```
import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs


X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)


wcss = []

cluster_range = range(1, 11)


for k in cluster_range:

    kmeans = KMeans(n_clusters=k, random_state=0)

    kmeans.fit(X)

    wcss.append(kmeans.inertia_)  # WCSS for the current K

wcss_table = pd.DataFrame({'Number of Clusters (K)': cluster_range, 'WCSS': wcss})

print(wcss_table)

# Plot the Elbow graph

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

plt.plot(cluster_range, wcss, marker='o', linestyle='--', color='b')

plt.xticks(cluster_range)

plt.xlabel('Number of Clusters (K)')

plt.ylabel('WCSS')

plt.title('The Elbow Method')

plt.grid(True)

plt.show()
```

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets import make_blobs


# Step 2: Generate synthetic dataset

X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)


# Step 3: Apply K-Means clustering

kmeans = KMeans(n_clusters=4, random_state=0)

kmeans.fit(X)

y_kmeans = kmeans.predict(X)


# Visualize before clustering

plt.figure(figsize=(12, 6))


# Before clustering

plt.subplot(1, 2, 1)

plt.scatter(X[:, 0], X[:, 1], c='gray', s=50, alpha=0.6)

plt.title('Data Before Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')


# After clustering

plt.subplot(1, 2, 2)

plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_

plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.75, marker='X')

plt.title('K-Means Clustering (4 Clusters)')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')
```

```
# Show both plots

plt.tight_layout()

plt.show()
```

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

# Step 1: Generate synthetic customer data
np.random.seed(42)
data = {
    'CustomerID': range(1, 201),
    'Annual_Spend': np.random.randint(500, 5000, 200),  # Annual spending in dollars
    'Shopping_Frequency': np.random.randint(1, 20, 200),  # Frequency per month
    'Promo_Sensitivity': np.random.uniform(0, 1, 200),  # Sensitivity to promotions (0-1)
}
customer_data = pd.DataFrame(data)

# Step 2: Prepare features for clustering
features = customer_data[['Annual_Spend', 'Shopping_Frequency', 'Promo_Sensitivity']]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Step 3: Apply K-Means Clustering
kmeans = KMeans(n_clusters=4, random_state=42)
customer_data['Cluster'] = kmeans.fit_predict(scaled_features)

# Step 4: Visualize the clusters
plt.figure(figsize=(10, 6))
plt.scatter(customer_data['Annual_Spend'],                    customer_data['Shopping_Frequency'],
c=customer_data['Cluster'], cmap='viridis', s=100)
plt.scatter(kmeans.cluster_centers_[:, 0] * scaler.scale_[0] + scaler.mean_[0],
            kmeans.cluster_centers_[:, 1] * scaler.scale_[1] + scaler.mean_[1],
        c='red', marker='X', s=200, label='Centroids')

plt.title('Customer Segmentation Clusters')

plt.xlabel('Annual Spend ($)')

plt.ylabel('Shopping Frequency (per month)')

plt.legend()

plt.show()
```

```
# Step 5: Analyze the clusters
cluster_summary = customer_data.groupby('Cluster').mean()
print("Cluster Summary:")
print(cluster_summary)

# Step 6: Interpret the clusters
interpretations = {
   0: "High spenders who shop frequently",
   1: "Budget-conscious customers who shop during sales",
   2: "Infrequent shoppers who spend a lot per visit",
   3: "Bargain hunters who purchase only during promotions"
}

print("\nCluster Interpretations:")
for cluster, desc in interpretations.items():
   print(f"Cluster {cluster}: {desc}")
```

```
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.ensemble import IsolationForest
    from sklearn.preprocessing import StandardScaler
    from sklearn.metrics import classification_report

    # For reproducibility
    np.random.seed(42)

    data = {
      "Transaction_Amount": np.random.normal(100, 30, 1000).tolist() + [500, 600, 700],  # Add
    anomalies
      "Transaction_Frequency": np.random.normal(20, 5, 1000).tolist() + [1, 1, 1],    # Add anomalies
    }

    df = pd.DataFrame(data)

    # Label anomalies for evaluation (optional, only for testing)
    df['Label'] = [0] * 1000 + [1, 1, 1]  # 0 = Normal, 1 = Anomaly

    print(df.head())

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df[["Transaction_Amount", "Transaction_Frequency"]])

# Initialize and fit Isolation Forest
model = IsolationForest(contamination=0.01, random_state=42)
df['Anomaly_Score'] = model.fit_predict(scaled_data)

# Map scores to labels: -1 (anomalies) to 1 (normal)
```

```
df['Detected_Anomaly'] = (df['Anomaly_Score'] == -1).astype(int)

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

# Normal transactions
plt.scatter(
    df[df['Detected_Anomaly'] == 0]['Transaction_Amount'],
    df[df['Detected_Anomaly'] == 0]['Transaction_Frequency'],
    c='blue',
    label='Normal Transactions'
)

# Anomalous transactions
plt.scatter(
    df[df['Detected_Anomaly'] == 1]['Transaction_Amount'],
    df[df['Detected_Anomaly'] == 1]['Transaction_Frequency'],
    c='red',
    label='Anomalies'
)

plt.xlabel('Transaction Amount')
plt.ylabel('Transaction Frequency')
plt.title('Transaction Anomaly Detection')
plt.legend()
plt.show()
```