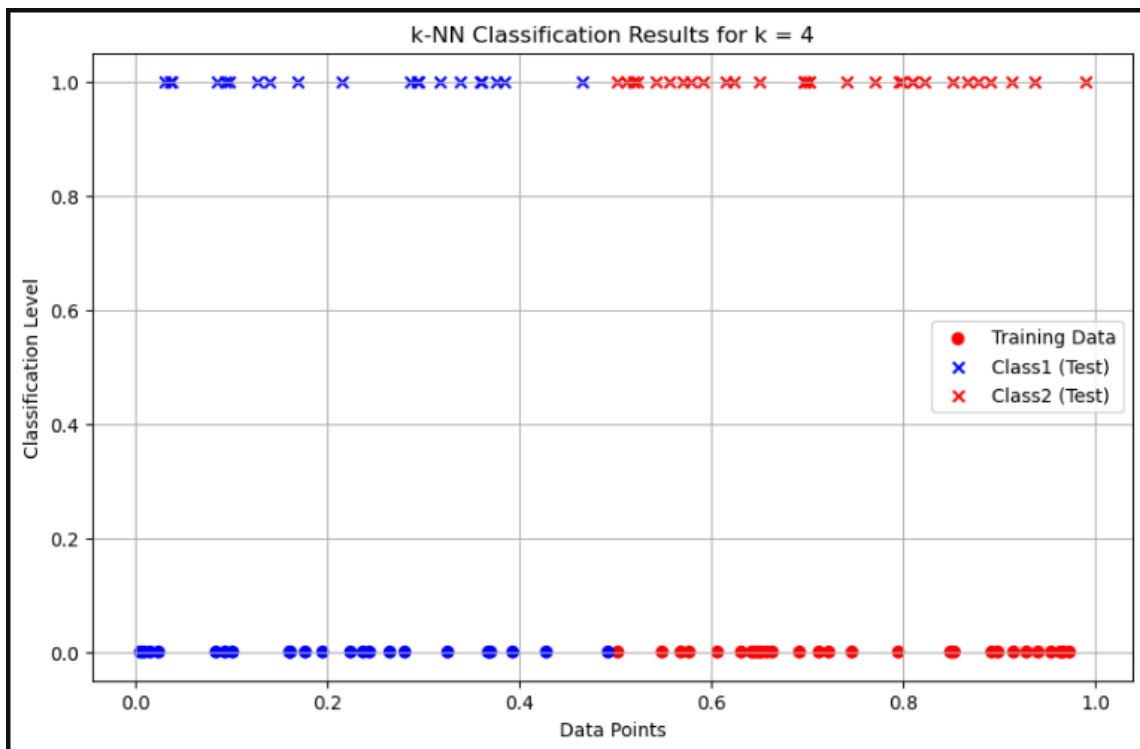
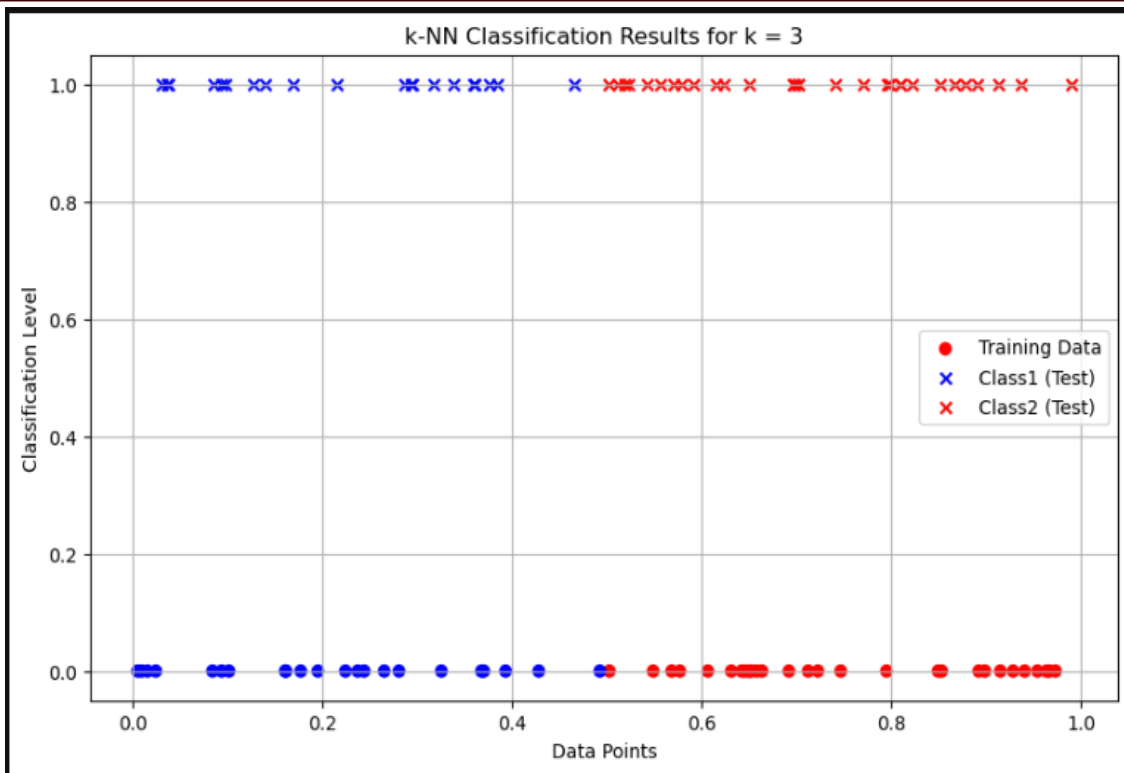
```
Results for k = 5:
Point x51 (value: 0.2944) is classified as Class1
Point x52 (value: 0.3851) is classified as Class1
Point x53 (value: 0.8511) is classified as Class2
Point x54 (value: 0.3169) is classified as Class1
Point x55 (value: 0.1695) is classified as Class1
Point x56 (value: 0.5568) is classified as Class2
Point x57 (value: 0.9362) is classified as Class2
Point x58 (value: 0.6960) is classified as Class2
Point x59 (value: 0.5701) is classified as Class2
Point x60 (value: 0.0972) is classified as Class1
Point x61 (value: 0.6150) is classified as Class2
Point x62 (value: 0.9901) is classified as Class2
Point x63 (value: 0.1401) is classified as Class1
Point x64 (value: 0.5183) is classified as Class2
Point x65 (value: 0.8774) is classified as Class2
Point x66 (value: 0.7408) is classified as Class2
Point x67 (value: 0.6970) is classified as Class2
Point x68 (value: 0.7025) is classified as Class2
Point x69 (value: 0.3595) is classified as Class1
Point x70 (value: 0.2936) is classified as Class1
Point x71 (value: 0.8094) is classified as Class2
Point x72 (value: 0.8101) is classified as Class2
Point x73 (value: 0.8671) is classified as Class2
Point x74 (value: 0.9132) is classified as Class2
Point x75 (value: 0.5113) is classified as Class2
Point x76 (value: 0.5015) is classified as Class2
Point x77 (value: 0.7983) is classified as Class2
Point x78 (value: 0.6500) is classified as Class2
Point x79 (value: 0.7020) is classified as Class2
Point x80 (value: 0.7958) is classified as Class2
Point x81 (value: 0.8900) is classified as Class2
Point x82 (value: 0.3380) is classified as Class1
```

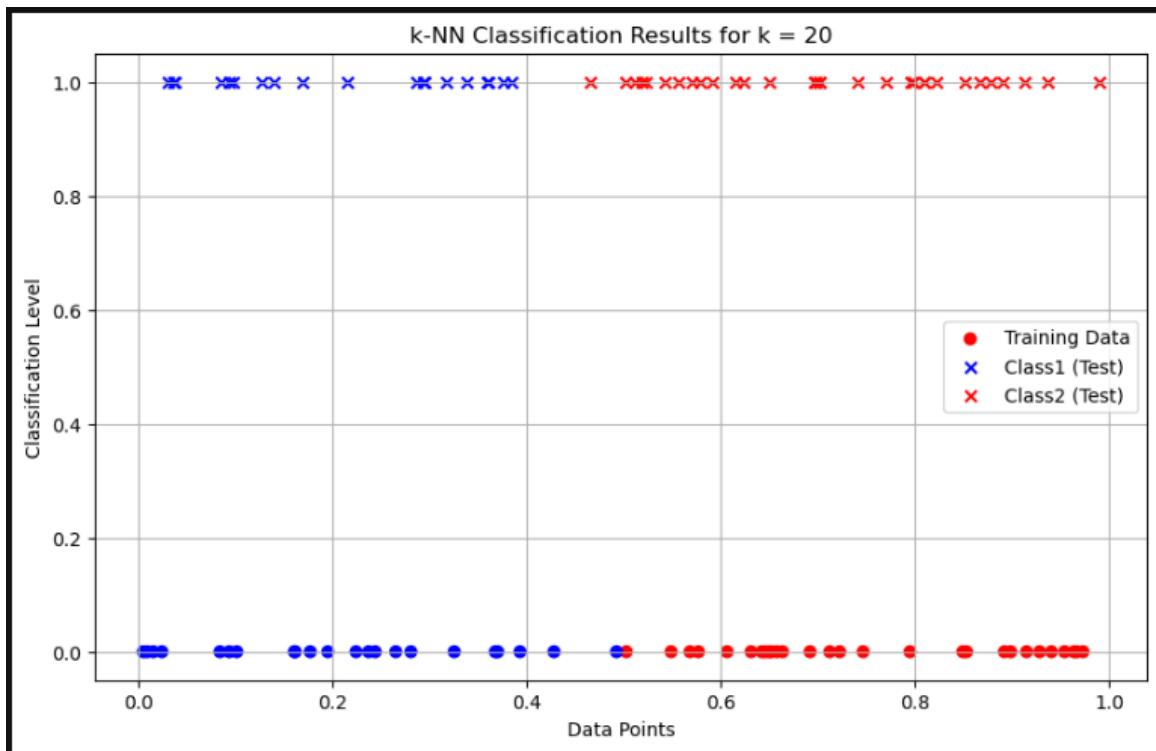
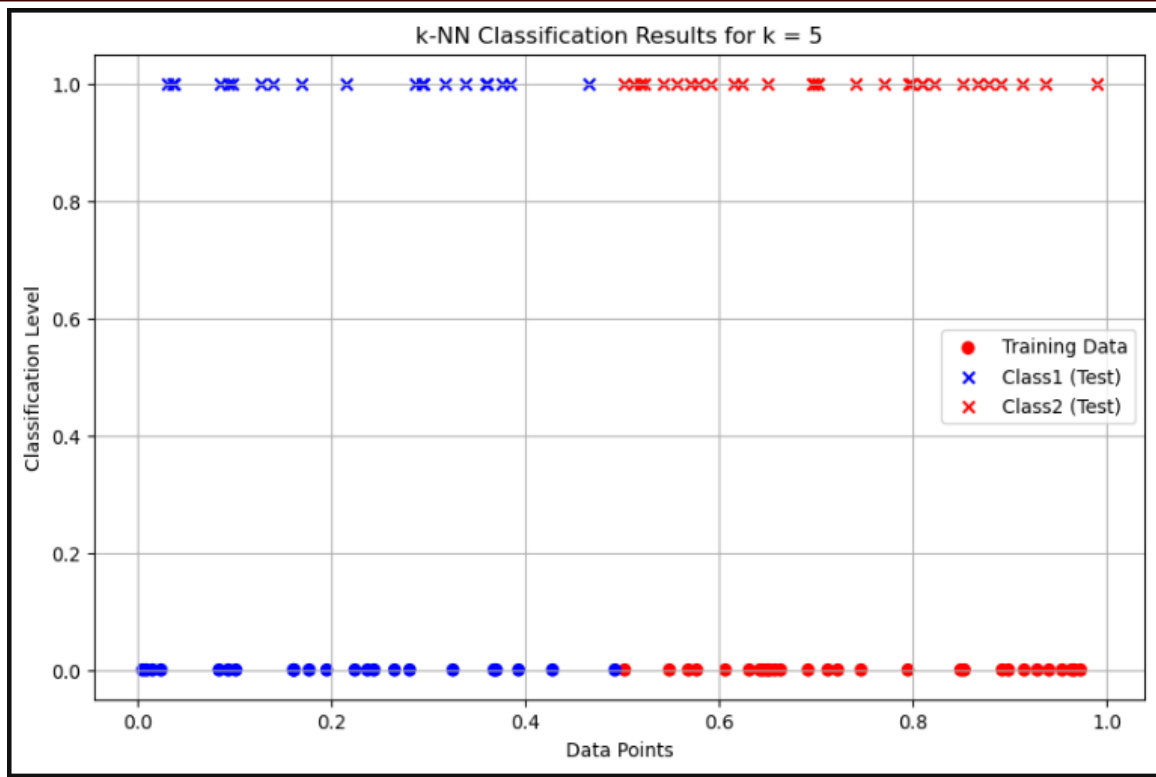
```
Results for k = 20:
Point x51 (value: 0.2944) is classified as Class1
Point x52 (value: 0.3851) is classified as Class1
Point x53 (value: 0.8511) is classified as Class2
Point x54 (value: 0.3169) is classified as Class1
Point x55 (value: 0.1695) is classified as Class1
Point x56 (value: 0.5568) is classified as Class2
Point x57 (value: 0.9362) is classified as Class2
Point x58 (value: 0.6960) is classified as Class2
Point x59 (value: 0.5701) is classified as Class2
Point x60 (value: 0.0972) is classified as Class1
Point x61 (value: 0.6150) is classified as Class2
Point x62 (value: 0.9901) is classified as Class2
Point x63 (value: 0.1401) is classified as Class1
Point x64 (value: 0.5183) is classified as Class2
Point x65 (value: 0.8774) is classified as Class2
Point x66 (value: 0.7408) is classified as Class2
Point x67 (value: 0.6970) is classified as Class2
Point x68 (value: 0.7025) is classified as Class2
Point x69 (value: 0.3595) is classified as Class1
Point x70 (value: 0.2936) is classified as Class1
Point x71 (value: 0.8094) is classified as Class2
Point x72 (value: 0.8101) is classified as Class2
Point x73 (value: 0.8671) is classified as Class2
Point x74 (value: 0.9132) is classified as Class2
Point x75 (value: 0.5113) is classified as Class2
Point x76 (value: 0.5015) is classified as Class2
Point x77 (value: 0.7983) is classified as Class2
Point x78 (value: 0.6500) is classified as Class2
Point x79 (value: 0.7020) is classified as Class2
Point x80 (value: 0.7958) is classified as Class2
Point x81 (value: 0.8900) is classified as Class2
```

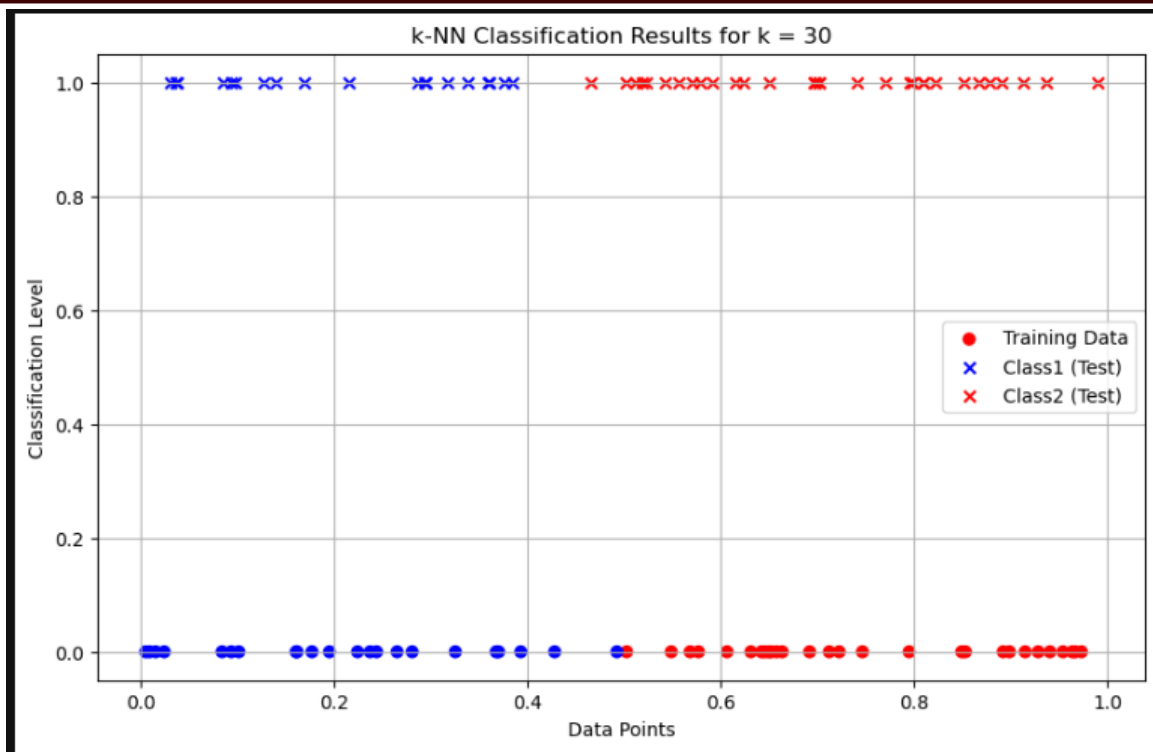
```
Results for k = 30:
Point x51 (value: 0.2944) is classified as Class1
Point x52 (value: 0.3851) is classified as Class1
Point x53 (value: 0.8511) is classified as Class2
Point x54 (value: 0.3169) is classified as Class1
Point x55 (value: 0.1695) is classified as Class1
Point x56 (value: 0.5568) is classified as Class2
Point x57 (value: 0.9362) is classified as Class2
Point x58 (value: 0.6960) is classified as Class2
Point x59 (value: 0.5701) is classified as Class2
Point x60 (value: 0.0972) is classified as Class1
Point x61 (value: 0.6150) is classified as Class2
Point x62 (value: 0.9901) is classified as Class2
Point x63 (value: 0.1401) is classified as Class1
Point x64 (value: 0.5183) is classified as Class2
Point x65 (value: 0.8774) is classified as Class2
Point x66 (value: 0.7408) is classified as Class2
Point x67 (value: 0.6970) is classified as Class2
Point x68 (value: 0.7025) is classified as Class2
Point x69 (value: 0.3595) is classified as Class1
Point x70 (value: 0.2936) is classified as Class1
Point x71 (value: 0.8094) is classified as Class2
Point x72 (value: 0.8101) is classified as Class2
Point x73 (value: 0.8671) is classified as Class2
Point x74 (value: 0.9132) is classified as Class2
Point x75 (value: 0.5113) is classified as Class2
Point x76 (value: 0.5015) is classified as Class2
Point x77 (value: 0.7983) is classified as Class2
Point x78 (value: 0.6500) is classified as Class2
Point x79 (value: 0.7020) is classified as Class2
Point x80 (value: 0.7958) is classified as Class2
Point x81 (value: 0.8900) is classified as Class2
Point x82 (value: 0.3380) is classified as Class1
Point x83 (value: 0.3756) is classified as Class1
Point x84 (value: 0.0940) is classified as Class1
Point x85 (value: 0.5783) is classified as Class2
Point x86 (value: 0.0359) is classified as Class1
Point x87 (value: 0.4656) is classified as Class2
Point x88 (value: 0.5426) is classified as Class2
Point x89 (value: 0.2865) is classified as Class1
Point x90 (value: 0.5908) is classified as Class2
Point x91 (value: 0.0305) is classified as Class1
Point x92 (value: 0.0373) is classified as Class1
Point x93 (value: 0.8226) is classified as Class2
Point x94 (value: 0.3602) is classified as Class1
Point x95 (value: 0.1271) is classified as Class1
Point x96 (value: 0.5222) is classified as Class2
Point x97 (value: 0.7700) is classified as Class2
Point x98 (value: 0.2158) is classified as Class1
Point x99 (value: 0.6229) is classified as Class2
Point x100 (value: 0.0853) is classified as Class1

Classification complete.
```











## Experiment No. 6

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

### Theory:

#### Locally Weighted Regression (LWR)

LWR is a **non-parametric** regression algorithm that gives more weight to nearby data points while making a prediction. This is achieved by using a kernel (typically a Gaussian kernel) to assign weights to the training points.

##### Mathematical Formulation

For a query point  $x_q$ , the prediction is obtained as:

$$\hat{y} = \theta^T X$$

where  $\theta$  is computed using **weighted least squares**:

$$\theta = (X^T W X)^{-1} X^T W y$$

- $W$  is a **diagonal weight matrix** where  $W_{ii} = \exp\left(\frac{-(x_i - x_q)^2}{2\tau^2}\right)$ .
- $\tau$  (bandwidth parameter) controls the locality: smaller  $\tau$  makes the model sensitive to nearby points.

### Explanation

1. **Data Generation:** Generates  $X$  values and their corresponding noisy  $y$  values using  $\sin(X)$ .
2. **Weight Matrix Calculation:** Uses a Gaussian kernel to assign weights based on distance.
3. **Locally Weighted Regression:** Computes weighted linear regression for each query point.
4. **Visualization:** Plots the original data along with LWR predictions for different values of  $\tau$ .

### Graph Analysis

- **Small tau (0.1)** → Highly localized, follows noise closely (overfitting).
- **Medium tau (0.5)** → Smoother curve, balances bias-variance tradeoff.
- **Large tau (1.0)** → More global averaging, leading to underfitting.

### Program:



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

def gaussian_kernel(X, x_query, tau):
    """Compute Gaussian weights for all training samples."""
    return np.exp(-np.sum((X - x_query) ** 2, axis=1) / (2 * tau ** 2))

def locally_weighted_regression(X, y, x_test, tau):
    """Perform Locally Weighted Regression (LWR)."""
    m, n = X.shape
    y_pred = np.zeros(x_test.shape[0])

    for i, x_q in enumerate(x_test):
        W = np.diag(gaussian_kernel(X, x_q, tau)) # Compute weights
        XTWX = X.T @ W @ X
        theta = np.linalg.inv(XTWX) @ X.T @ W @ y # Compute theta
        y_pred[i] = x_q @ theta # Predict y for x_q
    return y_pred

# Generate Data
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] # Add bias term

# Test points
x_test = np.linspace(0, 2 * np.pi, 200)
x_test_bias = np.c_[np.ones(x_test.shape), x_test]

# Experiment with different tau values
taus = [0.1, 0.5, 1.0]
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='red', label='Training Data', alpha=0.6)
```

for tau in taus:

```
y_pred = locally_weighted_regression(X_bias, y, x_test_bias, tau)
```

```
plt.plot(x_test, y_pred, label=f'LWR Fit (tau={tau})', linewidth=2)
```

```
plt.xlabel('X', fontsize=12)
```

```
plt.ylabel('y', fontsize=12)
```

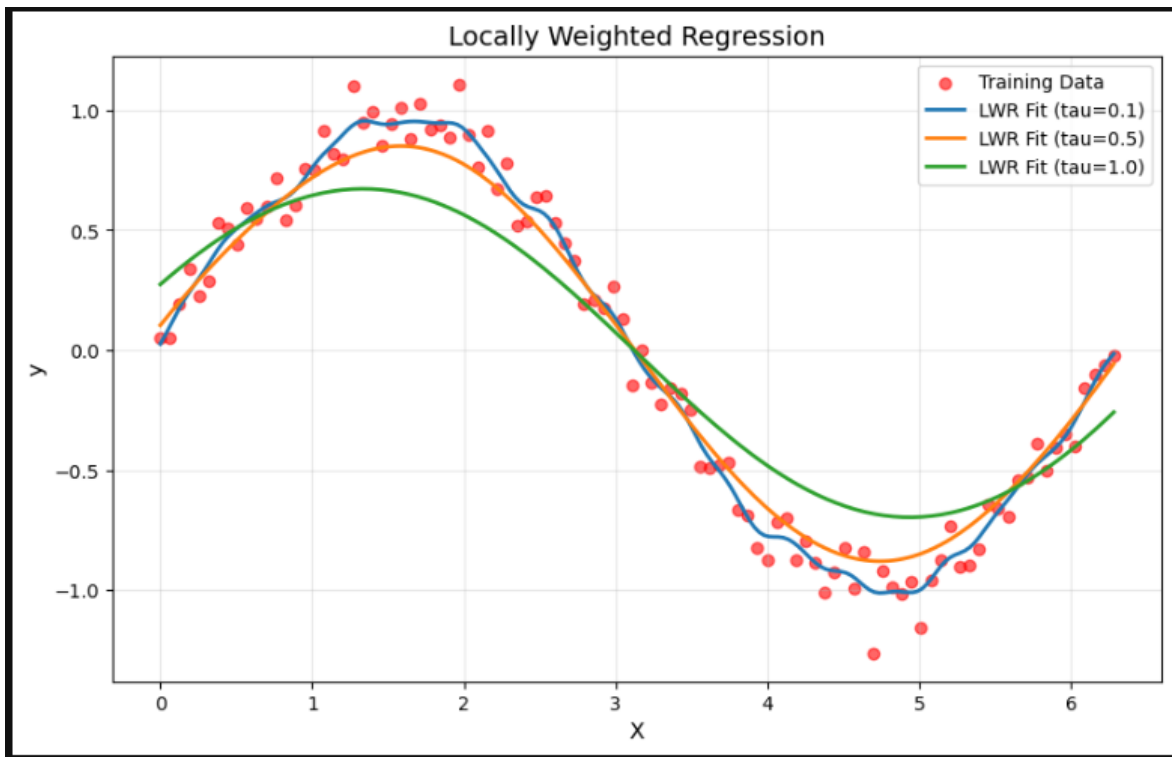
```
plt.title('Locally Weighted Regression', fontsize=14)
```

```
plt.legend(fontsize=10)
```

```
plt.grid(alpha=0.3)
```

```
plt.show()
```

### Output:



## Experiment No. 7

**Aim:** 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.

### Theory:

#### Linear Regression (Boston Housing Dataset)

Linear Regression models the relationship between a dependent variable (y) and an independent variable (X) using a straight-line equation:



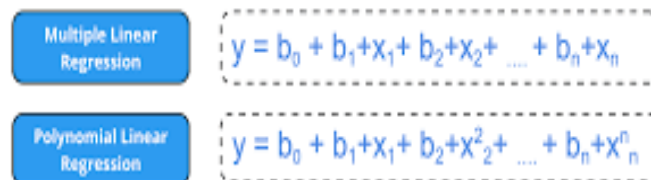
A diagram illustrating Simple Linear Regression. It consists of a solid blue rounded rectangle on the left containing the text "Simple Linear Regression". To its right is a dashed blue rounded rectangle containing the equation  $y = b_0 + b_1x_1$ .

#### Boston Housing Dataset:

Used to predict house prices based on features like crime rate, number of rooms, and accessibility.

#### Polynomial Regression (Auto MPG Dataset)

Polynomial Regression extends Linear Regression by adding polynomial terms:



A diagram illustrating Polynomial Regression. It shows two rows. The first row has a solid blue rounded rectangle labeled "Multiple Linear Regression" next to a dashed blue rounded rectangle containing the equation  $y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$ . The second row has a solid blue rounded rectangle labeled "Polynomial Linear Regression" next to a dashed blue rounded rectangle containing the equation  $y = b_0 + b_1x_1 + b_2x_2^2 + \dots + b_nx_n^n$ .

#### Auto MPG Dataset:

Used to predict vehicle fuel efficiency (MPG) based on features like engine displacement and weight.

### Conclusion

- **Linear Regression** is best for simple linear relationships (e.g., house prices).
- **Polynomial Regression** captures non-linear patterns (e.g., fuel efficiency).

### Program:

### a) Boston Housing Dataset for Linear Regression

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

# Load the dataset from CSV
data1 = "Boston_housing_dataset.csv"
data = pd.read_csv(data1)

# Display first few rows of the dataset
print(data.head())

# Define features and target variable
X = data.drop(columns=["medv"]) # Assuming 'MEDV' is the target column
y = data["medv"]

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared Score: {r2}")
```

```
# Plot actual vs predicted values
```

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

```
plt.scatter(y_test, y_pred, alpha=0.5, label="Actual vs Predicted")
```

```
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', label="Ideal Fit Line")
```

```
plt.xlabel("Actual Prices")
```

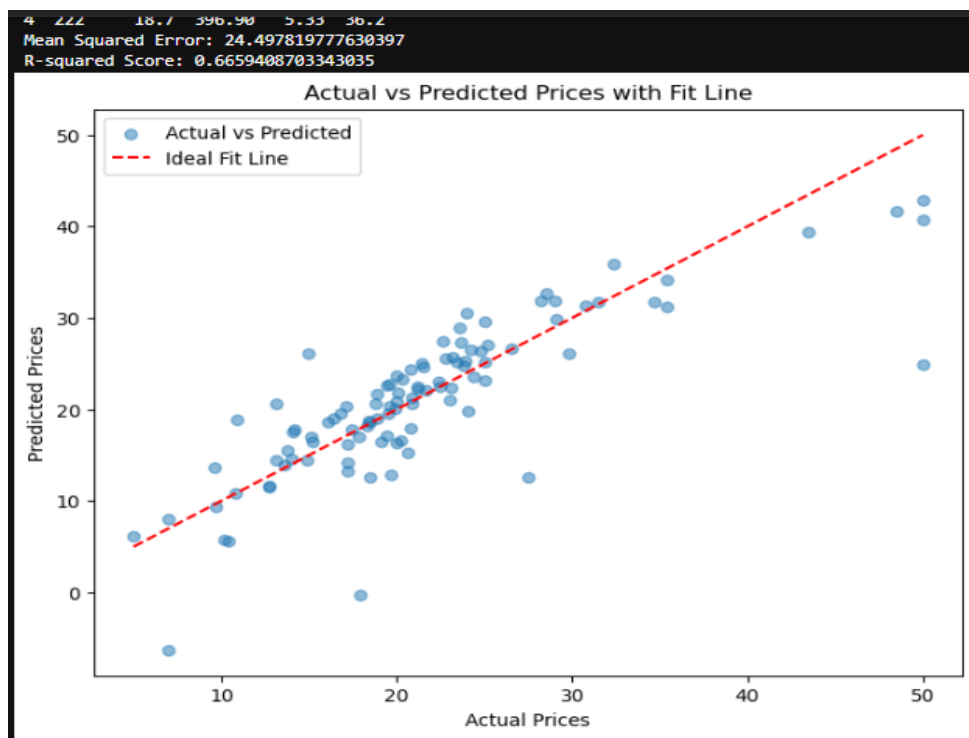
```
plt.ylabel("Predicted Prices")
```

```
plt.title("Actual vs Predicted Prices with Fit Line")
```

```
plt.legend()
```

```
plt.show()
```

### Output:



**b) Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.**

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

# Load the dataset
df = sns.load_dataset("mpg")

# Drop missing values
df = df.dropna()

# Select features and target variable
X = df["horsepower"].values.reshape(-1, 1) # Independent variable
y = df["mpg"].values # Dependent variable

# Convert horsepower to numeric (some versions have it as object)
X = X.astype(float)

# Apply Polynomial Transformation (degree 2 for quadratic fit)
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Train the model
model = LinearRegression()
model.fit(X_poly, y)

# Predict values
y_pred = model.predict(X_poly)

# Calculate R2 score and Mean Squared Error (MSE)
r2 = r2_score(y, y_pred)
```

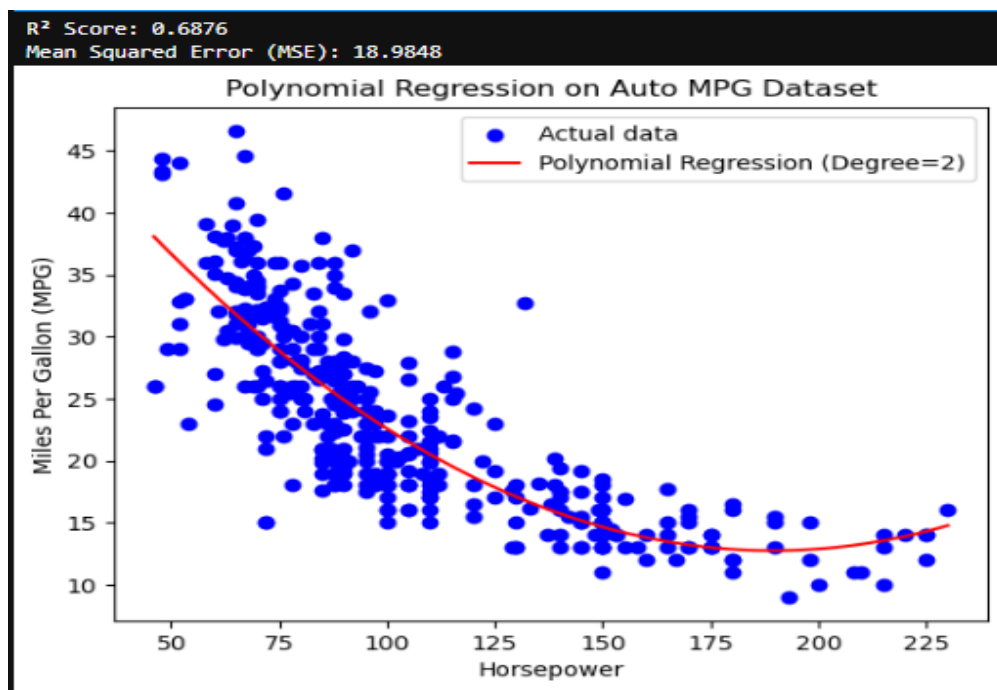
```
mse = mean_squared_error(y, y_pred)

# Print results
print(f"R2 Score: {r2:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")

# Visualization
X_range = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
X_range_poly = poly.transform(X_range)
y_range_pred = model.predict(X_range_poly)

plt.scatter(X, y, color='blue', label="Actual data")
plt.plot(X_range, y_range_pred, color='red', label="Polynomial Regression (Degree=2)")
plt.xlabel("Horsepower")
plt.ylabel("Miles Per Gallon (MPG)")
plt.title("Polynomial Regression on Auto MPG Dataset")
plt.legend()
plt.show()
```

### Output:



## Experiment No. 8

**Aim:** 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.

### Theory:

A **Decision Tree** is a supervised machine learning algorithm used for classification and regression tasks. It is a tree-like structure where:

- Each **internal node** represents a decision based on a feature.
- Each **branch** represents an outcome of the decision.
- Each **leaf node** represents a class label (for classification) or a numerical value (for regression).

### Working of Decision Tree Algorithm

1. **Selecting the Best Feature:** It selects the most significant feature to split the dataset based on metrics like:
  - **Gini Index:** Measures impurity (lower is better).
  - **Entropy (Information Gain):** Measures randomness in data (higher gain is better).
2. **Splitting the Dataset:** It divides the data into subsets based on the selected feature.
3. **Recursively Building the Tree:** The process repeats until a stopping condition is met (e.g., all nodes are pure or the tree reaches maximum depth).
4. **Classifying New Data:** To predict, the algorithm starts at the root node and follows the decision path to a leaf node.

### Classification Using Decision Tree

Classification is a supervised learning task where the goal is to categorize new data points into predefined labels (e.g., spam or not spam). The Decision Tree algorithm is widely used in classification due to its simplicity and interpretability.



### Steps for Classification Using a Decision Tree

1. **Collect and Preprocess Data:** Clean the dataset, handle missing values, and encode categorical variables if needed.
2. **Split the Data:** Divide the dataset into **training** and **testing** sets.
3. **Train the Decision Tree Model:** Use training data to construct the tree.
4. **Make Predictions:** Use the trained model to predict the class of test data.
5. **Evaluate Performance:** Measure accuracy using metrics like:
  - **Confusion Matrix**
  - **Accuracy, Precision, Recall, F1-score**
  - **ROC-AUC Curve (for binary classification)**

### Program:

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

# Load Breast Cancer dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Decision Tree model
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

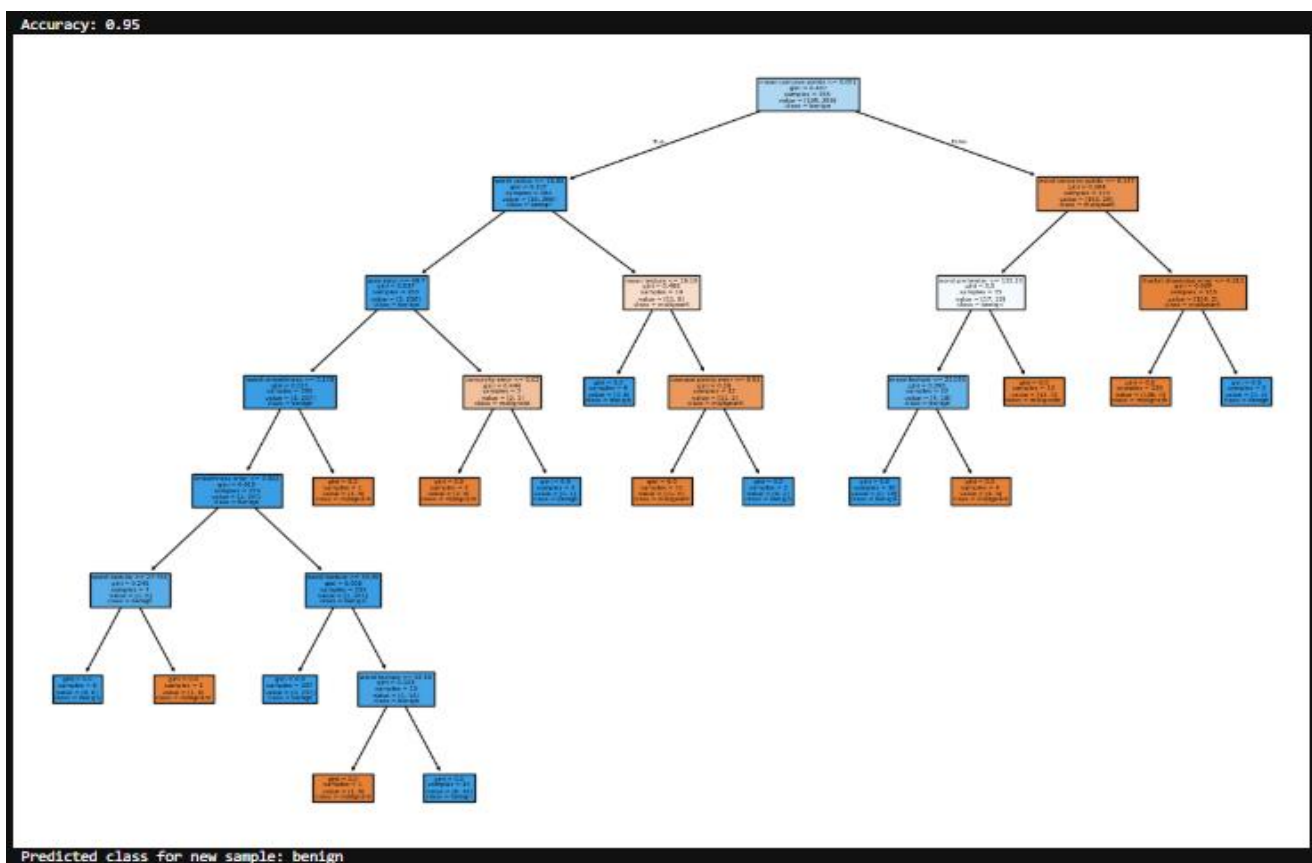
# Evaluate the model
```

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

# Visualizing the Decision Tree
plt.figure(figsize=(15, 10))
plot_tree(clf, filled=True, feature_names=data.feature_names, class_names=data.target_names)
plt.show()

# Classify a new sample
new_sample = np.array([X_test[0]]) # Taking first test sample
predicted_class = clf.predict(new_sample)
print(f"Predicted class for new sample: {data.target_names[predicted_class[0]]}")
```

### Output:



## Experiment No. 9

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

### Theory:

#### Naïve Bayes Classifier

Naïve Bayes is a **probabilistic classifier** based on **Bayes' Theorem**, assuming that features are **independent** (hence "naïve"). It is particularly useful for **classification tasks** like text classification, spam detection, and face recognition.

#### Bayes' Theorem Formula:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Where:

- $P(A|B)P(A | B) \rightarrow$  Probability of class **A** given features **B** (posterior).
- $P(B|A)P(B | A) \rightarrow$  Probability of features **B** given class **A** (likelihood).
- $P(A)P(A) \rightarrow$  Prior probability of class **A**.
- $P(B)P(B) \rightarrow$  Probability of features **B** (evidence).

#### Types of Naïve Bayes Classifiers:

1. **Gaussian Naïve Bayes** – Assumes data follows a normal distribution.
2. **Multinomial Naïve Bayes** – Used for text classification (word frequencies).
3. **Bernoulli Naïve Bayes** – Works with binary features (e.g., spam detection).

#### Olivetti Faces Dataset (Short Overview)

- **Contains:** 400 grayscale images (40 people, 10 images per person).
- **Image Size:**  $64 \times 64$  pixels (flattened into 4096 features per image).

- **Usage:** Face recognition, machine learning classification, and image processing tasks.
- **Available in:** `sklearn.datasets.fetch_olivetti_faces`.

### **Program:**

```
# Import necessary libraries
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 # Evaluation
metrics

import matplotlib.pyplot as plt

# Load the Olivetti Faces dataset (400 images of 40 people, each with 10 images)
data = fetch_olivetti_faces(shuffle=True, random_state=42)

# Extract features (X) and labels (y)
X = data.data # Flattened image pixels (64x64 => 4096 features per image)
y = data.target # Labels (40 unique individuals)

# Split the dataset into training (70%) and testing (30%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Gaussian Naïve Bayes classifier
gnb = GaussianNB()

# Train the classifier using the training set
gnb.fit(X_train, y_train)

# Predict labels for the test set
y_pred = gnb.predict(X_test)

# Calculate accuracy of the model
```

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

# Print the classification report (precision, recall, f1-score)
print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=1))

# Print the confusion matrix to analyze misclassifications
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Perform 5-fold cross-validation to evaluate model performance
cross_val_accuracy = cross_val_score(gnb, X, y, cv=5, scoring='accuracy')
print(f'\nCross-validation accuracy: {cross_val_accuracy.mean() * 100:.2f}%')

# Visualizing some test images with predicted and actual labels
fig, axes = plt.subplots(3, 5, figsize=(12, 8)) # Create a 3x5 grid for images
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) # Reshape 1D array back to 64x64 image
    ax.set_title(f"True: {label}, Pred: {prediction}") # Display actual and predicted labels
    ax.axis('off') # Hide axes for better visualization

# Show the plotted images
plt.show()
```

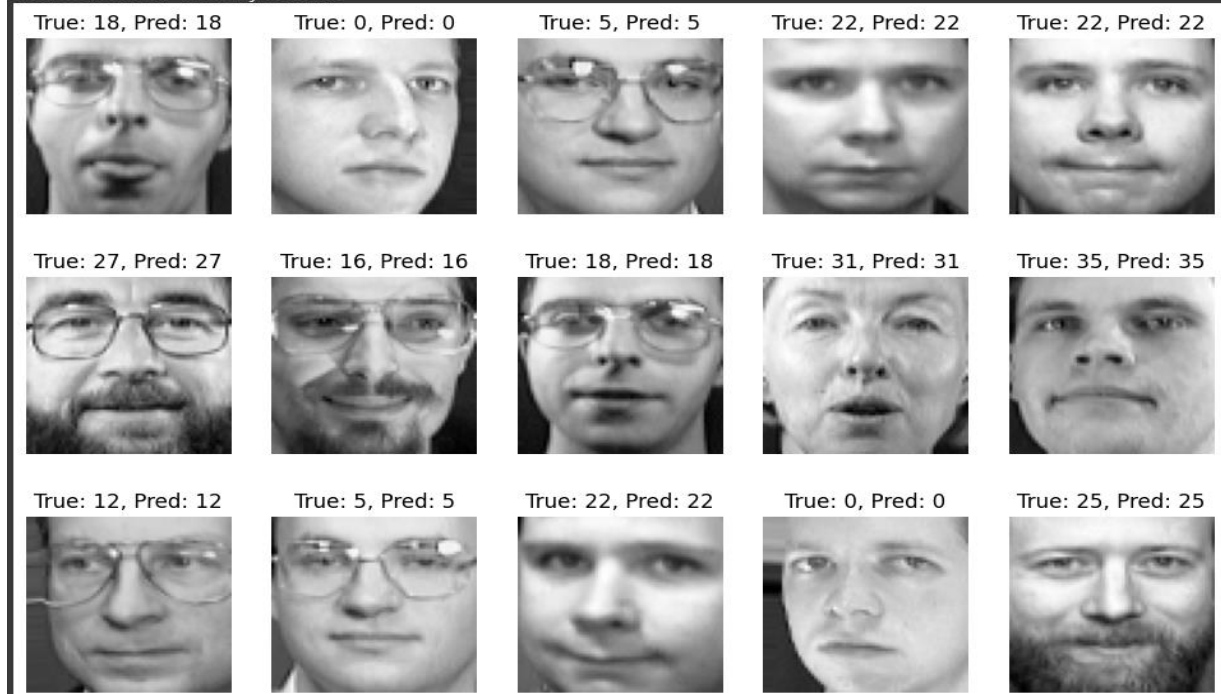
Output:

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

Cross-validation accuracy: 87.25%



## Experiment No. 10

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

### Theory:

#### **K-Means Clustering:**

K-Means is an unsupervised machine learning algorithm used to group similar data points into **K clusters**. It works by:

1. **Choosing K cluster centers (centroids)** randomly.
2. **Assigning each data point** to the nearest centroid based on Euclidean distance.
3. **Updating centroids** by computing the mean of all points assigned to each cluster.
4. **Repeating steps 2 & 3** until centroids no longer change significantly or a stopping criterion is met.

This method is widely used in pattern recognition and data segmentation tasks.

#### **Wisconsin Breast Cancer Dataset (WBCD)**

The **Wisconsin Breast Cancer Dataset (WBCD)** is a well-known medical dataset used for cancer detection. It contains **569 samples** with **30 numerical features** extracted from cell nuclei images obtained through **fine needle aspiration (FNA) biopsy**. These features describe **the shape, texture, and compactness** of the cell nuclei.

The dataset has **two target classes**:

- **Benign (0)** – Non-cancerous tumors.
- **Malignant (1)** – Cancerous tumors.

Since **K-Means is an unsupervised algorithm**, it does not use the actual labels but groups similar data points into **two clusters**. The accuracy of clustering is later evaluated by comparing predicted clusters with the true labels using a **confusion matrix** and **classification report**.

### **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

# Load the dataset
data = load_breast_cancer()
X = data.data # Features
y = data.target # Actual labels

# Standardizing the data for better clustering performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying K-Means Clustering with 2 clusters (since we have two classes)
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
y_kmeans = kmeans.fit_predict(X_scaled)

# Printing Confusion Matrix and Classification Report
print("Confusion Matrix:")
print(confusion_matrix(y, y_kmeans))
print("\nClassification Report:")
print(classification_report(y, y_kmeans))

# Performing PCA for visualization (reducing to 2D)
```



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

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

# Plotting the K-Means Clusters
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()

# Plotting True Labels for comparison
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()

# Plotting Clusters with Centroids
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:  
[[ 36 176]  
 [339  18]]  
  
Classification Report:  
              precision    recall  f1-score   support  
  
      0       0.10       0.17       0.12       212  
      1       0.09       0.05       0.07       357  
  
 accuracy          0.09          0.11          0.09          569  
 macro avg          0.09          0.11          0.09          569  
 weighted avg       0.09          0.09          0.09          569
```

