USN:

Name:

TW1. Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.

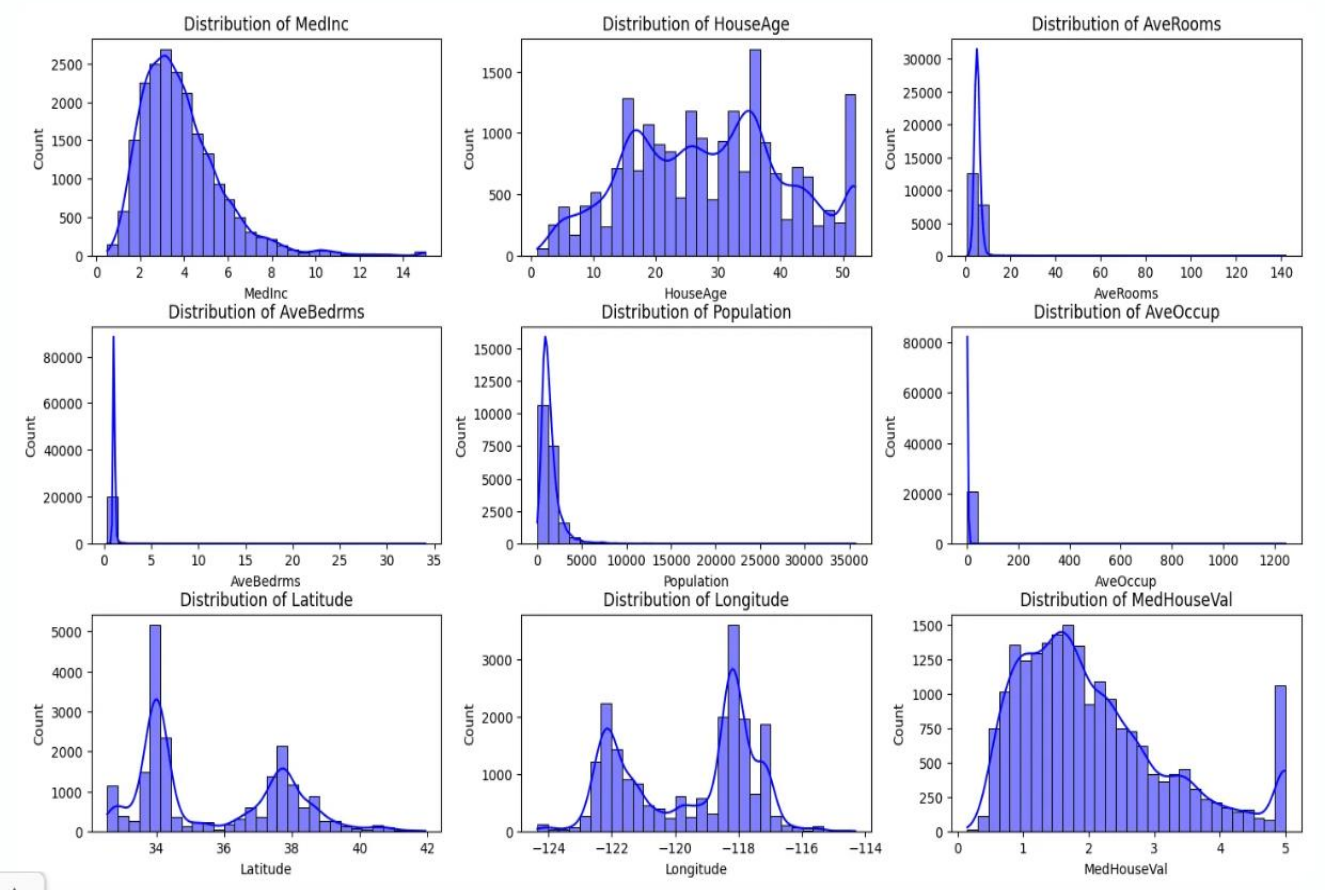
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv(r'C:housing.csv')
df.head( )
df.info()

#Data Cleaning
df.isnull().sum()
df.duplicated().sum()
df['total_bedrooms'].median()

# Handling missing values
df['total_bedrooms'].fillna(df['total_bedrooms'].median(), inplace=True)

#Feature Engineering
for i in df.iloc[:,2:7]:
    df[i] = df[i].astype('int')

df.head()

# **Disciptive Statistics**
df.describe().T

Numerical = df.select_dtypes(include=[np.number]).columns
print(Numerical)

# **Uni-Variate Analysis**
#plot histogram
for col in Numerical:
    plt.figure(figsize=(10, 6))

    df[col].plot(kind='hist', title=col, bins=60, edgecolor='black')
    plt.ylabel('Frequency')

    plt.show()

#genearte box plots for numerical features
for col in Numerical:
    plt.figure(figsize=(6, 6))
```

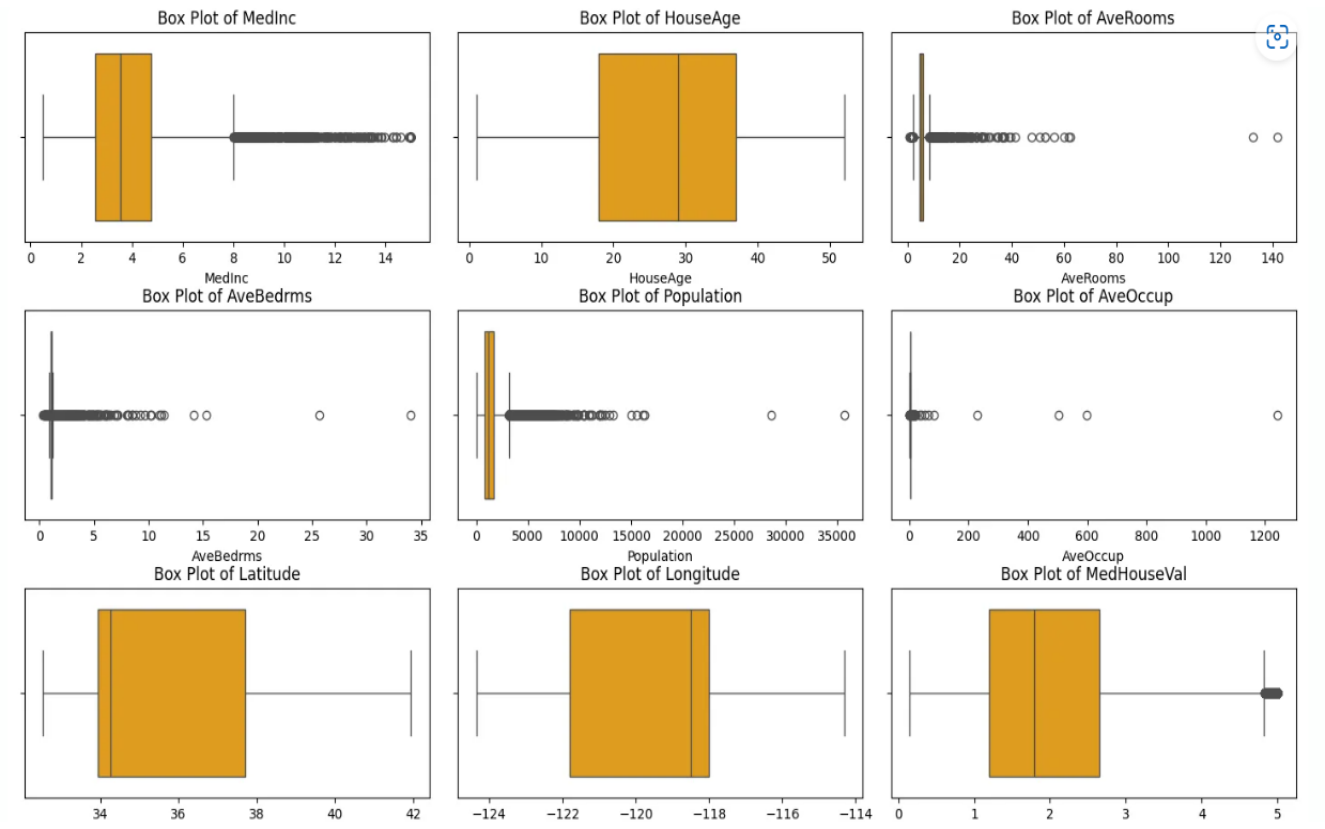
```
sns.boxplot(df[col], color='blue')
plt.title(col)
plt.ylabel(col)

plt.show()
```

USN:

Name:

OUTPUT:





USN:

Name:

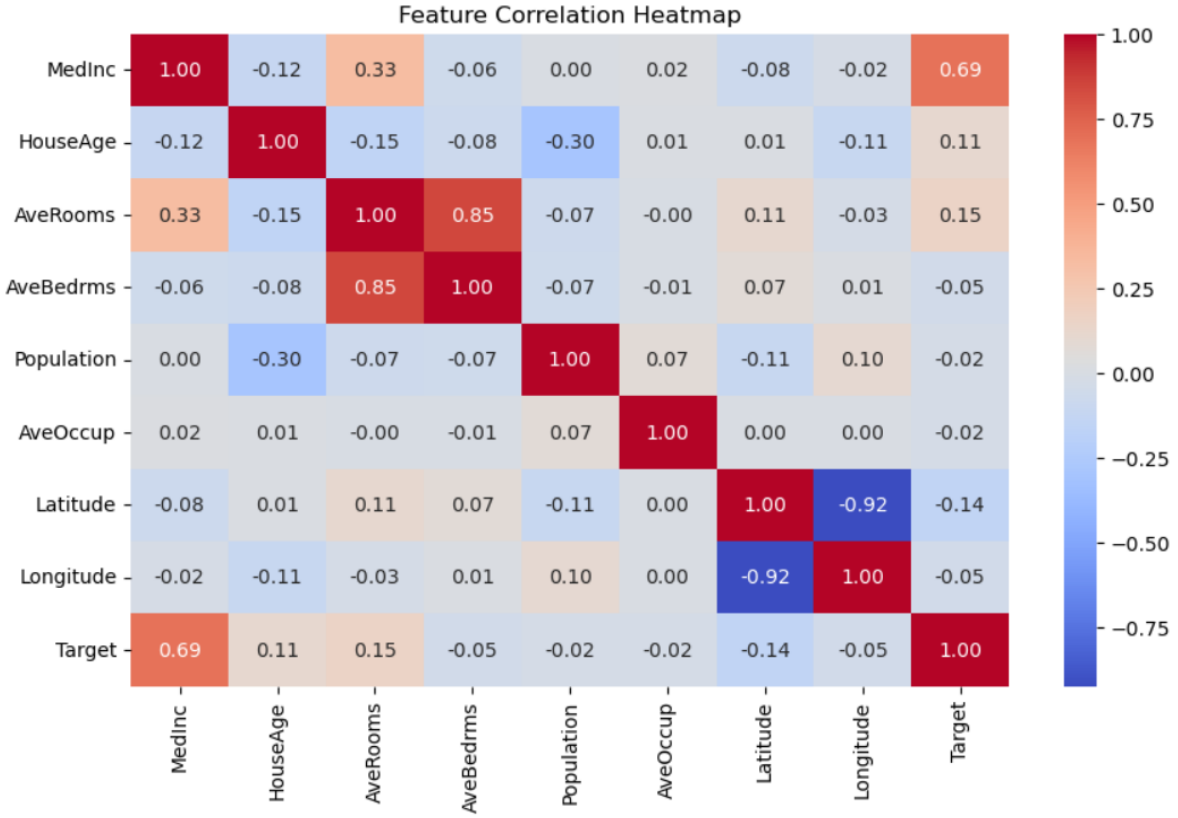
TW- 2: Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features.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

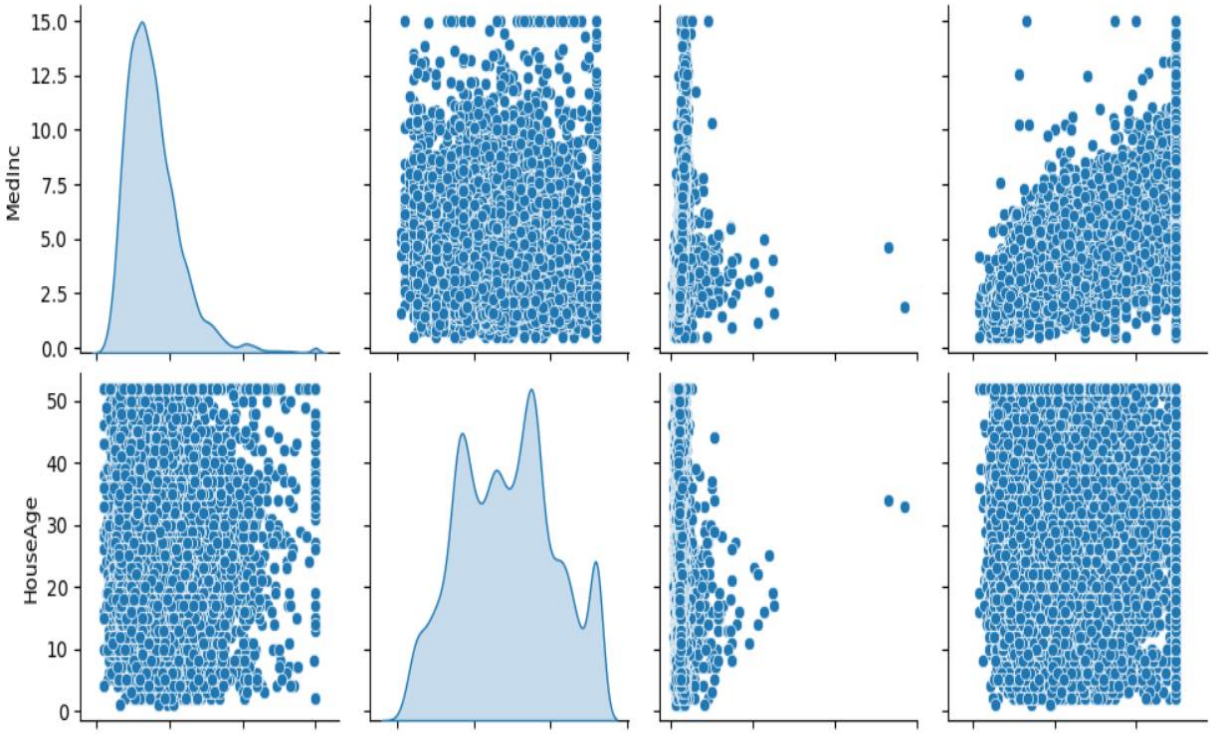
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing

# Load California Housing dataset
data = fetch_california_housing()

# Convert to DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target  # Adding the target variable (median house value)

# Correlation Matrix
plt.figure(figsize=(10, 6))
corr_matrix = df.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Feature Correlation Heatmap")
plt.show()

# Pairplot to analyze feature relationships (only a subset for clarity)
sns.pairplot(df[['MedInc', 'HouseAge', 'AveRooms', 'Target']], diag_kind='kde')
plt.show()

# Insights from Data Exploration
print("\nKey Insights:")
print("1. The dataset has", df.shape[0], "rows and", df.shape[1], "columns.")
print("2. No missing values were found in the dataset.")
print("3. Histograms show skewed distributions in some features like 'MedInc'.")
print("4. Boxplots indicate potential outliers in 'AveRooms' and 'AveOccup'.")
print("5. Correlation heatmap shows 'MedInc' has the highest correlation with house prices.")
```

USN:

Name:

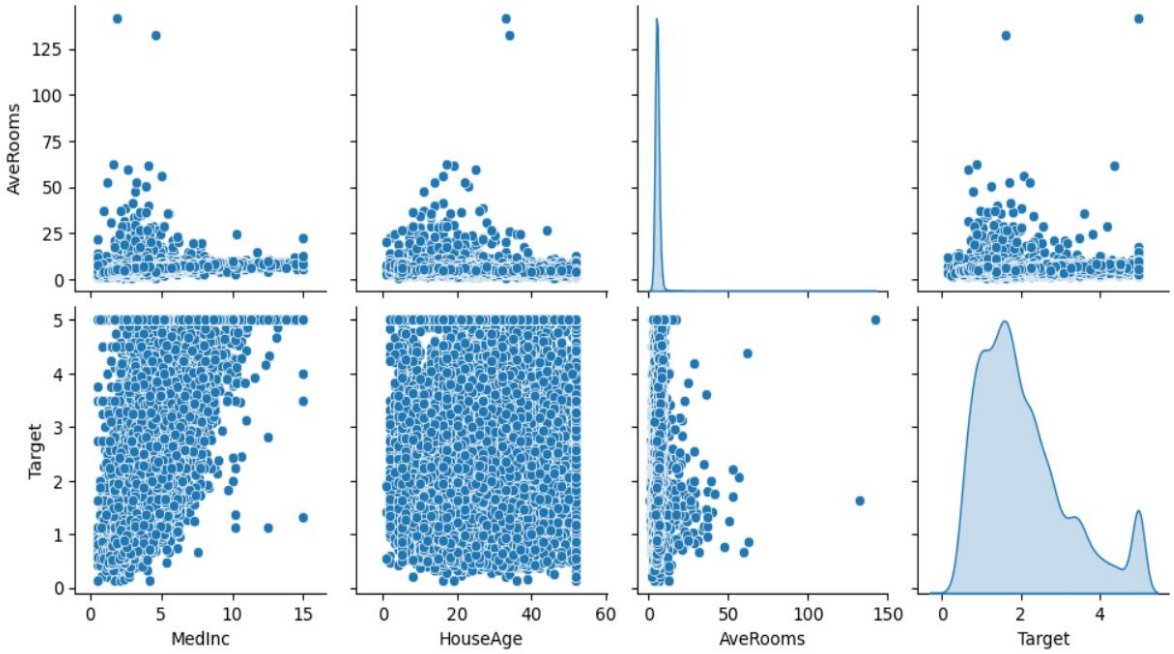
OUTPUT





USN:

Name:



Key Insights:
1. The dataset has 20640 rows and 9 columns.
2. No missing values were found in the dataset.
3. Histograms show skewed distributions in some features like 'MedInc'.
4. Boxplots indicate potential outliers in 'AveRooms' and 'AveOccup'.
5. Correlation heatmap shows 'MedInc' has the highest correlation with house prices.

USN:

Name:

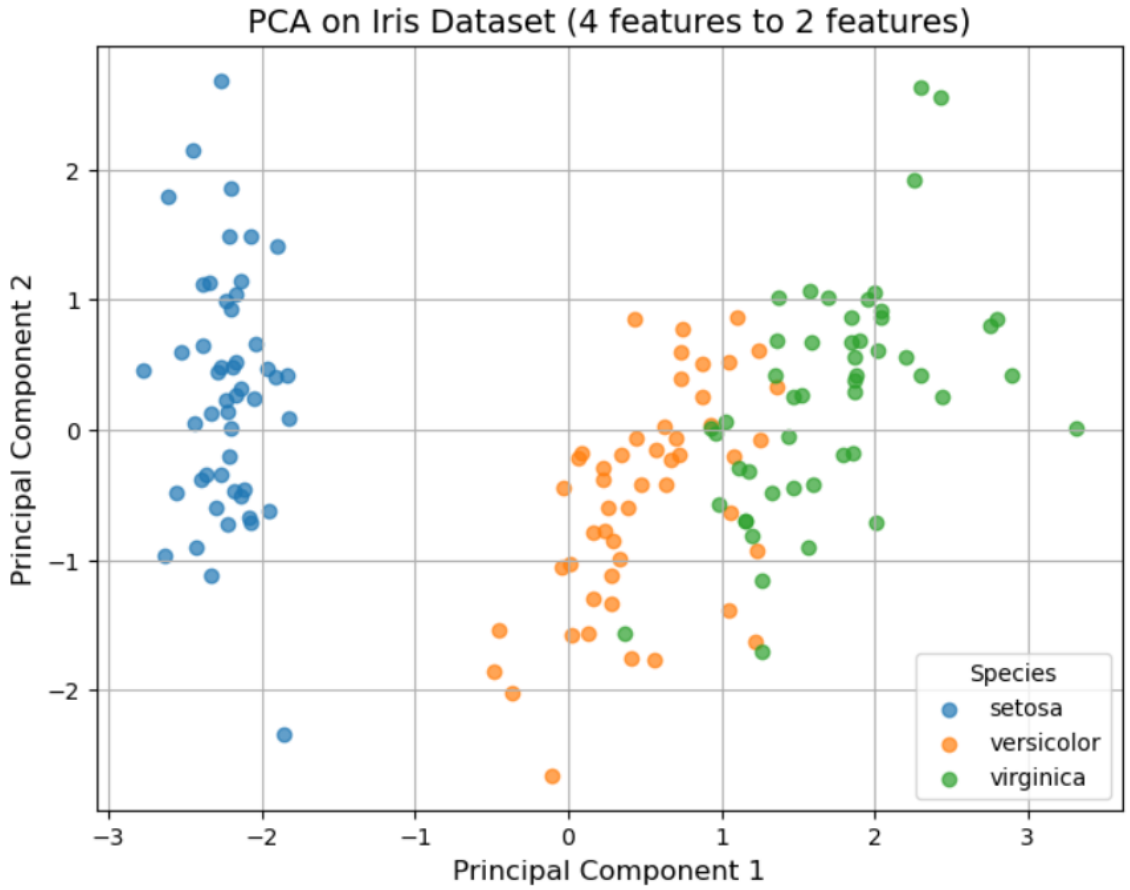
TW-3: Develop a program to implement Principal Component Analysis (PCA) for reduction the dimensionality of the Iris dataset from 4 features to 2.

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

# Step 1: Load the Iris dataset
iris = load_iris()
features = iris.data  # The 4 features: Sepal Length, Sepal Width, Petal Length, Petal Width
target = iris.target  # The target class (species)

# Step 2: Standardize the features
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)

# Step 3: Apply PCA to reduce to 2 components
pca = PCA(n_components=2)
features_pca = pca.fit_transform(features_standardized)

# Step 4: Create a DataFrame for the reduced data
pca_df = pd.DataFrame(data=features_pca, columns=["Principal Component 1", "Principal Component 2"])
pca_df["Target"] = target

# Step 5: Visualize the results
plt.figure(figsize=(8, 6))
for label, color in zip(iris.target_names, ["red", "green", "blue"]):
    plt.scatter(
        pca_df.loc[pca_df["Target"] == list(iris.target_names).index(label), "Principal Component 1"],
        pca_df.loc[pca_df["Target"] == list(iris.target_names).index(label), "Principal Component 2"],
        label=label,
        alpha=0.7
    )


plt.title("PCA on Iris Dataset (4 features to 2 features)", fontsize=14)
plt.xlabel("Principal Component 1", fontsize=12)
plt.ylabel("Principal Component 2", fontsize=12)
plt.legend(title="Species")
plt.grid()
plt.show()
```

USN:

Name:

OUTPUT:



USN:

Name:

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

```
import pandas as pd

def find_s_algorithm(file_path):
    data = pd.read_csv(file_path)

    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 hypothesis
file_path = 'training_data.csv'
hypothesis = find_s_algorithm(file_path)
print("\nThe final hypothesis is:", hypothesis)
```

DATASET

	Outlook	Temperature	Humidity	Windy	PlayTennis
1	Sunny	Hot	High	FALSE	No
2	Sunny	Hot	High	TRUE	No
3	Overcast	Hot	High	FALSE	Yes
4	Rain	Cold	High	FALSE	Yes
5	Rain	Cold	High	TRUE	No
6	Overcast	Hot	High	TRUE	Yes
7	Sunny	Hot	High	FALSE	No

	Experience	Qualification	Skill	Age	Hired
1	Yes	Masters	Python	30	Yes
2	Yes	Bachelors	Python	25	Yes
3	No	Bachelors	Java	28	No
4	Yes	Masters	Java	40	Yes
5	No	Masters	Python	35	No

OUTPUT:

The final hypothesis is: ['Overcast', 'Hot', 'High', '?']

The final hypothesis is: ['Yes', 'Masters', '?', 40]

USN:

Name:

TW-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

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

# Step 1: Generate 100 random values in the range [0,1]
x_values = np.random.rand(100)

# Step 2: Label the first 50 points
labels = np.array(["Class1" if x <= 0.5 else "Class2" for x in x_values[:50]])

print(x_values)
print("---------------------------------------")
print(labels)

# Step 3: Define the KNN function
def knn_classify(x_train, y_train, x_test, k):
    predictions = []

    for x in x_test:
        # Compute distances from x to all x_train points
        distances = np.abs(x_train - x)

        # Get indices of k nearest neighbors
        k_nearest_indices = np.argsort(distances)[:k]

        # Get the labels of k nearest neighbors
        k_nearest_labels = y_train[k_nearest_indices]

        # Determine the most common class among neighbors
        most_common = Counter(k_nearest_labels).most_common(1)[0][0]

        # Store the predicted class
        predictions.append(most_common)

    return np.array(predictions)

# Step 4: Classify the remaining 50 points using KNN for different values of k
k_values = [1, 2, 3, 4, 5, 20, 30]
results = {}

for k in k_values:
    predicted_labels = knn_classify(x_values[:50], labels, x_values[50:], k)
    results[k] = predicted_labels
```

 # Step 5: Visualization with clusters

```
plt.figure(figsize=(10, 6))
for k in k_values:
    plt.figure(figsize=(10, 6))

    # Plot labeled data
    plt.scatter(x_values[:50], [1]*50, c=["blue" if lbl == "Class1" else "red" for lbl in labels], label="Labeled Data")

    # Plot classified data
    plt.scatter(x_values[50:], [2]*50, c=["blue" if lbl == "Class1" else "red" for lbl in results[k]], label=f"Classified Data (k={k})")

    plt.xlabel("x values")
    plt.ylabel("Classified/Unclassified")
    plt.title(f"KNN Classification Clusters (k={k})")
    plt.legend()
    plt.show()
```

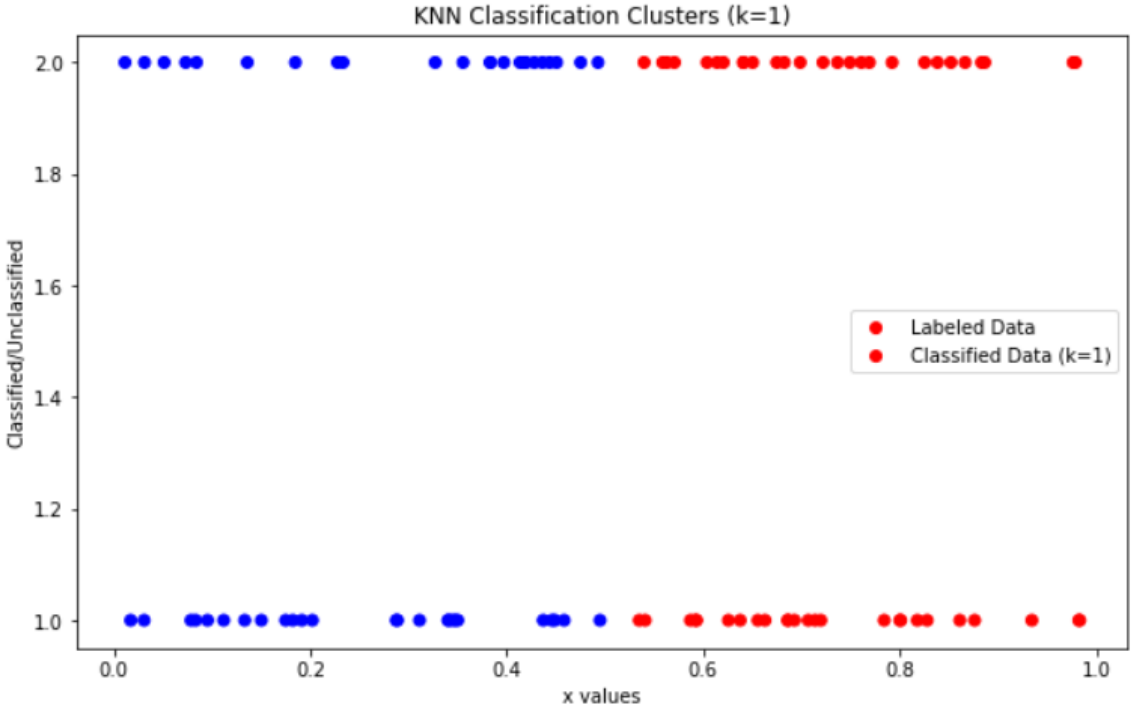
 # Step 6: Print classification results

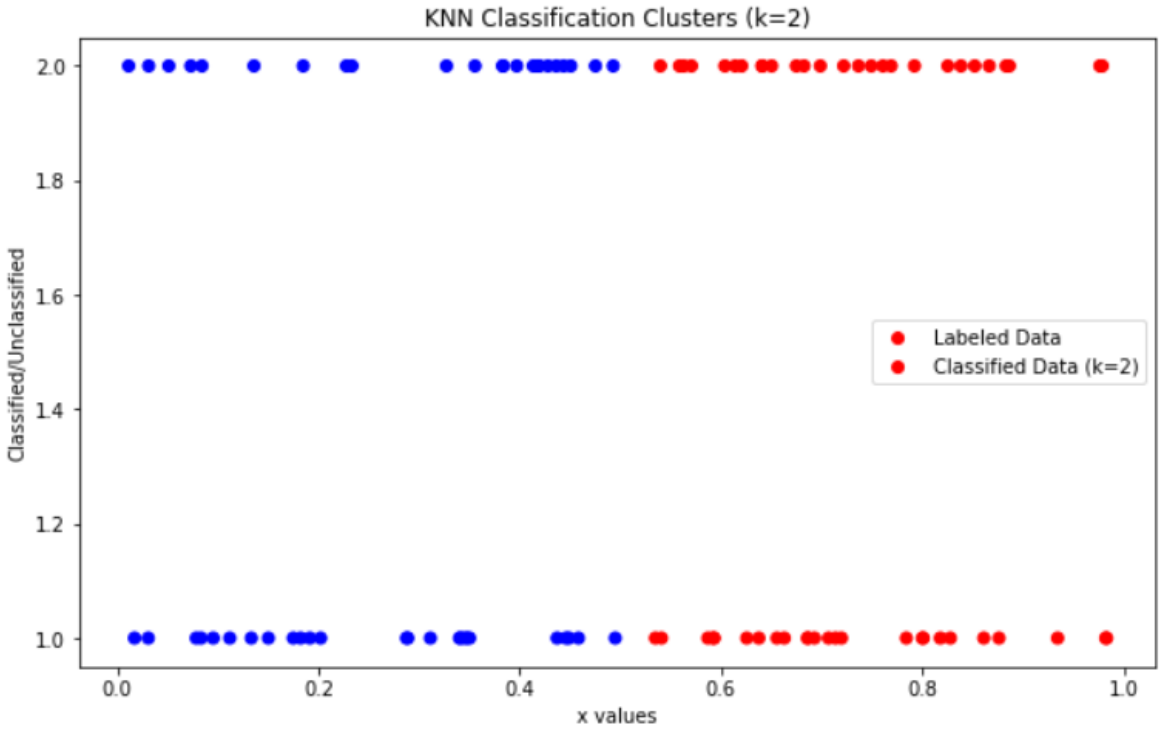
```
for k, preds in results.items():
    print(f"Results for k={k}:")
    print(preds)
    print("-")
```

USN:

Name:

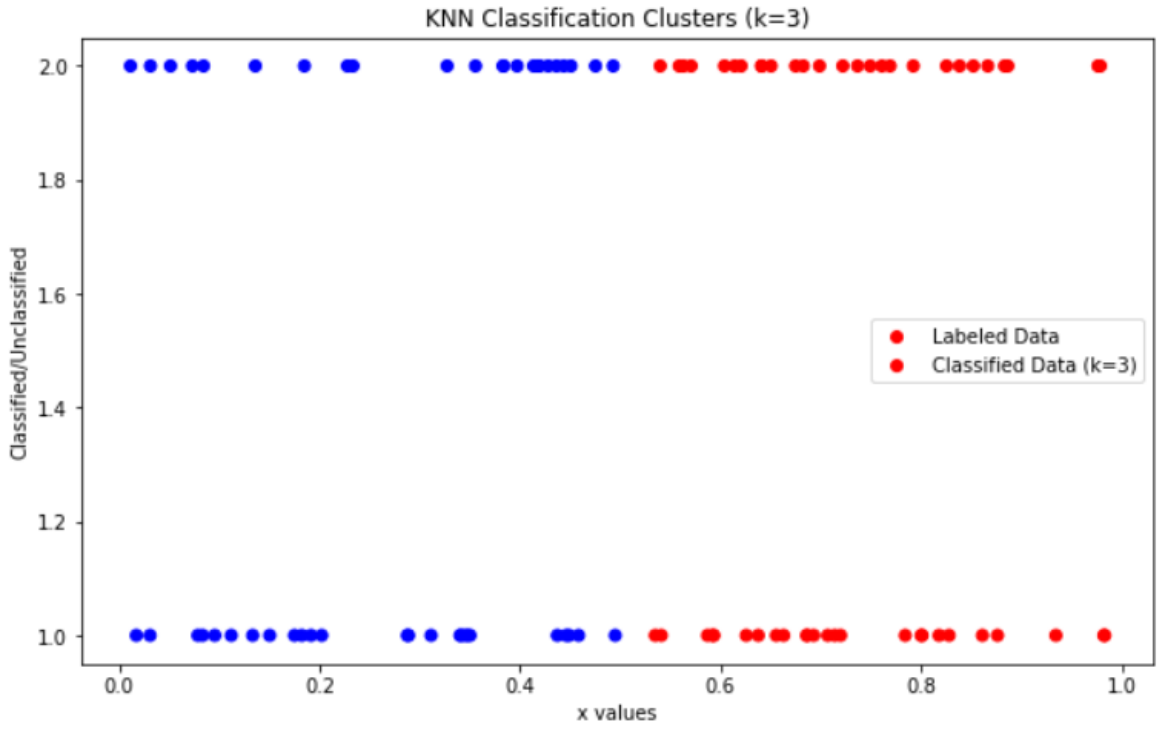
OUTPUT:

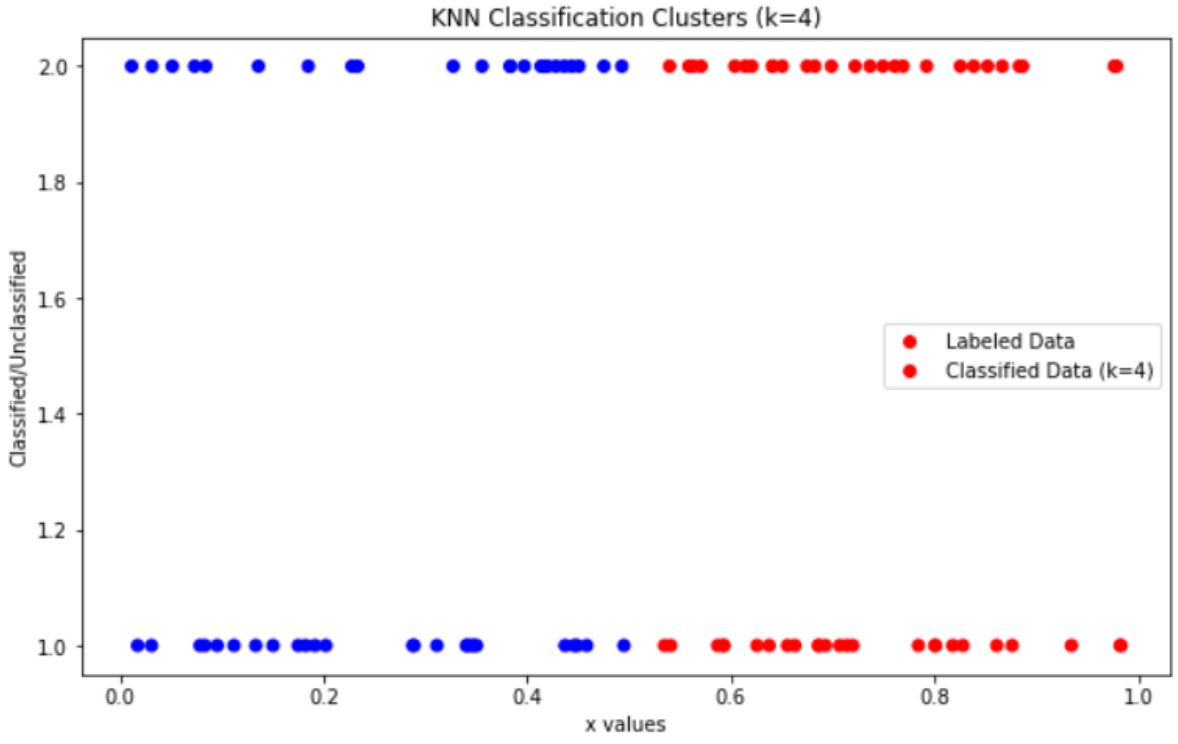




USN:

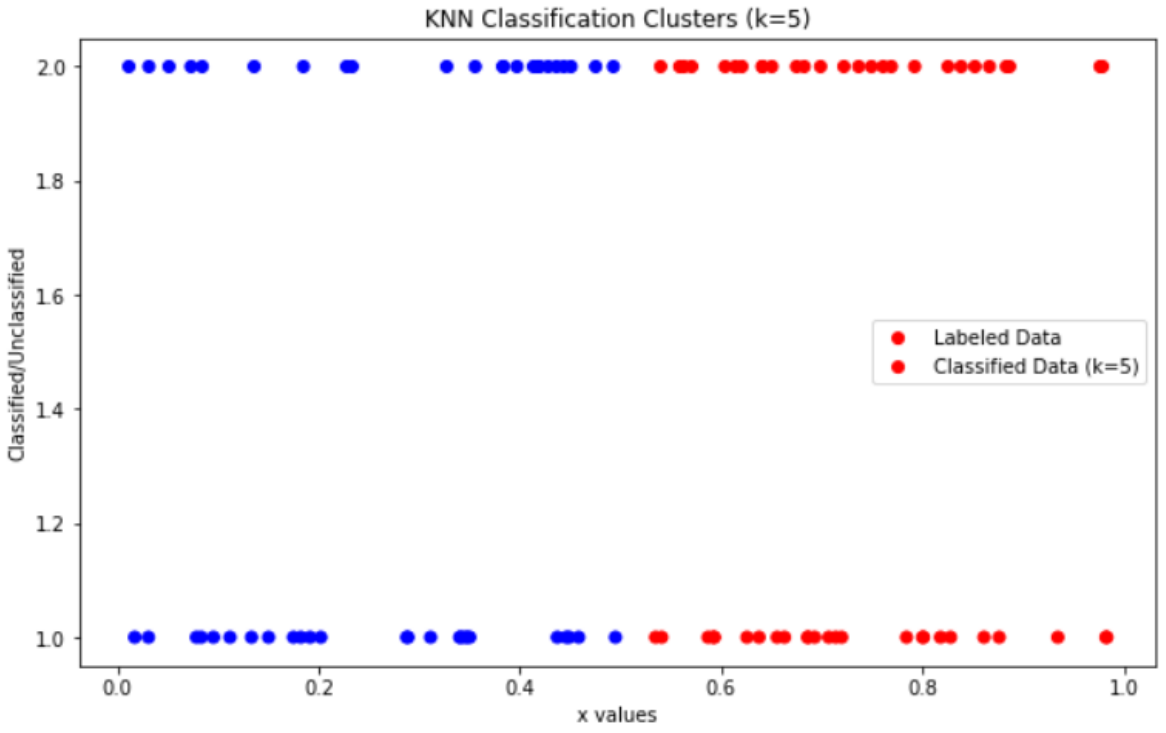
Name:

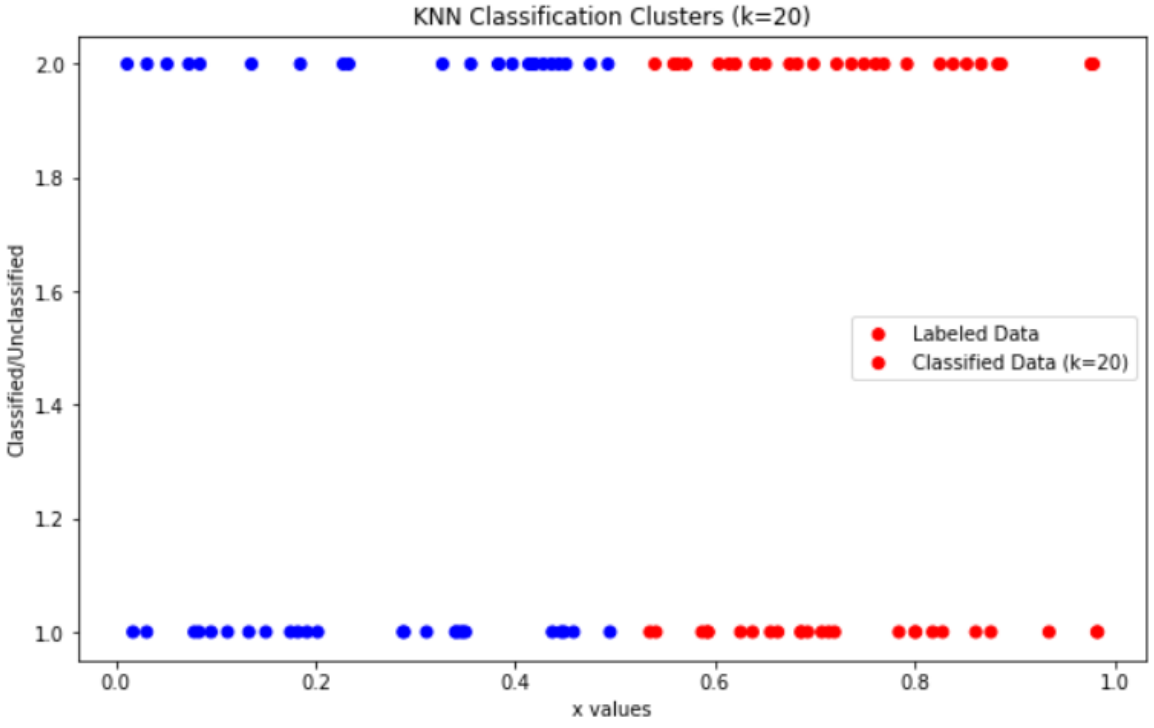

KNN Classification Clusters (k=3)
Classified/Unclassified
x values
Labeled Data
Classified Data (k=3)
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.2
0.4
0.6
0.8
1.0


KNN Classification Clusters (k=4)
Classified/Unclassified
x values
Labeled Data
Classified Data (k=4)
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.2
0.4
0.6
0.8
1.0

USN:

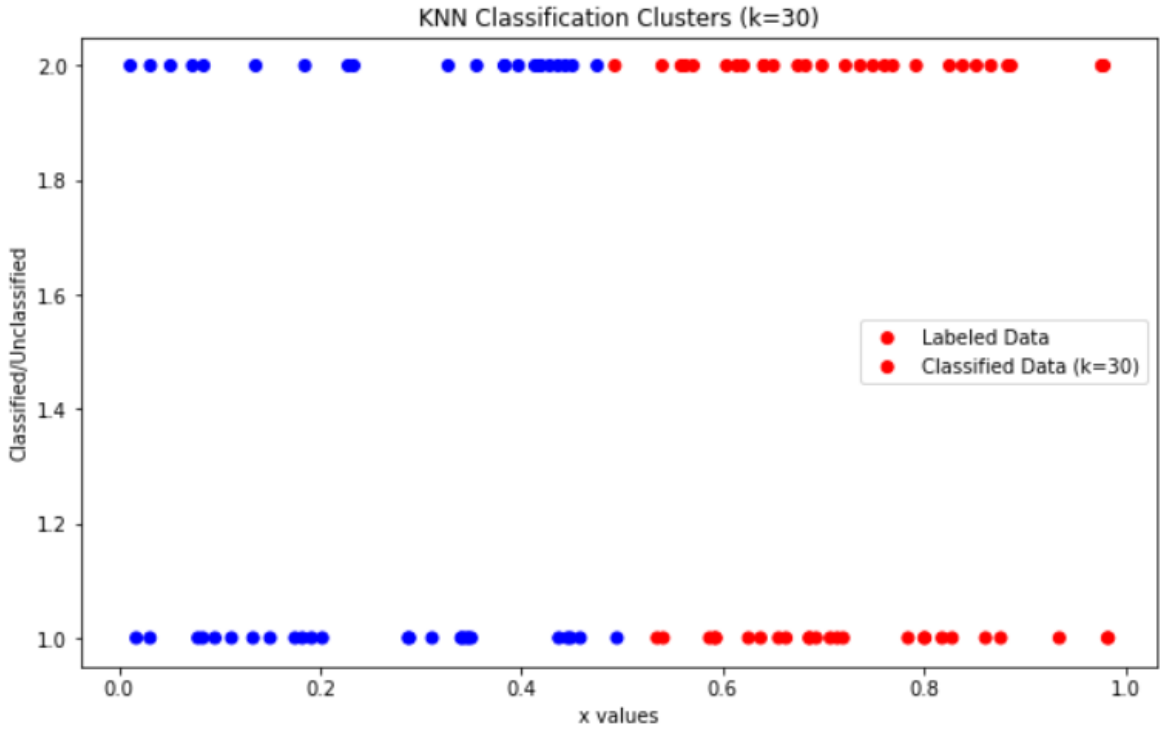
Name:


KNN Classification Clusters (k=5)
Classified/Unclassified
x values
Labeled Data
Classified Data (k=5)
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.2
0.4
0.6
0.8
1.0


KNN Classification Clusters (k=20)
Classified/Unclassified
x values
Labeled Data
Classified Data (k=20)
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.2
0.4
0.6
0.8
1.0

USN:

Name:



--- k-Nearest Neighbours Classification ---
Training dataset: First 50 points labelled based on the rule (x <= 0.5 -> Class1, x > 0.5 -> Class2)

Results for k=1:
['Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2''Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1']
-
Results for k=2:
['Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1']
-
Results for k=3:
['Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2' 'Class2' 'Class1']
-

Results for k=4:
['Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1'
'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
'Class2' 'Class1']
-
Results for k=5:
['Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1'
'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
'Class2' 'Class1']
-
Results for k=20:
['Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1'
'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
'Class2' 'Class1']
-
Results for k=30:
['Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class2' 'Class1'
'Class1' 'Class2' 'Class1' 'Class1' 'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class2' 'Class1' 'Class1' 'Class2' 'Class1'
'Class1' 'Class1' 'Class2' 'Class2' 'Class2' 'Class1' 'Class2' 'Class1' 'Class2' 'Class1' 'Class1' 'Class2'
'Class2' 'Class1']

USN:

Name:

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

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from scipy.spatial.distance import cdist
```

 # Load datasets

```
df_lwr = pd.read_csv("lwr_dataset.csv")
# Locally Weighted Regression (LWR)
def gaussian_kernel(x, X, tau):
    return np.exp(-cdist([[x]], X, 'sqeuclidean') / (2 * tau**2))

def locally_weighted_regression(X_train, y_train, tau=0.5):
    X_train = np.hstack([np.ones((X_train.shape[0], 1)), X_train])  # Add intercept
    X_range = np.linspace(X_train[:, 1].min(), X_train[:, 1].max(), 100)
    y_pred = []

    for x in X_range:
        x_vec = np.array([1, x])  # Intercept term
        weights = gaussian_kernel(x, X_train[:, 1:], tau).flatten()
        W = np.diag(weights)

        theta = np.linalg.pinv(X_train.T @ W @ X_train) @ (X_train.T @ W @ y_train)
        y_pred.append(x_vec @ theta)  # Use dot product for prediction

    plt.scatter(X_train[:, 1], y_train, label='Data')
    plt.plot(X_range, y_pred, color='red', label='LWR')
    plt.legend()
    plt.title("Locally Weighted Regression")
    plt.show()
```

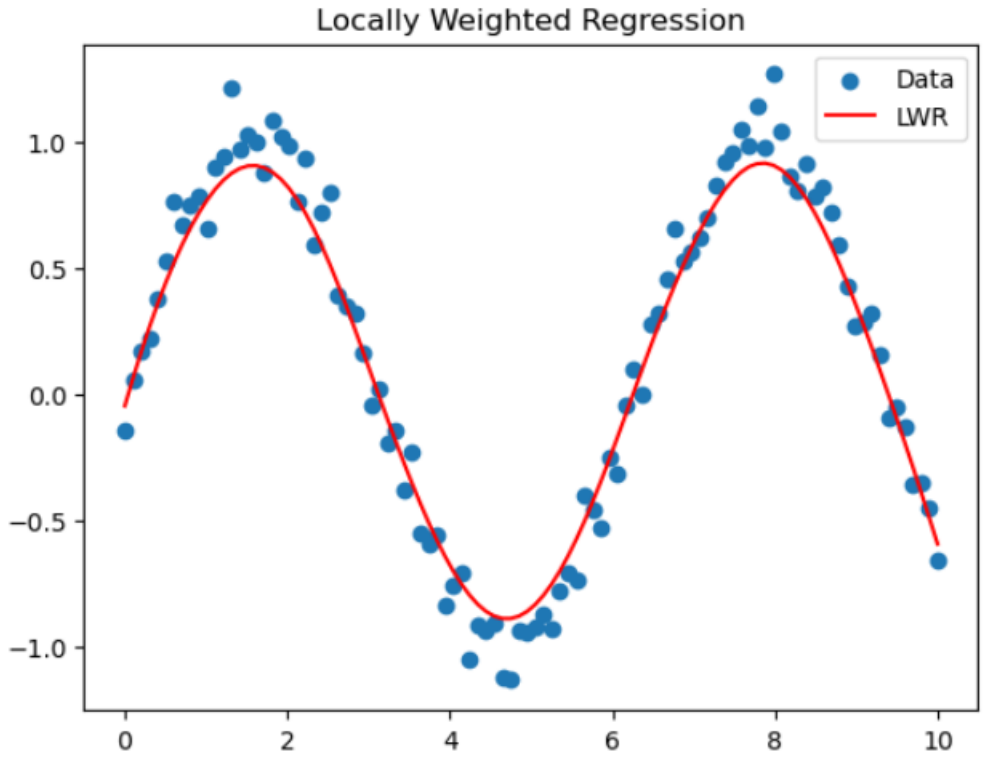
 # Run the models

```
locally_weighted_regression(df_lwr[['X']].values, df_lwr['Y'].values)
```

USN:

Name:

OUTPUT:



USN:

Name:

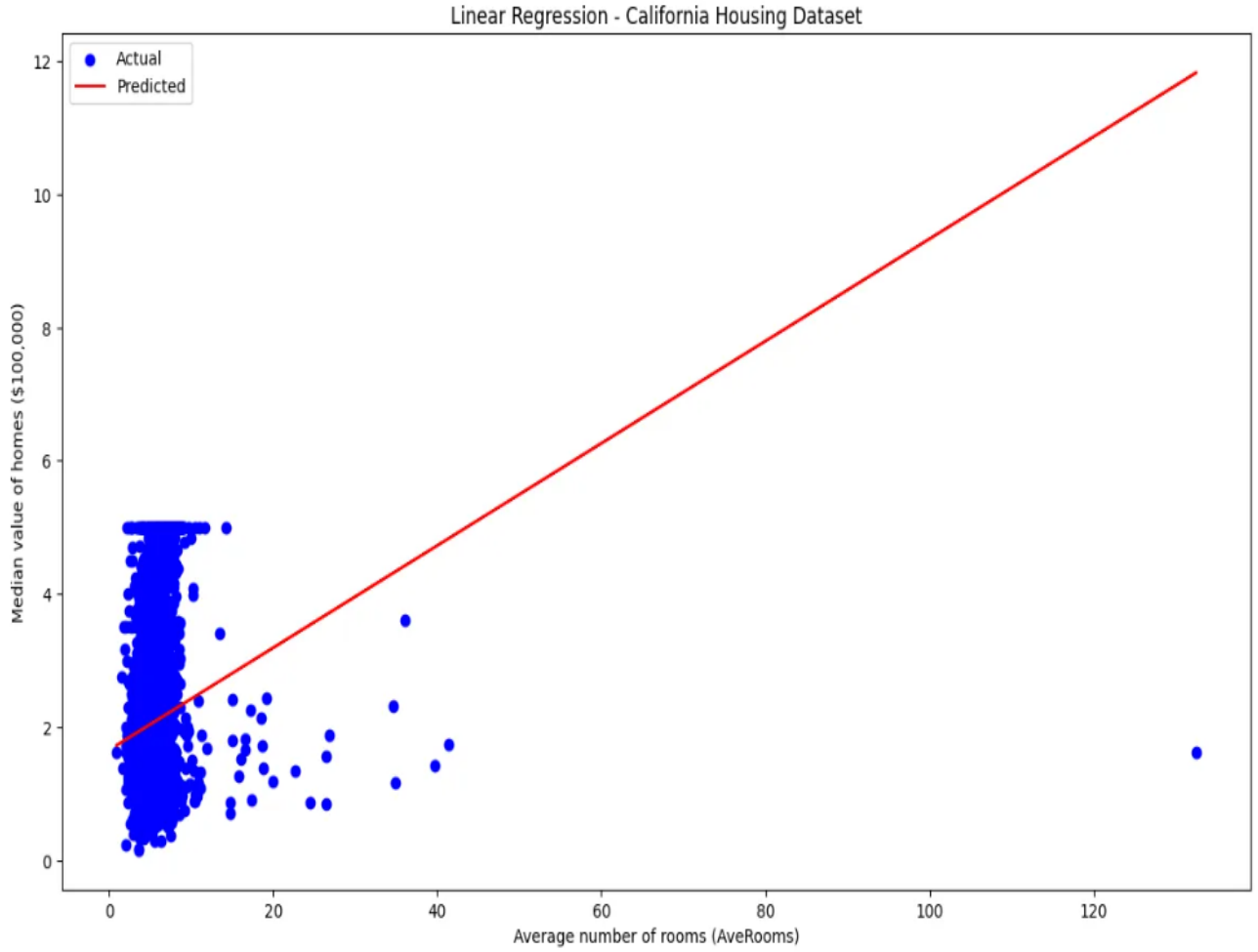
TW-7. Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.

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

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

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

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

    y_pred = model.predict(X_test)

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

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


def polynomial_regression_auto_mpg():
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
    column_names = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration",
"model_year", "origin"]
    data = pd.read_csv(url, sep='\s+', names=column_names, na_values="?")
    data = data.dropna()

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

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

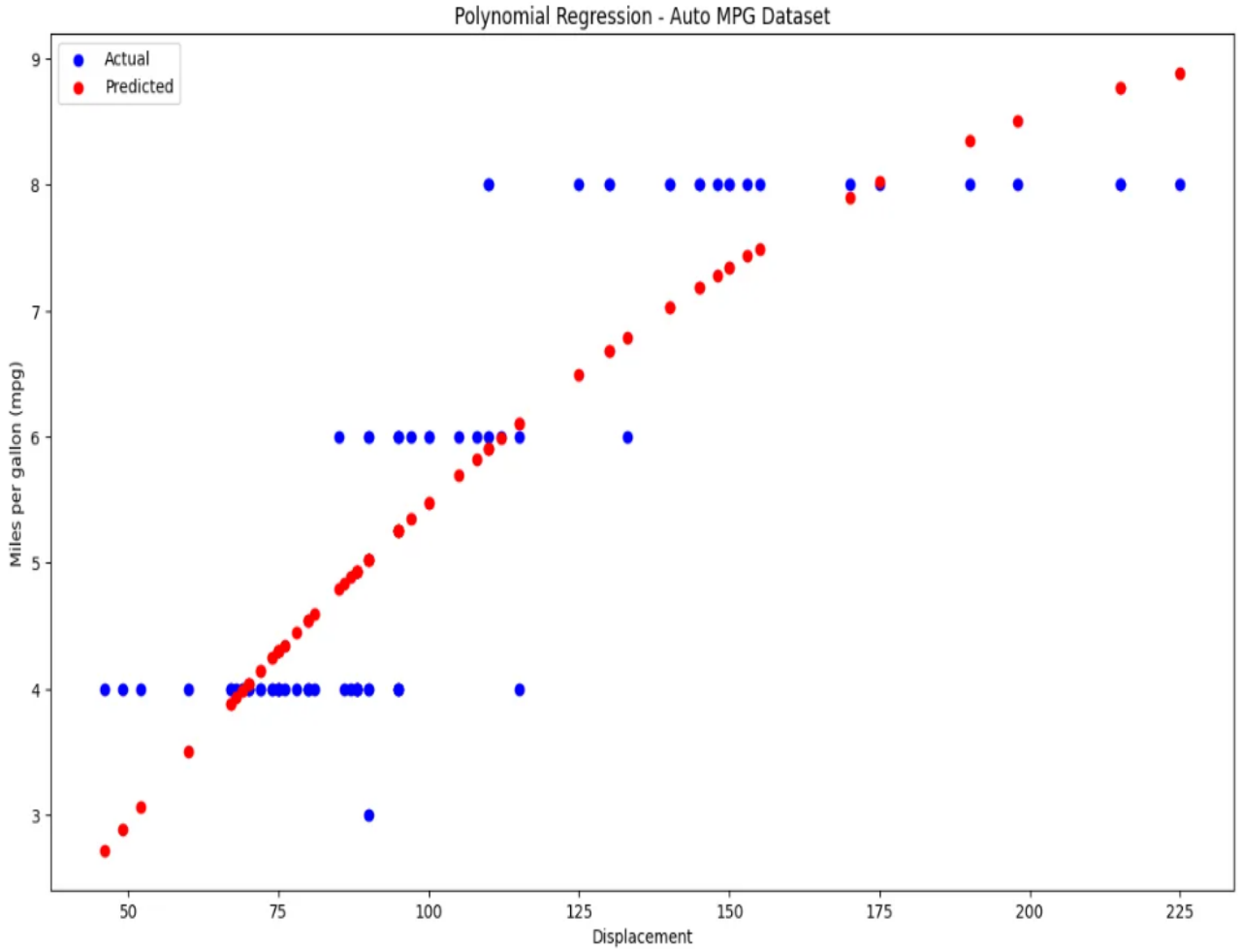
```
    poly_model = make_pipeline(PolynomialFeatures(degree=2), StandardScaler(), LinearRegression())
    poly_model.fit(X_train, y_train)

    y_pred = poly_model.predict(X_test)

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

    print("Polynomial Regression - Auto MPG Dataset")
    print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
    print("R^2 Score:", r2_score(y_test, y_pred))


if __name__ == "__main__":
    print("Demonstrating Linear Regression and Polynomial Regression\n")
    linear_regression_california()
    polynomial_regression_auto_mpg()
```

OUTPUT:

Demonstrating Linear Regression and Polynomial Regression



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



Polynomial Regression - Auto MPG Dataset
Mean Squared Error: 0.7431490557205862
R^2 Score: 0.7505650609469626

USN:

Name:

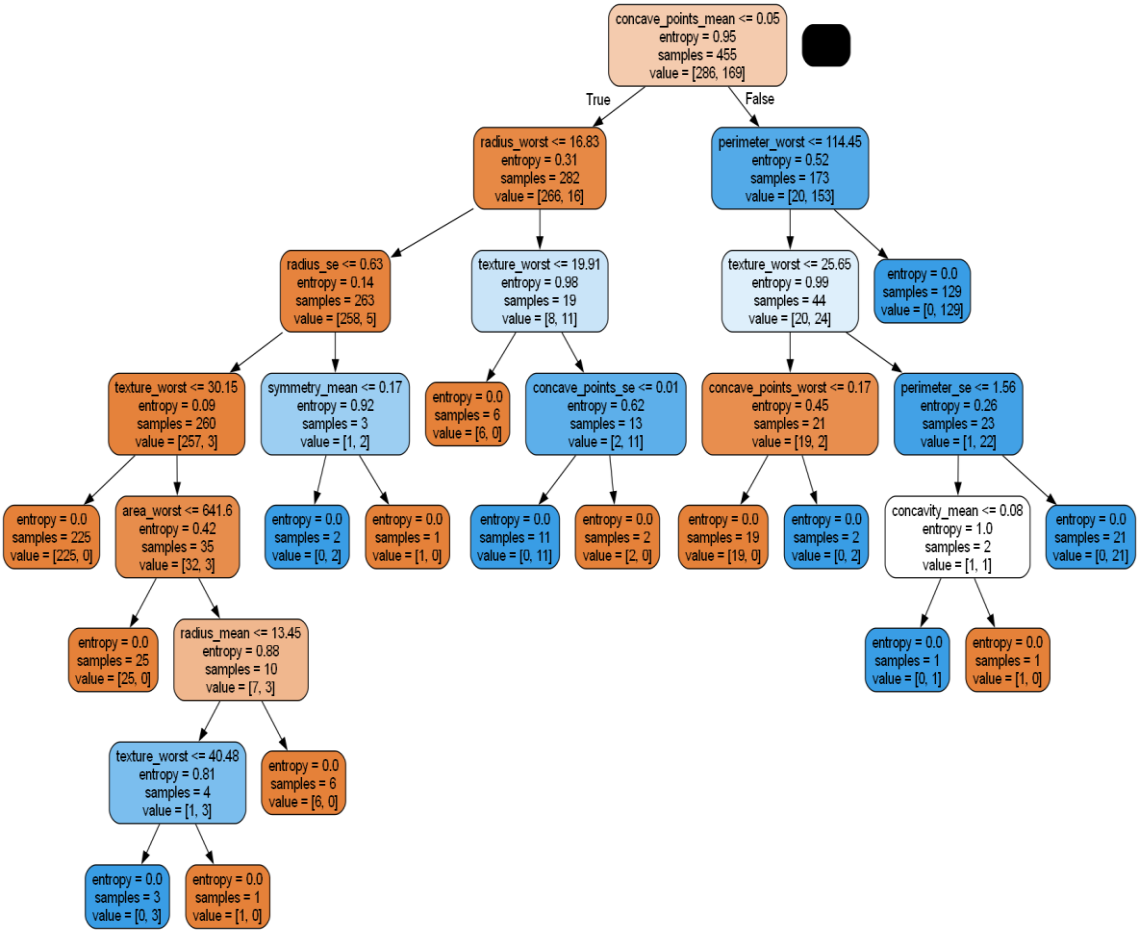
TW-8: Develop a program to demonstrate the working of decision tree algorithm. Use Breast Cancer Data Set for building decision tree and applying this knowledge to classify an new sample

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

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

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

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

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

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

USN:

Name:

OUTPUT:



USN:

Name:

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

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

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

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

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

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

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

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

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

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

plt.show()
```

USN:

Name:

OUTPUT:

USN:

Name:

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

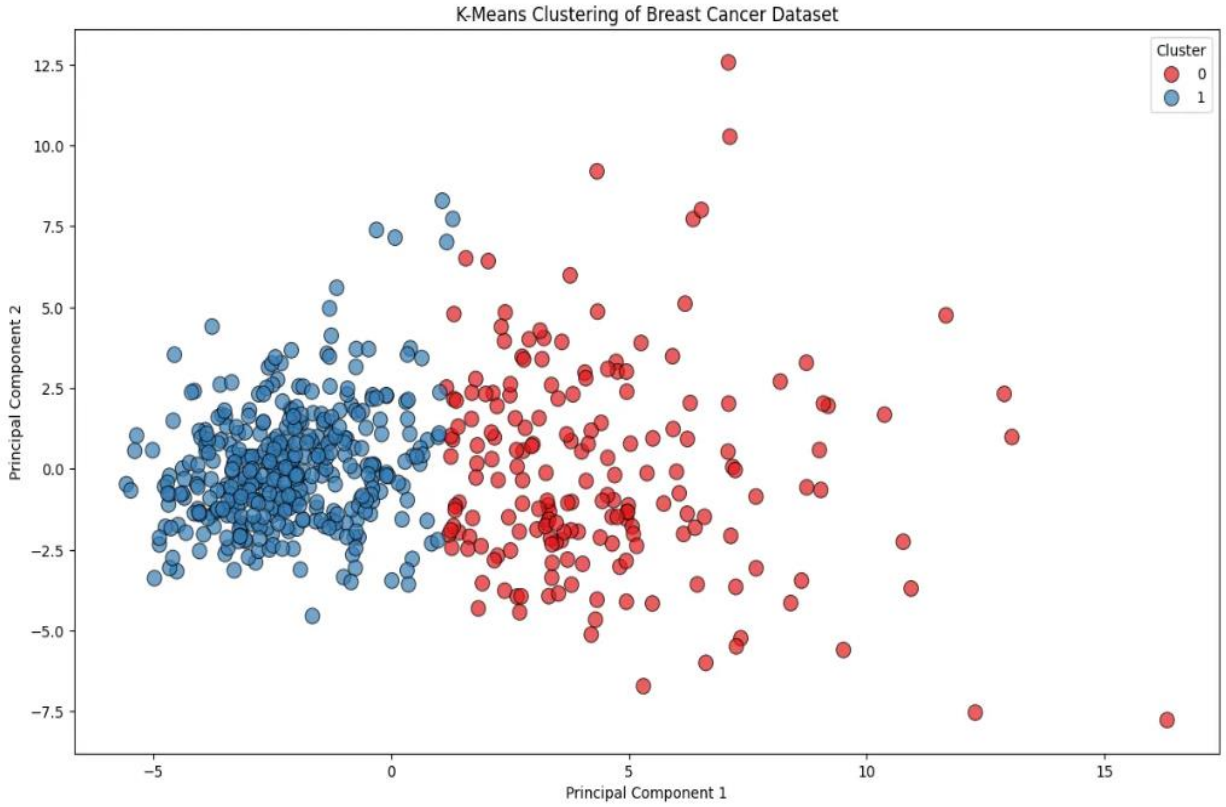
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix,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

import warnings
warnings.filterwarnings('ignore')

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

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

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

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

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

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

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

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='True Label', palette='coolwarm', s=100, edgecolor='black', alpha=0.7)
```

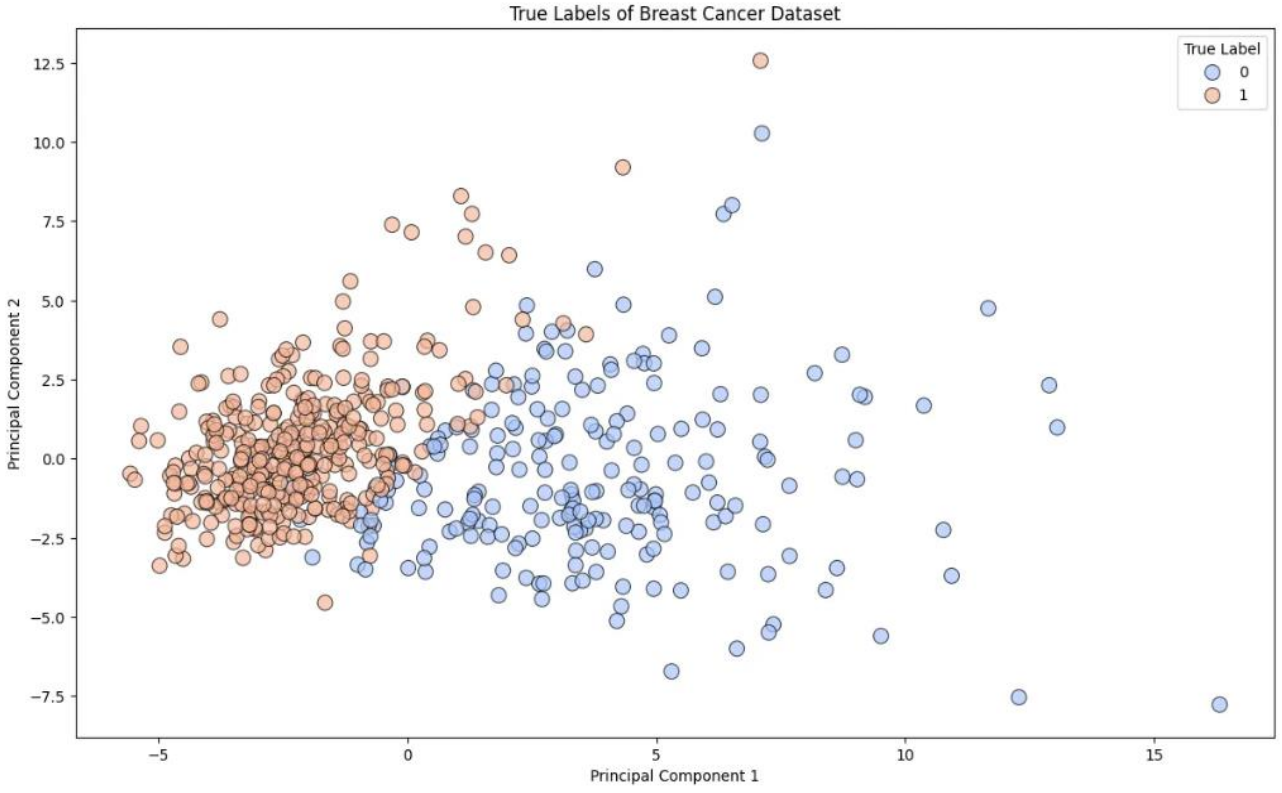
```
plt.title('True Labels of Breast Cancer Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title="True Label")
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PC1', y='PC2', hue='Cluster', palette='Set1', s=100, edgecolor='black', alpha=0.7)

centers = pca.transform(kmeans.cluster_centers_)
plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X', label='Centroids')

plt.title('K-Means Clustering with Centroids')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.legend(title="Cluster")
plt.show()
```

USN:

Name:

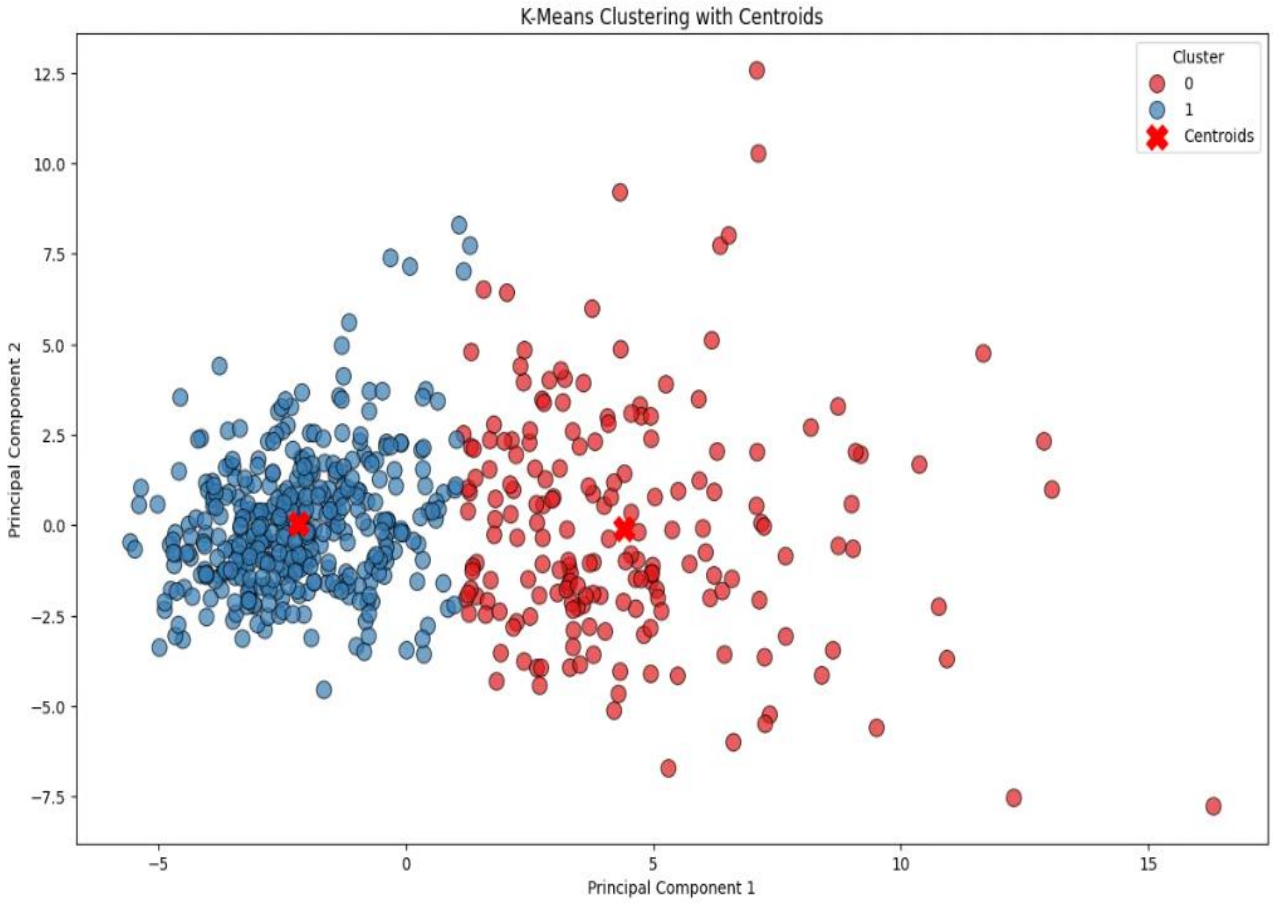
OUTPUT:





USN:

Name:


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