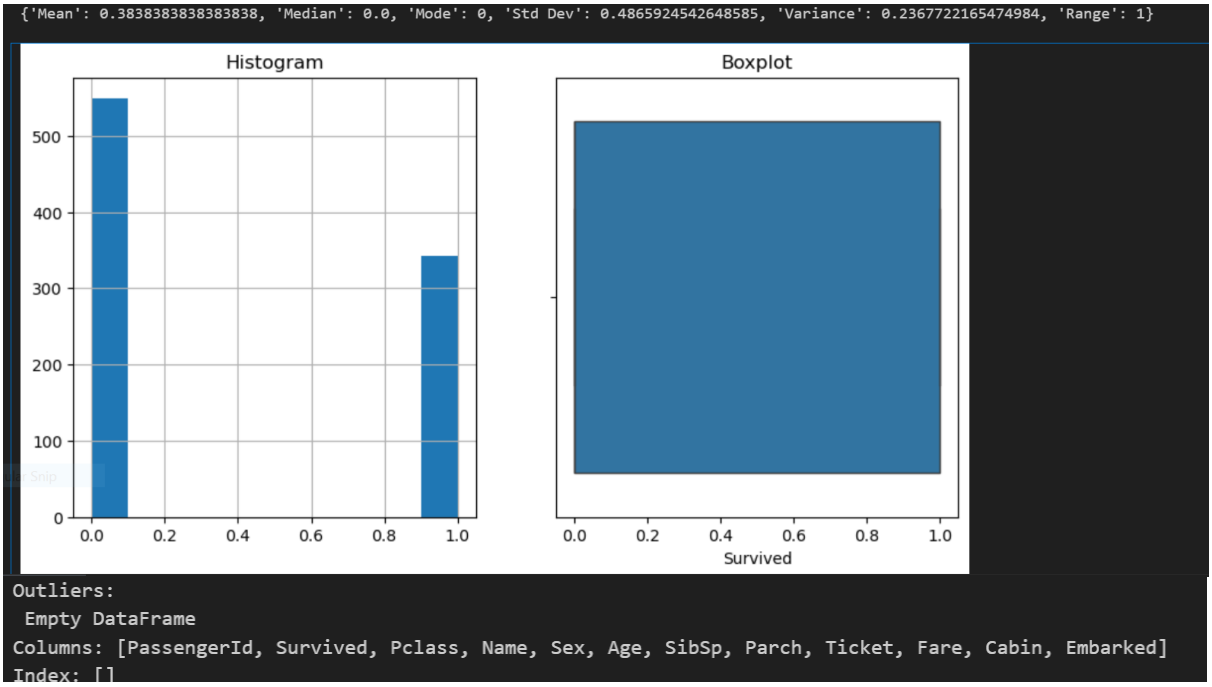
Program 1:

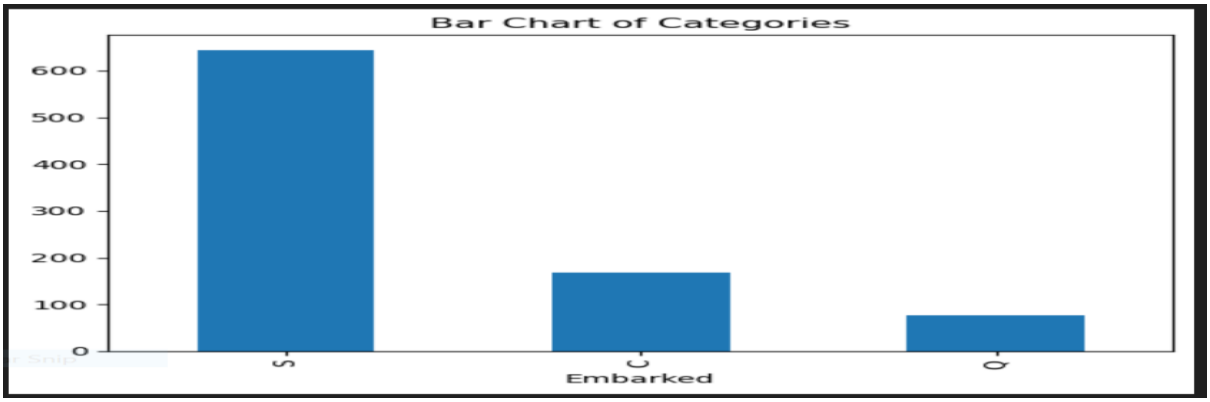
Develop a program to Load a dataset and select one numerical column. Compute mean, median, mode, standard deviation, variance, and range for a given numerical column in a dataset. Generate a histogram and boxplot to understand the distribution of the data. Identify any outliers in the data using IQR. Select a categorical variable from a dataset. Compute the frequency of each category and display it as a bar chart or pie chart.

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

# Load dataset
data = pd.read_csv('titanic.csv')

# Select a numerical column
column = 'Survived'
stats = {
    'Mean': data[column].mean(),
    'Median': data[column].median(),
    'Mode': data[column].mode()[0],
    'Std Dev': data[column].std(),
    'Variance': data[column].var(),
    'Range': data[column].max() - data[column].min()
}
print(stats)

# Histogram and Boxplot
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
data[column].hist()
plt.title('Histogram')

plt.subplot(1, 2, 2)
sns.boxplot(x=data[column])
plt.title('Boxplot')
plt.show()

# Outlier Detection using IQR
Q1 = data[column].quantile(0.25)
Q3 = data[column].quantile(0.75)
```

```
IQR = Q3 - Q1
outliers = data[(data[column] < Q1 - 1.5 * IQR) | (data[column] > Q3 + 1.5 * IQR)]
print("Outliers:\n", outliers)

# Categorical Analysis
cat_col = 'Embarked'  # actual categorical column name
category_counts = data[cat_col].value_counts()
category_counts.plot(kind='bar', title='Bar Chart of Categories')
plt.show()
```

Output:





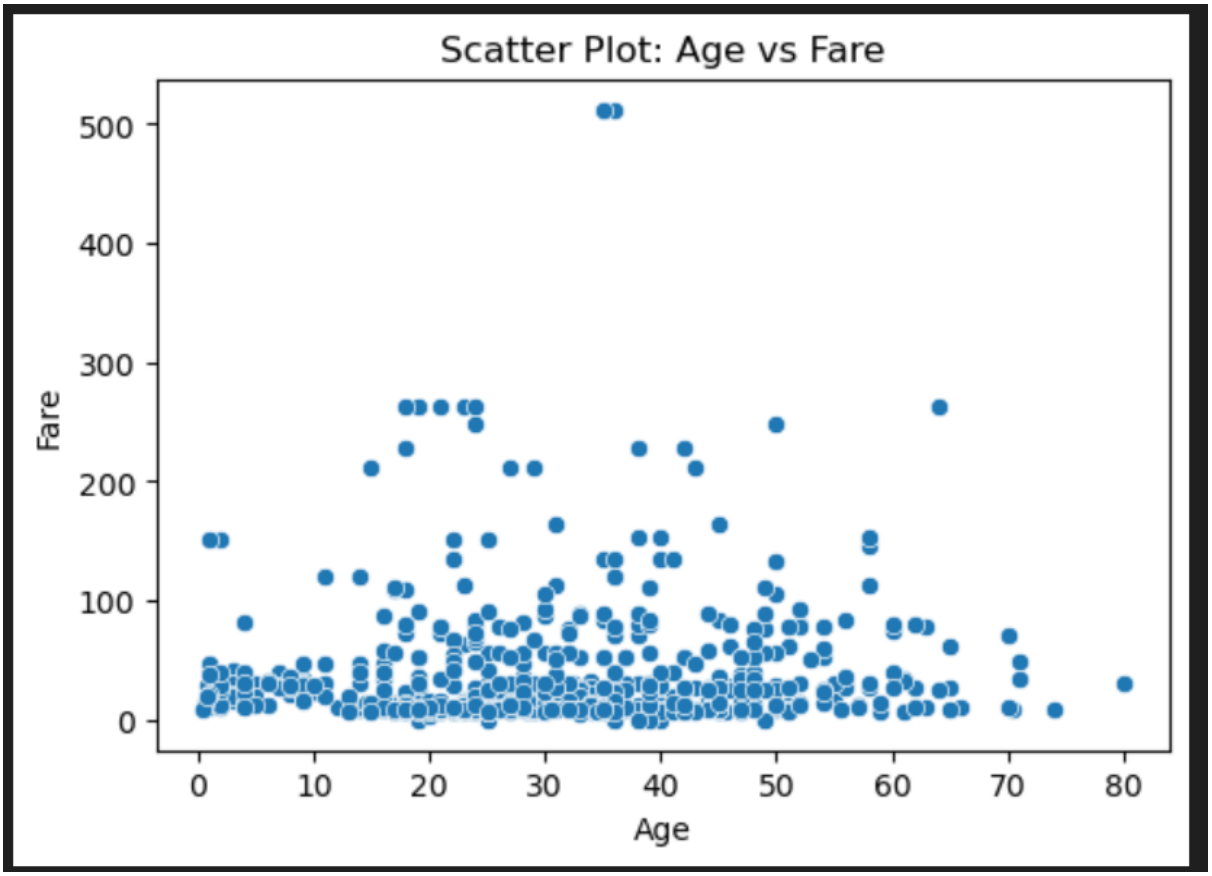
Program 2:

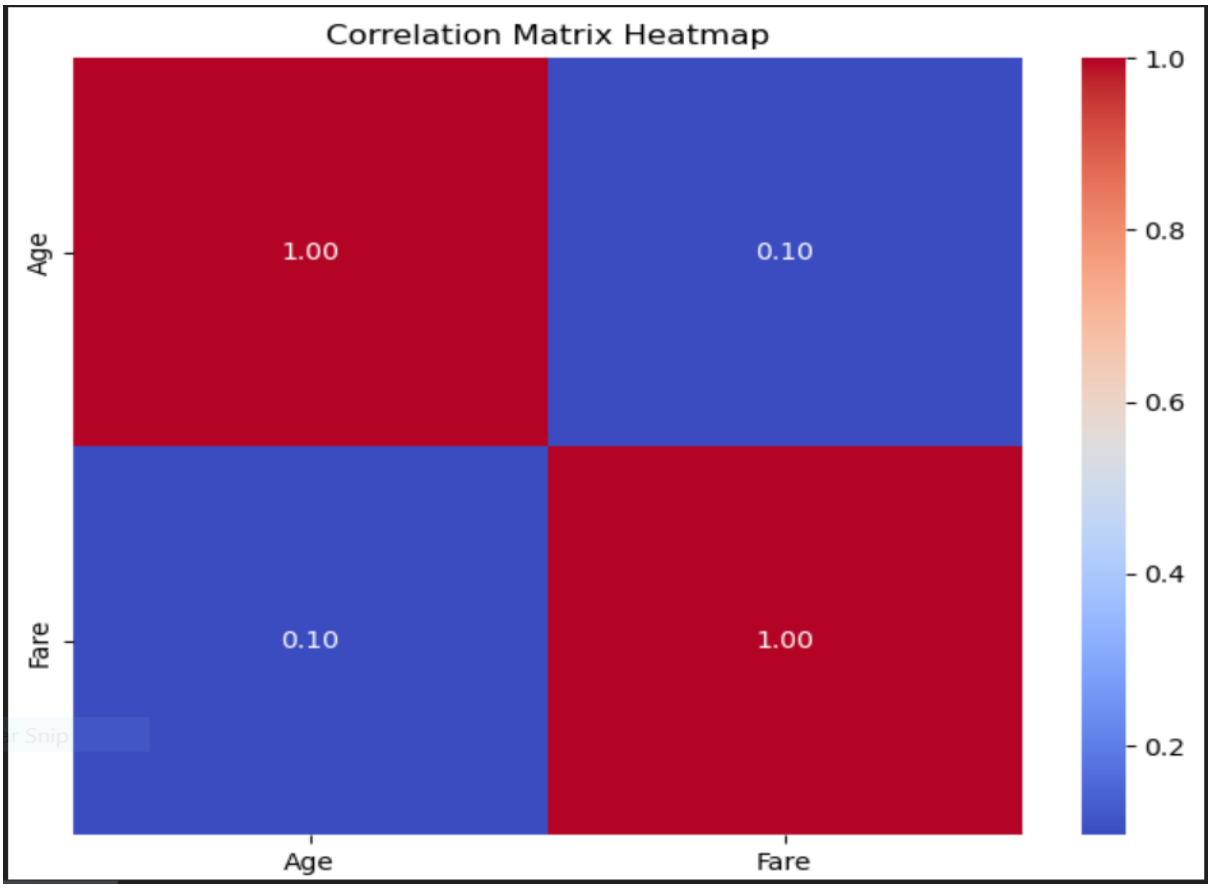
Develop a program to Load a dataset with at least two numerical columns (e.g., Iris, Titanic). Plot a scatter plot of two variables and calculate their Pearson correlation coefficient. Write a program to compute the covariance and correlation matrix for a dataset. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations.

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

# Load dataset (Replace 'titanic.csv' with your actual dataset)
data = pd.read_csv('titanic.csv')

# Select numerical columns for correlation analysis
numerical_columns = ['Age', 'Fare']  # Modify as per dataset
df = data[numerical_columns].dropna()  # Drop missing values

# Compute Correlation Matrix
corr_matrix = df.corr()
print("Correlation Matrix:\n", corr_matrix)

# Compute Covariance Matrix
cov_matrix = df.cov()
print("\nCovariance Matrix:\n", cov_matrix)

# Scatterplot to visualize relationships
plt.figure(figsize=(6, 4))
sns.scatterplot(x=df['Age'], y=df['Fare'])
plt.xlabel('Age')
plt.ylabel('Fare')
plt.title('Scatter Plot: Age vs Fare')
plt.show()

# Correlation Matrix Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()
```

Output:

```
Correlation Matrix:
             Age      Fare
Age     1.000000  0.096067
Fare    0.096067  1.000000

Covariance Matrix:
               Age         Fare
Age     211.019125    73.84903
Fare     73.849030  2800.41310
```



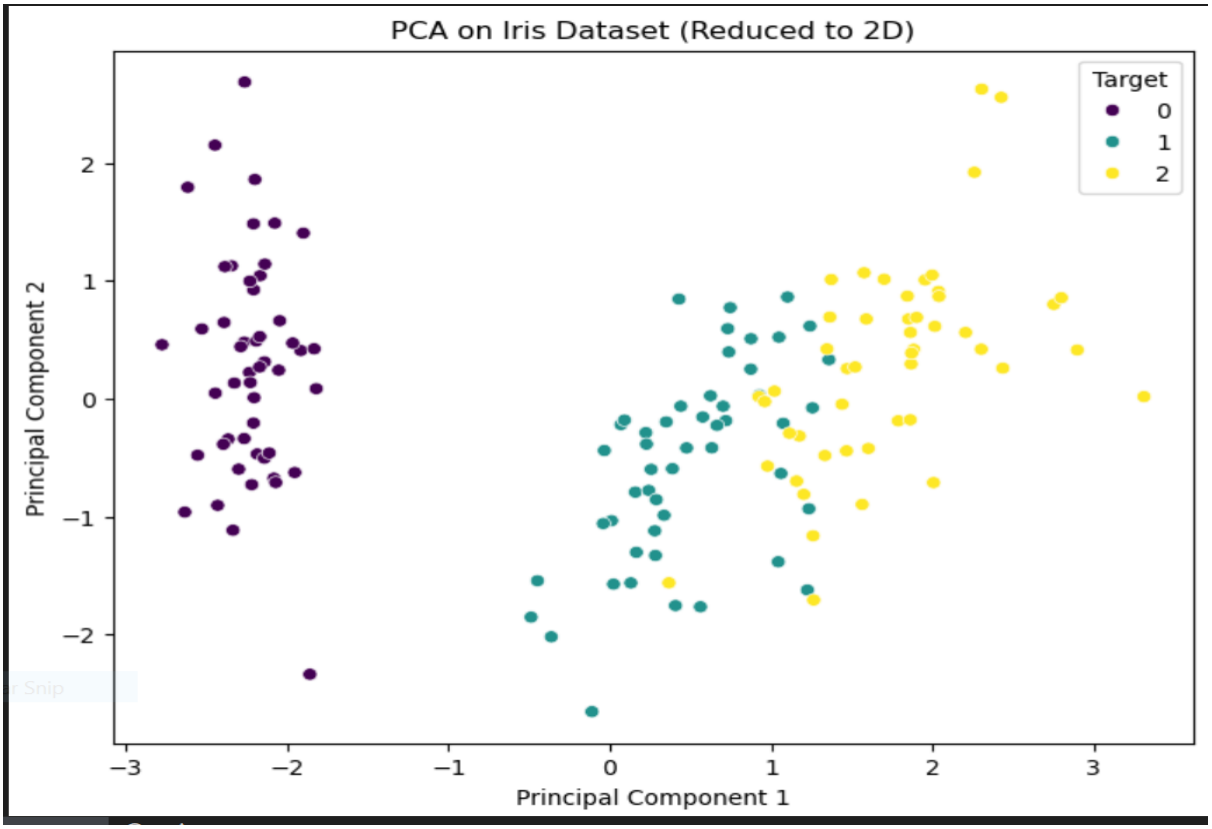

Correlation Matrix Heatmap
Age
Fare
1.00
0.10
0.10
1.00
Age
Fare
1.0
0.8
0.6
0.4
0.2

Program 3:

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

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

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Extract feature matrix (4 features)
y = iris.target  # Target labels

# Standardizing the data (PCA works better with normalized data)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to reduce from 4D to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Convert PCA result into a DataFrame
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['Target'] = y  # Add target labels

# Scatter plot of PCA results
plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_df['PC1'], y=pca_df['PC2'], hue=pca_df['Target'], palette='viridis',
legend=True)
plt.title('PCA on Iris Dataset (Reduced to 2D)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Explained variance ratio
print(f'Explained Variance Ratio: {pca.explained_variance_ratio_}')
```

Output:


PCA on Iris Dataset (Reduced to 2D)
Target
0
1
2
Principal Component 2
Principal Component 1
2
1
0
−1
−2
−3
−2
−1
0
1
2
3

Program 4:

Develop a program to load the Iris dataset. Implement the k-Nearest Neighbors (k-NN) algorithm for classifying flowers based on their features. Split the dataset into training and testing sets and evaluate the model using metrics like accuracy and F1-score. Test it for different values of k (e.g., k=1,3,5) and evaluate the accuracy. Extend the k-NN algorithm to assign weights based on the distance of neighbors (e.g., $weight=1/d2$). Compare the performance of weighted k-NN and regular k-NN on a synthetic or real-world dataset.

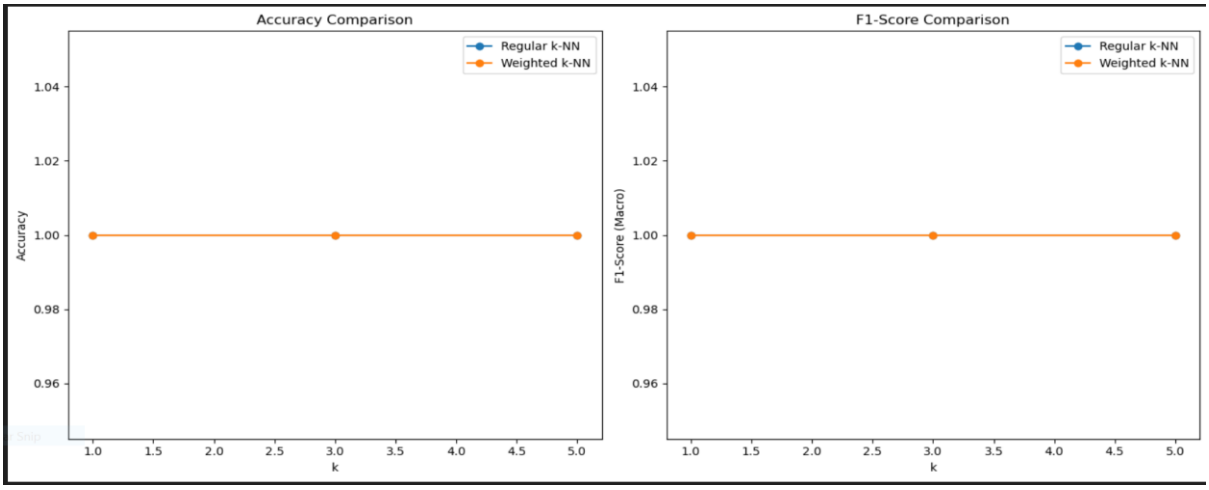
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from collections import Counter

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

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

# Euclidean distance function
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

# k-NN algorithm (regular and weighted)
def knn(X_train, y_train, X_test, k=3, weighted=False):
    y_pred = []
    for test_point in X_test:
        distances = [euclidean_distance(test_point, train_point) for train_point in X_train]
        k_indices = np.argsort(distances)[:k]
        k_nearest_labels = [y_train[i] for i in k_indices]

        if weighted:
            # Compute weights (1/d^2)
            weights = [1 / (distances[i]**2) if distances[i] != 0 else 1e-10 for i in k_indices]
            # Weighted majority voting
            class_votes = {}
            for label, weight in zip(k_nearest_labels, weights):
                if label not in class_votes:
                    class_votes[label] = 0
                class_votes[label] += weight
```

```
            y_pred.append(max(class_votes, key=class_votes.get))
        else:
            # Regular majority voting
            y_pred.append(Counter(k_nearest_labels).most_common(1)[0][0])
    return np.array(y_pred)

# Test the models for different k values and compare
k_values = [1, 3, 5]
results = {'k': [], 'Accuracy (Regular k-NN)': [], 'F1-Score (Regular k-NN)': [],
           'Accuracy (Weighted k-NN)': [], 'F1-Score (Weighted k-NN)': []}

for k in k_values:
    # Regular k-NN
    y_pred_knn = knn(X_train, y_train, X_test, k=k, weighted=False)
    accuracy_knn = accuracy_score(y_test, y_pred_knn)
    f1_knn = f1_score(y_test, y_pred_knn, average='macro')

    # Weighted k-NN
    y_pred_wknn = knn(X_train, y_train, X_test, k=k, weighted=True)
    accuracy_wknn = accuracy_score(y_test, y_pred_wknn)
    f1_wknn = f1_score(y_test, y_pred_wknn, average='macro')

    # Store results
    results['k'].append(k)
    results['Accuracy (Regular k-NN)'].append(accuracy_knn)
    results['F1-Score (Regular k-NN)'].append(f1_knn)
    results['Accuracy (Weighted k-NN)'].append(accuracy_wknn)
    results['F1-Score (Weighted k-NN)'].append(f1_wknn)

# Convert results to DataFrame
results_df = pd.DataFrame(results)
print(results_df)

# Plot comparison of accuracy and F1-score for different k values
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Accuracy comparison
axes[0].plot(results_df['k'], results_df['Accuracy (Regular k-NN)'], label='Regular k-NN',
marker='o')
axes[0].plot(results_df['k'], results_df['Accuracy (Weighted k-NN)'], label='Weighted k-NN',
marker='o')
axes[0].set_title('Accuracy Comparison')
axes[0].set_xlabel('k')
axes[0].set_ylabel('Accuracy')
axes[0].legend()
```

```
# F1-Score comparison
axes[1].plot(results_df['k'], results_df['F1-Score (Regular k-NN)'], label='Regular k-NN',
marker='o')
axes[1].plot(results_df['k'], results_df['F1-Score (Weighted k-NN)'], label='Weighted k-NN',
marker='o')
axes[1].set_title('F1-Score Comparison')
axes[1].set_xlabel('k')
axes[1].set_ylabel('F1-Score (Macro)')
axes[1].legend()

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

Output:

```
   k  Accuracy (Regular k-NN)  F1-Score (Regular k-NN)  \
0  1                      1.0                      1.0
1  3                      1.0                      1.0
2  5                      1.0                      1.0

   Accuracy (Weighted k-NN)  F1-Score (Weighted k-NN)
0                       1.0                       1.0
1                       1.0                       1.0
2                       1.0                       1.0
```



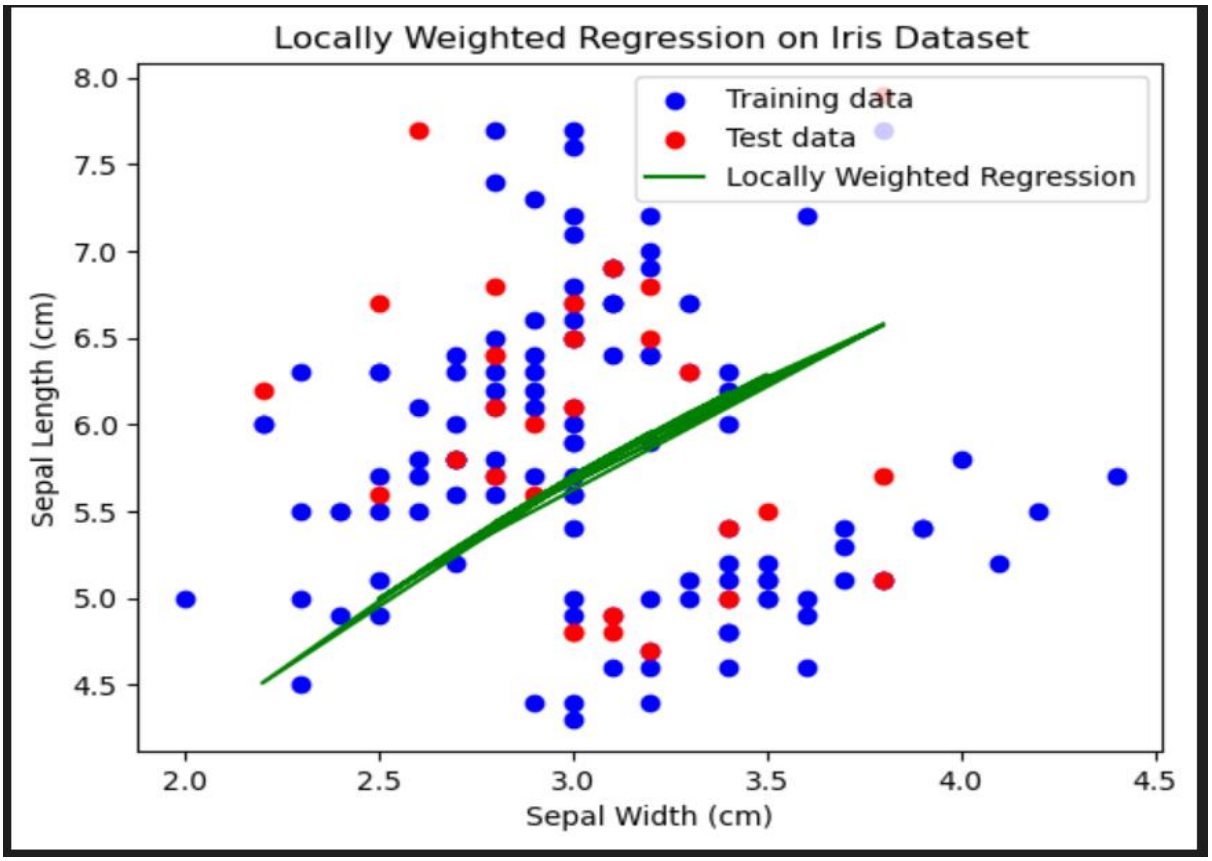
Program 6:

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

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)

# Let's predict the 'sepal length' based on 'sepal width' for simplicity
X = data[['sepal width (cm)']]  # Independent variable (sepal width)
y = data['sepal length (cm)']   # Dependent variable (sepal length)

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

# Locally Weighted Regression function
def locally_weighted_regression(X_train, y_train, X_test, tau=1.0):
    m = X_train.shape[0]
    y_pred = np.zeros(X_test.shape[0])

    for i in range(X_test.shape[0]):
        # Compute the weight matrix
        weights = np.exp(-np.sum((X_train - X_test.iloc[i]) ** 2, axis=1) / (2 * tau ** 2))

        # Create weighted X matrix and y vector
        X_weighted = X_train * weights[:, np.newaxis]
        y_weighted = y_train * weights

        # Calculate the regression coefficients using weighted least squares
        X_weighted_transpose = X_weighted.T
        theta = np.linalg.inv(X_weighted_transpose @ X_weighted) @ X_weighted_transpose @
y_weighted

        # Predict for the i-th test point
        y_pred[i] = np.dot(X_test.iloc[i], theta)

    return y_pred

# Perform Locally Weighted Regression
```

```
y_pred = locally_weighted_regression(X_train, y_train, X_test, tau=1.0)

# Visualize the results
plt.scatter(X_train, y_train, color='blue', label='Training data')
plt.scatter(X_test, y_test, color='red', label='Test data')
plt.plot(X_test, y_pred, color='green', label='Locally Weighted Regression')
plt.xlabel('Sepal Width (cm)')
plt.ylabel('Sepal Length (cm)')
plt.legend()
plt.title('Locally Weighted Regression on Iris Dataset')
plt.show()
```

Output:



Program 7:

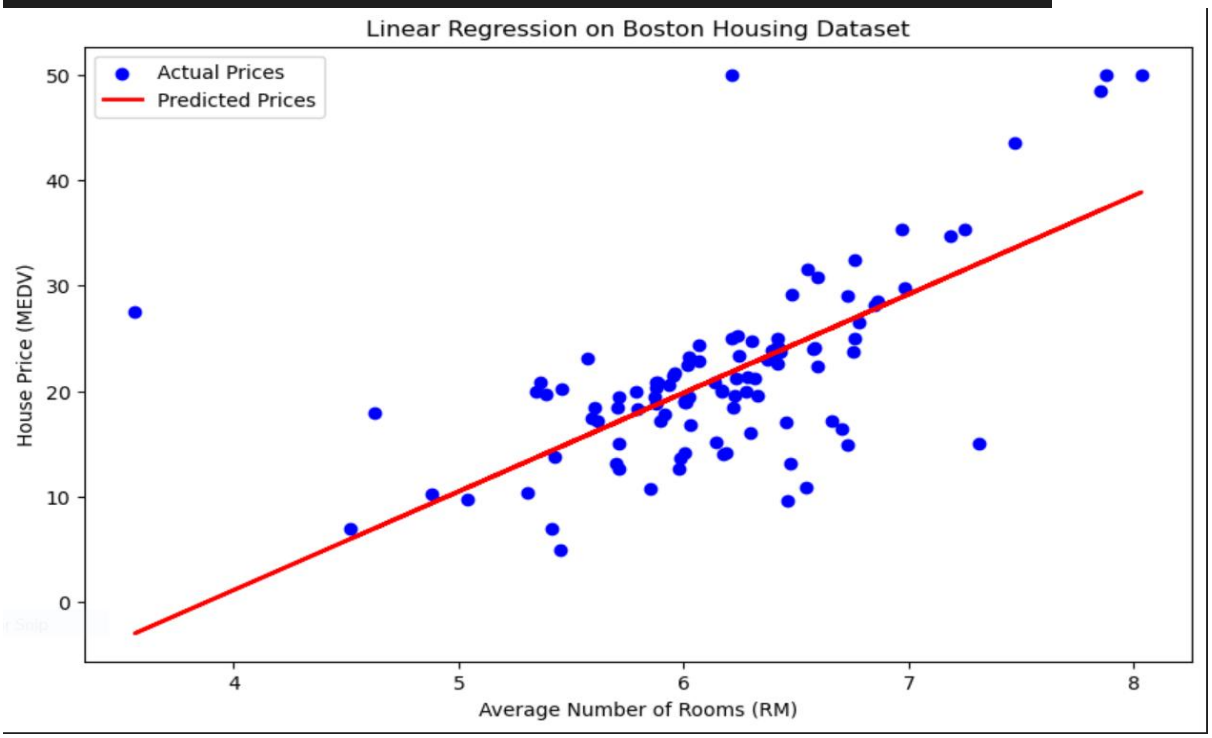
Develop a program to demonstrate the working of Linear Regression and Polynomial Regression. Use Boston Housing Dataset for Linear Regression and Auto MPG Dataset (for vehicle fuel efficiency prediction) for Polynomial Regression.

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

# ------------------------------
# Load Boston Housing Dataset Manually (Fix for load_boston())
# ------------------------------
url = "https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv"
boston_df = pd.read_csv(url)

# Selecting feature (RM = Average number of rooms) and target (MEDV = House Price)
X_boston_feature = boston_df[['rm']]
y_boston = boston_df['medv']

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X_boston_feature, y_boston, test_size=0.2,
random_state=42)

# Linear Regression Model
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)

# Predictions
y_pred = linear_reg.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Linear Regression Results (Boston Housing):")
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual Prices')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted Prices')
```

```
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('House Price (MEDV)')
plt.title('Linear Regression on Boston Housing Dataset')
plt.legend()
plt.show()
```

Output:

```
Linear Regression Results (Boston Housing):
Mean Squared Error: 46.144775347317264
R^2 Score: 0.3707569232254778
```



Program 8:

Develop a program to load the Titanic dataset. Split the data into training and test sets. Train a decision tree classifier. Visualize the tree structure. Evaluate accuracy, precision, recall, and F1-score.

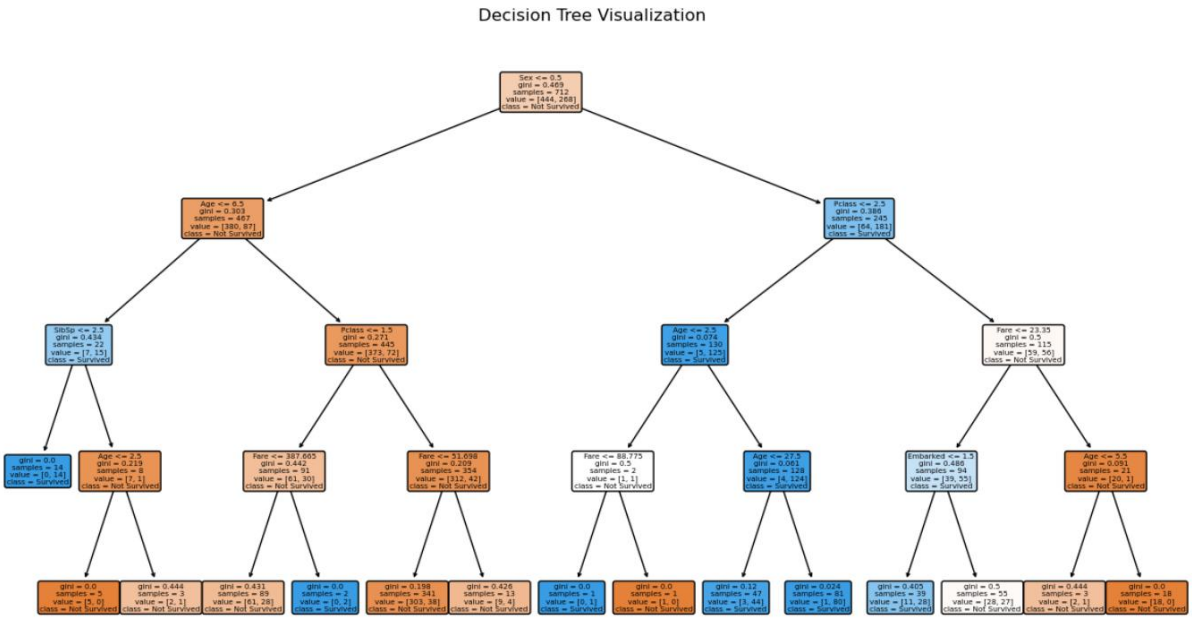
```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# ------------------------------
# Load the Titanic Dataset
# ------------------------------
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)

# Display first few rows
print("Dataset Preview:\n", data.head())

# ------------------------------
# Data Preprocessing
# ------------------------------
# Select relevant features
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
data = data[features + ['Survived']]

# Handle missing values
data['Age'].fillna(data['Age'].median(), inplace=True)
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Convert categorical variables to numeric
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})  # Male = 0, Female = 1
data['Embarked'] = data['Embarked'].map({'C': 0, 'Q': 1, 'S': 2})  # Encoding Embarked

# ------------------------------
# Splitting Data into Train & Test Sets
# ------------------------------
X = data.drop(columns=['Survived'])
y = data['Survived']

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

```
# ------------------------------
# Train Decision Tree Classifier
# ------------------------------
dtree = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=42)
dtree.fit(X_train, y_train)

# ------------------------------
# Visualizing the Decision Tree
# ------------------------------
plt.figure(figsize=(15, 8))
plot_tree(dtree, feature_names=X.columns.tolist(), class_names=['Not Survived', 'Survived'],
filled=True, rounded=True)
plt.title("Decision Tree Visualization")
plt.show()

# ------------------------------
# Model Evaluation
# ------------------------------
y_pred = dtree.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Display Metrics
print("\nDecision Tree Model Evaluation:")
print(f"Accuracy  : {accuracy:.4f}")
print(f"Precision : {precision:.4f}")
print(f"Recall    : {recall:.4f}")
print(f"F1 Score  : {f1:.4f}")
```

Output:

```
Decision Tree Model Evaluation:
Accuracy  : 0.7989
Precision : 0.8393
Recall    : 0.6351
F1 Score  : 0.7231
```

```
Dataset Preview:
    PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```



Program 9:

Develop a program to implement the Naive Bayesian classifier considering Iris dataset for training. Compute the accuracy of the classifier, considering the test data.

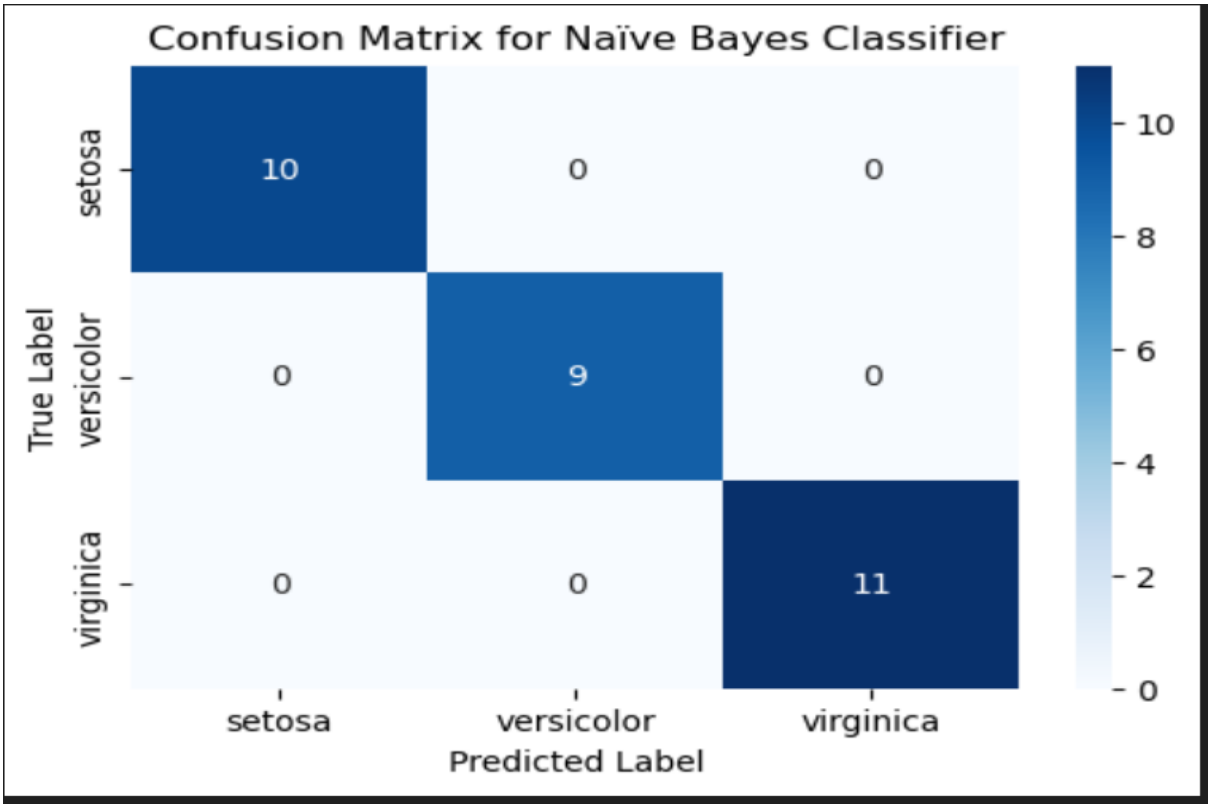
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# ------------------------------
# Load the Iris Dataset
# ------------------------------
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Labels (0 = Setosa, 1 = Versicolor, 2 = Virginica)

# Convert dataset to DataFrame for better readability
df = pd.DataFrame(X, columns=iris.feature_names)
df['species'] = y

# Display dataset info
print("Dataset Preview:\n", df.head())

# ------------------------------
# Splitting Data into Train & Test Sets
# ------------------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ------------------------------
# Train Naïve Bayes Classifier
# ------------------------------
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# ------------------------------
# Model Predictions
# ------------------------------
y_pred = nb_classifier.predict(X_test)

# ------------------------------
# Model Evaluation
```

```
# ------------------------------
accuracy = accuracy_score(y_test, y_pred)
print(f"\nNaïve Bayes Classifier Accuracy: {accuracy:.4f}")

# Display Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Display Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d', xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix for Naïve Bayes Classifier")
plt.show()
```

Output:

```
Dataset Preview:
    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   species
0        0
1        0
2        0
3        0
4        0

Naïve Bayes Classifier Accuracy: 1.0000
```

```
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



Program 10:

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

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

# ---------------------------------------------------------------------
# 1. Load the Wisconsin Breast Cancer Dataset from "data.csv"
# ---------------------------------------------------------------------
data = pd.read_csv("data.csv")  # Ensure "data.csv" is in the same directory
print("Original Data Shape:", data.shape)
print(data.head())

# --------------------------------
# 2. Data Preprocessing
# --------------------------------
# Drop ID column if it exists
if "id" in data.columns:
    data.drop(columns=["id"], inplace=True)

# Convert diagnosis column to numeric (Malignant: 1, Benign: 0)
if "diagnosis" in data.columns:
    data["diagnosis"] = data["diagnosis"].map({"M": 1, "B": 0})

# Separate features and labels (if available)
if "diagnosis" in data.columns:
    X = data.drop(columns=["diagnosis"])  # Features
    true_labels = data["diagnosis"].values  # Actual class labels
else:
    X = data.copy()
    true_labels = None

# ------------------------------
# 3. Feature Scaling
# ------------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# ------------------------------------
# 4. Apply K-Means Clustering
# ------------------------------------

# Use 2 clusters (Benign & Malignant) and set n_init='auto' for compatibility
kmeans = KMeans(n_clusters=2, random_state=42, n_init='auto')
clusters = kmeans.fit_predict(X_scaled)
data["Cluster"] = clusters  # Store cluster assignments in DataFrame


# ------------------------------------------------------------
# 5. (Optional) Compare Clusters with Actual Diagnosis
# ------------------------------------------------------------

if true_labels is not None:
    # Since K-Means assigns labels arbitrarily, check both alignments
    acc1 = np.mean(clusters == true_labels)
    acc2 = np.mean((1 - clusters) == true_labels)
    best_accuracy = max(acc1, acc2)
    print(f"\nBest Clustering Accuracy: {best_accuracy:.4f}")

# ------------------------------------------------
# 6. Visualization using PCA (2D Projection)
# ------------------------------------------------

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

plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=clusters, palette="coolwarm", alpha=0.8)
plt.title("K-Means Clustering on Wisconsin Breast Cancer Data (PCA Projection)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Cluster")
plt.show()
```

Output:

```
Original Data Shape: (569, 33)
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622
1  ...          23.41           158.80      1956.0            0.1238
```