Machine Learning

LAB MANUAL

VI Semester AI&DS

Designed By,

1. **Dr. Aijaz Qazi**
2. **Prof. Vaibhav Chavan**

Institute Vision

To become premier institute committed to **academic excellence** and **global competence** for the **holistic development** of students.

Key words: academic excellence, global competence, holistic development.

Institute Mission

M1: Develop competent human resources, adopt outcome based education (OBE) and implement cognitive assessment of students.

M2: Inculcate the traits of global competencies amongst the students.

M3: Nurture and train our students to have domain knowledge, develop the qualities of global professionals and to have social consciousness for holistic development.

Department Vision

To deliver a **quality and responsive** education in the field of artificial intelligence and data science emphasizing **professional skills** to face **global challenges** in the evolving IT paradigm.

Key words: quality and responsive, professional skills, global challenges.

Department Mission

M1: Leverage multiple pedagogical approaches to impart knowledge on the current and emerging AI technologies.

M2: Develop an inclusive and holistic ambiance that bolsters problem solving, cognitive abilities and critical thinking.

M3: Enable students to develop trust worthiness, team spirit, understanding law-of-the-land, social behavior to be a global stake holder.

Program Specific Outcomes (PSOs):

PSO1: To apply core knowledge of Artificial Intelligence, Machine Learning, Deep Learning, Data Science, Big Data Analytics and Statistical Learning to develop effective solutions for real-world problems.

PSO2: To demonstrate proficiency in specialized and emerging technologies such as Natural Language Processing, Cloud Computing, Robotic Process Automation, Storage Area Networks and the Internet of Things to meet the stringent and diverse professional challenges.

PSO3: To imbibe managerial skills, social responsibility, ethical and moral values through courses in Management and Entrepreneurship, Software Engineering Principles, Universal Human Values and Ability Enhancement Programs to meet the industry and societal expectations.

Program Educational Objectives (PEOs)

PEO 1: Build a strong foundation in mathematics, core programming, artificial intelligence, machine learning, and data science to enable graduates to analyze, design, and implement intelligent systems for solving complex real-world problems.

PEO 2: Foster creativity, cognitive and research skills to analyze the requirements and technical specifications of software to articulate novel engineering solutions for an efficient product design.

PEO 3: Prepare graduates for dynamic career opportunities in AI and Data Science by equipping them with interdisciplinary knowledge, adaptability, and practical exposure to tools and techniques required for industry and research.

PEO 4: Instill a strong sense of ethics, professional responsibility, and human values, empowering graduates to contribute positively to society and lead with integrity in their professional domains.

PEO 5: Encourage graduates to pursue higher education, certification program, entrepreneurial ventures, etc. by nurturing a mindset of continuous learning and awareness of global trends and challenges.

INDEX

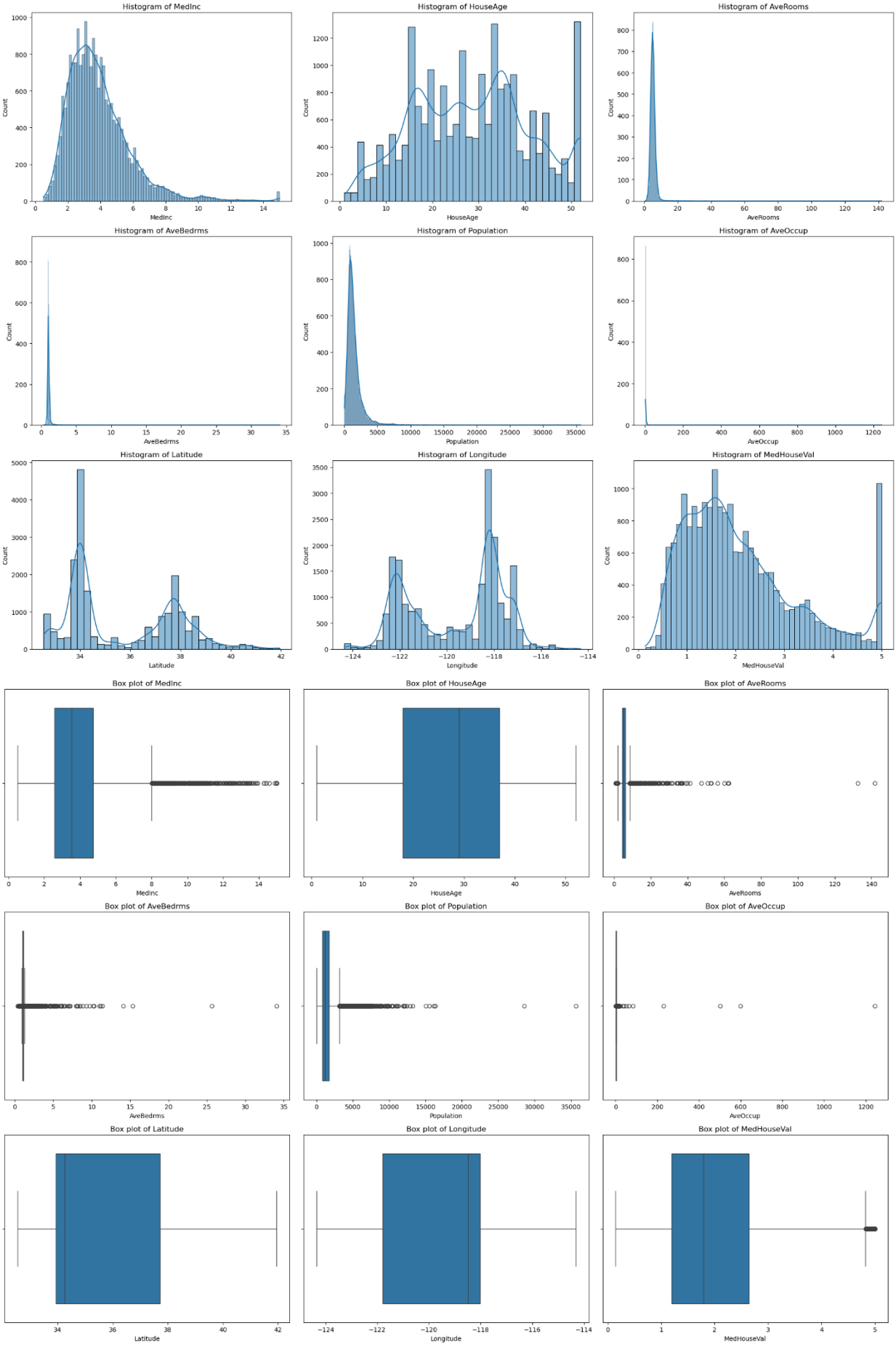
Experiment 1. Develop a program to create histograms for all numerical features and analyze the distribution of each feature. Generate box plots for all numerical features and identify any outliers. Use California Housing dataset.

Program:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing(as_frame=True)
df = data.frame
print(df.head())
numerical_features = df.select_dtypes(include=['float64', 'int64']).columns.tolist()
print(f'Numerical features: {numerical_features}')
plt.figure(figsize=(20, 15))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.histplot(df[feature], kde=True)
    plt.title(f'Histogram of {feature}')
plt.tight_layout()
plt.show()
plt.figure(figsize=(20, 15))
for i, feature in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(x=df[feature])
    plt.title(f'Box plot of {feature}')
plt.tight_layout()
plt.show()
```

Output:

```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  MedHouseVal
0    -122.23        4.526
1    -122.22        3.585
2    -122.24        3.521
3    -122.25        3.413
4    -122.25        3.422
Numerical features: ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude', 'MedHouseVal']
```

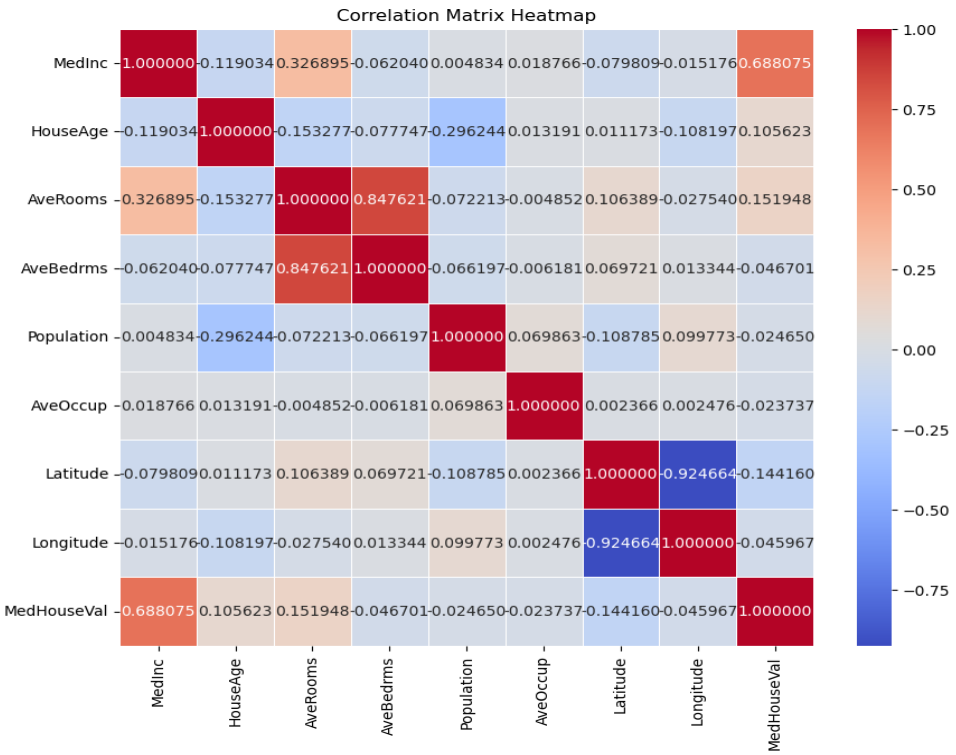

Histogram of MedInc
Count
MedInc
Histogram of HouseAge
Count
HouseAge
Histogram of AveRooms
Count
AveRooms
Histogram of AveBedrms
Count
AveBedrms
Histogram of Population
Count
Population
Histogram of AveOccup
Count
AveOccup
Histogram of Latitude
Count
Latitude
Histogram of Longitude
Count
Longitude
Histogram of MedHouseVal
Count
MedHouseVal
Box plot of MedInc
MedInc
Box plot of HouseAge
HouseAge
Box plot of AveRooms
AveRooms
Box plot of AveBedrms
AveBedrms
Box plot of Population
Population
Box plot of AveOccup
AveOccup
Box plot of Latitude
Latitude
Box plot of Longitude
Longitude
Box plot of MedHouseVal
MedHouseVal

Experiment 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.

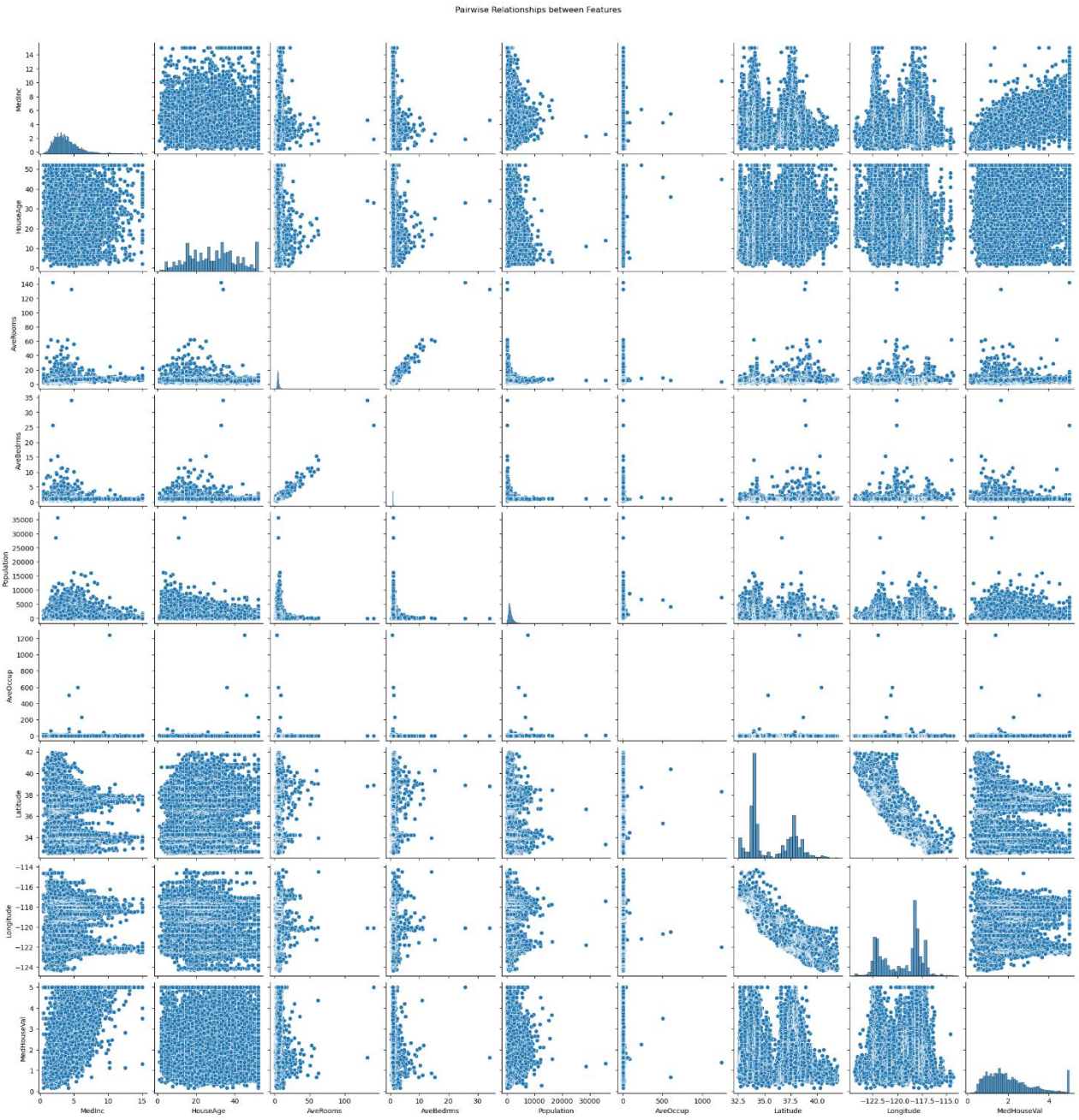
Program:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing(as_frame=True)
df = data.frame
print("First few rows of the dataset:")
print(df.head())
corr_matrix = df.corr()
print("\nCorrelation matrix:")
print(corr_matrix)
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='2f', linewidths=0.5)
plt.title("Correlation Matrix Heatmap")
plt.show()
sns.pairplot(df)
plt.suptitle('Pairwise Relationships between Features', y=1.02)
plt.show()
```

Output:



```
First few rows of the dataset:
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85

   Longitude  MedHouseVal
0    -122.23        4.526
1    -122.22        3.585
2    -122.24        3.521
3    -122.25        3.413
4    -122.25        3.422

Correlation matrix:
              MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  \
MedInc       1.000000 -0.119034  0.326895  -0.062040    0.004834  0.018766
HouseAge    -0.119034  1.000000 -0.153277  -0.077747   -0.296244  0.013191
AveRooms     0.326895 -0.153277  1.000000   0.847621   -0.072213 -0.004852
AveBedrms   -0.062040 -0.077747  0.847621   1.000000   -0.066197 -0.006181
Population   0.004834 -0.296244 -0.072213  -0.066197    1.000000  0.069863
AveOccup     0.018766  0.013191 -0.004852  -0.006181    0.069863  1.000000
Latitude    -0.079809  0.011173  0.106389   0.069721   -0.108785  0.002366
Longitude   -0.015176 -0.108197 -0.027540   0.013344    0.099773  0.002476
MedHouseVal  0.688075  0.105623  0.151948  -0.046701   -0.024650 -0.023737

              Latitude  Longitude  MedHouseVal
MedInc       -0.079809  -0.015176     0.688075
HouseAge      0.011173  -0.108197     0.105623
AveRooms      0.106389  -0.027540     0.151948
AveBedrms     0.069721   0.013344    -0.046701
Population   -0.108785   0.099773    -0.024650
AveOccup      0.002366   0.002476    -0.023737
Latitude      1.000000  -0.924664    -0.144160
Longitude    -0.924664   1.000000    -0.045967
MedHouseVal  -0.144160  -0.045967     1.000000
```

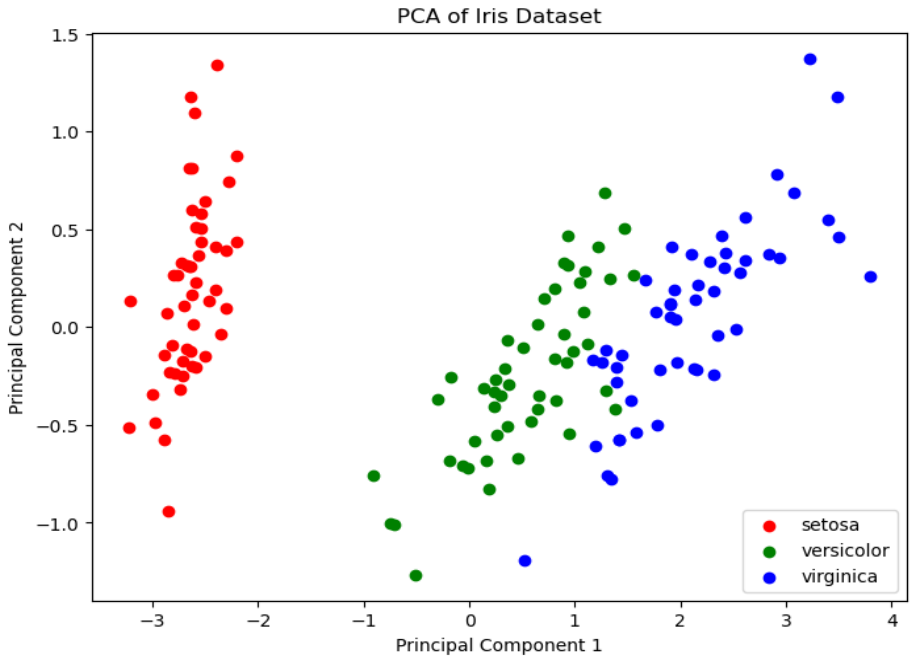

Pairwise Relationships between Features
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal

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

Program:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
data = load_iris()
iris_df = pd.DataFrame(data.data, columns=data.feature_names)
iris_df['species'] = pd.Categorical.from_codes(data.target, data.target_names)
pca = PCA(n_components=2)
principal_components = pca.fit_transform(data.data)
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['species'] = iris_df['species']
plt.figure(figsize=(8, 6))
colors = ['red', 'green', 'blue']
species = data.target_names
for i, color in enumerate(colors):
    subset = pca_df[pca_df['species'] == species[i]]
    plt.scatter(subset['PC1'], subset['PC2'], color=color, label=species[i])
plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

Output:



Experiment 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.

Program:

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

Output:

```
Training data:
           Sky  Temp Humidity    Wind Water Forecast PlayTennis
0        Sunny   Hot     High    Weak  Warm     Same         No
1        Sunny   Hot     High  Strong  Warm     Same         No
2        Rainy   Hot     High    Weak  Cool   Change        Yes
3        Sunny  Mild     High    Weak  Warm     Same        Yes
4        Sunny  Cool   Normal    Weak  Warm     Same        Yes
..         ...   ...      ...     ...   ...      ...        ...
96       Rainy  Cool      Low  Strong  Warm   Change         No
97   Overcast  Mild      Low    Weak  Warm   Change         No
98       Rainy   Hot      Low  Strong  Warm     Same        Yes
99       Sunny  Cool     High  Strong  Cool     Same         No
100      Sunny   Hot      Low  Strong  Warm   Change         No

[101 rows x 7 columns]

The final hypothesis is: ['Rainy', '?', 'Low', 'Strong', 'Warm', 'Same']
```

Experiment 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 ε Class1, else xi ε Class1

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

Program:

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
data = np.random.rand(100)
labels = np.zeros(100)
labels[:50] = np.where(data[:50] <= 0.5, 1, 2)
train_data = data[:50].reshape(-1, 1)
train_labels = labels[:50]
test_data = data[50:].reshape(-1, 1)
k_values = [1, 2, 3, 4, 5, 20, 30]
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(train_data, train_labels)
    predicted_labels = knn.predict(test_data)
        print(f"K = {k}")
    print("Predicted Labels:", predicted_labels)
    print()
```

Output:

```
K = 1
Predicted Labels: [1. 2. 1. 2. 2. 1. 1. 2. 1. 1. 2. 2. 1. 2. 2. 1. 1. 1. 2. 2. 2. 2. 1. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 2. 2.
 1. 2.]

K = 2
Predicted Labels: [1. 2. 1. 2. 2. 1. 1. 2. 1. 1. 2. 2. 1. 2. 2. 1. 1. 1. 2. 2. 2. 1. 1. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 2. 2.
 1. 2.]

K = 3
Predicted Labels: [1. 2. 1. 2. 2. 1. 1. 2. 1. 1. 2. 2. 1. 2. 2. 1. 1. 1. 2. 2. 2. 2. 1. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 2. 2.
 1. 2.]

K = 4
Predicted Labels: [1. 2. 1. 2. 2. 1. 1. 2. 1. 1. 2. 2. 1. 2. 2. 1. 1. 1. 2. 2. 2. 1. 1. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 2. 2.
 1. 2.]

K = 5
Predicted Labels: [1. 2. 1. 2. 2. 1. 1. 2. 1. 1. 2. 2. 1. 2. 2. 1. 1. 1. 2. 2. 2. 2. 1. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 2. 2.
 1. 2.]

K = 20
Predicted Labels: [1. 2. 1. 2. 2. 1. 1. 2. 1. 1. 2. 2. 1. 2. 2. 1. 1. 1. 2. 2. 2. 2. 1. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 1. 1. 1. 2. 2. 2. 1. 1. 2. 1. 2. 1. 1. 2. 2.
 1. 2.]

K = 30
Predicted Labels: [1. 2. 1. 2. 2. 1. 1. 2. 1. 1. 2. 2. 1. 2. 2. 2. 1. 1. 2. 2. 2. 2. 1. 2.
 2. 2. 2. 2. 1. 2. 1. 1. 1. 2. 1. 1. 2. 2. 2. 1. 2. 2. 1. 2. 1. 1. 2. 2.
 1. 2.]
```

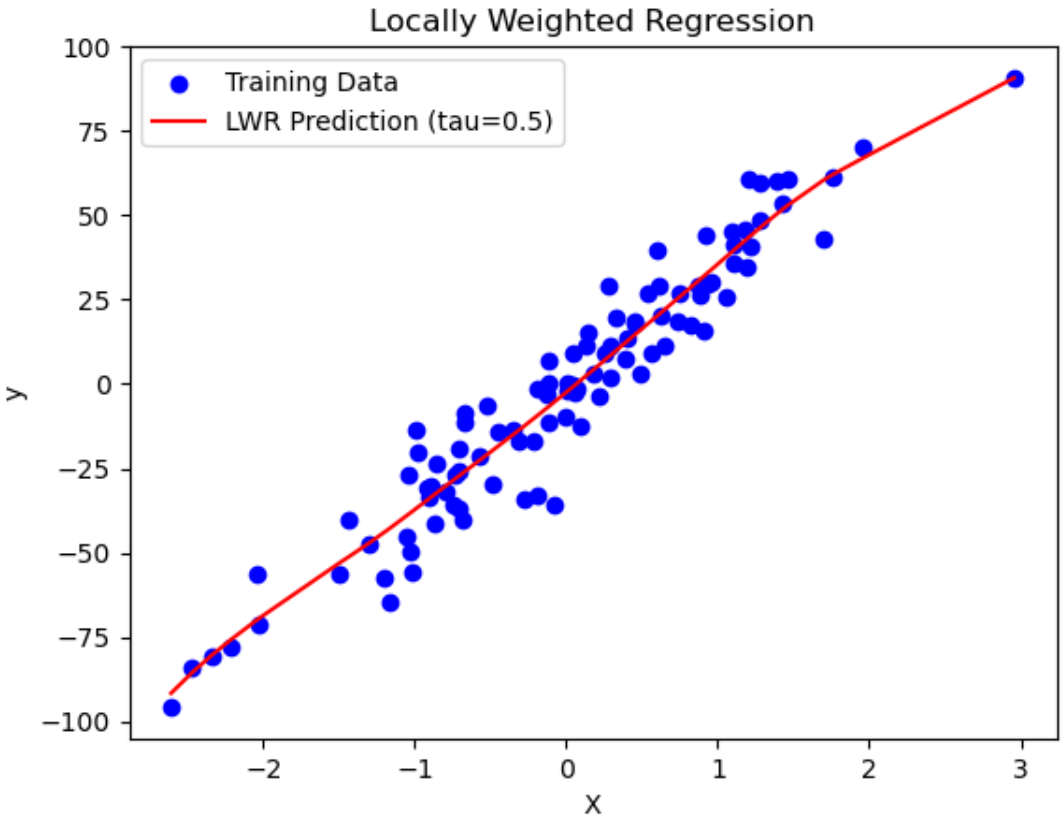
Experiment 6. Implement the non-parametric Locally Weighted Regression algorithm to fit data points. Select appropriate data set for your experiment and draw graphs.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
def kernel(point, x, tau):
    m = x.shape[0]
    weights = np.mat(np.eye(m))
    for i in range(m):
        diff = point - x[i]
        weights[i, i] = np.exp(diff @ diff.T / (-2.0 * tau ** 2))
    return weights
def locally_weighted_regression(test_point, x, y, tau):
    x_mat = np.mat(x)
    y_mat = np.mat(y).T
    weights = kernel(test_point, x_mat, tau)
    x_tx = x_mat.T * (weights * x_mat)
    if np.linalg.det(x_tx) == 0.0:
        print("Singular Matrix")
        return
    theta = x_tx.I * (x_mat.T * (weights * y_mat))
    return test_point @ theta
def lwr_predictions(x_test, x_train, y_train, tau):
    m = x_test.shape[0]
    y_pred = np.zeros(m)
    for i in range(m):
        y_pred[i] = locally_weighted_regression(x_test[i], x_train, y_train, tau)
    return y_pred
X, y = make_regression(n_samples=100, n_features=1, noise=10)
X = np.array(X)
y = np.array(y)
X_with_intercept = np.hstack((np.ones((X.shape[0], 1)), X))
sort_idx = X[:, 0].argsort()
X_sorted = X[sort_idx]
X_sorted_with_intercept = X_with_intercept[sort_idx]
y_sorted = y[sort_idx]
tau = 0.5  # bandwidth
y_pred = lwr_predictions(X_sorted_with_intercept, X_with_intercept, y, tau)
plt.scatter(X, y, label="Training Data", color="blue")
plt.plot(X_sorted, y_pred, color='red', label=f'LWR Prediction (tau={tau})')
plt.title("Locally Weighted Regression")
plt.xlabel("X")
```

```
plt.ylabel("y")
plt.legend()
plt.show()
```

Output:



Experiment 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.

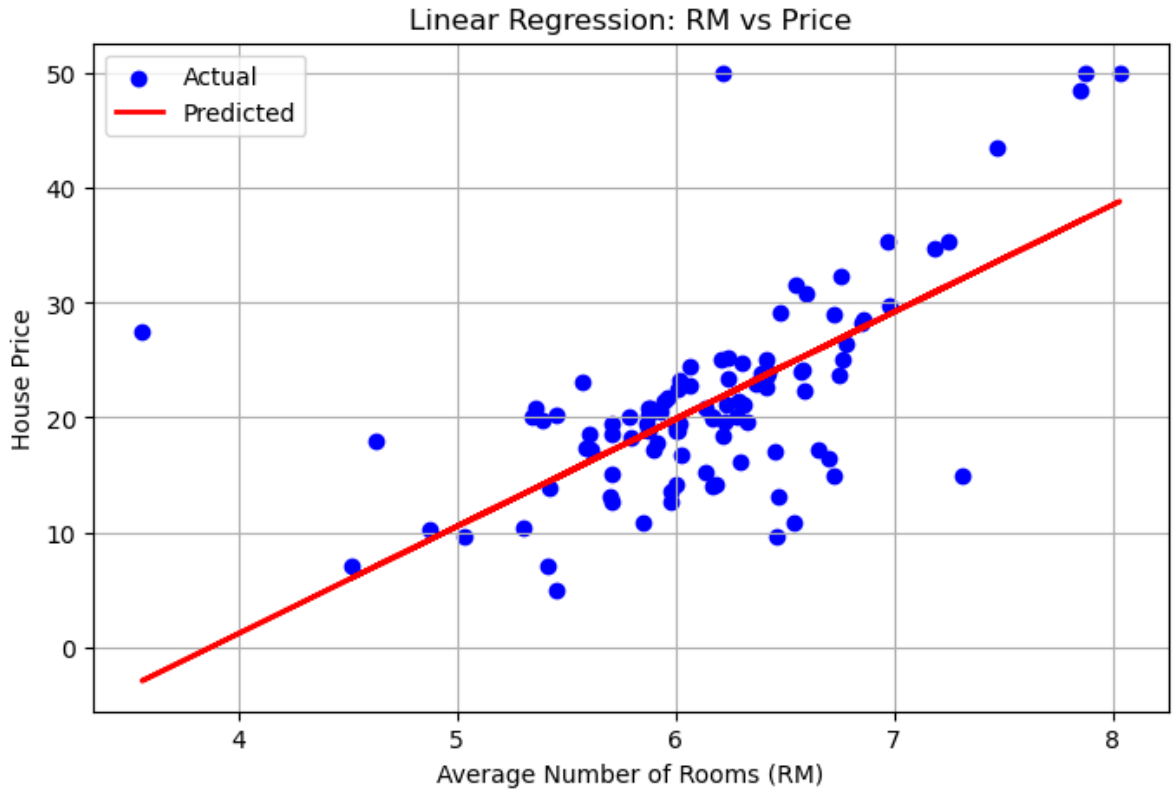
Program:

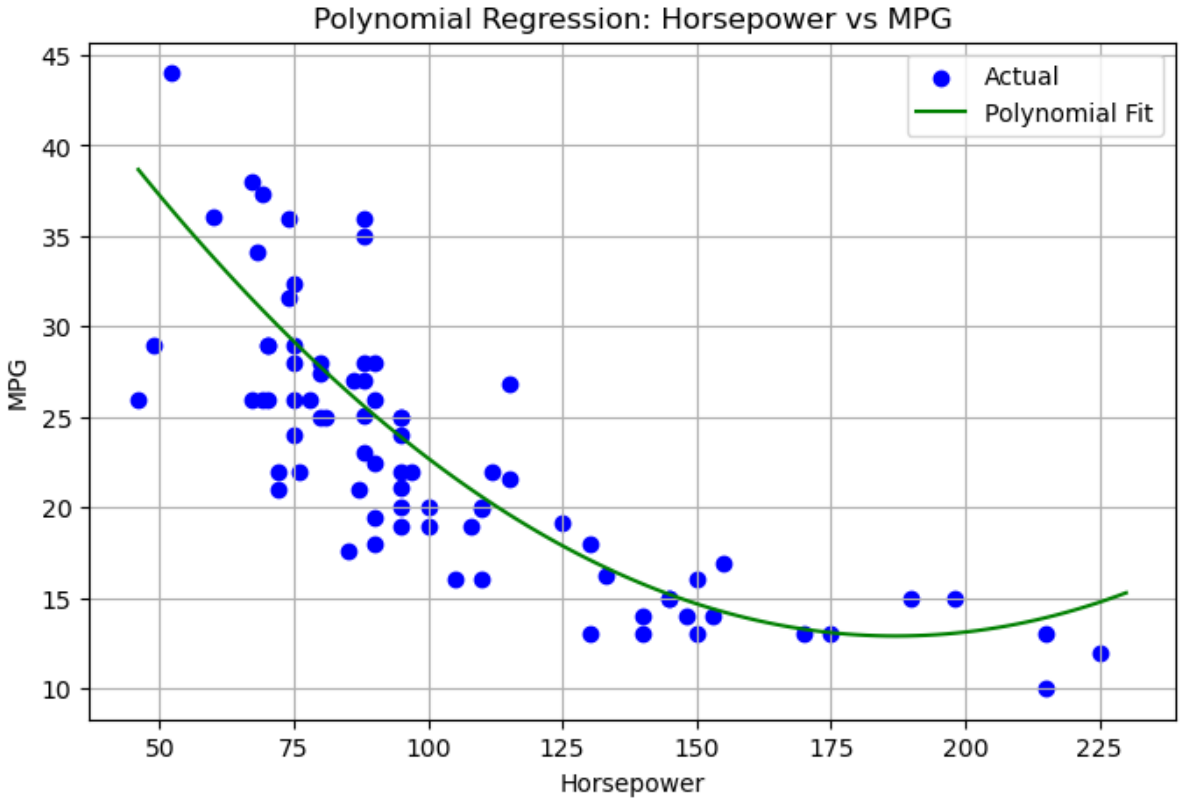
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import seaborn as sns
print("=== Linear Regression: Boston Housing Dataset ===")
boston_url =
"https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv"
boston_df = pd.read_csv(boston_url)
X_boston = boston_df.drop(columns='medv')
y_boston = boston_df['medv']
X_rm = X_boston[['rm']]
X_train, X_test, y_train, y_test = train_test_split(X_rm, y_boston, test_size=0.2,
random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
plt.figure(figsize=(8, 5))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('House Price')
plt.title('Linear Regression: RM vs Price')
plt.legend()
plt.grid(True)
plt.show()

print("\n=== Polynomial Regression: Auto MPG Dataset ===")
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-
mpg.data"
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                'acceleration', 'model year', 'origin', 'car name']
df = pd.read_csv(url, names=column_names, delim_whitespace=True, na_values='?')
```

```
df.dropna(inplace=True)
X_auto = df[['horsepower']]
y_auto = df['mpg']
X_train_auto, X_test_auto, y_train_auto, y_test_auto = train_test_split(X_auto, y_auto,
test_size=0.2, random_state=42)
degree = 2
poly_model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
poly_model.fit(X_train_auto, y_train_auto)
y_pred_auto = poly_model.predict(X_test_auto)
print(f"Polynomial Regression (degree={degree}) - MSE:
{mean_squared_error(y_test_auto, y_pred_auto):.4f}")
print(f"R² Score: {r2_score(y_test_auto, y_pred_auto):.4f}")
plt.figure(figsize=(8, 5))
plt.scatter(X_test_auto, y_test_auto, color='blue', label='Actual')
x_range = np.linspace(X_auto.min(), X_auto.max(), 100).reshape(-1, 1)
plt.plot(x_range, poly_model.predict(x_range), color='green', label='Polynomial Fit')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.title('Polynomial Regression: Horsepower vs MPG')
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
=== Linear Regression: Boston Housing Dataset ===
Mean Squared Error: 46.144775347317264
R² Score: 0.3707569232254778
```





Experiment 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.

Program:

```
import numpy as np
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, classification_report, confusion_matrix

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)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=data.target_names))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
new_sample = np.array([[17.99, 10.38, 122.8, 1001.0, 0.1184, 0.2776, 0.3001,
                        0.1471, 0.2419, 0.07871, 1.095, 0.9053, 8.589,
                        153.4, 0.006399, 0.04904, 0.05373, 0.01587,
                        0.03003, 0.006193, 25.38, 17.33, 184.6,
                        2019.0, 0.1622, 0.6656, 0.7119, 0.2654,
                        0.4601, 0.1189]])
prediction = clf.predict(new_sample)
print("\nNew Sample Prediction:\nClass:", data.target_names[prediction][0])
```

Output:

```
Accuracy: 0.9473684210526315

Classification Report:
              precision    recall  f1-score   support

   malignant       0.93      0.93      0.93        43
      benign       0.96      0.96      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114


Confusion Matrix:
 [[40  3]
 [ 3 68]]

New Sample Prediction:
Class: malignant
Accuracy: 0.9473684210526315

Classification Report:
              precision    recall  f1-score   support

   malignant       0.93      0.93      0.93        43
      benign       0.96      0.96      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114


Confusion Matrix:
 [[40  3]
 [ 3 68]]

New Sample Prediction:
Class: malignant
```

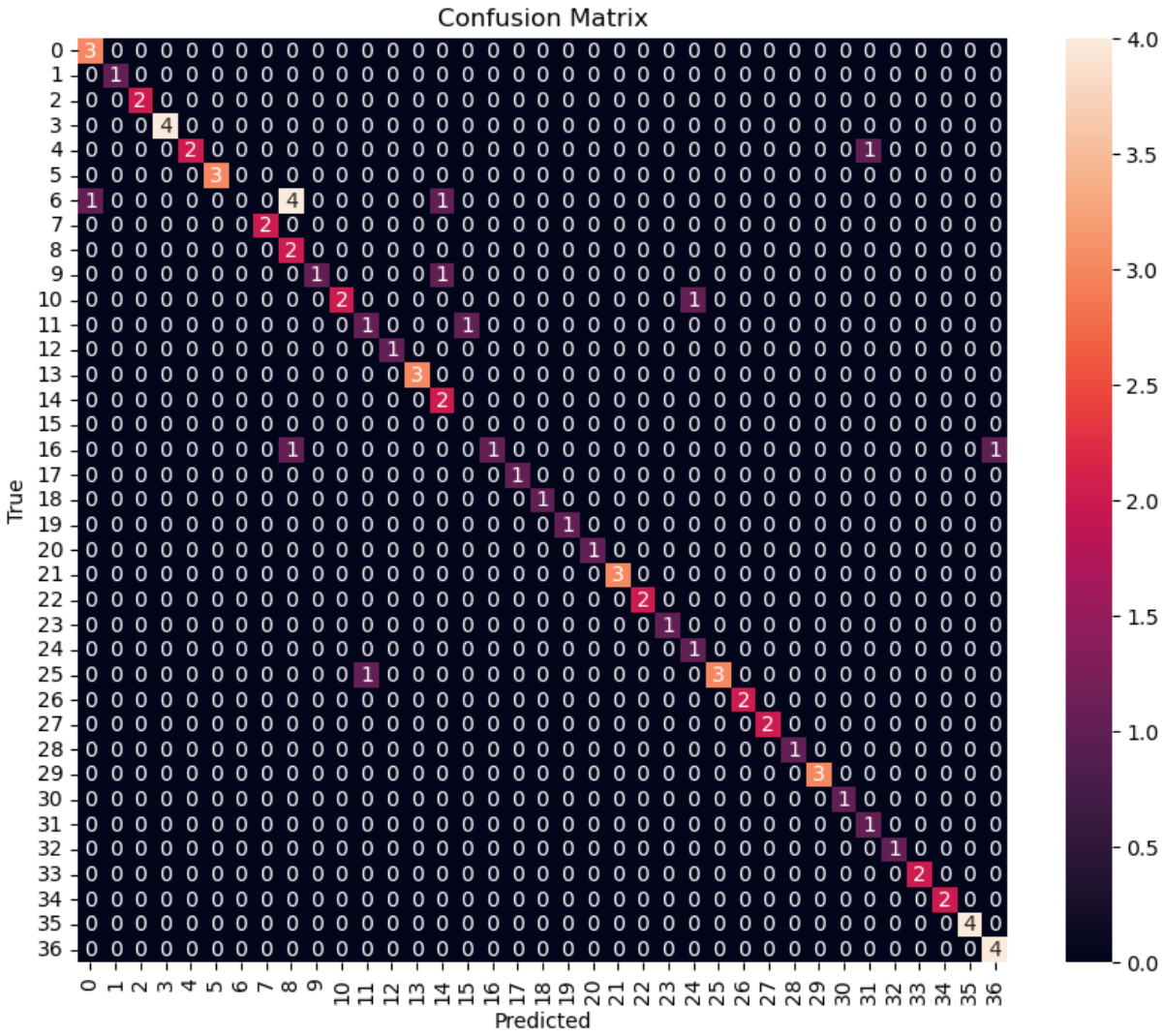
Experiment 9. Develop a program to implement the Naive Bayesian classifier considering Olivetti Face Data set for training. Compute the accuracy of the classifier, considering a few test data sets.

Program:

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

faces = fetch_olivetti_faces()
X = faces.data
y = faces.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

Output:

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to C:\Users\Sourabh\scikit_learn_data
Accuracy: 83.75%

Classification Report:
              precision    recall  f1-score   support

           0       0.75      1.00      0.86         3
           1       1.00      1.00      1.00         1
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         4
           4       1.00      0.67      0.80         3
           5       1.00      1.00      1.00         3
           7       0.00      0.00      0.00         6
           8       1.00      1.00      1.00         2
           9       0.29      1.00      0.44         2
          10       1.00      0.50      0.67         2
          11       1.00      0.67      0.80         3
          12       0.50      0.50      0.50         2
          13       1.00      1.00      1.00         1
          14       1.00      1.00      1.00         3
          15       0.50      1.00      0.67         2
          16       0.00      0.00      0.00         0
          17       1.00      0.33      0.50         3
          18       1.00      1.00      1.00         1
          19       1.00      1.00      1.00         1
          20       1.00      1.00      1.00         1
          21       1.00      1.00      1.00         1
          22       1.00      1.00      1.00         3
          23       1.00      1.00      1.00         2
          24       1.00      1.00      1.00         1
          25       0.50      1.00      0.67         1
          26       1.00      0.75      0.86         4
          27       1.00      1.00      1.00         2
          28       1.00      1.00      1.00         2
          29       1.00      1.00      1.00         1
          32       1.00      1.00      1.00         3
          33       1.00      1.00      1.00         1
          34       0.50      1.00      0.67         1
          35       1.00      1.00      1.00         1
          36       1.00      1.00      1.00         2
          37       1.00      1.00      1.00         2
          38       1.00      1.00      1.00         4
          39       0.80      1.00      0.89         4

    accuracy                           0.84        80
   macro avg       0.86      0.88      0.85        80
weighted avg       0.85      0.84      0.82        80


Confusion Matrix:
 [[3 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 2 ... 0 0 0]
 ...
 [0 0 0 ... 2 0 0]
 [0 0 0 ... 0 4 0]
 [0 0 0 ... 0 0 4]]
```


Confusion Matrix
True
Predicted
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0

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

Program:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
X['Cluster'] = clusters
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap='viridis', alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
        c='red', marker='x', s=200, label='Centroids')
plt.title('K-Means Clustering on Breast Cancer Dataset')
plt.xlabel(data.feature_names[0])
plt.ylabel(data.feature_names[1])
plt.legend()
plt.show()
```

Output: