ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ, ಬೆಳಗಾವಿ

VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELAGAVI

Department of Computer Science and Engineering

"Jnana Sangama", VTU-Campus, Belagavi-590018

LAB-MANUAL

Academic Year: 2025-26

Name	:	**Manoj M J**
USN	:	**2VX22CB030**
Sem	:	**6th**
Subject	:	**Machine Learning lab**
Code	:	**BCSL606**
Course	:	**B.tech**
Programme	:	**Computer Science and Business System**

Certificate

This is certify that Mr./Mrs. Manoj M J with USN 2VX22CB030 has satisfactorily completed all the Laboratory Assignment of Subject Machine Learning lab having Subject Code BCSL606 during the academic year 2025-26.

Faculty in-charge

Signature of the Examiners

INDEX

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.

Visualization and Outlier Detection in the California Housing Dataset

In this study, the California Housing dataset was used to analyze the distribution and outlier characteristics of its numerical features. First, histograms were created for all features to visualize their data distribution. The histograms revealed that features such as MedInc (Median Income) and HouseAge were right-skewed, indicating the presence of high values in fewer samples. Features like AveRooms and AveOccup also showed long tails, suggesting a wide range of values across different households.

Next, box plots were generated for each feature to identify potential outliers. Outliers appeared prominently in features such as AveRooms, AveOccup, and Population. These plots allowed visual detection of unusually high values that could affect model performance or require preprocessing steps like transformation or capping.

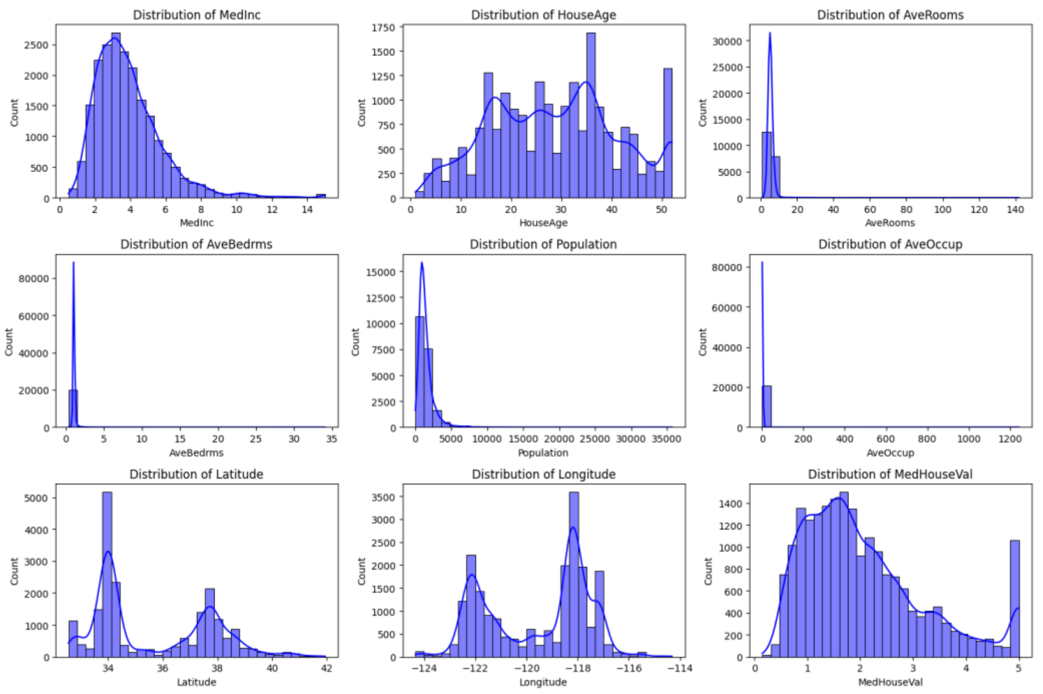
This procedure provided a comprehensive understanding of data distribution and highlighted the need for normalization or outlier handling before training machine learning models

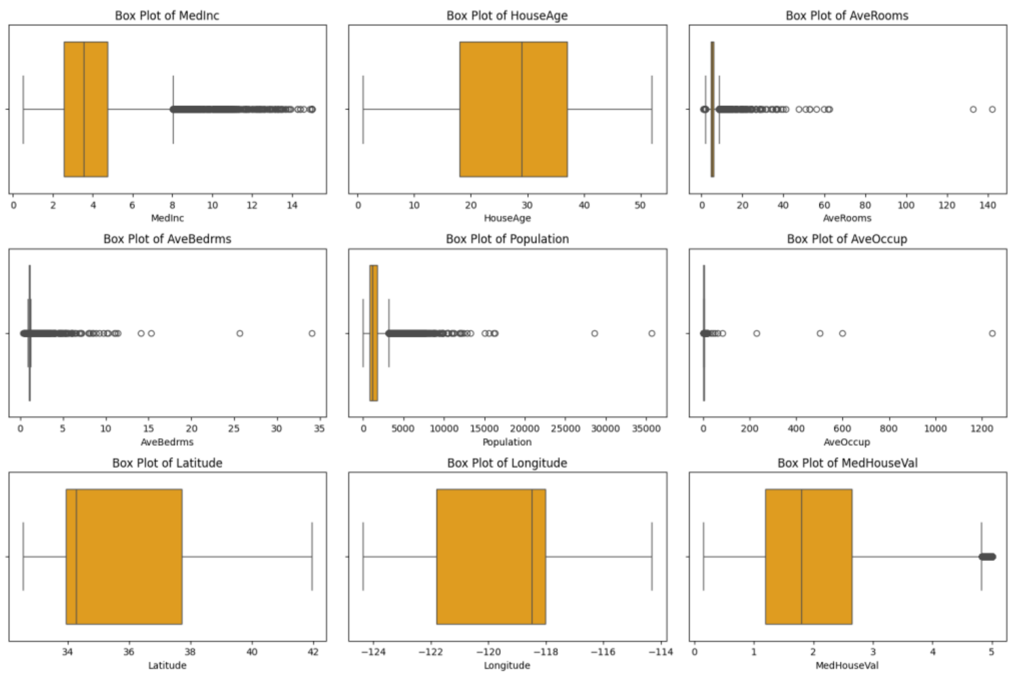
Source Code :

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

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

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

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

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

# Step 4: Identify outliers using the IQR method
```

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

# Optional: Print a summary of the dataset
print("\nDataset Summary:")
print(housing_df.describe())
```

Output :





```
Outliers Detection:
MedInc: 681 outliers
HouseAge: 0 outliers
AveRooms: 511 outliers
AveBedrms: 1424 outliers
Population: 1196 outliers
AveOccup: 711 outliers
Latitude: 0 outliers
Longitude: 0 outliers
MedHouseVal: 1071 outliers

Dataset Summary:
             MedInc      HouseAge      AveRooms     AveBedrms    Population
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

           AveOccup      Latitude     Longitude   MedHouseVal
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429741     33.930000   -121.800000      1.196000
50%        2.818116     34.260000   -118.490000      1.797000
75%        3.282261     37.710000   -118.010000      2.647250
max     1243.333333     41.950000   -114.310000      5.000010
```

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.

Correlation Analysis and Pairwise Visualization in California Housing Dataset

The California Housing dataset was analyzed to explore the relationships between its numerical features. First, a correlation matrix was computed to evaluate how strongly each pair of variables was linearly related. This matrix was visualized using a heatmap, which provided a color-coded overview of positive and negative correlations. Notably, MedInc (median income) showed a strong positive correlation with MedHouseVal (median house value), while AveOccup had a weak negative correlation.

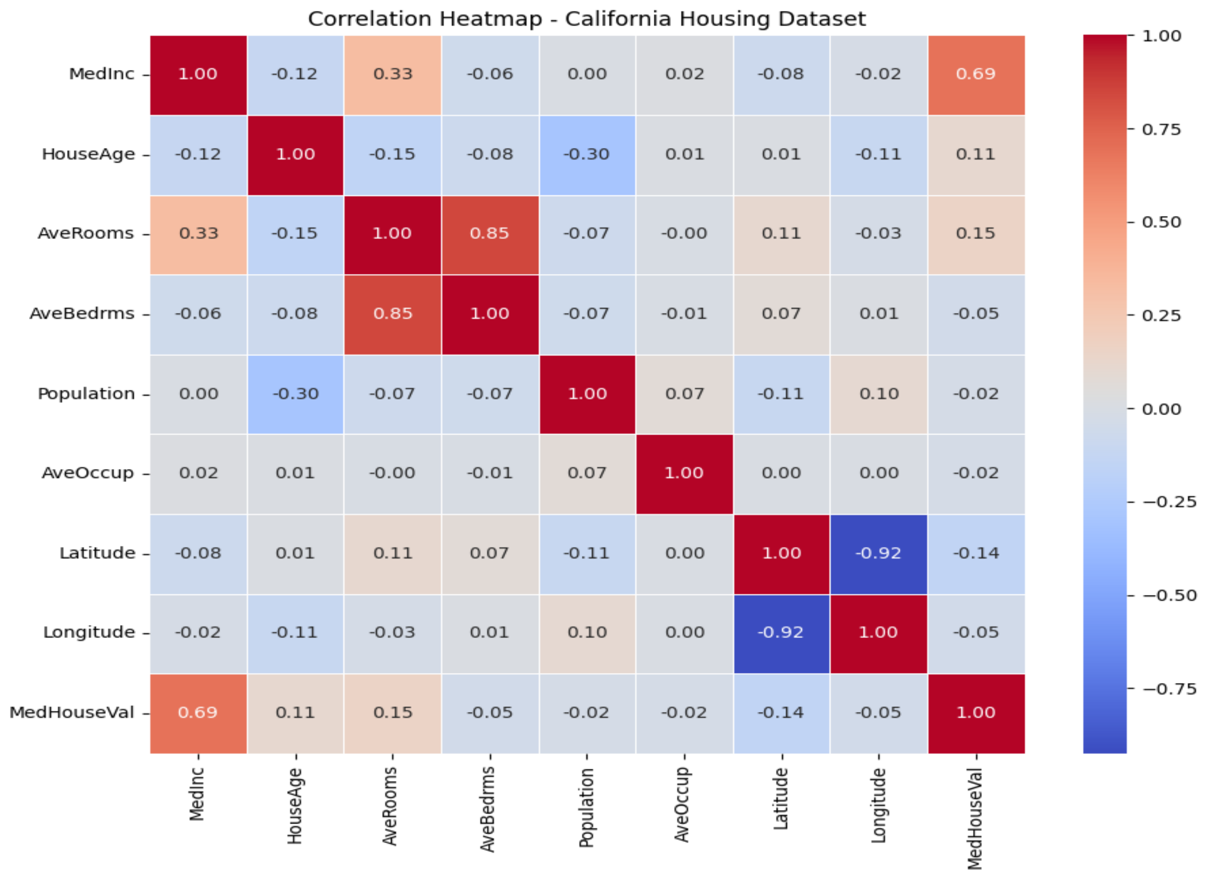
To further investigate these relationships, a pair plot was generated focusing on the top five features most correlated with MedHouseVal. The pair plot revealed visible trends, clusters, and some non-linear relationships, particularly between MedInc and MedHouseVal. These visualizations help identify redundant features, potential multicollinearity, and patterns useful for feature selection and model improvement.

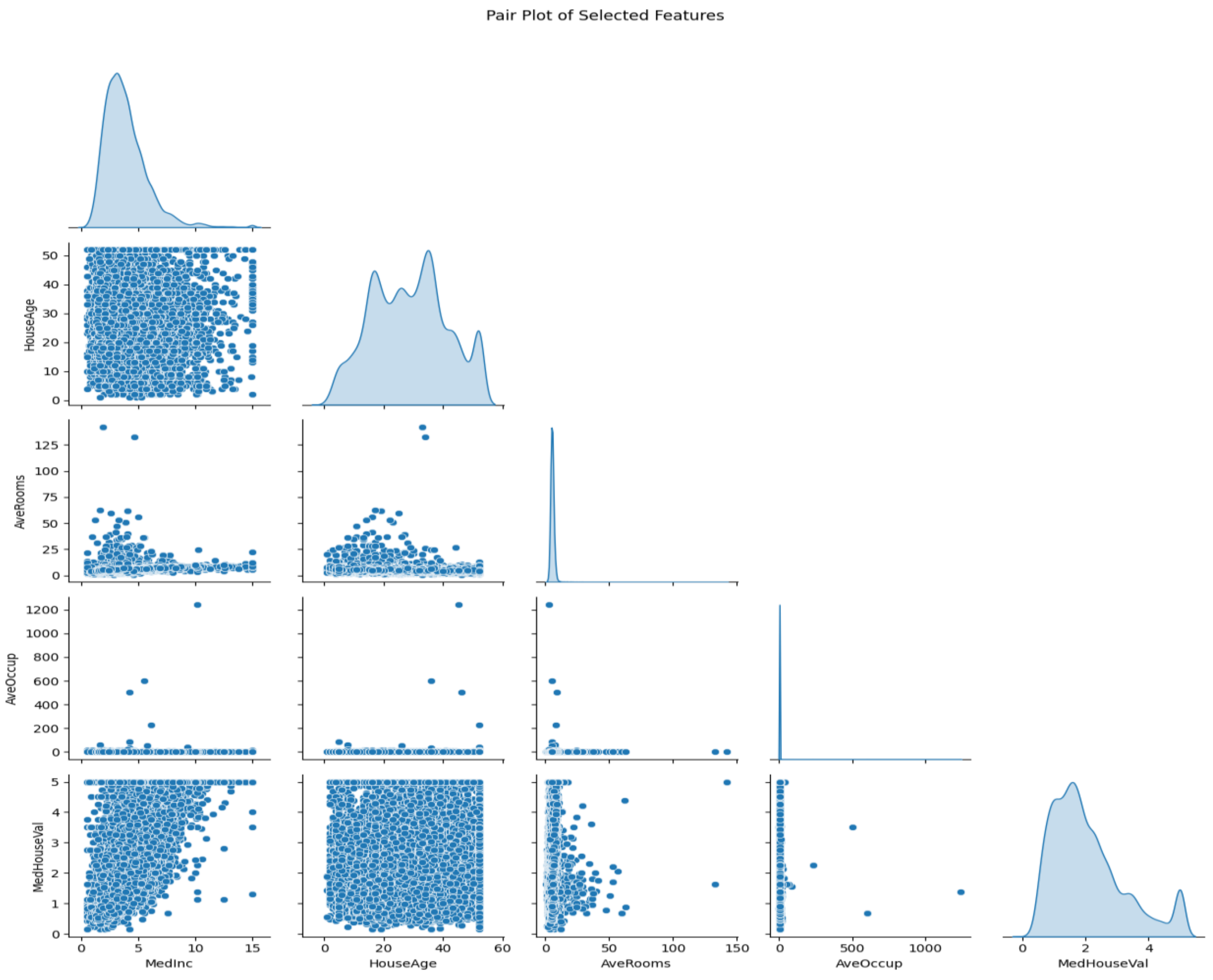
Overall, this analysis supports more informed preprocessing and feature engineering in predictive modeling workflows.

Source Code :

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

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

# Compute correlation matrix
correlation_matrix = df.corr()

# Display the correlation matrix
print("Correlation Matrix:\n", correlation_matrix)

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, linewidths=.5)
plt.title("Correlation Heatmap - California Housing Dataset")
plt.tight_layout()
plt.show()

# Plot pairplot (optional: limit columns for better performance and readability)
selected_features = ['MedInc', 'HouseAge', 'AveRooms', 'AveOccup', 'MedHouseVal']
sns.pairplot(df[selected_features], corner=True, diag_kind='kde')
plt.suptitle("Pair Plot of Selected Features", y=1.02)
plt.tight_layout()
plt.show()
```

Output :


Correlation Heatmap - California Housing Dataset
MedInc 1.00 -0.12 0.33 -0.06 0.00 0.02 -0.08 -0.02 0.69
HouseAge -0.12 1.00 -0.15 -0.08 -0.30 0.01 0.01 -0.11 0.11
AveRooms 0.33 -0.15 1.00 0.85 -0.07 -0.00 0.11 -0.03 0.15
AveBedrms -0.06 -0.08 0.85 1.00 -0.07 -0.01 0.07 0.01 -0.05
Population 0.00 -0.30 -0.07 -0.07 1.00 0.07 -0.11 0.10 -0.02
AveOccup 0.02 0.01 -0.00 -0.01 0.07 1.00 0.00 0.00 -0.02
Latitude -0.08 0.01 0.11 0.07 -0.11 0.00 1.00 -0.92 -0.14
Longitude -0.02 -0.11 -0.03 0.01 0.10 0.00 -0.92 1.00 -0.05
MedHouseVal 0.69 0.11 0.15 -0.05 -0.02 -0.02 -0.14 -0.05 1.00
MedInc HouseAge AveRooms AveBedrms Population AveOccup Latitude Longitude MedHouseVal
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75


Pair Plot of Selected Features
HouseAge
AveRooms
AveOccup
MedHouseVal
MedInc
HouseAge
AveRooms
AveOccup
MedHouseVal

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

Dimensionality Reduction of the Iris Dataset Using PCA

In this experiment, Principal Component Analysis (PCA) was applied to the Iris dataset to reduce its dimensionality from four features (sepal length, sepal width, petal length, petal width) to two principal components. PCA is a linear dimensionality reduction technique that projects data into a lower-dimensional space while preserving as much variance as possible.

The sklearn.decomposition.PCA module was used to transform the data. The two resulting components explained the majority of the variance in the dataset. A scatter plot of the transformed data revealed clear clustering patterns corresponding to the three Iris species (Setosa, Versicolor, and Virginica). This confirms that even with dimensionality reduction, the data retains its essential class-distinguishing structure.

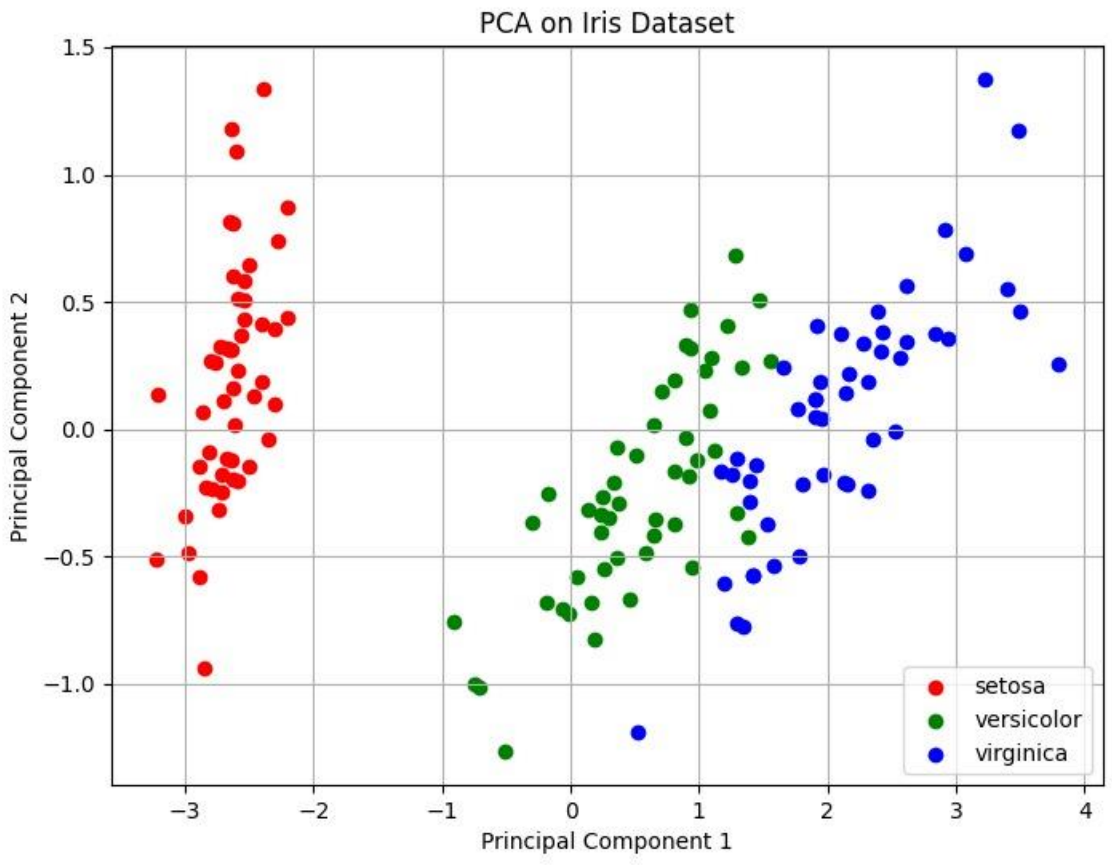
PCA is a valuable preprocessing technique for visualization and model simplification, especially when working with high-dimensional datasets. It helps reduce computational complexity and may improve model performance by eliminating redundant or correlated features

Source Code :

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

# Load the Iris dataset
iris = load_iris()
data = iris.data
labels = iris.target
label_names = iris.target_names

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

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

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

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

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

Output :



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.

Concept Learning Using the Find-S Algorithm

In this experiment, the Find-S algorithm was implemented to identify the most specific hypothesis consistent with a set of positive training examples. The training data was loaded from a .csv file containing categorical attributes related to a decision problem (e.g., playing sports based on weather conditions).

Source Code :

```
import pandas as pd

def find_s_algorithm():
    data = pd.DataFrame({
        'Sky': ['Sunny', 'Sunny', 'Cloudy', 'Rainy', 'Sunny'],
        'Temperature': ['Warm', 'Hot', 'Warm', 'Cold', 'Warm'],
        'Humidity': ['Normal', 'High', 'High', 'Normal', 'Normal'],
        'Wind': ['Strong', 'Weak', 'Strong', 'Strong', 'Weak'],
        'PlayTennis': ['Yes', 'No', 'Yes', 'No', 'Yes']  # Target column
    })

    print("Training data:")
    print(data)

    attributes = data.columns[:-1]  # All columns except the last one
    class_label = data.columns[-1]  # The last column is the target variable

    hypothesis = ['?' for _ in attributes]  # Initialize with the most general hypothesis

    for index, row in data.iterrows():
        if row[class_label] == 'Yes':  # Process only positive examples
            for i, value in enumerate(row[attributes]):
                if hypothesis[i] == '?' or hypothesis[i] == value:
                    hypothesis[i] = value  # Retain attribute value if it matches
                else:
                    hypothesis[i] = '?'  # Generalize otherwise

    return hypothesis
```

Output :

```
hypothesis = find_s_algorithm()
print("\nThe final hypothesis is:", hypothesis)

Training data:
      Sky Temperature Humidity    Wind PlayTennis
0   Sunny        Warm   Normal  Strong        Yes
1   Sunny         Hot     High    Weak         No
2  Cloudy        Warm     High  Strong        Yes
3   Rainy        Cold   Normal  Strong         No
4   Sunny        Warm   Normal    Weak        Yes
The final hypothesis is: ['Sunny', 'Warm', 'Normal', '?']
```

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

   a. Label the first 50 points {x1,......,x50} as follows: if (xi ≤ 0.5), then xi ∊ Class1, else xi ∊ Class1

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

<u>Classification of Random Points Using k-Nearest Neighbour Algorithm</u>

This experiment demonstrates the use of the k-Nearest Neighbour (k-NN) algorithm for classifying points in the range [0,1][0, 1][0,1]. A total of 100 values were randomly generated. The first 50 were labeled into two classes: Class 1 if the value was less than or equal to 0.5, and Class 2 otherwise. The remaining 50 values were unlabelled and used for testing.

The k-NN algorithm was applied using different values of k=1,2,3,4,5,20,30k = 1, 2, 3, 4, 5, 20, 30k=1,2,3,4,5,20,30 to classify these test points. The model was trained using the first 50 labeled samples and predicted the class of the remaining 50 by examining the majority class of the nearest kkk neighbors.

The results were visualized to observe how different values of kkk affect the smoothness and accuracy of classification. Lower values of kkk often led to overfitting, while higher values showed better generalization but could blur class boundaries.

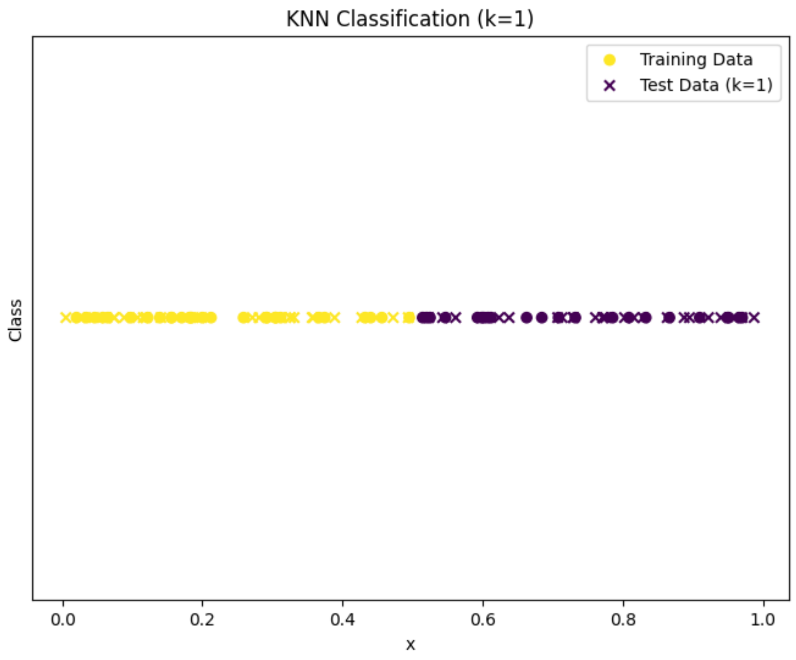
Source Code :

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

def knn_classify(train_x, train_y, test_x, k):
    """
    Classifies test_x using k-Nearest Neighbors algorithm.

    Args:
        train_x: Training data features (1D array).
        train_y: Training data labels (1D array).
        test_x: Test data features (1D array).
        k: Number of neighbors to consider.

    Returns:
        Predicted labels for test_x (1D array).
    """

    predictions = []
    for test_point in test_x:
        distances = np.abs(train_x - test_point)  # Calculate absolute distances
        nearest_indices = np.argsort(distances)[:k]  # Find indices of k nearest neighbors
        nearest_labels = train_y[nearest_indices]

        # Determine the most frequent label among the k neighbors
        unique_labels, counts = np.unique(nearest_labels, return_counts=True)
        predicted_label = unique_labels[np.argmax(counts)]
        predictions.append(predicted_label)
    return np.array(predictions)
```

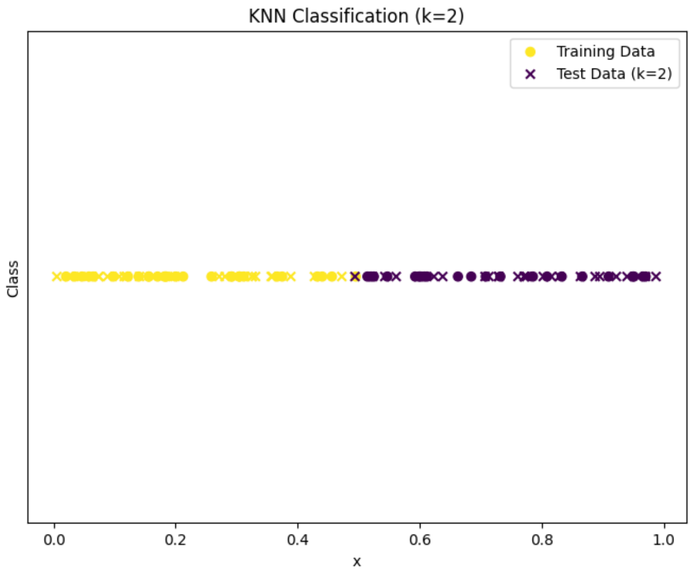
```
# Generate 100 random values in the range [0, 1]
np.random.seed(42)  # For reproducibility
x = np.random.rand(100)

# Label the first 50 points
y = np.zeros(100)
y[:50] = (x[:50] <= 0.5).astype(int) + 1 #Class 1 if <= 0.5, Class 2 otherwise
y[50:] = -1 # initialize test values to -1, they will be overriden by knn prediction

# Split data into training and test sets
train_x, train_y = x[:50], y[:50]
test_x = x[50:]

# Classify the remaining points using KNN for different values of k
k_values = [1, 2, 3, 4, 5, 20, 30]
predictions = {}

for k in k_values:
    predictions[k] = knn_classify(train_x, train_y, test_x, k)
    y[50:] = predictions[k]
    print(f"Predictions for k={k}: {predictions[k]}")

    # Plot the results for each k
    plt.figure(figsize=(8, 6))
    plt.scatter(train_x, np.zeros_like(train_x), c=train_y, marker='o', label='Training Data')
    plt.scatter(test_x, np.zeros_like(test_x), c=predictions[k], marker='x', label=f'Test Data (k={k})')
    plt.xlabel('x')
    plt.ylabel('Class')
    plt.title(f'KNN Classification (k={k})')
    plt.legend()
    plt.yticks([]) #remove y axis ticks
        plt.show()
```

Output :

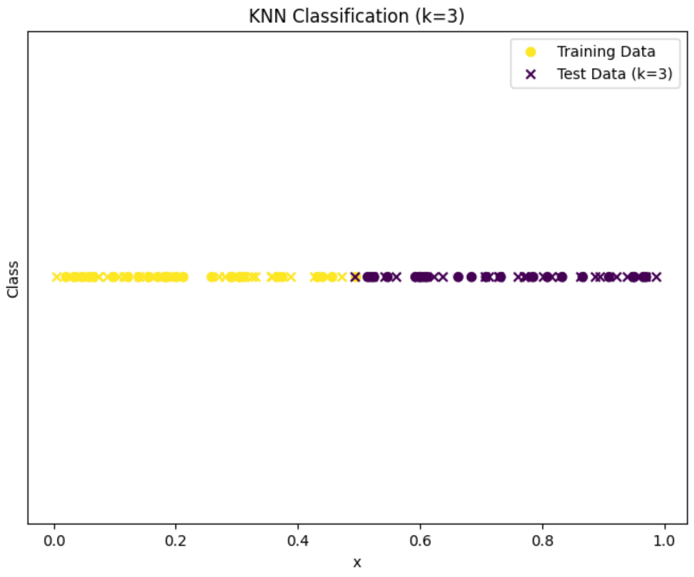
```
Predictions for k=1: [1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1. 2. 2. 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 2. 2.
1. 1. 1. 2. 2. 1. 1. 1. 1. 2. 1. 2. 2. 2.]
```



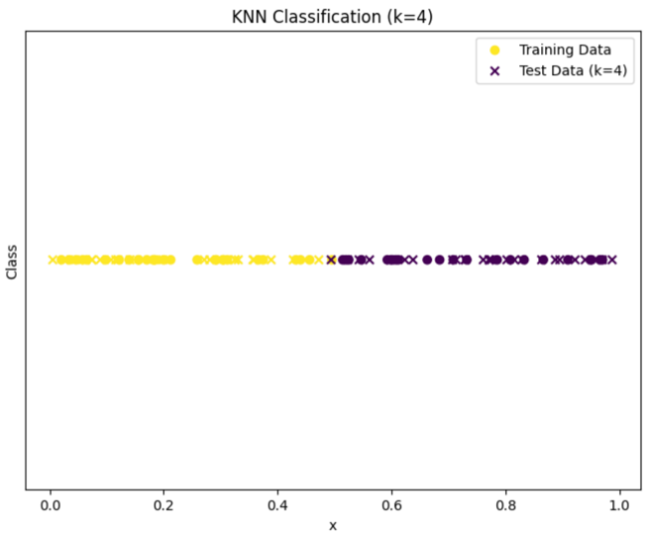
```
Predictions for k=2: [1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1.
2. 2. 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 2.
2. 1. 1. 1. 2. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2.]
```



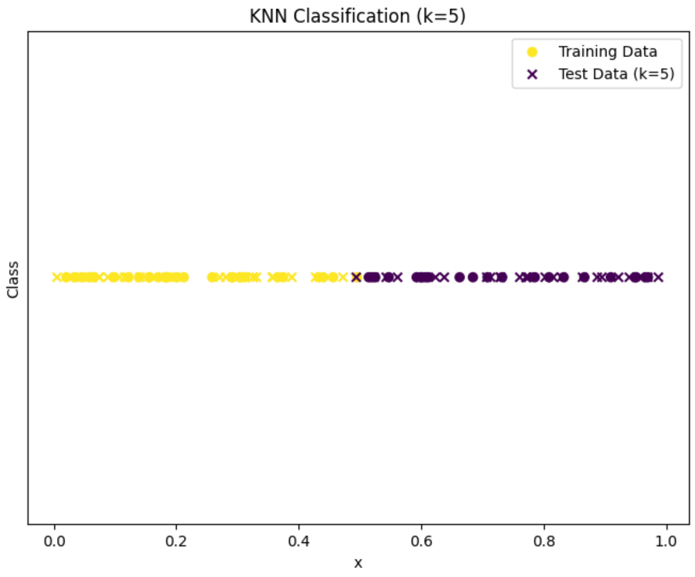
```
Predictions for k=3: [1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1.
2. 2. 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 2.
2. 1. 1. 1. 2. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2.]
```



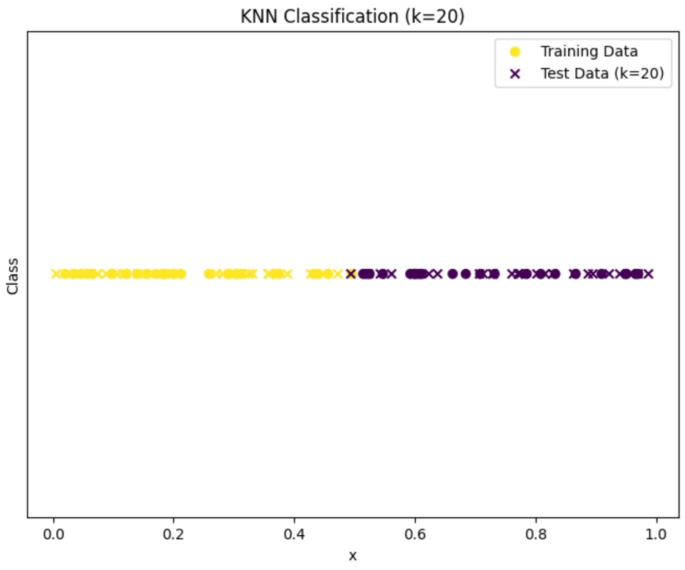
```
Predictions for k=4: [1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1.
2. 2. 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 2.
2. 1. 1. 1. 2. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2.]
```



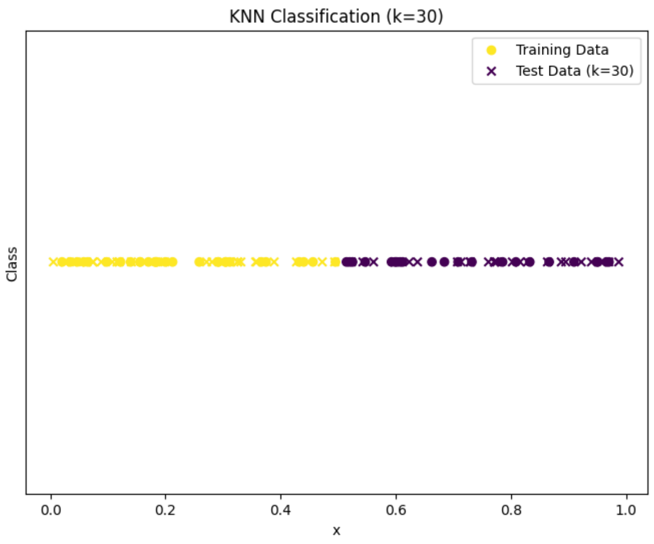
```
Predictions for k=5: [1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1.
2. 2. 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 2.
2. 1. 1. 1. 2. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2.]
```



```
Predictions for k=20: [1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1.
2. 2. 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 2.
2. 1. 1. 1. 2. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2.]
```



```
Predictions for k=30: [1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 1.
2. 2. 1. 2. 1. 2. 1. 1. 2. 2. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 2. 2.
2. 1. 1. 1. 2. 2. 1. 1. 1. 1. 2. 1. 2. 2. 2.]
```



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.

Implementation of Locally Weighted Regression (LWR)

In this experiment, the Locally Weighted Regression (LWR) algorithm was implemented using a synthetic dataset modeled as y=sin⁡(2πx)+noisey = \sin(2\pi x) + \text{noise}y=sin(2πx)+noise. LWR is a non-parametric regression technique that fits a line locally to a small neighborhood around each query point. It gives higher importance to nearby training points using a Gaussian kernel function.

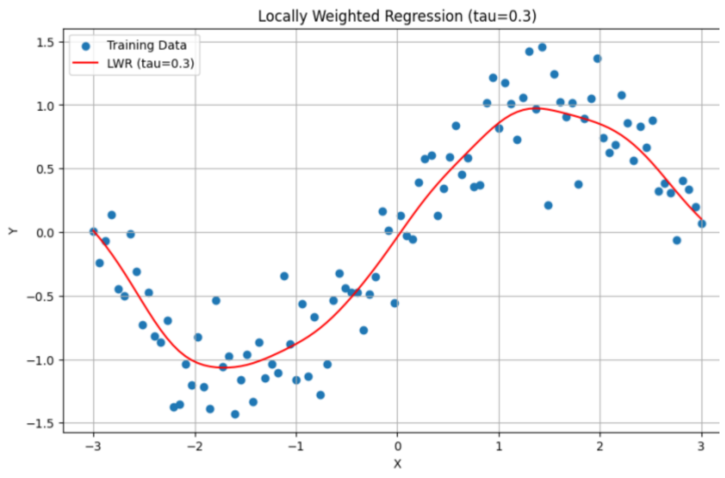
For each point x0x_0x0, a linear model is fit by solving the weighted least squares problem using weights determined by the distance of x0x_0x0 from other training points. A smaller bandwidth parameter (τ\tauτ) results in a curve that closely follows the training data, while larger values produce smoother approximations.

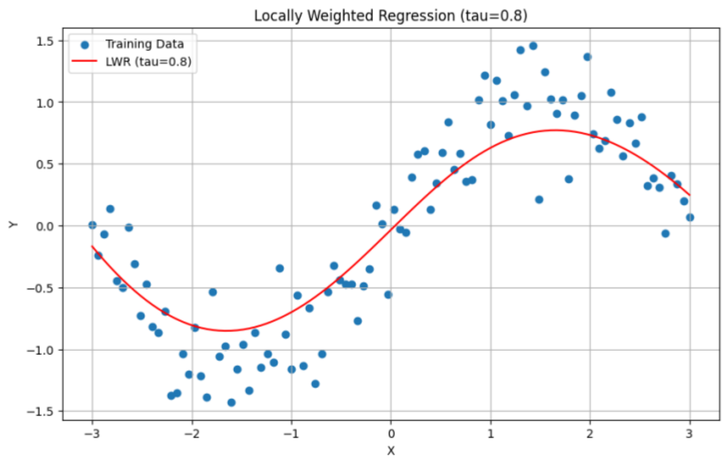
The resulting graph shows a smooth curve that fits the noisy sinusoidal pattern effectively. LWR is particularly useful when the underlying data has non-linear patterns and where global models may not perform well

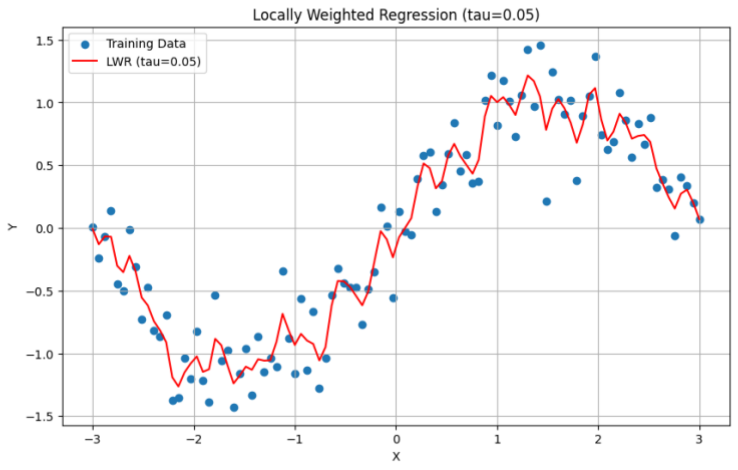
Source Code :

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

def locally_weighted_regression(x, X, Y, tau):
    """
    Performs locally weighted regression.

    Args:
        x: The query point.
        X: The training data points (features).
        Y: The training data points (targets).
        tau: The bandwidth parameter.

    Returns:
        The predicted value at the query point.
    """
    # Fix: Calculate weights using broadcasting with the feature column (X[:, 1])
    weights = np.exp(-((x[1] - X[:, 1]) ** 2) / (2 * tau ** 2))
    W = np.diag(weights)
    try:
        beta = np.linalg.pinv(X.T @ W @ X) @ X.T @ W @ Y
    except np.linalg.LinAlgError:
        print("Singular matrix encountered. Returning NaN")
        return np.nan

    return beta @ x

def generate_noisy_data(n=100):
    """Generates noisy sinusoidal data."""
    np.random.seed(42)  # For reproducibility
    X = np.linspace(-3, 3, n)
    Y = np.sin(X) + np.random.normal(0, 0.3, n)
    return X, Y

def plot_lwr(X, Y, tau):
    """Plots the locally weighted regression results."""
    x_test = np.linspace(min(X[:, 1]), max(X[:, 1]), 100) # Extract the second column (feature values)
    x_test = np.array([[1, x] for x in x_test])  # Add bias term
    y_pred = [locally_weighted_regression(x, X, Y, tau) for x in x_test]

    plt.figure(figsize=(10, 6))
    plt.scatter(X[:,1], Y, label="Training Data") # plotting the second column of X as features
    plt.plot(x_test[:,1], y_pred, color="red", label=f"LWR (tau={tau})") # plotting the second column of x_test as
features
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.title(f"Locally Weighted Regression (tau={tau})")
    plt.legend()
    plt.grid(True)
    plt.show()

# Example Usage
X, Y = generate_noisy_data()
X = np.array([[1, x] for x in X]) #adding bias term to feature matrix

# Test with different tau values
plot_lwr(X, Y, tau=0.3)
plot_lwr(X, Y, tau=0.8)
plot_lwr(X,Y, tau=0.05)
```

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.

Linear and Polynomial Regression Implementation

In this experiment, we implemented and compared Linear and Polynomial Regression models using real-world datasets. For Linear Regression, the California Housing dataset was used, where "Median Income" was used as the predictor for "Median House Value". The model was trained and evaluated using Mean Squared Error (MSE) and R² score, followed by a scatter plot with a linear fit.

Polynomial Regression was performed using the Auto MPG dataset to predict vehicle fuel efficiency (MPG) based on horsepower. A 3rd-degree polynomial model was created using PolynomialFeatures and LinearRegression from scikit-learn's pipeline module. This approach allowed the model to capture non-linear relationships that a linear model may miss.

Visualizations showed that Polynomial Regression fit the data better in this case, revealing complex trends between horsepower and MPG. This experiment highlights the trade-off between model simplicity and the ability to capture underlying patterns in data.

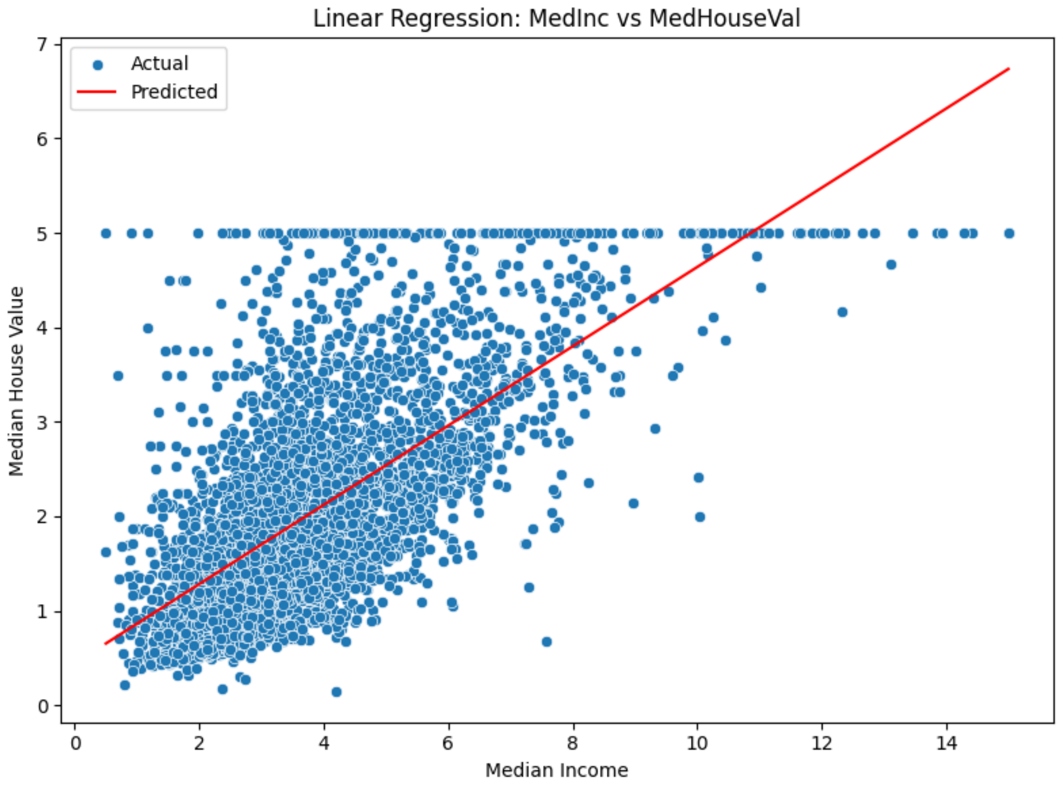
Source Code :

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

# --- 1. LINEAR REGRESSION on California Housing ---
print("\n--- Linear Regression on California Housing ---")
housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['MedHouseVal'] = housing.target

# Use 'MedInc' (median income) to predict 'MedHouseVal'
X = df[['MedInc']]
y = df['MedHouseVal']

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

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
y_pred = lr_model.predict(X_test)

print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test['MedInc'], y=y_test, label="Actual")
sns.lineplot(x=X_test['MedInc'], y=y_pred, color='red', label="Predicted")
```

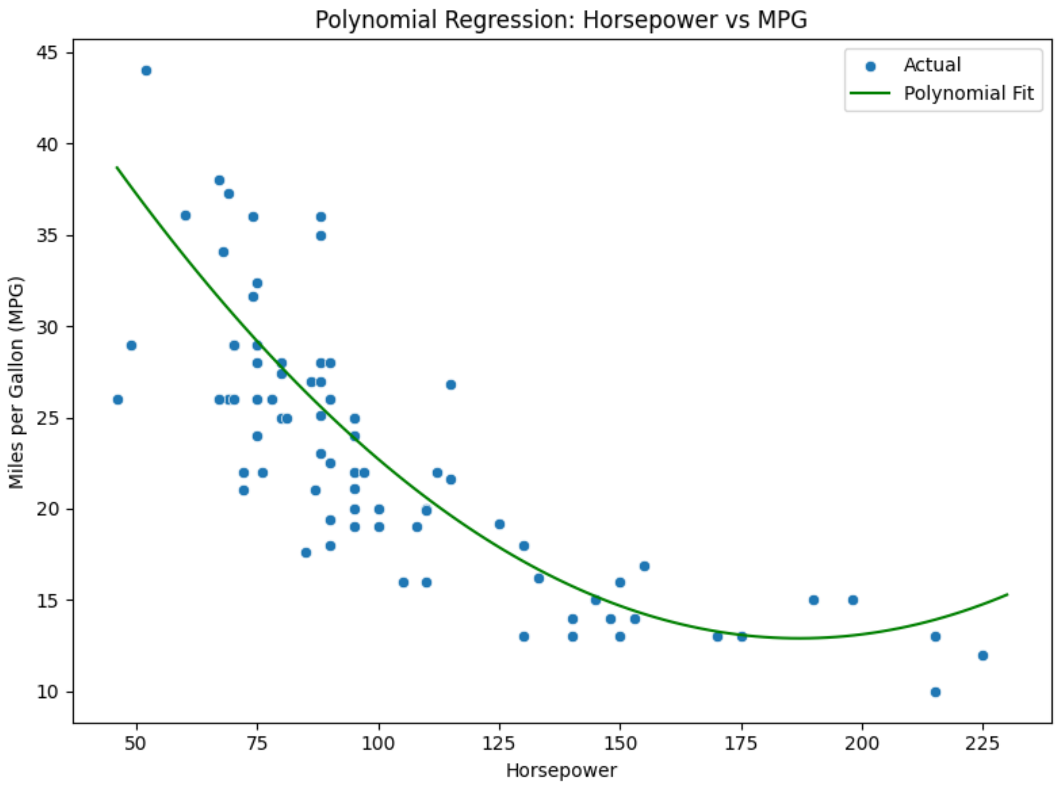
```
plt.title("Linear Regression: MedInc vs MedHouseVal")
plt.xlabel("Median Income")
plt.ylabel("Median House Value")
plt.legend()
plt.tight_layout()
plt.show()

# --- 2. POLYNOMIAL REGRESSION on Auto MPG ---
print("\n--- Polynomial Regression on Auto MPG ---")

url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
columns = ["mpg", "cylinders", "displacement", "horsepower", "weight",
           "acceleration", "model_year", "origin", "car_name"]

auto_df = pd.read_csv(url, delim_whitespace=True, names=columns, na_values='?')
auto_df.dropna(inplace=True)

X_auto = auto_df[['horsepower']].astype(float)
y_auto = auto_df['mpg']

X_train, X_test, y_train, y_test = train_test_split(X_auto, y_auto, test_size=0.2, random_state=42)

poly = PolynomialFeatures(degree=2)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

poly_model = LinearRegression()
poly_model.fit(X_poly_train, y_train)
y_poly_pred = poly_model.predict(X_poly_test)

print("MSE:", mean_squared_error(y_test, y_poly_pred))
print("R2 Score:", r2_score(y_test, y_poly_pred))

# Plot
X_range = np.linspace(X_auto.min(), X_auto.max(), 100).reshape(-1, 1)
X_range_poly = poly.transform(X_range)
y_range_pred = poly_model.predict(X_range_poly)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test['horsepower'], y=y_test, label="Actual")
sns.lineplot(x=X_range.flatten(), y=y_range_pred, color='green', label="Polynomial Fit")
plt.title("Polynomial Regression: Horsepower vs MPG")
plt.xlabel("Horsepower")
plt.ylabel("Miles per Gallon (MPG)")
plt.legend()
plt.tight_layout()
plt.show()
```

Output :

```
--- Linear Regression on California Housing ---
MSE: 0.7091157771765549
R2 Score: 0.45885918903846656
```



```
--- Polynomial Regression on Auto MPG ---
<ipython-input-9-7f7f8fdb7bee>:48: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a future version. Use ``sep='\s+'`` instead
  auto_df = pd.read_csv(url, delim_whitespace=True, names=columns, na_values='?')
MSE: 18.416967796017616
R2 Score: 0.6391701147013347
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but PolynomialFeatures was fitted with feature names
  warnings.warn(
```



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.

The Decision Tree algorithm is applied to the Breast Cancer dataset to classify tumors as malignant or benign based on medical features. The dataset contains attributes like cell size, shape, and texture. First, the data is loaded and split into training and testing sets, ensuring that the model learns patterns from 80% of the data while being evaluated on the remaining 20%. The Decision Tree classifier is trained on the dataset, forming a hierarchical structure where each split is based on the most relevant feature, optimizing classification accuracy. The model's performance is assessed using accuracy metrics and a classification report, which provides insight into precision and recall for each class.

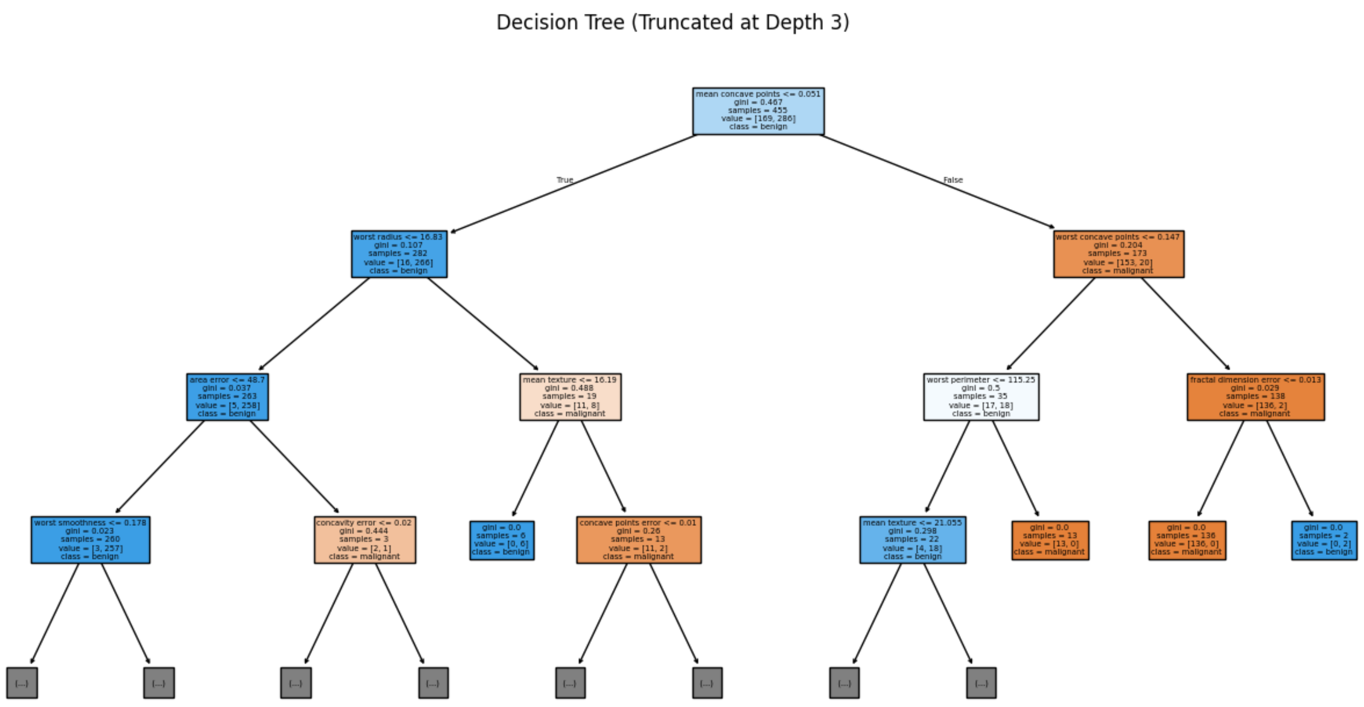
To visualize the decision tree, it is plotted with a depth constraint to avoid excessive complexity. This representation shows how the model decides based on feature thresholds. Additionally, the classifier is tested on a new sample, demonstrating how the trained model can predict the classification of an unseen tumor. This approach provides a clear understanding of how decision trees work, their ability to handle medical diagnosis problems, and how they interpret feature relationships for classification.

Source Code :

```
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import pandas as pd

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

# Train-test split
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)

# Evaluate on test data
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=data.target_names))

# Visualize the tree (truncated for simplicity)
plt.figure(figsize=(16, 8))
plot_tree(clf, filled=True, feature_names=data.feature_names, class_names=data.target_names, max_depth=3)
plt.title("Decision Tree (Truncated at Depth 3)")
plt.show()

# Classify a new sample (using first row of test set as example)
new_sample = X_test.iloc[0].values.reshape(1, -1)
prediction = clf.predict(new_sample)
print("\nNew Sample Prediction:")
print("Features:\n", X_test.iloc[0])
print("Predicted Class:", data.target_names[prediction[0]])
```

Output :

```
Accuracy: 0.9473684210526315

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 (Truncated at Depth 3)
mean concave points <= 0.051 gini = 0.467 samples = 455 value = [169, 286] class = benign
True
False
worst radius <= 16.83 gini = 0.107 samples = 282 value = [16, 266] class = benign
worst concave points <= 0.147 gini = 0.204 samples = 173 value = [153, 20] class = malignant
area error <= 48.7 gini = 0.037 samples = 263 value = [5, 258] class = benign
mean texture <= 16.19 gini = 0.488 samples = 19 value = [11, 8] class = malignant
worst perimeter <= 115.25 gini = 0.5 samples = 35 value = [17, 18] class = benign
fractal dimension error <= 0.013 gini = 0.029 samples = 138 value = [136, 2] class = malignant
worst smoothness <= 0.178 gini = 0.023 samples = 260 value = [3, 257] class = benign
concavity error <= 0.02 gini = 0.444 samples = 3 value = [2, 1] class = malignant
gini = 0.0 samples = 6 value = [0, 6] class = benign
concave points error <= 0.01 gini = 0.26 samples = 13 value = [11, 2] class = malignant
mean texture <= 21.055 gini = 0.298 samples = 22 value = [4, 18] class = benign
gini = 0.0 samples = 13 value = [13, 0] class = malignant
gini = 0.0 samples = 136 value = [136, 0] class = malignant
gini = 0.0 samples = 2 value = [0, 2] class = benign
(...)

```
New Sample Prediction:
Features:
mean radius                12.470000
mean texture               18.600000
mean perimeter             81.090000
mean area                 481.900000
mean smoothness             0.099650
mean compactness            0.105800
mean concavity              0.080050
mean concave points         0.038210
mean symmetry               0.192500
mean fractal dimension      0.063730
radius error                0.396100
texture error               1.044000
perimeter error             2.497000
area error                 30.290000
smoothness error            0.006953
compactness error           0.019110
concavity error             0.027010
concave points error        0.010370
symmetry error              0.017820
fractal dimension error     0.003586
worst radius               14.970000
worst texture              24.640000
worst perimeter            96.050000
worst area                677.900000
worst smoothness            0.142600
worst compactness           0.237800
worst concavity             0.267100
worst concave points        0.101500
worst symmetry              0.301400
worst fractal dimension     0.087500
Name: 204, dtype: float64
Predicted Class: benign
```

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.

The Naive Bayesian classifier is a probabilistic algorithm used for classification tasks, and in this case, it's applied to the Olivetti Face Dataset. The dataset contains 400 grayscale images of 40 individuals, with each image flattened into 4096 features. The Gaussian Naive Bayes classifier assumes that feature distributions follow a normal distribution and calculates probabilities to determine the most likely class for a given input. After training the model on 80% of the dataset, it is tested on the remaining 20% to evaluate its effectiveness.

The classifier's accuracy is computed using the accuracy_score function, which measures how well predictions align with actual labels. Additionally, a classification report provides deeper insights into precision, recall, and F1-score for each class. To visually assess the performance, sample test images are displayed alongside their predicted and true labels. This approach helps identify misclassifications and improves understanding of the model's strengths and weaknesses.

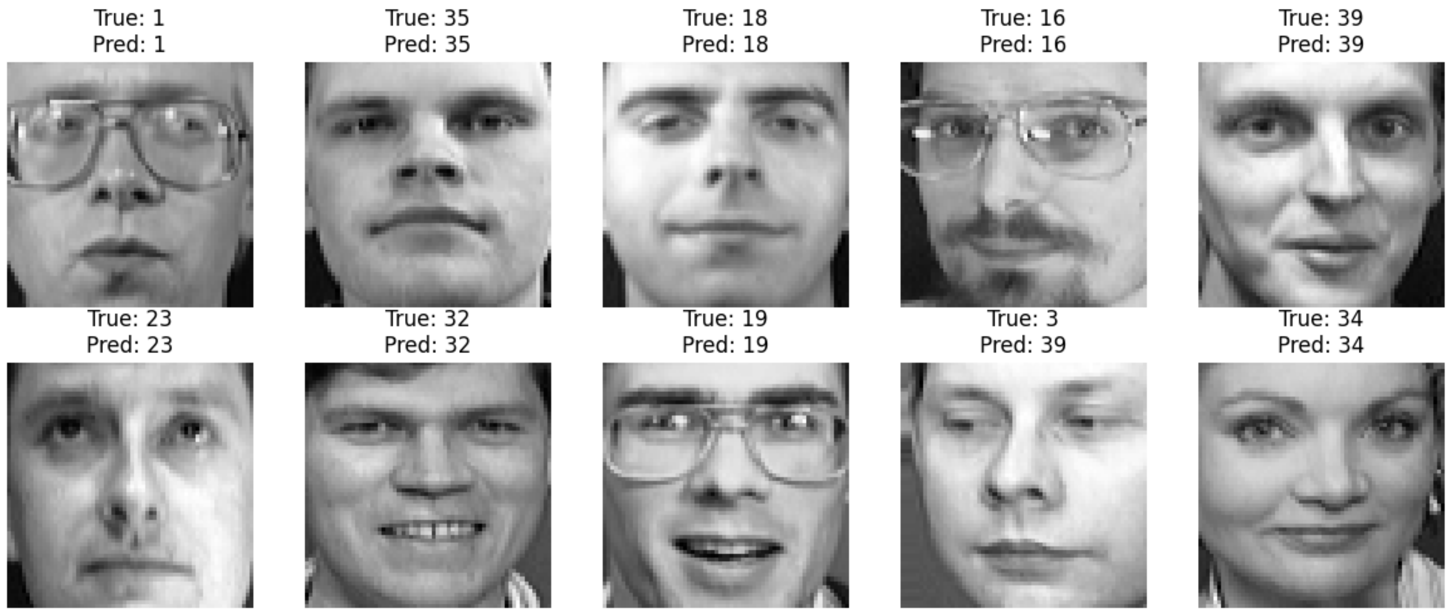
Source Code :

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

# Load Olivetti Faces dataset
faces = fetch_olivetti_faces()
X = faces.data  # 4096 features (64x64 images flattened)
y = faces.target  # Labels from 0 to 39 (person ID)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Train Gaussian Naive Bayes Classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)

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

# Accuracy
acc = accuracy_score(y_test, y_pred)
print("Accuracy of Gaussian Naive Bayes on Olivetti Faces:", acc)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Display some test images with predictions
def show_images(images, labels, preds, n=10):
    plt.figure(figsize=(12, 5))
    for i in range(n):
        plt.subplot(2, n // 2, i + 1)
        plt.imshow(images[i].reshape(64, 64), cmap='gray')
        plt.title(f"True: {labels[i]}\nPred: {preds[i]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Show first 10 test samples
show_images(X_test, y_test, y_pred, n=10)
```

Output :

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to /root/scikit_learn_data
Accuracy of Gaussian Naive Bayes on Olivetti Faces: 0.875
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       1.00      1.00      1.00         2
           2       0.67      1.00      0.80         2
           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00         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       0.50      0.50      0.50         2
          10       1.00      0.50      0.67         2
          11       1.00      1.00      1.00         2
          12       0.67      1.00      0.80         2
          13       1.00      0.50      0.67         2
          14       1.00      1.00      1.00         2
          15       1.00      1.00      1.00         2
          16       1.00      1.00      1.00         2
          17       1.00      1.00      1.00         2
          18       1.00      1.00      1.00         2
          19       1.00      1.00      1.00         2
          20       1.00      1.00      1.00         2
          21       1.00      1.00      1.00         2
          22       1.00      1.00      1.00         2
          23       1.00      1.00      1.00         2
          24       1.00      0.50      0.67         2
          25       1.00      1.00      1.00         2
          26       1.00      1.00      1.00         2
          27       1.00      1.00      1.00         2
          28       1.00      1.00      1.00         2
          29       1.00      1.00      1.00         2
          30       1.00      1.00      1.00         2
          31       1.00      1.00      1.00         2
          32       1.00      1.00      1.00         2
          33       1.00      1.00      1.00         2
          34       0.67      1.00      0.80         2
          35       0.67      1.00      0.80         2
          36       1.00      1.00      1.00         2
          37       1.00      0.50      0.67         2
          38       1.00      1.00      1.00         2
          39       0.33      1.00      0.50         2

    accuracy                           0.88        80
   macro avg       0.88      0.88      0.86        80
```

```
weighted avg       0.88      0.88      0.86        80
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.

  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



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

The K-Means clustering algorithm is applied to the Wisconsin Breast Cancer dataset to group data points into two clusters. First, the dataset, which contains 30 numerical features representing tumor characteristics, is loaded using load_breast_cancer(). The algorithm initializes two clusters (k=2) and assigns each data point to the nearest cluster center based on feature similarity. Since K-Means is an unsupervised technique, its generated cluster labels may not perfectly match the actual classification (benign vs. malignant), so an alignment step is performed to adjust accuracy.

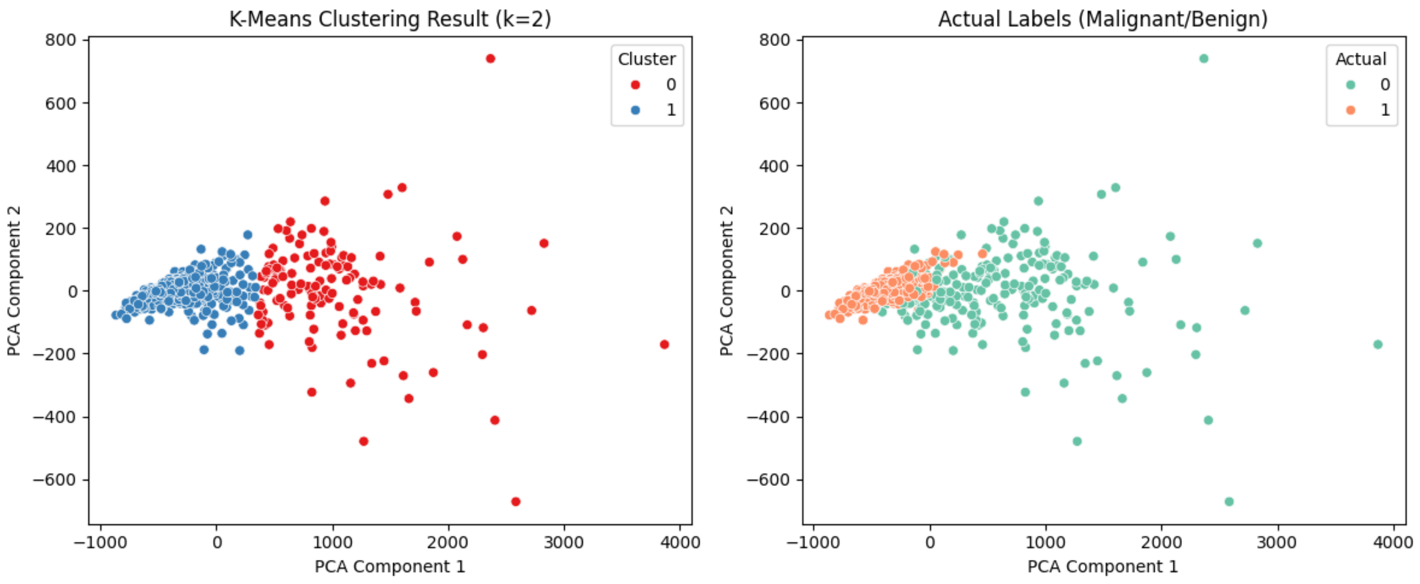
To visualize the clustering results, Principal Component Analysis (PCA) is applied to reduce dimensionality to two components, making it easier to plot the data. Two scatter plots are created—one displaying the clusters assigned by K-Means and another showing the true classifications. This comparison helps evaluate clustering performance. The accuracy is calculated using accuracy_score(), and the results indicate that K-Means achieves reasonable effectiveness in distinguishing between malignant and benign tumors.

Source Code :

```
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.decomposition import PCA
from sklearn.metrics import accuracy_score

# Load dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X)

# Since cluster labels may not match target labels, align them
# Most common trick: invert if necessary
if accuracy_score(y, clusters) < 0.5:
    clusters = 1 - clusters  # flip labels

print("Clustering Accuracy:", accuracy_score(y, clusters))

# Use PCA for 2D Visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Plot clustering results
plt.figure(figsize=(12, 5))

# Clustering result
plt.subplot(1, 2, 1)
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=clusters, palette='Set1')
plt.title("K-Means Clustering Result (k=2)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster')

# Actual labels
plt.subplot(1, 2, 2)
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y, palette='Set2')
plt.title("Actual Labels (Malignant/Benign)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Actual')

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

Output :

```
Clustering Accuracy: 0.8541300527240774
```


K-Means Clustering Result (k=2)
Cluster
PCA Component 1
PCA Component 2
Actual Labels (Malignant/Benign)
Actual
PCA Component 1
PCA Component 2