# Task Title:

Clustering and Dimensionality Reduction in Machine Learning

# Objective:

To explore and implement unsupervised learning techniques including clustering and
dimensionality reduction, using a real-world dataset.

# Dataset Used:

I used the **Wholesale Customers Dataset** for this project. This dataset contains information on the annual spending habits of wholesale clients across different product categories, including:

- **Fresh**
- **Milk**
- **Grocery**
- **Frozen**
- **Detergents_Paper**
- **Delicassen**

Additionally, the dataset includes two categorical columns: **Channel** (1 = Horeca, 2 = Retail) and **Region** (1 = Lisbon, 2 = Oporto, 3 = Other)

# Steps Performed:

## 1-Importing Libararies

I began the task by importing essential Python libraries required for data handling, preprocessing, visualization, clustering, and evaluation:

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

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA from
sklearn.manifold import TSNE
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score
```

## 2-Loading Dataset

The dataset was uploaded to **Google Colab** using the file upload feature. After uploading:
I explored the first few rows using df.head() to understand its structure.

```
df = pd.read_csv('/content/Wholesale customers data.csv')
df.head()
```

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 440,\n  \"fields\": [\
n    {\n      \"column\": \"Channel\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 2,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Region\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 1,\n
\"max\": 3,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          3,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Fresh\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 12647,\n        \"min\": 3,\n
\"max\": 112151,\n        \"num_unique_values\": 433,\n
\"samples\": [\n          21117,\n          20398\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Milk\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 7380,\n        \"min\": 55,\n
\"max\": 73498,\n        \"num_unique_values\": 421,\n
\"samples\": [\n          8384,\n          7184\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Grocery\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 9503,\n
\"min\": 3,\n        \"max\": 92780,\n        \"num_unique_values\":
430,\n        \"samples\": [\n          5160,\n          3\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Frozen\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4854,\n        \"min\": 25,\n        \"max\": 60869,\n
\"num_unique_values\": 426,\n        \"samples\": [\n          269,\n
7530\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Detergents_Paper\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4767,\n        \"min\": 3,\n
\"max\": 40827,\n        \"num_unique_values\": 417,\n
\"samples\": [\n          302,\n          6740\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Delicassen\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2820,\n        \"min\": 3,\n        \"max\": 47943,\n
\"num_unique_values\": 403,\n        \"samples\": [\n          14472,\
n          172\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}

# 3-Initial Data Exploration

To understand the dataset better, I explored the following:

- Data types and non-null entries with df.info()
- Statistical summary with df.describe()
- Checked for missing values and duplicate rows:

```
df.info() df.describe()
df.isnull().sum()
df.duplicated().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
0    Channel           440 non-null    int64
1    Region            440 non-null    int64
2    Fresh             440 non-null    int64
3    Milk              440 non-null    int64
4    Grocery           440 non-null    int64
5    Frozen            440 non-null    int64
6    Detergents_Paper  440 non-null    int64
     7   Delicassen        440 non-null
     int64
```

dtypes: int64(8) memory usage: 27.6 KB np.int64(0)

# 4-Preprocessing

```
df = df.drop_duplicates()
df_numerical = df.drop(['Channel', 'Region'], axis=1)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_numerical)
<bound method NDFrame.head of      Channel  Region  Fresh   Milk
Grocery  Frozen  Detergents_Paper  \

0          2       3  12669   9656     7561     214              2674
```

To prepare the dataset for clustering:

- I **removed duplicate entries** using `df.drop_duplicates()`
- Dropped categorical columns `Channel` and `Region` as clustering was applied on numerical data
- **Standardized the data** using `StandardScaler()` to bring all features to a similar scale, which is essential before applying PCA and clustering algorithms.

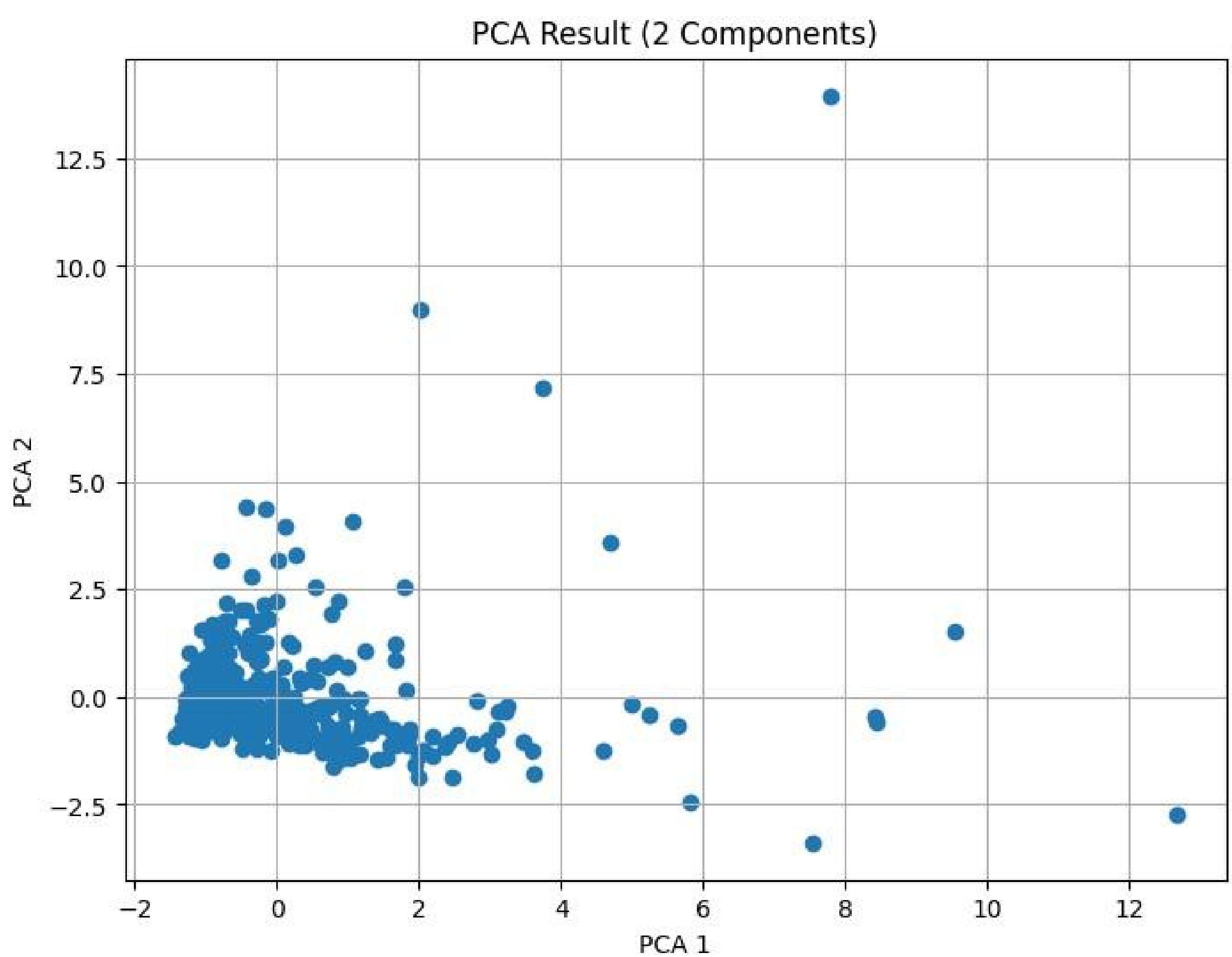
```
1     2        3   7057   9810      9568    1762                 3293

2     2        3   6353   8808      7684    2405                 3516

3     1        3  13265   1196      4221    6404                  507

4     2                    3     22615        5410              7198          3915

1777  ..         ...       ...     ...     ...        ...      ...                  ...

435   1        3  29703  12051     16027   13135                  182

436   1        3  39228   1431       764    4510                   93

437   2        3  14531  15488     30243     437                14841

438   1          3   10290    1981       2232     1038                    168   439
1        3   2787   1698      2510       65                 477
     Delicassen
0     1338
1     1776
2     7844
3     1788
4     5185  ..           ...
435          2204
436          2346
437          1867
438          2125
439          52

[440 rows x 8 columns]>
```

## 5-PCA for Dimensionality Reduction

I applied **Principal Component Analysis (PCA)** to reduce the high-dimensional dataset into 2 dimensions for easier visualization and analysis:

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_data)
print("Explained variance ratio:", pca.explained_variance_ratio_)
plt.figure(figsize=(8, 6))
plt.scatter(pca_result[:, 0], pca_result[:, 1])
plt.xlabel('PCA 1') plt.ylabel('PCA 2')
plt.title('PCA Result (2 Components)')
plt.grid(True) plt.show()
```

```
Explained variance ratio: [0.44082893 0.283764  ]
```



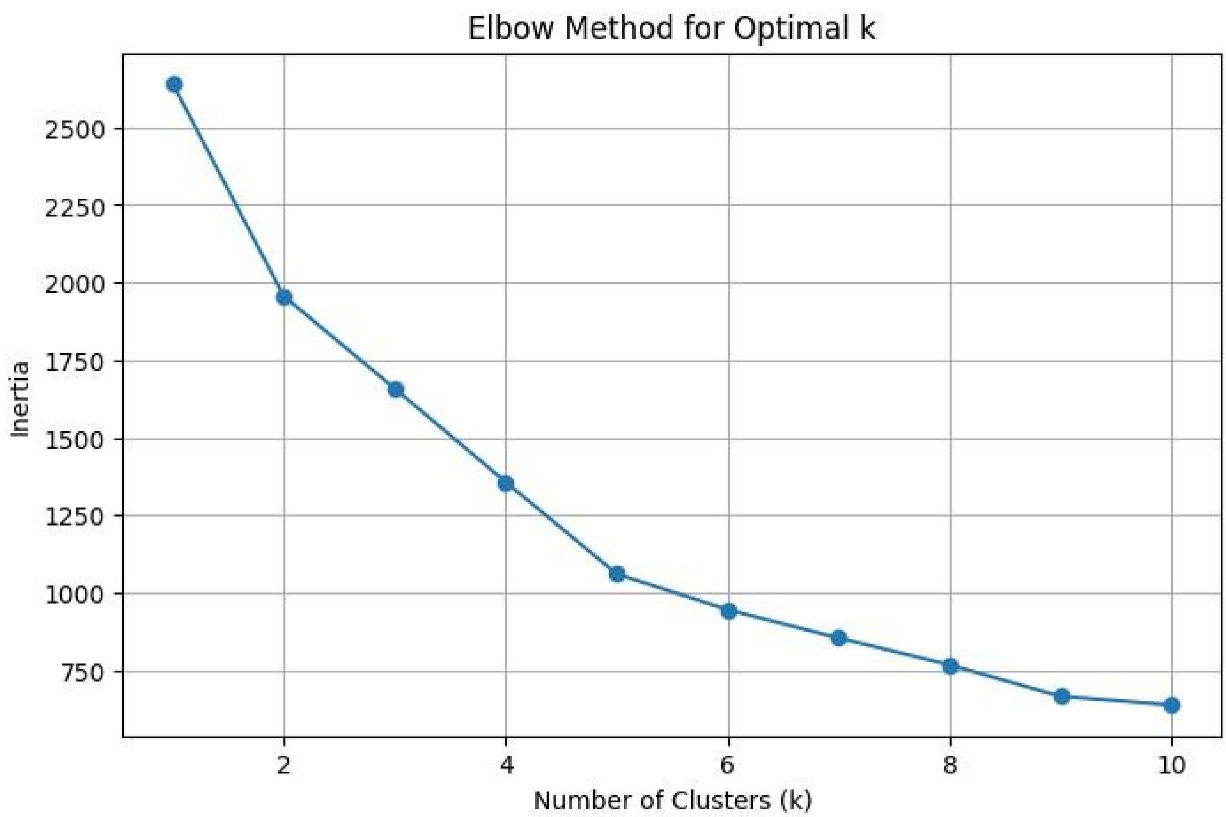
· The first two principal components explained **72.46%** of the variance.

· The reduced dataset was visualized using a **2D scatter plot**.

## 6-K-Means Clustering + Elbow Method

I used the **Elbow Method** to determine the optimal number of clusters for K-Means:

```
inertia = [] k_range = range(1, 11) for k in
k_range:     kmeans = KMeans(n_clusters=k,
random_state=0)     kmeans.fit(scaled_data)
inertia.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.grid(True) plt.show()
```

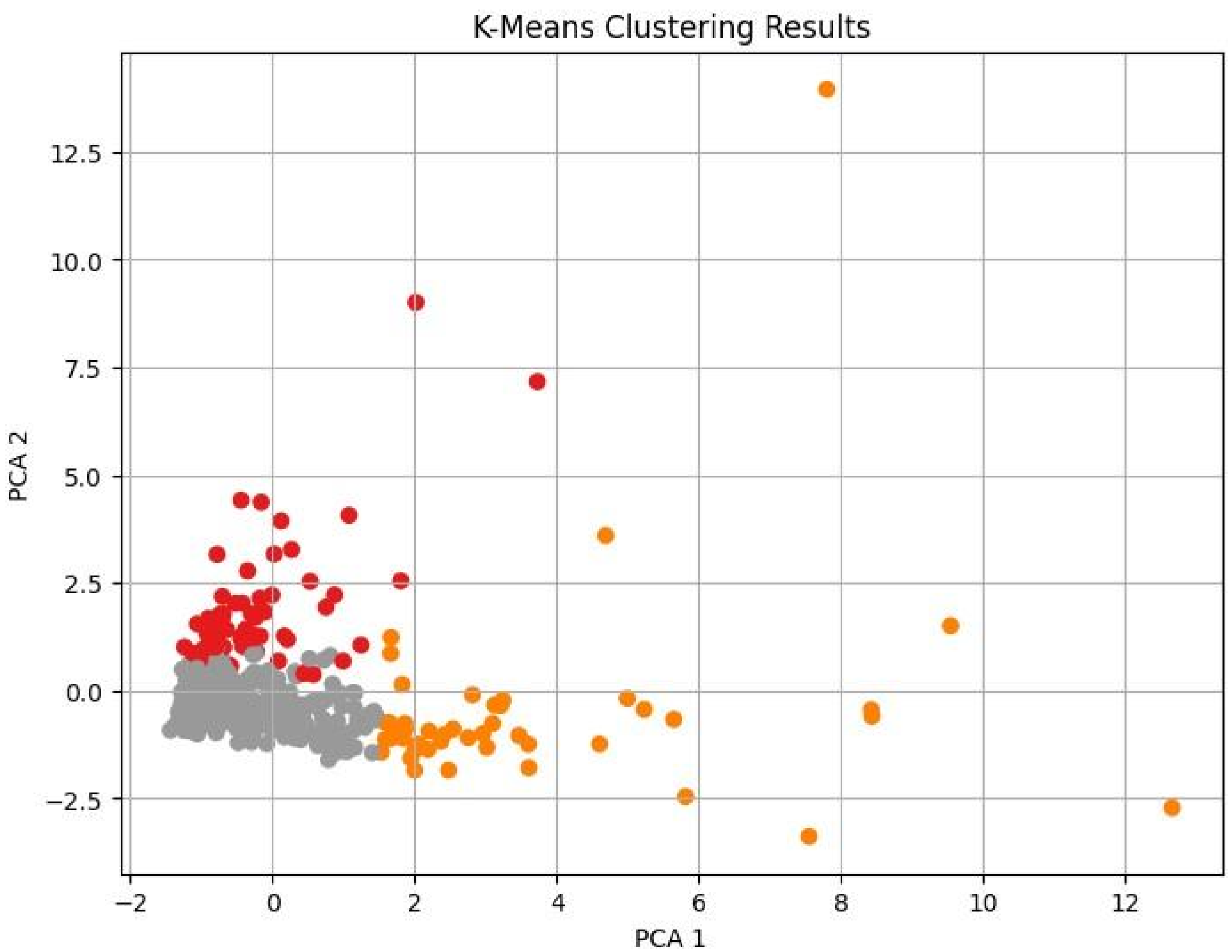


- Based on the elbow point, I chose **k = 3** clusters.

- Applied **KMeans** and visualized the clusters on the PCA-reduced data.

## 7-Apply Clustering Algorithms

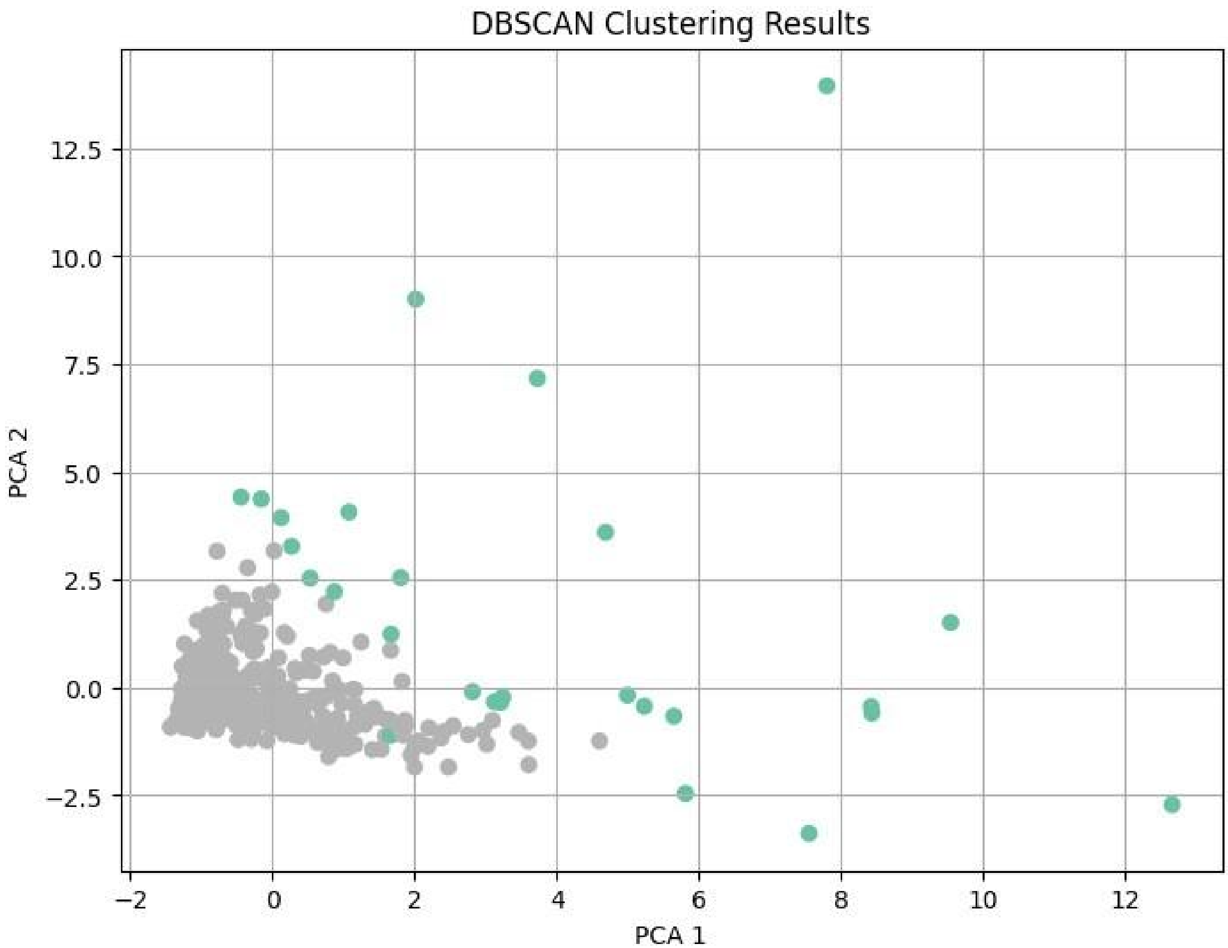
```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans_labels = kmeans.fit_predict(scaled_data)
plt.figure(figsize=(8,6))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=kmeans_labels,
cmap='Set1')
plt.title('K-Means Clustering Results')
plt.xlabel('PCA 1') plt.ylabel('PCA 2')
plt.grid(True) plt.show()
```



## 7-DBSCAN

I applied **DBSCAN (Density-Based Spatial Clustering)** as a second clustering technique:

```
dbscan = DBSCAN(eps=1.5, min_samples=5)
db_labels = dbscan.fit_predict(scaled_data)

plt.figure(figsize=(8,6))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=db_labels,
cmap='Set2')
plt.title('DBSCAN Clustering Results')
plt.xlabel('PCA 1') plt.ylabel('PCA
2') plt.grid(True) plt.show()
```



- DBSCAN was able to identify cluster structure based on density and noise.

- The clusters were visualized using the same 2D PCA-reduced plot.

## 8-Evaluation

To evaluate the quality of the clusters formed by both KMeans and DBSCAN, I calculated the **Silhouette Score** and **Davies-Bouldin Index**:

**K-Means Results:**

- **Silhouette Score:** 0.3916
- **Davies-Bouldin Index:** 1.2494

### DBSCAN Results:

- **Silhouette Score:** 0.6602
- **Davies-Bouldin Index:** 1.4808

> **Insight**: DBSCAN produced a higher silhouette score indicating better-defined clusters. However, it had a slightly higher Davies-Bouldin score. Overall, DBSCAN performed better in terms of clustering tightness and separation.

---

```
print("K-Means Silhouette Score:", silhouette_score(scaled_data,
kmeans_labels))
print("K-Means Davies-Bouldin Index:",
davies_bouldin_score(scaled_data, kmeans_labels)) if
len(set(db_labels)) > 1:     print("DBSCAN Silhouette Score:",
silhouette_score(scaled_data, db_labels))
    print("DBSCAN Davies-Bouldin Index:",
davies_bouldin_score(scaled_data, db_labels)) else:
print("DBSCAN formed only one cluster or failed to cluster.")
```

```
K-Means Silhouette Score: 0.3916016573908254
K-Means Davies-Bouldin Index: 1.2494010354240845
DBSCAN Silhouette Score: 0.6602347586576692
DBSCAN Davies-Bouldin Index: 1.480843288827392
```