1) A botanist is studying three species of the Iris flower: Iris-setosa, Iris-versicolor, and Iris-virginica. She collects data on the sepal and petal dimensions of several flowers and wants to predict the species of a flower based on these measurements.

The dataset includes 20 samples with the following columns:

Sepal Length (cm)

Sepal Width (cm)|

Petal Length (cm)

Petal Width (cm)

a) Create a histogram for the Sepal Length. What is the distribution of Sepal Length values among the 20 samples?
b) Plot a boxplot for Sepal Width. Are there any outliers in the Sepal Width measurements?
c) Plot a frequency bar chart for the Species column. Which species is most represented in the dataset?
d) Plot a bar chart showing the average Petal Width for each species. Which species has the widest petals on average?
e) Plot a scatter plot between Sepal Length and Petal Length. Is there any correlation between these two features?
f) Create a boxplot of Petal Length grouped by Species. Which species has the longest petals on average?
g) Plot a line graph to compare Sepal Length and Sepal Width for each sample. Are there any trends observed?
h) Create a heatmap of the correlation matrix for all numerical columns. Which pair of variables are most strongly correlated?
i) Create a 3D scatter plot with Sepal Length, Petal Length, and Petal Width as axes, colored by Species. Can you observe distinct clusters for each species?
j) Generate a pair plot for all numerical features. What patterns or clusters can be observed for each species?

Program:

1) import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

```
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D

# Example Data
data = {
    'Sepal Length (cm)': np.random.uniform(4.3, 7.9, 20),
    'Sepal Width (cm)': np.random.uniform(2.0, 4.4, 20),
    'Petal Length (cm)': np.random.uniform(1.0, 6.9, 20),
    'Petal Width (cm)': np.random.uniform(0.1, 2.5, 20),
    'Species': np.random.choice(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], 20)
}

# Create DataFrame
df = pd.DataFrame(data)

a) plt.figure(figsize=(8, 6))
plt.hist(df['Sepal Length (cm)'], bins=8, color='skyblue', edgecolor='black')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Frequency')
plt.title('Histogram of Sepal Length')
plt.show()

b)  plt.figure(figsize=(8, 6))
sns.boxplot(y=df['Sepal Width (cm)'], color='lightgreen')
plt.title('Boxplot of Sepal Width')
plt.ylabel('Sepal Width (cm)')
plt.show()
c)  plt.figure(figsize=(8, 6))
df['Species'].value_counts().plot(kind='bar', color=['orange', 'blue', 'green'])
```

```
plt.xlabel('Species')
plt.ylabel('Frequency')
plt.title('Frequency Bar Chart for Species')
plt.show()
```

d)
```
plt.figure(figsize=(8, 6))
df.groupby('Species')['Petal Width (cm)'].mean().plot(kind='bar', color=['orange', 'blue', 'green'])
plt.xlabel('Species')
plt.ylabel('Average Petal Width (cm)')
plt.title('Average Petal Width per Species')
plt.show()
```

e)
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Sepal Length (cm)', y='Petal Length (cm)', hue='Species', data=df)
plt.xlabel('
```

f)
```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Species', y='Petal Length (cm)', data=df, palette='Set2')
plt.xlabel('Species')
plt.ylabel('Petal Length (cm)')
plt.title('Boxplot of Petal Length Grouped by Species')
plt.show()
```

g)
```
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Sepal Length (cm)'], marker='o', label='Sepal Length', color='blue')
plt.plot(df.index, df['Sepal Width (cm)'], marker='o', label='Sepal Width', color='green')
```

```
plt.xlabel('Sample Index')
plt.ylabel('Length/Width (cm)')
plt.title('Comparison of Sepal Length and Sepal Width')
plt.legend()
plt.grid()
plt.show()
```

h)
```
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Sepal Length (cm)'], marker='o', label='Sepal Length',
color='blue')
plt.plot(df.index, df['Sepal Width (cm)'], marker='o', label='Sepal Width', color='green')
plt.xlabel('Sample Index')
plt.ylabel('Length/Width (cm)')
plt.title('Comparison of Sepal Length and Sepal Width')
plt.legend()
plt.grid()
plt.show()
```

i)
```
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(
    df['Sepal Length (cm)'],
    df['Petal Length (cm)'],
    df['Petal Width (cm)'],
    c=pd.Categorical(df['Species']).codes, cmap='viridis', s=100
)
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Petal Length (cm)')
ax.set_zlabel('Petal Width (cm)')
```

```
plt.title('3D Scatter Plot of Iris Dataset')
plt.show()
```

```
j) sns.pairplot(df, hue='Species', diag_kind='kde', palette='Set1')
plt.suptitle('Pair Plot of Numerical Features', y=1.02)
plt.show()
```

2) A teacher wants to predict student exam scores based on the number of hours they study. She collected data on 20 students, recording the number of hours they studied and their scores on a recent test. Use the Least Squares Method to build a simple linear regression model to predict exam scores based on study hours.

Program:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Load dataset (replace this with your own dataset)
# For example: df = pd.read_csv('your_dataset.csv')
data = {
    "Hours Studied": [2.5, 5.1, 3.2, 8.5, 3.5, 1.5, 9.2, 5.5, 8.3, 2.7, 7.7, 5.9, 4.5, 3.3, 1.1, 8.9, 2.5, 1.9, 6.1, 7.4],
    "Exam Scores": [21, 47, 27, 75, 30, 20, 88, 60, 81, 25, 85, 62, 41, 42, 17, 95, 30, 24, 67, 69]
}
df = pd.DataFrame(data)

# Extract features and target
X = df["Hours Studied"].values.reshape(-1, 1)
y = df["Exam Scores"]

# Build linear regression model
model = LinearRegression()
model.fit(X, y)

# Predicted scores
y_pred = model.predict(X)

# Model parameters
```

```
slope = model.coef_[0]
intercept = model.intercept_

# Print the model equation
print(f"Model Equation: Exam Score = {slope:.2f} * Hours Studied +
{intercept:.2f}")

# Visualization: scatter plot and regression line
plt.figure(figsize=(10, 6))
plt.scatter(df["Hours Studied"], df["Exam Scores"], color="blue", label="Actual
Data")
plt.plot(df["Hours Studied"], y_pred, color="red", label="Regression Line")
plt.title("Linear Regression: Exam Scores vs. Hours Studied")
plt.xlabel("Hours Studied")
plt.ylabel("Exam Score")
plt.legend()
plt.grid(True)
plt.show()
```

3) A fitness trainer wants to analyze the relationship between an individual's daily calorie intake and their weight. She collected data from 30 clients, recording their average daily calorie intake (in kilocalories) and their body weight (in kilograms).Build a simple linear regression model to predict a client's weight based on their daily calorie intake.

Program:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Sample dataset (you can replace this with your actual data or load from a
file)
data = {
   "Calorie Intake (kcal)": [2000, 2200, 1800, 2500, 2700, 2100, 1900, 3000,
2300, 2400,
                          2600, 2800, 1850, 2050, 2250, 2150, 2350, 2900, 3100,
1950,
                          2450, 2650, 1750, 1850, 2050, 2200, 2750, 2850, 2950,
2500],
   "Weight (kg)": [65, 68, 60, 75, 80, 67, 63, 85, 70, 73,
                78, 82, 62, 66, 69, 68, 72, 88, 90, 64,
                74, 77, 58, 61, 65, 69, 81, 84, 86, 75]
}
```

```
df = pd.DataFrame(data)

# Features (Calorie Intake) and Target (Weight)
X = df["Calorie Intake (kcal)"].values.reshape(-1, 1)
y = df["Weight (kg)"]

# Build the Linear Regression Model
model = LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

# Model parameters
slope = model.coef_[0]
intercept = model.intercept_

# Print the regression equation
print(f"Model Equation: Weight = {slope:.2f} * Calorie Intake + {intercept:.2f}")

# Visualization
plt.figure(figsize=(10, 6))
plt.scatter(df["Calorie Intake (kcal)"], df["Weight (kg)"], color="blue",
label="Actual Data")
plt.plot(df["Calorie Intake (kcal)"], y_pred, color="red", label="Regression
Line")
plt.title("Linear Regression: Weight vs. Calorie Intake")
plt.xlabel("Calorie Intake (kcal)")
plt.ylabel("Weight (kg)")
plt.legend()
plt.grid(True)
plt.show()
```

4) A university wants to predict whether a student will Pass or Fail an exam based on the number of hours they study. The university collected data on 20 students, recording the number of hours they studied and whether they passed the exam. Use a Logistic Regression Model to classify students as "Pass" or 'Fail."

Program:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Sample dataset: hours studied and exam outcome (Pass = 1, Fail = 0)
data = {
    "Hours Studied": [1, 2, 1.5, 3, 2.5, 4, 5, 6, 7, 8, 1.2, 2.8, 3.5, 5.5, 6.8, 7.5,
8.2, 9, 1.1, 0.5],
    "Pass/Fail": [0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0]
}
df = pd.DataFrame(data)

# Features (Hours Studied) and Target (Pass/Fail)
X = df["Hours Studied"].values.reshape(-1, 1)
y = df["Pass/Fail"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Build the Logistic Regression Model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Decision boundary
plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(x="Hours Studied", y="Pass/Fail", data=df, hue="Pass/Fail",
palette="coolwarm", s=100)
plt.axhline(0.5, color="black", linestyle="--", label="Decision Boundary")
plt.title("Logistic Regression: Pass/Fail vs. Hours Studied")
plt.xlabel("Hours Studied")
plt.ylabel("Pass/Fail")
plt.legend(title="Outcome", loc="upper left")
plt.grid(True)
plt.show()
```

5) A real estate analyst wants to predict house prices based on factors such as location, size, and amenities. She collected data on 100 houses, recording their area, bedroom, bathroom, available amenities, and prices. Use a Logistic regression model to analyze this data and predict house prices based on these features.

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Sample dataset: Replace with actual data or load from a CSV
data = {
    "Area (sq ft)": [1200, 1500, 800, 2200, 1800, 1400, 1600, 2100, 1000, 2000,
                     1300, 1700, 1900, 2500, 900, 2300, 1200, 1100, 2400, 3000],
    "Bedrooms": [2, 3, 2, 4, 3, 2, 3, 4, 2, 4, 3, 3, 4, 5, 1, 4, 2, 2, 5, 6],
    "Bathrooms": [1, 2, 1, 3, 2, 1, 2, 3, 1, 2, 2, 2, 3, 4, 1, 3, 1, 1, 4, 5],
    "Amenities": [1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1],
```

```
    "Price Category": ["Medium", "High", "Low", "High", "High", "Medium", "Medium", "High",
                       "Low", "High", "Medium", "Medium", "High", "High", "Low", "High",
                       "Medium", "Low", "High", "High"]
}
df = pd.DataFrame(data)


# Preprocessing
X = df[["Area (sq ft)", "Bedrooms", "Bathrooms", "Amenities"]]
y = df["Price Category"]


# Encode categorical target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)  # Converts "Low", "Medium", "High" to 0, 1, 2


# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)


# Standardize features for better model performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Build Logistic Regression Model
model = LogisticRegression(multi_class="ovr", solver="liblinear")
model.fit(X_train_scaled, y_train)


# Predictions
```

```
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

# Visualization: Confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

6) An energy company wants to classify whether daily energy consumption will be high or low based on historical data. They collected data over 50 days, recording the following features:

a. Temperature (*C)

b. Humidity (%)

c. Wind Speed (km/h)

Build a logistic regression model to classify daily energy consumption as either

High or Low.

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Synthetic dataset: Replace this with your actual dataset if available
data = {
    "Temperature (°C)": [22, 25, 19, 30, 28, 21, 20, 27, 32, 35, 23, 24, 19, 31, 26, 22, 20, 29, 33, 34,
                         25, 30, 19, 28, 27, 21, 22, 31, 20, 34, 29, 26, 25, 33, 23, 24, 32, 30, 28, 20,
                         21, 19, 35, 33, 22, 31, 29, 28, 27, 25],
    "Humidity (%)": [55, 60, 65, 45, 50, 70, 80, 55, 40, 35, 58, 62, 68, 47, 52, 71, 75, 49, 42, 37,
                     59, 48, 66, 50, 53, 72, 74, 46, 76, 38, 41, 54, 63, 39, 56, 61, 43, 44, 51, 78,
                     73, 64, 36, 41, 57, 45, 47, 49, 52, 53],
    "Wind Speed (km/h)": [15, 18, 12, 25, 22, 14, 10, 20, 30, 35, 16, 17, 11, 28, 24, 13, 9, 26, 31, 34,
                          19, 27, 8, 23, 21, 12, 14, 29, 11, 33, 25, 22, 18, 32, 15, 16, 30, 24, 23, 12,
                          13, 10, 34, 33, 14, 28, 26, 25, 21, 20],
    "Energy Consumption": ["Low", "High", "Low", "High", "High", "Low", "Low", "High", "High", "High",
                           "Low", "High", "Low", "High", "High", "Low", "Low", "High", "High", "High",
```

```
                "Low", "High", "Low", "High", "High", "Low", "Low", "High", "Low",
"High",
                "High", "High", "Low", "High", "Low", "High", "High", "High", "High",
"Low",
                "Low", "Low", "High", "High", "Low", "High", "High", "High", "High",
"High"]
}

df = pd.DataFrame(data)

# Preprocessing
X = df[["Temperature (°C)", "Humidity (%)", "Wind Speed (km/h)"]]
y = df["Energy Consumption"]

# Encode the target variable (High = 1, Low = 0)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3,
random_state=42)

# Standardize the features for better model performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build Logistic Regression Model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)
```

```
# Predictions
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

# Visualization: Confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

7) A bank wants to predict whether a loan application will be approved or rejected based on the applicant's income and loan amount requested. The bank collected data from 20 past applicants, including their income, loan amount, and loan approval status. Use a Single-Layer Perceptron to classify the loan applications as "Approved" or "Rejected."

   Program:

   ```
   import pandas as pd
   ```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Sample dataset: Replace with actual data or load from a CSV
data = {
   "Income": [3000, 4500, 2000, 5000, 4000, 3800, 2500, 6000, 3200,
4200,
            3100, 4800, 3600, 5200, 2800, 4100, 2700, 5900, 3500, 4700],
   "Loan Amount": [1000, 1500, 800, 2000, 1200, 1100, 700, 2500, 900,
1400,
                950, 1700, 1300, 2200, 850, 1350, 750, 2400, 1250, 1650],
   "Approval Status": ["Approved", "Approved", "Rejected", "Approved",
"Approved",
                       "Rejected", "Rejected", "Approved", "Rejected",
"Approved",
                       "Rejected", "Approved", "Rejected", "Approved",
"Rejected",
                       "Approved", "Rejected", "Approved", "Rejected",
"Approved"]
}
df = pd.DataFrame(data)

# Preprocessing
X = df[["Income", "Loan Amount"]]
y = df["Approval Status"]

# Encode the target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)  # "Approved" -> 1, "Rejected"
-> 0

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.3, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)

# Build the Single-Layer Perceptron Model
model = MLPClassifier(hidden_layer_sizes=(), max_iter=1000,
random_state=42, activation='logistic')
model.fit(X_train_scaled, y_train)

# Predictions
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

8) An online retail store wants to predict whether a customer will make a purchase based on their browsing behavior. The store has collected data on 1000 customers, recording the following features:

a. Time Spent on Website (in minutes)
b. Number of Pages Visited
c. Age Group (1 = 18-25, 2 = 26-35, 3 = 36-50, 4 = 51+)
d. Previous Purchase (1 = Yes, 0 = No)

Build a Single-Layer Perceptron (SLP) model to predict whether a customer will make a purchase based on their browsing activity.

Program:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Generate synthetic dataset: Replace with actual data if available
np.random.seed(42)
data = {
    "Time Spent on Website (min)": np.random.randint(5, 50, 1000),
    "Number of Pages Visited": np.random.randint(1, 20, 1000),
    "Age Group": np.random.choice([1, 2, 3, 4], 1000),
    "Previous Purchase": np.random.choice([0, 1], 1000, p=[0.7, 0.3]),
    "Purchase": np.random.choice([0, 1], 1000, p=[0.6, 0.4])
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Time Spent on Website (min)", "Number of Pages Visited", "Age
Group", "Previous Purchase"]]
y = df["Purchase"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the Single-Layer Perceptron Model
model = MLPClassifier(hidden_layer_sizes=(), max_iter=1000,
random_state=42, activation='logistic')
model.fit(X_train_scaled, y_train)
```

```
# Predictions
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["No Purchase", "Purchase"], yticklabels=["No Purchase",
"Purchase"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

9) A hospital wants to predict whether a patient is at high risk of heart disease based on their lifestyle and medical data. The hospital has collected data from 500 patients with the following features:
a. Age (in years)
b. Cholesterol Level (mg/dL)
c. Blood Pressure (mm Hg)
d. Number of Cigarettes Smoked per Day
e. Physical Activity Level (1 = Sedentary, 0 = Active)
Build a Single-Layer Perceptron (SLP) model to classify whether a patient is at high risk of heart disease or not based on the provided features.

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Generating synthetic dataset: Replace with actual data if available
np.random.seed(42)
data = {
    "Age": np.random.randint(25, 80, 500),  # Age between 25 and 80
    "Cholesterol Level (mg/dL)": np.random.randint(150, 300, 500),  #
Cholesterol level between 150 and 300
    "Blood Pressure (mm Hg)": np.random.randint(90, 180, 500),  # Blood
pressure between 90 and 180
    "Cigarettes Smoked per Day": np.random.randint(0, 30, 500),  #
Number of cigarettes smoked per day
    "Physical Activity Level": np.random.choice([0, 1], 500),  # 1 =
Sedentary, 0 = Active
    "Heart Disease Risk": np.random.choice([0, 1], 500)  # 1 = High risk, 0 =
Low risk
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Age", "Cholesterol Level (mg/dL)", "Blood Pressure (mm Hg)",
"Cigarettes Smoked per Day", "Physical Activity Level"]]
y = df["Heart Disease Risk"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the Single-Layer Perceptron Model
model = MLPClassifier(hidden_layer_sizes=(), max_iter=1000,
random_state=42, activation='logistic')
model.fit(X_train_scaled, y_train)
```

```
# Predictions
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Low Risk", "High Risk"], yticklabels=["Low Risk", "High Risk"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

10) A telecommunications company wants to predict whether a customer will
Churn (leave the service) based on their monthly bill amount and the number of complaints they've raised. The company collected data from 20 customers, including their monthly bill, number of complaints, and churn status. Use a Multi-Layer Perceptron (MLP) with Backpropagation to classify customers as
"Churn" or "No Churn."

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Sample dataset: Replace with actual data if available
np.random.seed(42)
data = {
   "Monthly Bill": np.random.randint(20, 150, 20),  # Monthly bill between
20 and 150
   "Number of Complaints": np.random.randint(0, 5, 20),  # Number of
complaints (0 to 4)
   "Churn": np.random.choice([0, 1], 20)  # 0 = No Churn, 1 = Churn
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Monthly Bill", "Number of Complaints"]]
y = df["Churn"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the Multi-Layer Perceptron (MLP) model with Backpropagation
model = MLPClassifier(hidden_layer_sizes=(5,), max_iter=1000,
random_state=42, activation='relu')
model.fit(X_train_scaled, y_train)

# Predictions
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["No Churn", "Churn"], yticklabels=["No Churn", "Churn"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

11) A company wants to predict whether an employee will leave the company (attrition) based on their job satisfaction level and years of tenure. The company has gathered data from 100 employees, which includes their job satisfaction level (on a scale from 1 to 10), years of tenure, and whether the employee stayed or left (attrition status).Use MLP network (number of layers,
neurons per layer, activation functions) to predict employee attrition, using job satisfaction and tenure as input features?

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Generate synthetic dataset: Replace with actual data if available
np.random.seed(42)
data = {
    "Job Satisfaction": np.random.randint(1, 11, 100),  # Job satisfaction
between 1 and 10
    "Years of Tenure": np.random.randint(1, 31, 100),  # Years of tenure
between 1 and 30
    "Attrition": np.random.choice([0, 1], 100)  # 0 = Stayed, 1 = Left (Attrition
status)
}

df = pd.DataFrame(data)
```

```
# Features and Target
X = df[["Job Satisfaction", "Years of Tenure"]]
y = df["Attrition"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the Multi-Layer Perceptron Model with multiple layers and neurons
# We use two hidden layers with 8 and 4 neurons, ReLU activation, and
the output layer uses the sigmoid function
model = MLPClassifier(hidden_layer_sizes=(8, 4), max_iter=1000,
random_state=42, activation='relu', solver='adam')

# Fit the model on the training data
model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Stayed", "Left"], yticklabels=["Stayed", "Left"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
```

```
plt.ylabel("Actual")
plt.show()
```

12) A retail company wants to categorize its customers into two segments: high spenders and low spenders. The company has collected data on customer age, monthly spending, and product category preferences. The company intends to use a Multi-Layer Perceptron (MLP) with backpropagation to classify customers into these two segments. Design the architecture for an MLP model to classify customers into either the "high spender' or low spender'.

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Generating synthetic dataset
np.random.seed(42)

# Simulating 100 customers' data
data = {
    "Age": np.random.randint(18, 70, 100),  # Customer age between 18
and 70
    "Monthly Spending": np.random.randint(50, 2000, 100),  # Monthly
spending between 50 and 2000
    "Product Category Preference": np.random.choice([1, 2, 3], 100),  # 1,
2, 3 represent different categories
    "Spender Type": np.random.choice([0, 1], 100)  # 0 = Low spender, 1 =
High spender
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Age", "Monthly Spending", "Product Category Preference"]]
y = df["Spender Type"]
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the Multi-Layer Perceptron Model with multiple layers and neurons
# Let's use two hidden layers with 16 and 8 neurons, ReLU activation
function
model = MLPClassifier(hidden_layer_sizes=(16, 8), max_iter=1000,
random_state=42, activation='relu', solver='adam')

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Low Spender", "High Spender"], yticklabels=["Low Spender",
"High Spender"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

13) .An email service provider wants to classify emails as Spam or Not Spam based on certain features of the email. The features include:
1. Email Length: The number of words in the email.
2 Number of Links: The count of hyperlinks in the email.
A dataset of 20 emails has been collected. Use a Support Vector Machine (SVM) classifier to predict whether an email is spam or not spam.

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Generate synthetic dataset for 20 emails
# Columns: Email Length (words), Number of Links, and Spam (1 = Spam, 0 = Not Spam)
np.random.seed(42)

data = {
    "Email Length": np.random.randint(50, 1000, 20),  # Email length between 50 and 1000 words
    "Number of Links": np.random.randint(0, 20, 20),  # Number of links between 0 and 20
    "Spam": np.random.choice([0, 1], 20)  # 0 = Not Spam, 1 = Spam
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Email Length", "Number of Links"]]
y = df["Spam"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the Support Vector Machine (SVM) model with a linear kernel
svm_model = SVC(kernel='linear', random_state=42)

# Train the model
svm_model.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test_scaled)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Not Spam", "Spam"], yticklabels=["Not Spam", "Spam"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

14) A bank wants to predict whether a loan application will be Approved or Rejected based on two features:
1. Annual Income: The annual income of the applicant in thousands of dollars.
2. Credit Score: The applicant's credit score.
The bank collected data from 20 applicants. Use AdaBoost Classifier to predict whether a loan application will be approved or rejected.

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate synthetic dataset for 20 loan applicants
# Columns: Annual Income (in thousands), Credit Score, and Loan
Approval Status (1 = Approved, 0 = Rejected)
np.random.seed(42)

data = {
    "Annual Income": np.random.randint(30, 150, 20),  # Annual income
between 30k and 150k
    "Credit Score": np.random.randint(300, 850, 20),  # Credit score
between 300 and 850
    "Loan Approval": np.random.choice([0, 1], 20)  # 0 = Rejected, 1 =
Approved
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Annual Income", "Credit Score"]]
y = df["Loan Approval"]

# 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)

# Build the AdaBoost Classifier with a base estimator (Decision Tree)
# AdaBoost uses weak classifiers (decision trees in this case) to improve
performance
base_estimator = DecisionTreeClassifier(max_depth=1)  # Decision stump
(shallow tree)
ada_boost_model = AdaBoostClassifier(base_estimator=base_estimator,
n_estimators=50, random_state=42)

# Train the AdaBoost model
```

```
ada_boost_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = ada_boost_model.predict(X_test)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Rejected", "Approved"], yticklabels=["Rejected", "Approved"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

15) A real estate company wants to predict the Price of a House based on its characteristics:
1. Size: The size of the house in square meters.
2. Number of Rooms: The number of rooms in the house.
The company has collected data from 20 houses. Use Gradient Boosting Regressor to predict the price of a house based on the given dataset.

Program:

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

# Generate synthetic dataset for 20 houses
np.random.seed(42)
```

```
data = {
    "Size (sq meters)": np.random.randint(50, 300, 20),  # House size
between 50 and 300 sq meters
    "Number of Rooms": np.random.randint(1, 10, 20),  # Number of rooms
between 1 and 10
    "Price": np.random.randint(50000, 500000, 20)  # Price of house
between 50k and 500k
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Size (sq meters)", "Number of Rooms"]]
y = df["Price"]

# 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)

# Build the Gradient Boosting Regressor model
gbr_model = GradientBoostingRegressor(n_estimators=100,
random_state=42)

# Train the model
gbr_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = gbr_model.predict(X_test)

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

# Print results
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Plotting Actual vs Predicted Prices
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle='--')
```

```
plt.title("Actual vs Predicted House Prices")
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.show()
```

16) A telecommunications company wants to predict whether a customer will
Churn (leave the service) or Stay (remain with the service) based on the following features:

1. Monthly Spend: The amount the customer spends per month on their subscription in dollars.
2. Years with Company: The number of years the customer has been with the company.

The company collected data from 20 customers. Use the K-Nearest Neighbors (KNN) model to predict whether a customer will churn or stay based on the given dataset.

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Generate synthetic dataset for 20 customers
np.random.seed(42)

data = {
    "Monthly Spend ($)": np.random.randint(30, 200, 20),  # Monthly spend between $30 and $200
    "Years with Company": np.random.randint(1, 10, 20),  # Years with company between 1 and 10
    "Churn": np.random.choice([0, 1], 20)  # 0 = Stay, 1 = Churn
}

df = pd.DataFrame(data)

# Features and Target
X = df[["Monthly Spend ($)", "Years with Company"]]
y = df["Churn"]
```

```
# 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)

# Build the KNN model
k = 3  # Number of neighbors
knn_model = KNeighborsClassifier(n_neighbors=k)

# Train the model
knn_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn_model.predict(X_test)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print results
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Visualization: Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
xticklabels=["Stay", "Churn"], yticklabels=["Stay", "Churn"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Visualization: Decision Boundary (Optional)
from matplotlib.colors import ListedColormap

# Create a mesh grid for visualization
h = 1  # Mesh step size
x_min, x_max = X["Monthly Spend ($)"].min() - 10, X["Monthly Spend
($)"].max() + 10
```

```
y_min, y_max = X["Years with Company"].min() - 1, X["Years with
Company"].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))

# Predict on mesh grid
Z = knn_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundary
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(["blue", "orange"]))
plt.scatter(X["Monthly Spend ($)"], X["Years with Company"], c=y,
edgecolor="k", cmap=ListedColormap(["blue", "orange"]))
plt.title(f"K-Nearest Neighbors (k={k}) Decision Boundary")
plt.xlabel("Monthly Spend ($)")
plt.ylabel("Years with Company")
plt.show()
```

17) A retail company wants to segment its customers based on their Annual Spending and Frequency of Purchases to improve marketing strategies. The company collected data from 20 customers. Use the K-Means Clustering algorithm to identify customer segments.

Program:

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

# Generate synthetic dataset for 20 customers
np.random.seed(42)

data = {
    "Annual Spending ($)": np.random.randint(500, 5000, 20),  # Spending
between $500 and $5000
    "Frequency of Purchases": np.random.randint(1, 50, 20)    # Purchases
between 1 and 50 times per year
}

df = pd.DataFrame(data)
```

```
# Preprocess the data (optional: scaling for better performance of K-Means)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Apply K-Means Clustering
k = 3  # Number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
df["Cluster"] = kmeans.fit_predict(scaled_data)

# Cluster centroids
centroids = scaler.inverse_transform(kmeans.cluster_centers_)

# Print clustered data
print("Clustered Data:")
print(df)

# Visualize the clusters
plt.figure(figsize=(10, 6))
for cluster in range(k):
    cluster_data = df[df["Cluster"] == cluster]
    plt.scatter(cluster_data["Annual Spending ($)"], cluster_data["Frequency of Purchases"], label=f"Cluster {cluster}")

# Plot cluster centroids
plt.scatter(centroids[:, 0], centroids[:, 1], color="red", marker="x", s=100, label="Centroids")

# Add labels and title
plt.title("Customer Segments")
plt.xlabel("Annual Spending ($)")
plt.ylabel("Frequency of Purchases")
plt.legend()
plt.grid()
plt.show()
```

18) A marketing company has collected data from 20 customers, including several features about their purchasing habits. They want to analyze the data in a more compact form while retaining as much information as

possible. For this, they decide to use Principal Component Analysis (PCA) to reduce the dimensionality of the dataset.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Generate synthetic dataset for 20 customers
np.random.seed(42)

data = {
    "Annual Spending ($)": np.random.randint(500, 5000, 20),
    "Frequency of Purchases": np.random.randint(1, 50, 20),
    "Number of Products Bought": np.random.randint(1, 20, 20),
    "Loyalty Score": np.random.randint(1, 100, 20)
}

df = pd.DataFrame(data)

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Apply PCA
pca = PCA(n_components=2)  # Reduce to 2 principal components
principal_components = pca.fit_transform(scaled_data)

# Create a DataFrame for the principal components
pca_df = pd.DataFrame(data=principal_components, columns=["Principal
Component 1", "Principal Component 2"])

# Explained Variance
explained_variance = pca.explained_variance_ratio_

# Add the principal components back to the original dataset
df_pca = pd.concat([df, pca_df], axis=1)

# Print the PCA results
print("Explained Variance by Each Principal Component:")
```

```
print(f"PC1: {explained_variance[0]*100:.2f}%, PC2:
{explained_variance[1]*100:.2f}%\n")
print("Dataset with Principal Components:")
print(df_pca)

# Plot the principal components
plt.figure(figsize=(10, 6))
plt.scatter(pca_df["Principal Component 1"], pca_df["Principal Component
2"], c='blue', s=50)
plt.title("PCA: Principal Component Analysis (2 Components)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid()
plt.show()
```