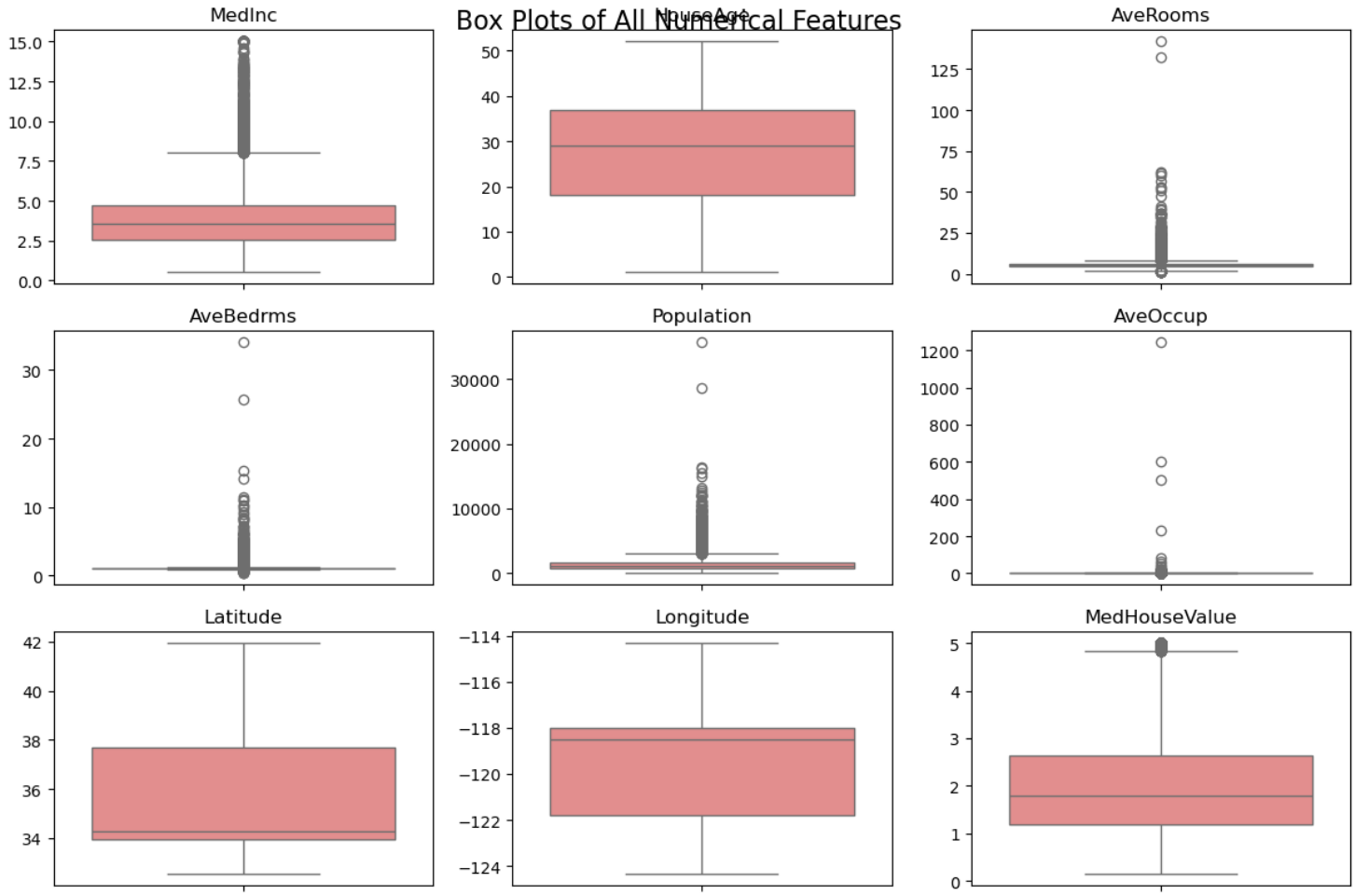
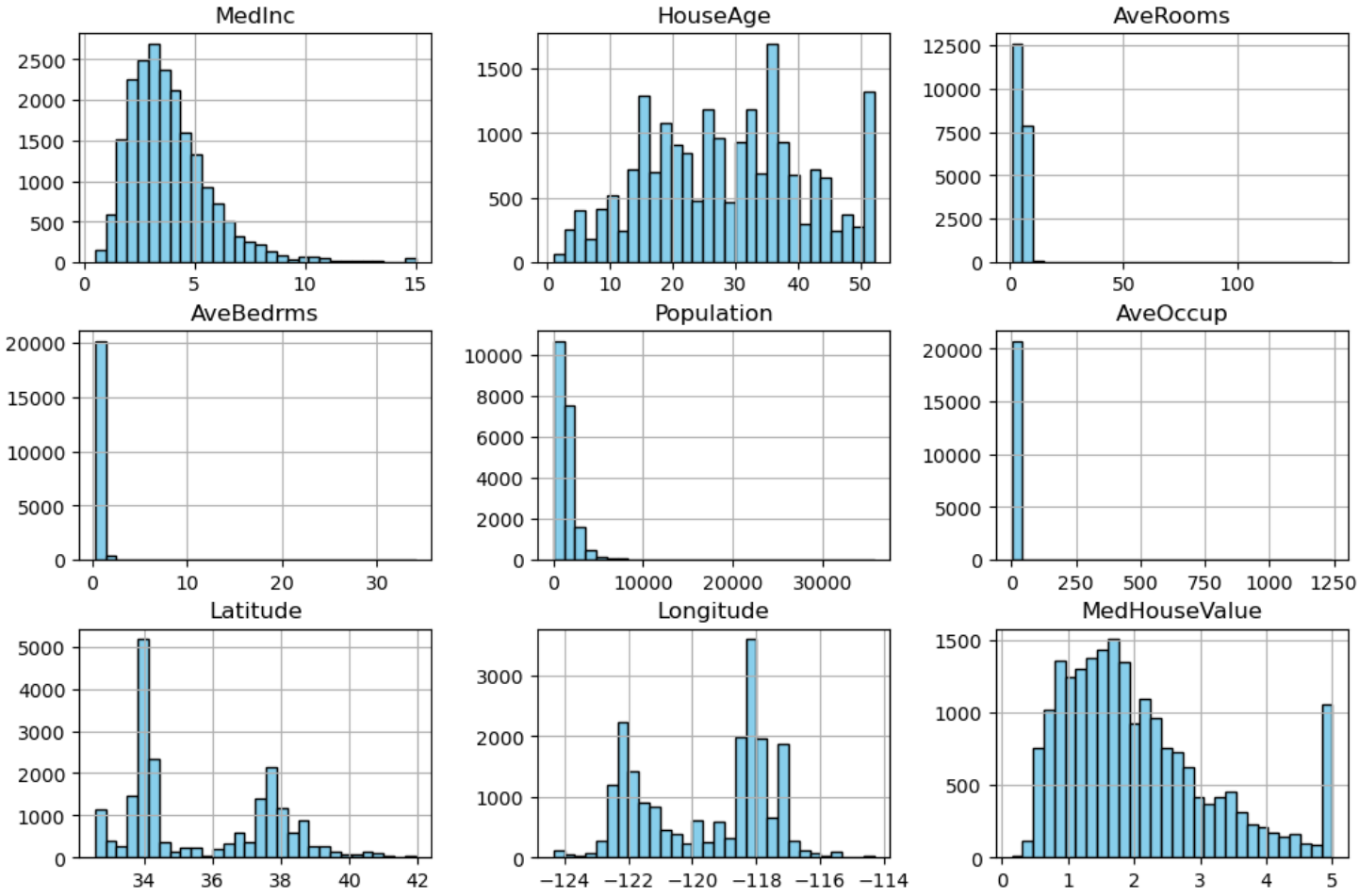
Experiment-01

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 necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset from local storage

data = pd.read_csv("california_housing.csv")

# Display the shape of the dataset
print("Dataset shape:", data.shape)

# Display column names
print("Column Names:", data.columns.tolist())

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

# Print the first 10 rows of the dataset
print("\nFirst 10 Rows of the Dataset:\n")
print(data.head(10))

# Generate Histograms for all numerical features
plt.figure(figsize=(12, 8))
data.hist(bins=30, figsize=(12, 8), edgecolor='black', color='skyblue')
plt.suptitle("Histograms of All Numerical Features", fontsize=16)
plt.show()

# Generate Box Plots to identify outliers
plt.figure(figsize=(12, 8))

for i, col in enumerate(data.columns):
    plt.subplot(3, 3, i+1)
    sns.boxplot(y=data[col], color='lightcoral')
    plt.title(col)
    plt.ylabel('')
    plt.tight_layout()

plt.suptitle("Box Plots of All Numerical Features", fontsize=16)
plt.show()
```

OUTPUT


Box Plots of All Numerical Features
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseValue

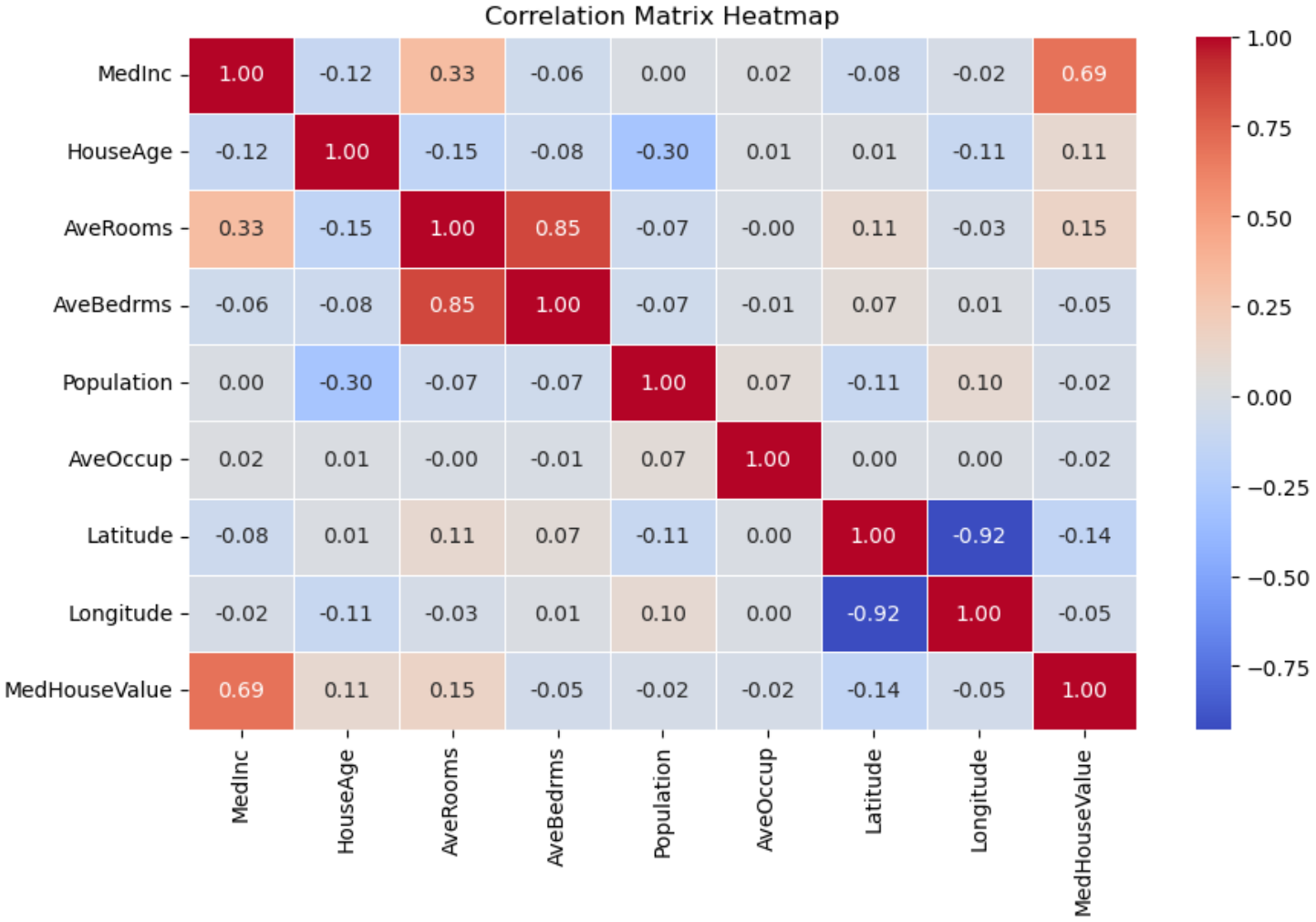

Histograms of All Numerical Features
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseValue

Experiment-02

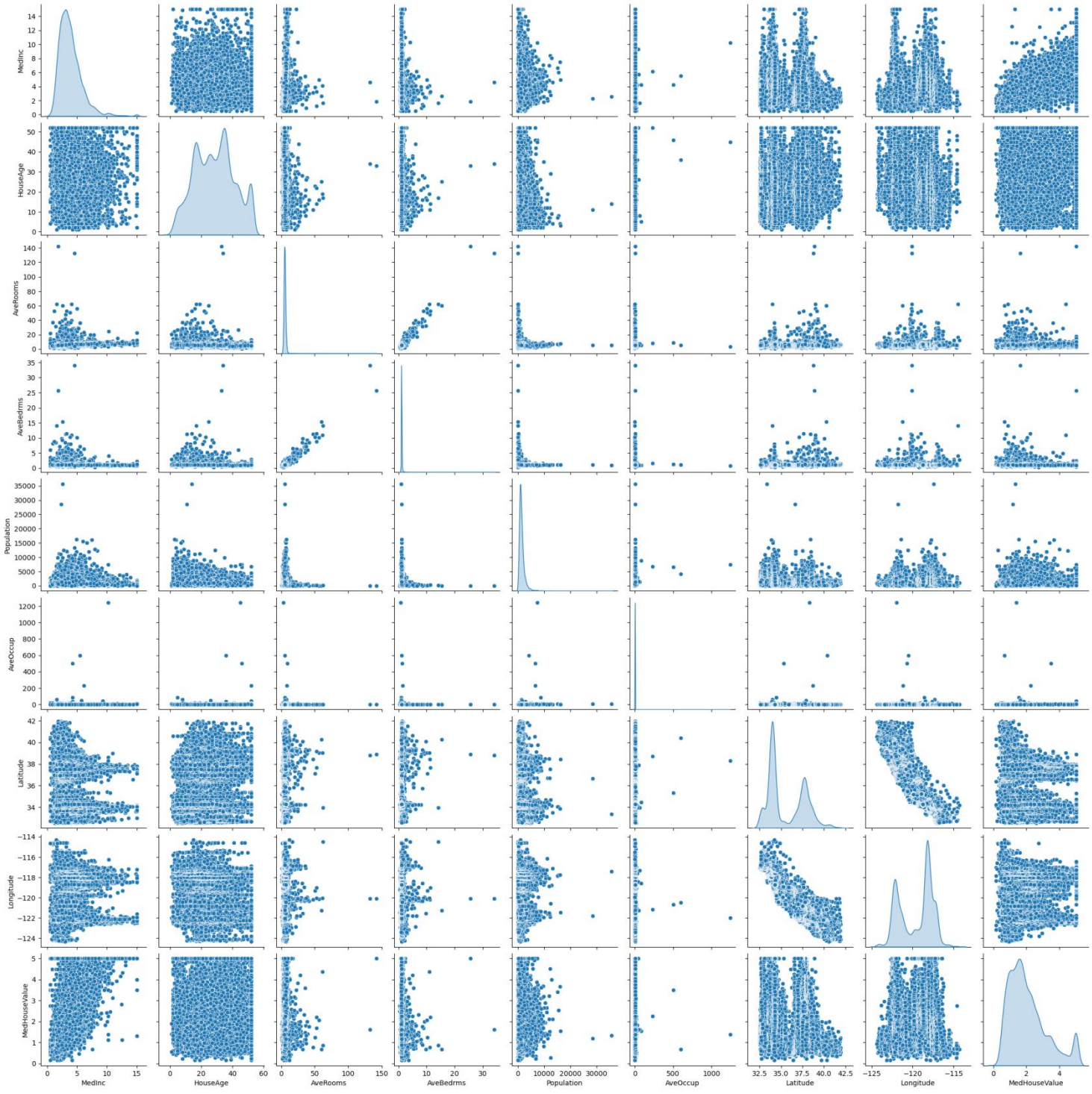
Develop a program to Compute the correlation matrix to understand the relationships between pairs of features. Visualize the correlation matrix using a heatmap to know which variables have strong positive/negative correlations. Create a pair plot to visualize pairwise relationships between features.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

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

# Load the dataset

df = pd.read_csv("california_housing.csv")

# Display basic information about the dataset
print(df.info())
print("\nFirst 5 Rows:")
print(df.head())

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

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()

# Create a pair plot to visualize relationships between features
sns.pairplot(df, diag_kind='kde')
plt.show()
```

output


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

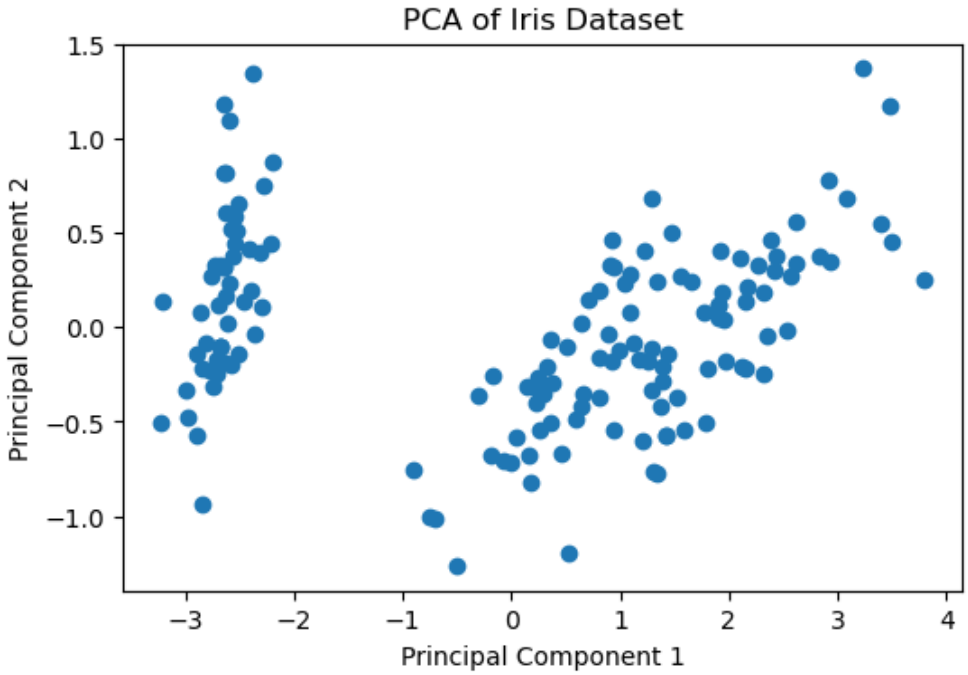
PAIRPLOTS


MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseValue

Experiment-03

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

```
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Read the Iris dataset
data = pd.read_csv("Iris.csv")

# Select the 4 feature columns (adjust the column names if necessary)
X = data[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]]

# Apply PCA to reduce to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Print the shape and the first few rows of the reduced data
print("Reduced data shape:", X_pca.shape)
print(X_pca[:5])

# Optional: Plot the 2D PCA result
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA of Iris Dataset")
plt.show()
```



Iris Dataset (Sample Records)

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	7	3.2	4.7	1.4	Iris-versicolor
11	6.4	3.2	4.5	1.5	Iris-versicolor
12	6.9	3.1	4.9	1.5	Iris-versicolor
13	5.5	2.3	4	1.3	Iris-versicolor
14	6.5	2.8	4.6	1.5	Iris-versicolor
15	5.7	2.8	4.5	1.3	Iris-versicolor
16	6.3	3.3	4.7	1.6	Iris-versicolor
17	4.9	2.4	3.3	1	Iris-versicolor
18	6.6	2.9	4.6	1.3	Iris-versicolor
19	5.2	2.7	3.9	1.4	Iris-versicolor
20	6.3	3.3	6	2.5	Iris-virginica
21	5.8	2.7	5.1	1.9	Iris-virginica
22	7.1	3	5.9	2.1	Iris-virginica
23	6.3	2.9	5.6	1.8	Iris-virginica
24	6.5	3	5.8	2.2	Iris-virginica
25	7.6	3	6.6	2.1	Iris-virginica
26	4.9	2.5	4.5	1.7	Iris-virginica
27	7.3	2.9	6.3	1.8	Iris-virginica
28	6.7	2.5	5.8	1.8	Iris-virginica
29	7.2	3.6	6.1	2.5	Iris-virginica

Experiment-04

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 csv
 # Load CSV directly into data
with open('enjoysport.csv', 'r') as file:
data = list(csv.reader(file))
# Initialize hypothesis with the first positive example
hypothesis = []
for example in data[1:]: # Skip header row
if example[-1].lower() == 'yes':
 hypothesis = example[:-1]
break
# Generalize hypothesis based on other positive examples
for example in data[1:]:
if example[-1].lower() == 'yes':
for i in range(len(hypothesis)):
if hypothesis[i] != example[i]:
hypothesis[i] = '?'
print("Final Hypothesis:", hypothesis)
```

Output:

Final Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Experiment-05

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

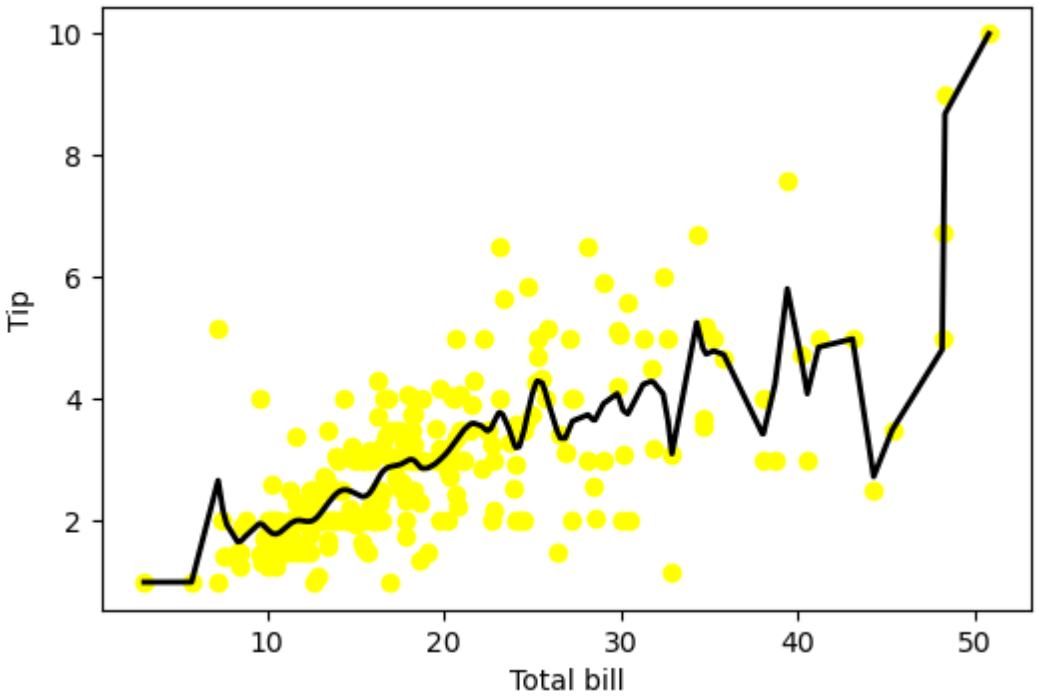
```
import numpy as np
from collections import Counter

# Generate 100 random values in [0, 1]
np.random.seed(42)
data = np.random.rand(100)

# Split into training (first 50) and test (last 50)
train_x = data[:50]
test_x = data[50:]

# Label training data
train_y = ['Class1' if x <= 0.5 else 'Class2' for x in train_x]

def  knn_predict(train_x, train_y, test_val, k):
distances = np.abs(train_x - test_val)
sorted_idx = np.argsort(distances)
nearest_labels = np.array(train_y)[sorted_idx][:k]
return Counter(nearest_labels).most_common(1)[0][0]

k_values = [1, 2, 3, 4, 5, 20, 30]

for k in k_values:
print(f"--- For k = {k} ---")
predictions = []
for i, x in enumerate(test_x, start=51):
pred = knn_predict(train_x, train_y, x, k)
predictions.append(pred)
print(f"Point x{i}: {x:.4f} -> Predicted: {pred}")
counts = Counter(predictions)
print(f"Summary for k = {k}: Class1 = {counts['Class1']}, Class2 = {counts['Class2']}\n")
```

Experiment-06

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 matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def  kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def  localWeight(point, xmat, ymat, k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def  localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

# load data points
data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing the data
mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='yellow')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'black', linewidth=2)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

OUTPUT



Tips Data Set(Sample Records):

total_bill	tip	gender	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3
16.97	3.5	Female	No	Sun	Dinner	3

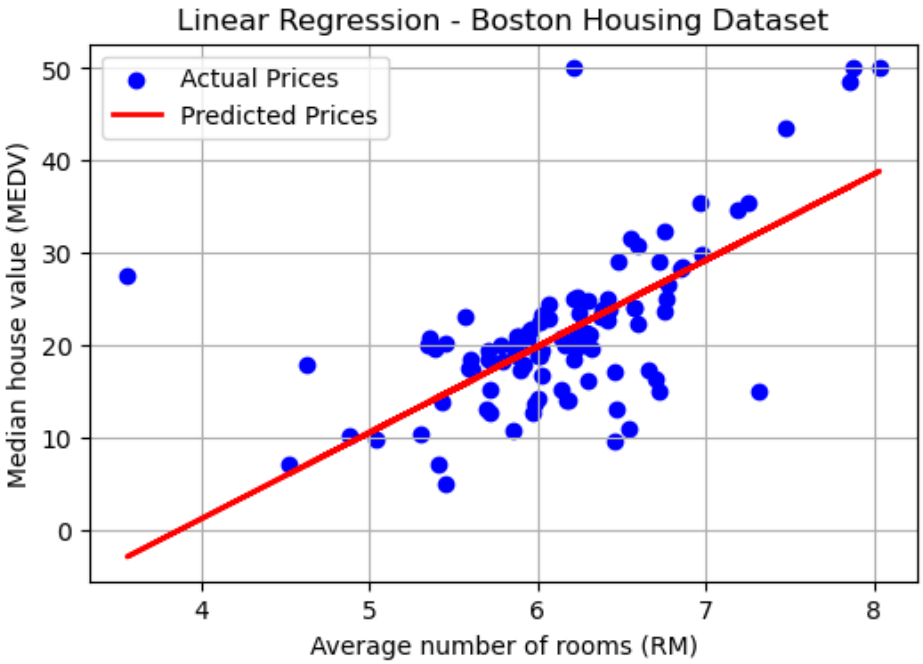
Experiment-07

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 pandas as pd
import numpy as np
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 data from local CSV
boston_df = pd.read_csv("BostonHousing.csv")
# Feature and target
X = boston_df[['RM']]        # Average number of rooms,
# uses double square brackets to return the result as a DataFrame, not a Series.
y = boston_df['MEDV']        # Median house price
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
# Plot Linear Regression output
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("Median house value (MEDV)")
plt.title("Linear Regression - Boston Housing Dataset")
plt.legend()
```

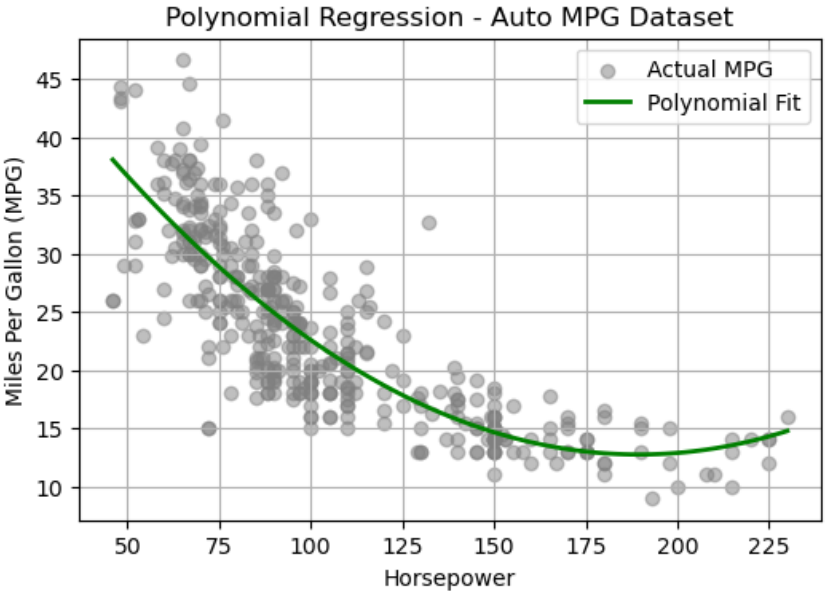
```
plt.grid(True)

plt.show()

# Evaluate

print("\nLinear Regression Evaluation:")

print("Mean Squared Error:", mean_squared_error(y_test, y_pred))

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

# ---------- POLYNOMIAL REGRESSION: Auto MPG Dataset ----------

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make_pipeline

# Load Auto MPG dataset from local CSV
auto_df = pd.read_csv("auto_mpg.csv")
auto_df = auto_df.dropna()  # Drop missing values

# Feature and target
X = auto_df[['horsepower']]
y = auto_df['mpg']

# Polynomial regression pipeline
poly_model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
poly_model.fit(X, y)
# Create input range with correct feature name to avoid warning

x_range = pd.DataFrame(np.linspace(X.min().values[0], X.max().values[0], 100), columns=["horsepower"])

# Predict

y_poly_pred = poly_model.predict(x_range)

# Plot Polynomial Regression output

plt.scatter(X, y, color='gray', alpha=0.5, label='Actual MPG')

plt.plot(x_range, y_poly_pred, color='green', linewidth=2, label='Polynomial Fit')

plt.xlabel("Horsepower")

plt.ylabel("Miles Per Gallon (MPG)")

plt.title("Polynomial Regression - Auto MPG Dataset")

plt.legend()

plt.grid(True)

plt.show()
```

OUTPUT



The model is suitable for **predictive purposes** when the trend is mostly linear and features like 'RM' are strong predictors.

2. Polynomial Regression – Auto MPG Dataset



Observation:

- The graph shows the relationship between **horsepower** and **miles per gallon (MPG)**.
- The **gray dots** are actual data points; the **green curve** is the polynomial regression prediction (degree 2).

Boston Housing Dataset (Sample Records)

The data was collected as part of a **housing study** conducted by the U.S. Census Bureau. Each row corresponds to a **small geographic area** (not an individual house) with **aggregate or average values** for that tract.

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	NA	36.2
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
0.08829	12.5	7.87	NA	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1
0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5
0.17004	12.5	7.87	NA	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15
0.11747	12.5	7.87	0	0.524	6.009	82.9	6.2267	5	311	15.2	396.9	13.27	18.9
0.09378	12.5	7.87	0	0.524	5.889	39	5.4509	5	311	15.2	390.5	15.71	21.7
0.62976	0	8.14	0	0.538	5.949	61.8	4.7075	4	307	21	396.9	8.26	20.4
0.63796	0	8.14	NA	0.538	6.096	84.5	4.4619	4	307	21	380.02	10.26	18.2
0.62739	0	8.14	0	0.538	5.834	56.5	4.4986	4	307	21	395.62	8.47	19.9
1.05393	0	8.14	0	0.538	5.935	29.3	4.4986	4	307	21	386.85	6.58	23.1
0.7842	0	8.14	0	0.538	5.99	81.7	4.2579	4	307	21	386.75	14.67	17.5
0.80271	0	8.14	0	0.538	5.456	36.6	3.7965	4	307	21	288.99	11.69	20.2

Experiment-08

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.

```
# Import necessary libraries
from sklearn.datasets import load_breast_cancer  # To load dataset
from sklearn.tree import DecisionTreeClassifier  # Decision Tree classifier
from sklearn.model_selection import train_test_split  # To split data
from sklearn.metrics import accuracy_score  # To check accuracy
# Step 1: Load the Breast Cancer dataset
data = load_breast_cancer()

# Step 2: Display the features and target names
print("Feature Names:\n", data.feature_names)
print("\nTarget Names:\n", data.target_names)

# Step 3: Separate features (X) and labels (y)
X = data.data  # Features
y = data.target  # Labels (0 = malignant, 1 = benign)

# Step 4: Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Create and train the Decision Tree classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Step 6: Test the model on the test data
y_pred = model.predict(X_test)
# Step 7: Print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy on test data:", round(accuracy * 100, 2), "%")

# Step 8: Classify a new (unseen) sample from the test set
new_sample = X_test[0].reshape(1, -1)  # Truly unseen sample
predicted_class = model.predict(new_sample)

# Step 9: Display the result
print("\nNew Sample Prediction (from test set):")
print("Predicted Class:", predicted_class[0])
print("Meaning:", "Malignant" if predicted_class[0] == 0 else "Benign")
```

Output and Analysis:

Model Accuracy on Test Data: 94.74%

New Sample Prediction (from X_test[0]):

Predicted Class: 1

Meaning: Benign

The decision tree classifier achieved a high accuracy, demonstrating its effectiveness in distinguishing between malignant and benign tumors. The model successfully classified a truly unseen sample from the test set, ensuring it wasn't part of the training data. This validates the model's ability to generalize well to new data.

Experiment-09

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 necessary libraries
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

# Load the Olivetti Faces dataset
faces = fetch_olivetti_faces(shuffle=True, random_state=42)
X = faces.data
y = faces.target

# Display 10 sample face images with their labels
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    ax.imshow(X[i].reshape(64, 64), cmap='gray')
    ax.set_title(f"Person {y[i]}")
    ax.axis('off')
plt.suptitle("Sample Images from Olivetti Faces Dataset")
plt.tight_layout()
plt.show()

# Split the data into training and testing sets (80%-20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Train the Naive Bayes classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Make predictions and calculate accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("\nNaive Bayes Classifier Accuracy: {:.2f}%".format(accuracy * 100))
# Visualize predictions
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    ax.imshow(X_test[i].reshape(64, 64), cmap='gray')
    ax.set_title(f"Pred: {y_pred[i]}\nActual: {y_test[i]}")
    ax.axis('off')
plt.suptitle("Predictions on Test Images")
plt.tight_layout()
plt.show()
```

Predictions on Test Images

Sample Images from Olivetti Faces Dataset

Experiment-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 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.preprocessing import StandardScaler
# Step 1: Load the Wisconsin Breast Cancer dataset
data = load_breast_cancer()
X = data.data            # Feature matrix (569 samples × 30 features)

# Step 2: Standardize the features (mean=0, std=1)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# Step 3: Apply K-Means clustering with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)
# clusters is an array of 0s and 1s assigning each sample to one of two clusters

# Step 4: Reduce data to 2 principal components for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# X_pca now has shape (569, 2), ready for 2D plotting

# Step 5: Transform cluster centroids into the same 2D PCA space
centroids_original = kmeans.cluster_centers_
centroids_pca = pca.transform(centroids_original)

# Step 6: Plot the clustered data in 2D
plt.figure(figsize=(8, 6))

# Define user-friendly names for the two clusters
cluster_names = {0: "Cluster 1", 1: "Cluster 2"}
colors = ["yellow", "blue"]
for cluster_id, color in zip(cluster_names, colors):
    mask = (clusters == cluster_id)
    plt.scatter(
        X_pca[mask, 0],        # x = PC1
        X_pca[mask, 1],        # y = PC2
        c=color,
        alpha=0.6,
        edgecolor='k',
        label=cluster_names[cluster_id]
```

```
    )

# Step 7: Plot the centroids as black 'X' markers
plt.scatter(
centroids_pca[:, 0],
    centroids_pca[:, 1],
    marker='X',
    s=200,
    c='black',
    label="Centroids"
)

# Step 8: Add titles and labels
plt.title("K-Means Clustering on Breast Cancer Data")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(loc="upper right")

# Step 9: Display the plot
plt.tight_layout()
plt.show()
```

OUTPUT