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

import 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)

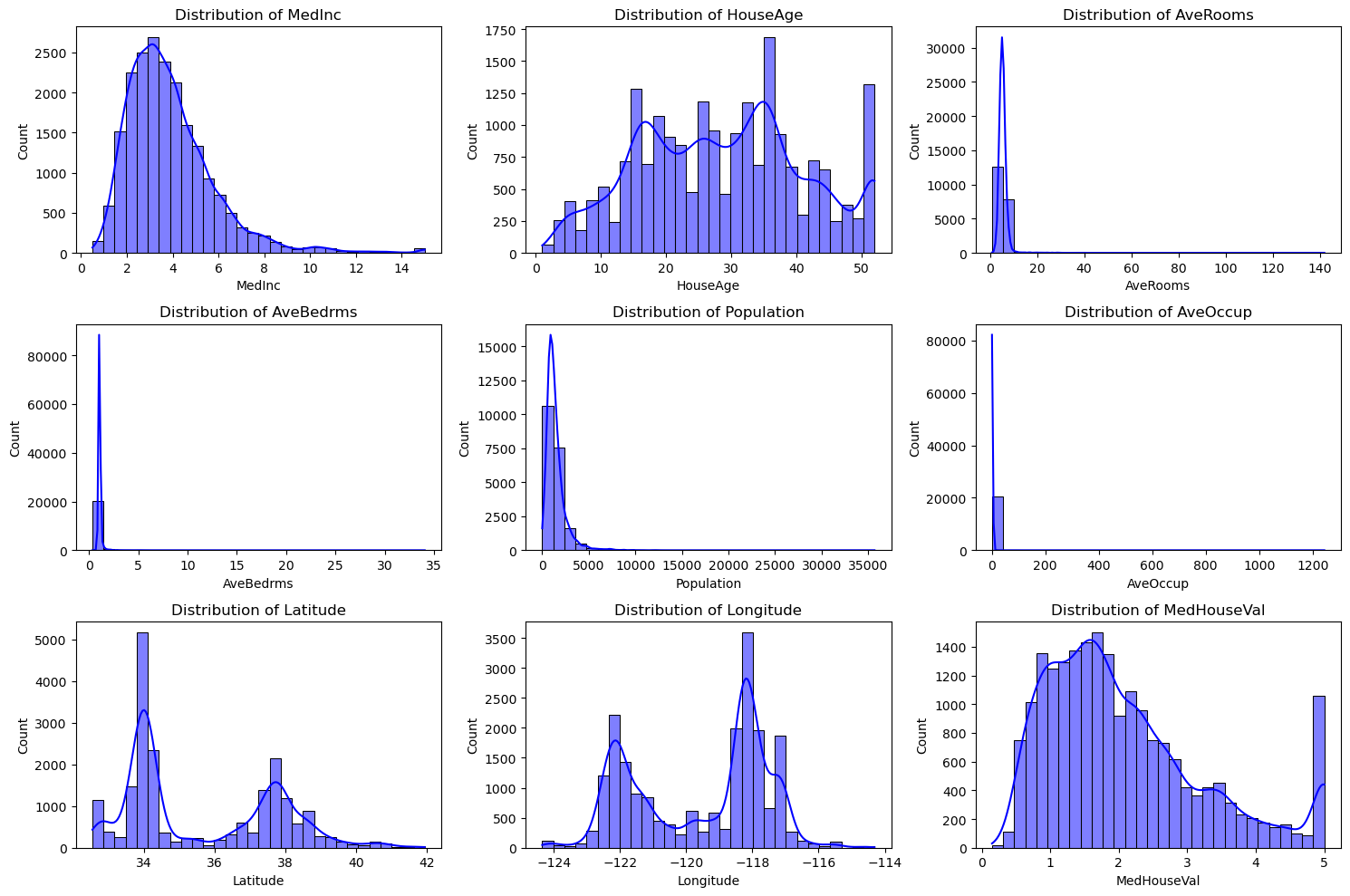
    print(f"{feature}: {len(outliers)} outliers")

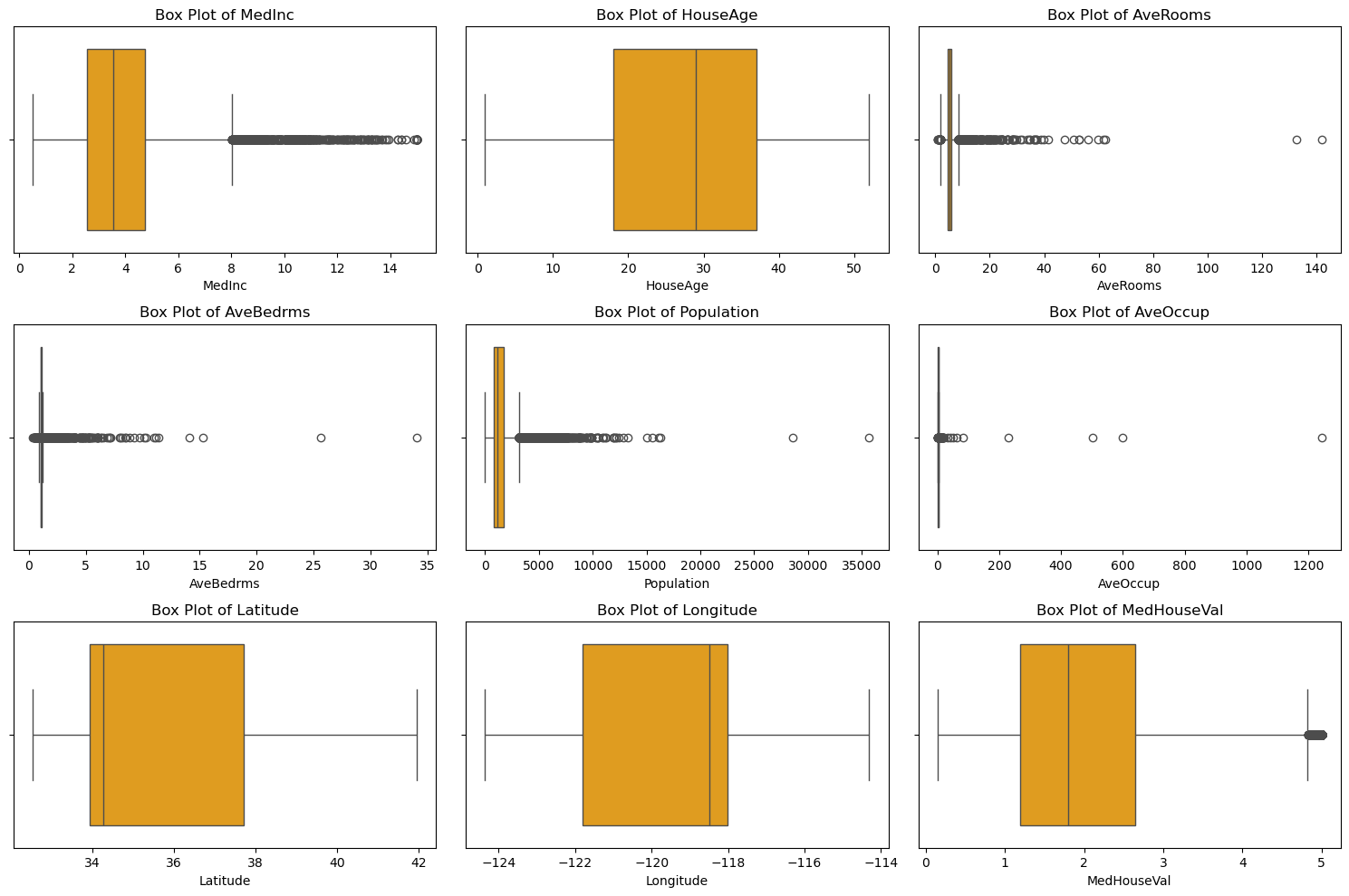
# Optional: Print a summary of the dataset

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.

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_california\_housing

# Step 1: Load the California Housing Dataset

california\_data = fetch\_california\_housing(as\_frame=True)

data = california\_data.frame

# Step 2: Compute the correlation matrix

correlation\_matrix = data.corr()

# Step 3: Visualize the correlation matrix using a heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Matrix of California Housing Features')

plt.show()

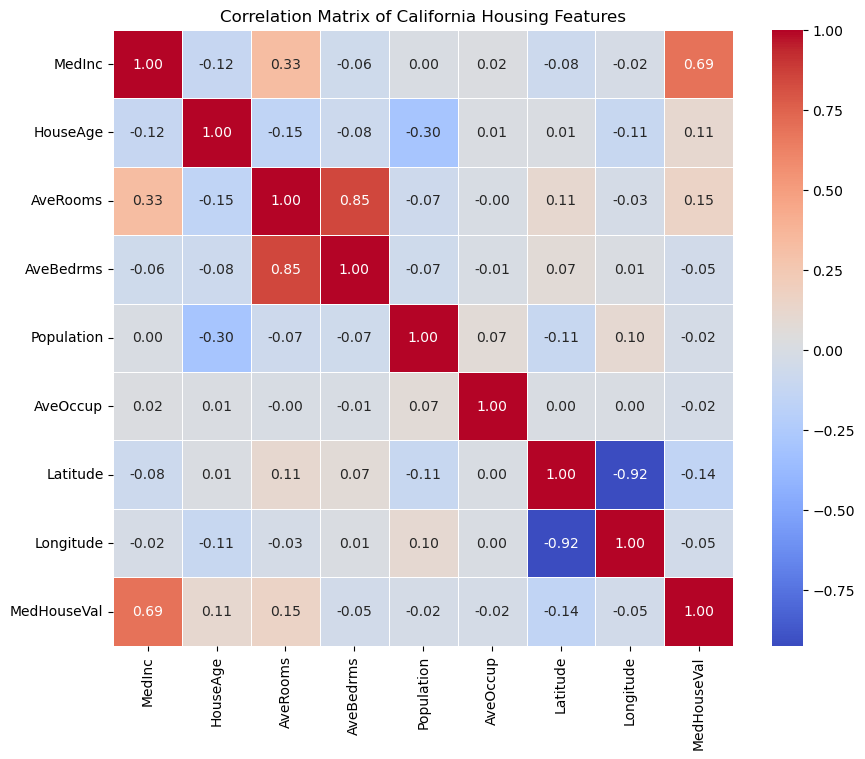
# Step 4: Create a pair plot to visualize pairwise relationships

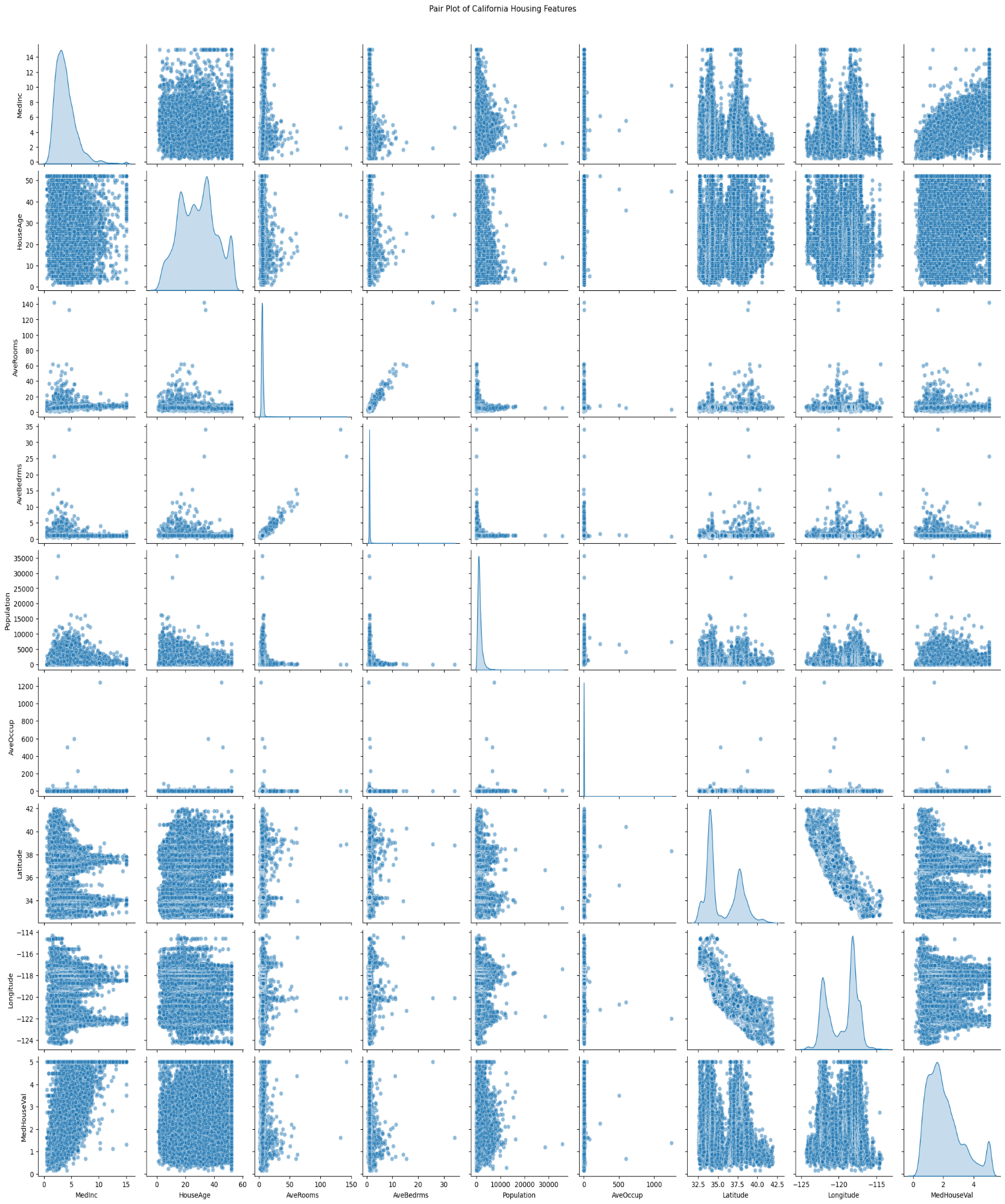
sns.pairplot(data, diag\_kind='kde', plot\_kws={'alpha': 0.5})

plt.suptitle('Pair Plot of California Housing Features', y=1.02)

plt.show()

Output:





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

import 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')

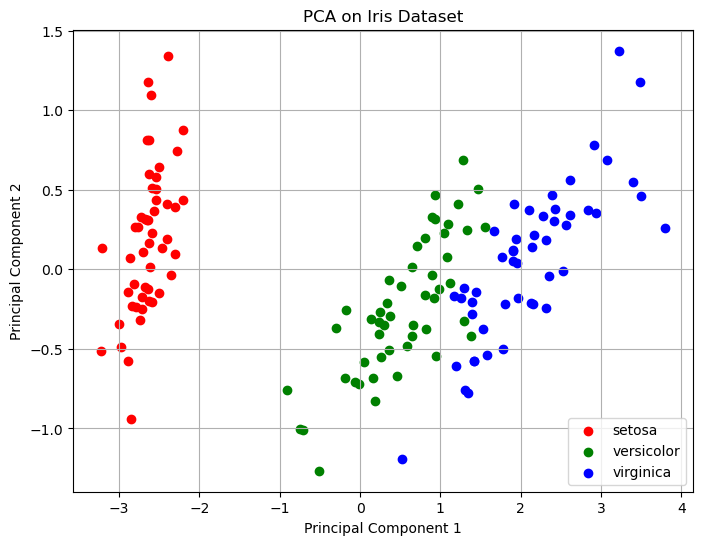
plt.ylabel('Principal Component 2')

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.

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)

Output:

Training data:

Outlook Temperature Humidity Windy PlayTennis

0 Sunny Hot High False No

1 Sunny Hot High True No

2 Overcast Hot High False Yes

3 Rain Cold High False Yes

4 Rain Cold High True No

5 Overcast Hot High True Yes

6 Sunny Hot High False No

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

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

data = np.random.rand(100)

labels = ["Class1" if x <= 0.5 else "Class2" for x in data[:50]]

def euclidean\_distance(x1, x2):

    return abs(x1 - x2)

def knn\_classifier(train\_data, train\_labels, test\_point, k):

    distances = [(euclidean\_distance(test\_point, train\_data[i]), train\_labels[i]) for i in range(len(train\_data))]

    distances.sort(key=lambda x: x[0])

    k\_nearest\_neighbors = distances[:k]

    k\_nearest\_labels = [label for \_, label in k\_nearest\_neighbors]

    return Counter(k\_nearest\_labels).most\_common(1)[0][0]

train\_data = data[:50]

train\_labels = labels

test\_data = data[50:]

k\_values = [1, 2, 3, 4, 5, 20, 30]

print("--- k-Nearest Neighbors Classification ---")

print("Training dataset: First 50 points labeled based on the rule (x <= 0.5 -> Class1, x > 0.5 -> Class2)")

print("Testing dataset: Remaining 50 points to be classified\n")

results = {}

for k in k\_values:

    print(f"Results for k = {k}:")

    classified\_labels = [knn\_classifier(train\_data, train\_labels, test\_point, k) for test\_point in test\_data]

    results[k] = classified\_labels

    for i, label in enumerate(classified\_labels, start=51):

        print(f"Point x{i} (value: {test\_data[i - 51]:.4f}) is classified as {label}")

    print("\n")

print("Classification complete.\n")

for k in k\_values:

    classified\_labels = results[k]

    class1\_points = [test\_data[i] for i in range(len(test\_data)) if classified\_labels[i] == "Class1"]

    class2\_points = [test\_data[i] for i in range(len(test\_data)) if classified\_labels[i] == "Class2"]

    plt.figure(figsize=(10, 6))

    plt.scatter(train\_data, [0] \* len(train\_data), c=["blue" if label == "Class1" else "red" for label in train\_labels],

                label="Training Data", marker="o")

    plt.scatter(class1\_points, [1] \* len(class1\_points), c="blue", label="Class1 (Test)", marker="x")

    plt.scatter(class2\_points, [1] \* len(class2\_points), c="red", label="Class2 (Test)", marker="x")

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

    plt.xlabel("Data Points")

    plt.ylabel("Classification Level")

    plt.legend()

    plt.grid(True)

    plt.show()

Output:

Classification rule: x ≤ 0.5 → Class1, x > 0.5 → Class2

k = 1:

Class1: 25 points

Class2: 25 points

k = 2:

Class1: 25 points

Class2: 25 points

k = 3:

Class1: 25 points

Class2: 25 points

k = 4:

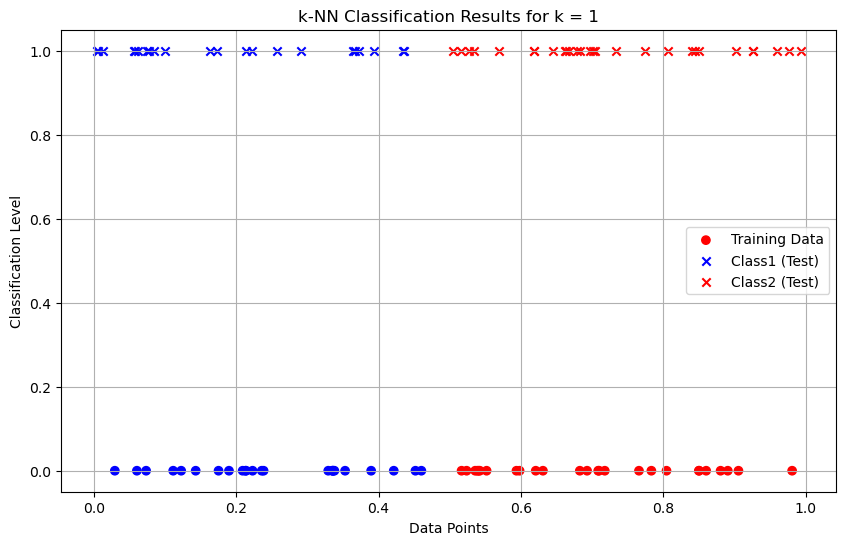
Class1: 25 points

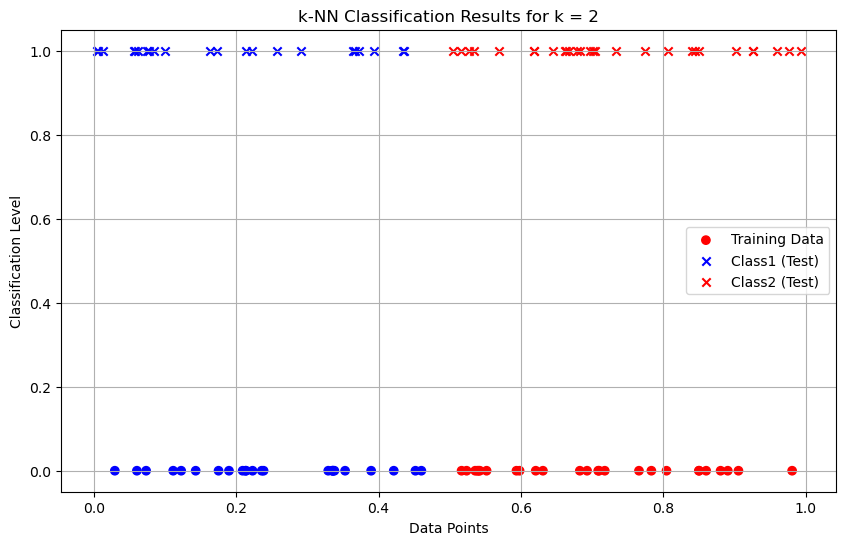
Class2: 25 points

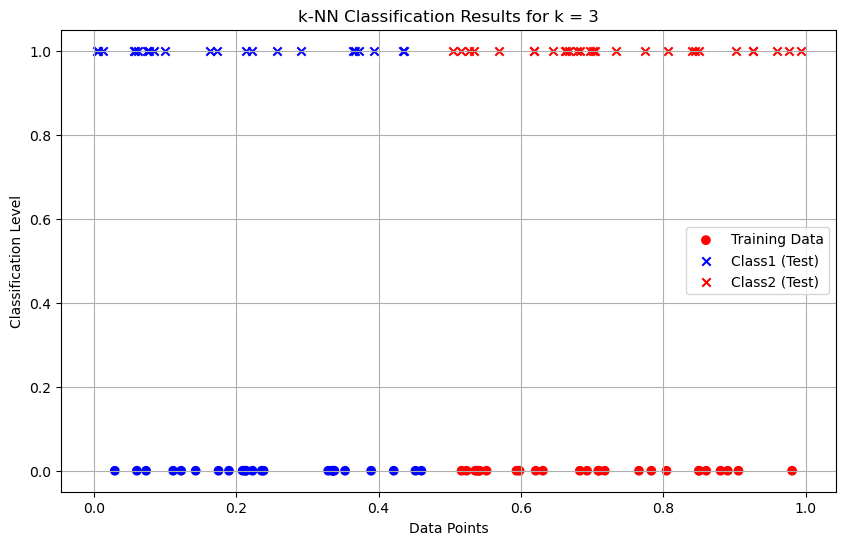
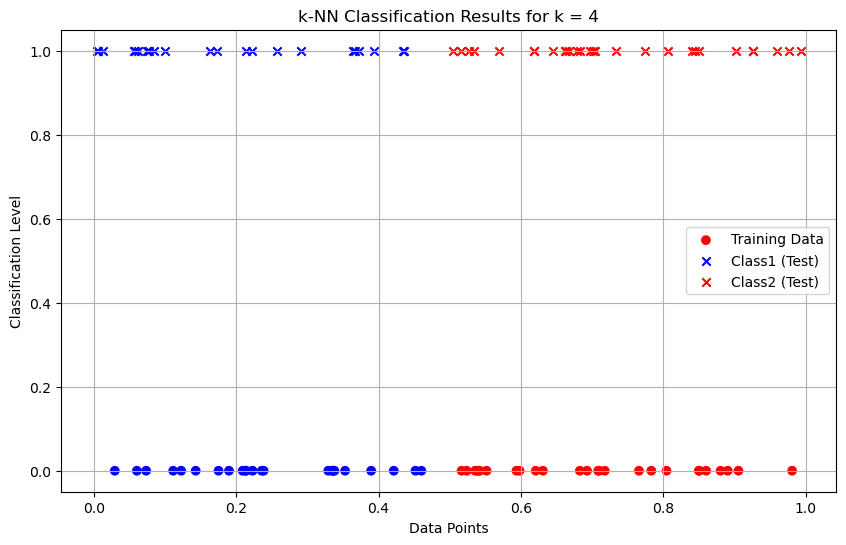
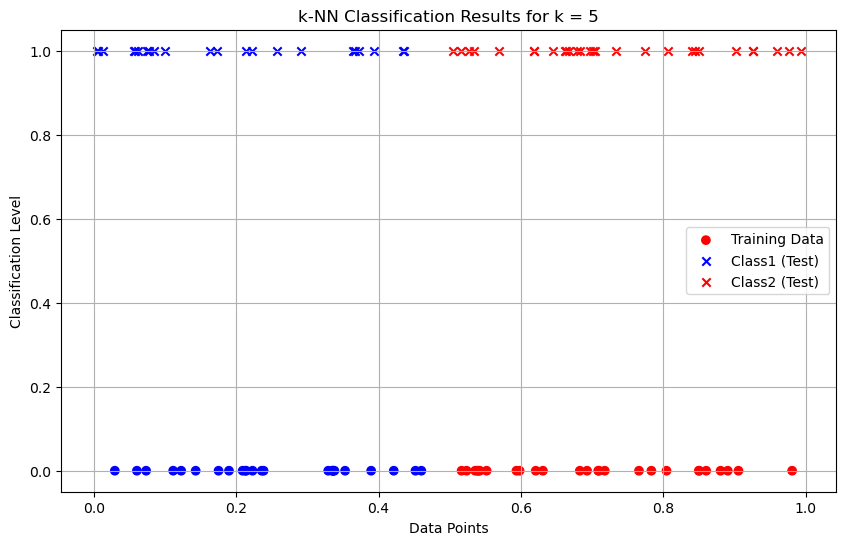
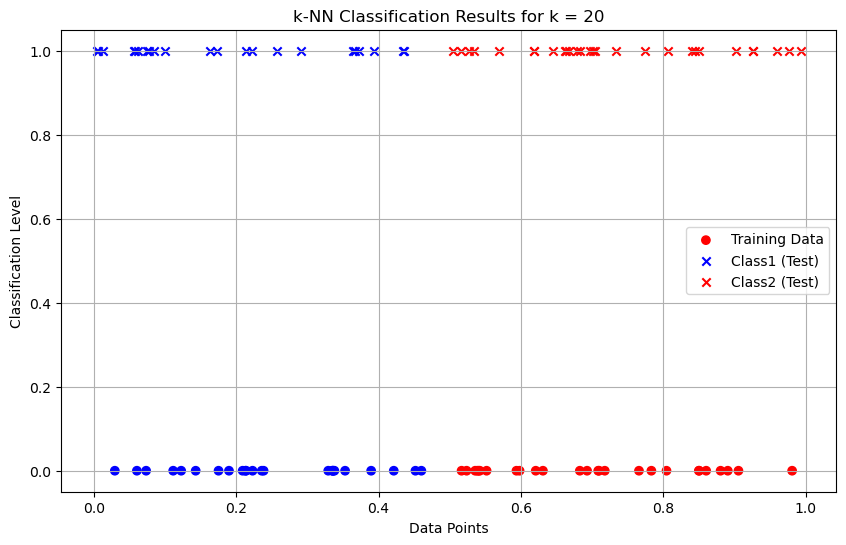
k = 5:

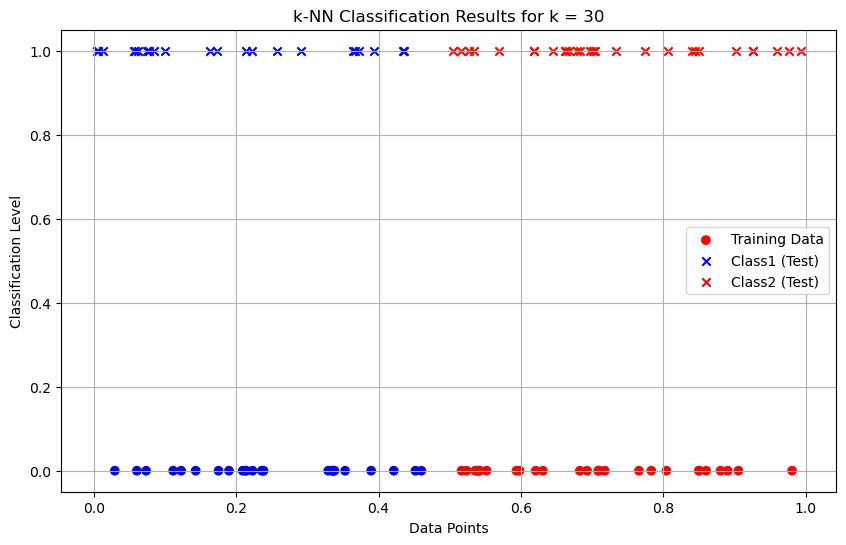
Class1: 25 points

Class2: 25 points







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 matplotlib.pyplot as plt

def gaussian\_kernel(x, xi, tau):

    return np.exp(-np.sum((x - xi) \*\* 2) / (2 \* tau \*\* 2))

def locally\_weighted\_regression(x, X, y, tau):

    m = X.shape[0]

    weights = np.array([gaussian\_kernel(x, X[i], tau) for i in range(m)])

    W = np.diag(weights)

    X\_transpose\_W = X.T @ W

    theta = np.linalg.inv(X\_transpose\_W @ X) @ X\_transpose\_W @ y

    return x @ theta

np.random.seed(42)

X = np.linspace(0, 2 \* np.pi, 100)

y = np.sin(X) + 0.1 \* np.random.randn(100)

X\_bias = np.c\_[np.ones(X.shape), X]

x\_test = np.linspace(0, 2 \* np.pi, 200)

x\_test\_bias = np.c\_[np.ones(x\_test.shape), x\_test]

tau = 0.5

y\_pred = np.array([locally\_weighted\_regression(xi, X\_bias, y, tau) for xi in x\_test\_bias])

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

plt.scatter(X, y, color='red', label='Training Data', alpha=0.7)

plt.plot(x\_test, y\_pred, color='blue', label=f'LWR Fit (tau={tau})', linewidth=2)

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

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

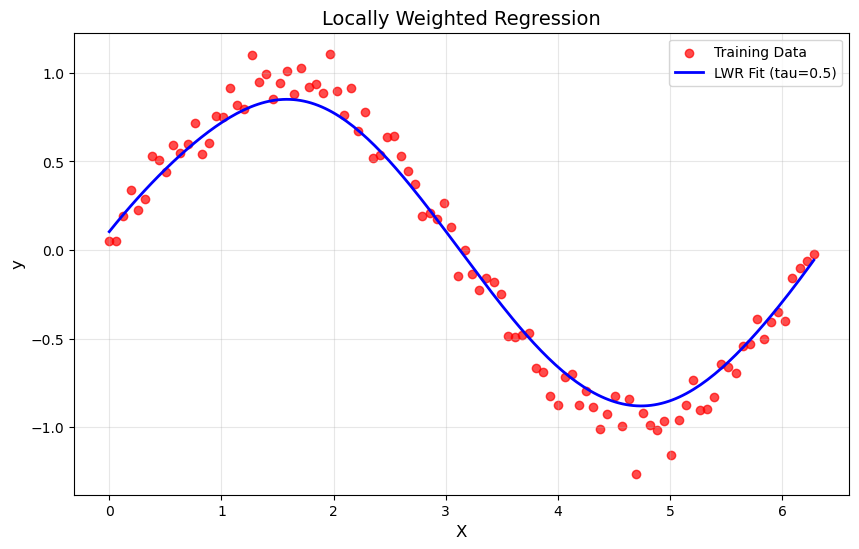
plt.title('Locally Weighted Regression', fontsize=14)

plt.legend(fontsize=10)

plt.grid(alpha=0.3)

plt.show()

Output:



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\_\_":

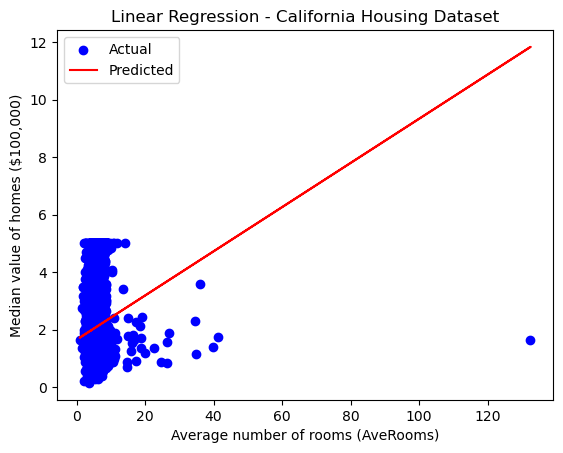
    print("Demonstrating Linear Regression and Polynomial Regression\n")

    linear\_regression\_california()

    polynomial\_regression\_auto\_mpg()

Output:

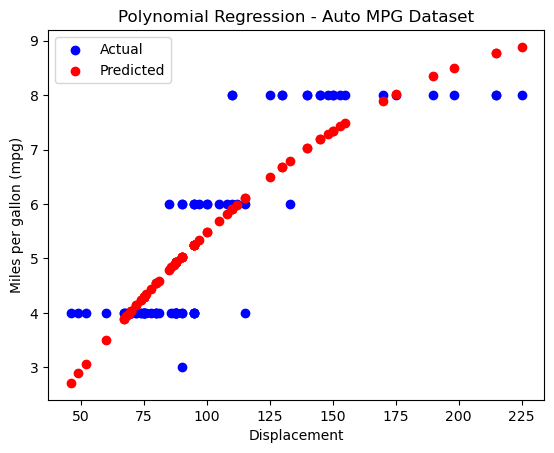
Demonstrating Linear Regression and Polynomial Regression



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

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.

# 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}")

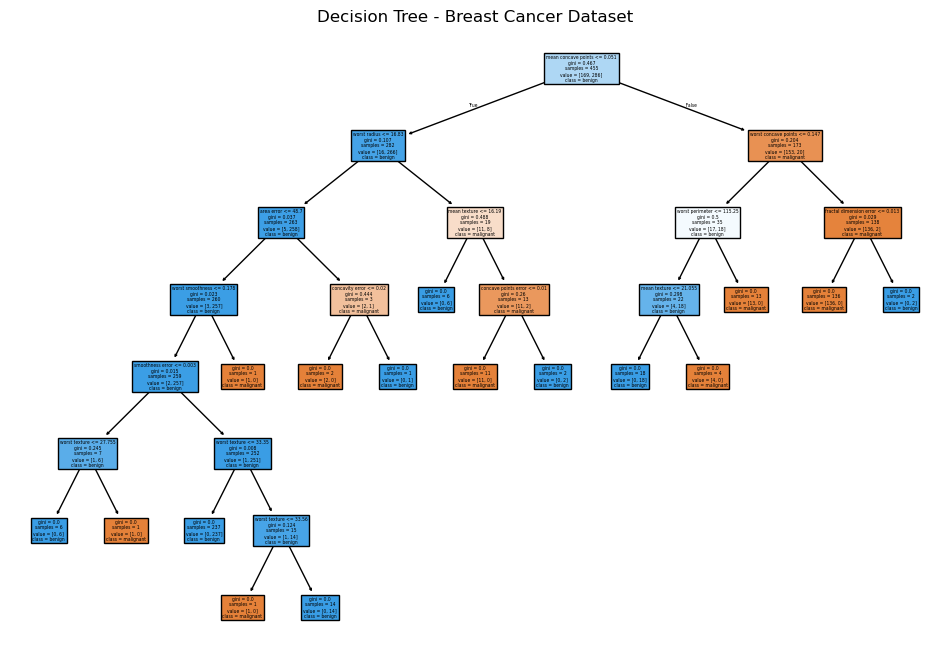
plt.figure(figsize=(12,8))

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()

Output:



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

import 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()

Output:

Accuracy: 80.83%

Classification Report:

precision recall f1-score support

0 0.67 1.00 0.80 2

1 1.00 1.00 1.00 2

2 0.33 0.67 0.44 3

3 1.00 0.00 0.00 5

4 1.00 0.50 0.67 4

5 1.00 1.00 1.00 2

7 1.00 0.75 0.86 4

8 1.00 0.67 0.80 3

9 1.00 0.75 0.86 4

10 1.00 1.00 1.00 3

11 1.00 1.00 1.00 1

12 0.40 1.00 0.57 4

13 1.00 0.80 0.89 5

14 1.00 0.40 0.57 5

15 0.67 1.00 0.80 2

16 1.00 0.67 0.80 3

17 1.00 1.00 1.00 3

18 1.00 1.00 1.00 3

19 0.67 1.00 0.80 2

20 1.00 1.00 1.00 3

21 1.00 0.67 0.80 3

22 1.00 0.60 0.75 5

23 1.00 0.75 0.86 4

24 1.00 1.00 1.00 3

25 1.00 0.75 0.86 4

26 1.00 1.00 1.00 2

27 1.00 1.00 1.00 5

28 0.50 1.00 0.67 2

29 1.00 1.00 1.00 2

30 1.00 1.00 1.00 2

31 1.00 0.75 0.86 4

32 1.00 1.00 1.00 2

34 0.25 1.00 0.40 1

35 1.00 1.00 1.00 5

36 1.00 1.00 1.00 3

37 1.00 1.00 1.00 1

38 1.00 0.75 0.86 4

39 0.50 1.00 0.67 5

accuracy 0.81 120

macro avg 0.89 0.85 0.83 120

weighted avg 0.91 0.81 0.81 120

Confusion Matrix:

[[2 0 0 ... 0 0 0]

[0 2 0 ... 0 0 0]

[0 0 2 ... 0 0 1]

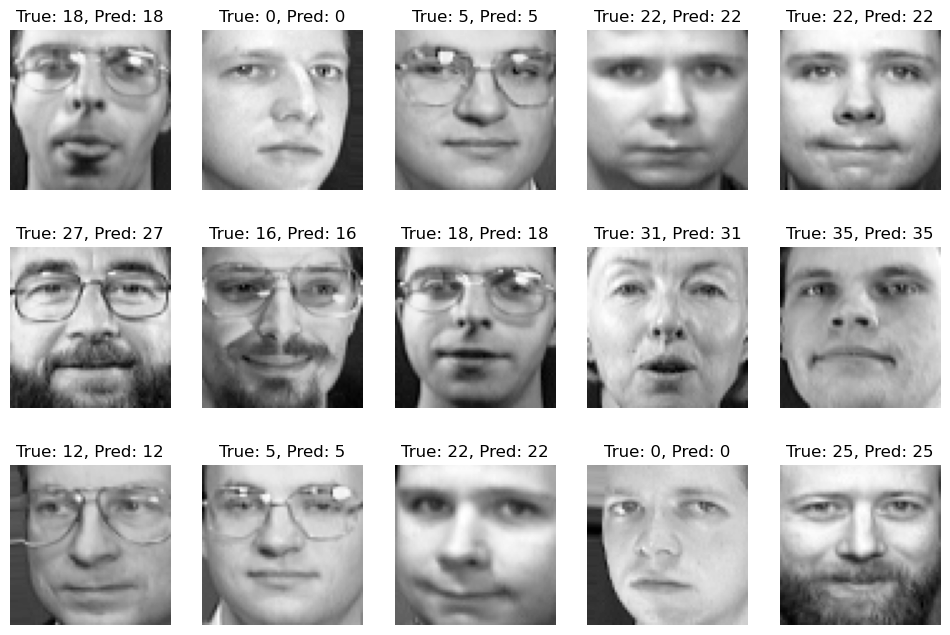
...

[0 0 0 ... 1 0 0]

[0 0 0 ... 0 3 0]

[0 0 0 ... 0 0 5]]

Cross-validation accuracy: 87.25%



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

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()

Output:

Confusion Matrix:

[[175 37]

[ 13 344]]

Classification Report:

precision recall f1-score support

0 0.93 0.83 0.88 212

1 0.90 0.96 0.93 357

accuracy 0.91 569

macro avg 0.92 0.89 0.90 569

weighted avg 0.91 0.91 0.91 569

