Day 13 – DBSCAN Clustering (Unsupervised Learning)

📒 Today's Highlights

We learned about **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** — a clustering technique used in unsupervised learning to find groups of data points in large datasets.
Unlike K-Means or Hierarchical Clustering, **DBSCAN does not require you to specify the number of clusters**. It is also capable of identifying **noise and outliers** effectively.

What is DBSCAN?

DBSCAN groups together points that are close to each other based on a **distance measurement (usually Euclidean)** and a **minimum number of points (min_samples)**.

📌 Core Concepts:

- **eps**: The maximum distance between two samples for them to be considered as in the same neighborhood.
- **min_samples**: The number of samples in a neighborhood for a point to be considered a **core point**.
- **Core Points**, **Border Points**, and **Noise**: Based on these two parameters, DBSCAN labels points accordingly.

📁 Dataset Used: `Mall_Customers.csv`

The dataset contained details of mall customers:

- Age
- Annual Income (k$)
- Spending Score (1–100)
- Gender (converted into numeric: 0 = Male, 1 = Female)

🔧 Step-by-Step Implementation

✅ 1. Import Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
```

```
from sklearn.metrics import silhouette_score
```

✅ 2. Load and Clean Data

```
df = pd.read_csv('/content/Mall_Customers (1).csv')
df.drop(columns=['CustomerID'], inplace=True)
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
```

📊 Exploratory Data Analysis (EDA)

```
sns.scatterplot(x='Age', y='Annual Income (k$)', data=df)
plt.show()

sns.boxplot(x='Gender', y='Age', data=df)
plt.show()
```

- We visualized distribution and relationships between features like age, income, and gender.

📏 Feature Scaling

DBSCAN is distance-based, so we scaled the features between 0 and 1.

```
scaler = MinMaxScaler()
df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']] =
scaler.fit_transform(
    df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]
)
```

🔍 Choosing the Right Epsilon (eps)

We used **K-Nearest Neighbors (KNN)** to determine the best value of `eps`.

```
knn = NearestNeighbors(n_neighbors=6)
nbrs = knn.fit(df[['Age', 'Annual Income (k$)', 'Spending Score (1-
100)']])
distances, indices = nbrs.kneighbors(df[['Age', 'Annual Income (k$)',
'Spending Score (1-100)']])

# Plot sorted distances
distances = np.sort(distances, axis=0)
distances = distances[:, 1]
plt.plot(distances)
plt.show()
```

📌 The elbow in the graph helped us select **eps = 0.13**

□ Applying DBSCAN

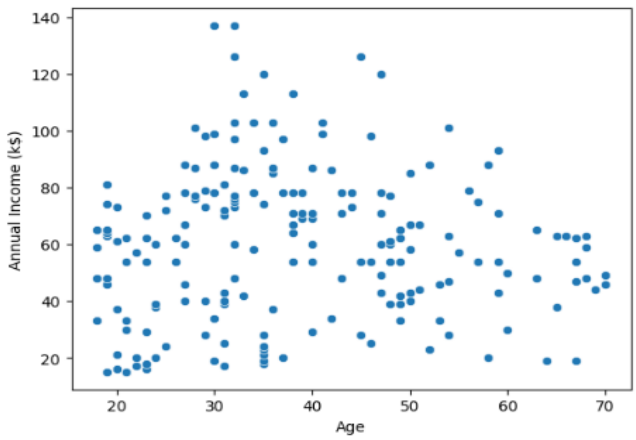
```
dbscan = DBSCAN(eps=0.13, min_samples=5, metric='euclidean')
df['cluster'] = dbscan.fit_predict(df[['Age', 'Annual Income (k$)',
'Spending Score (1-100)']])
```

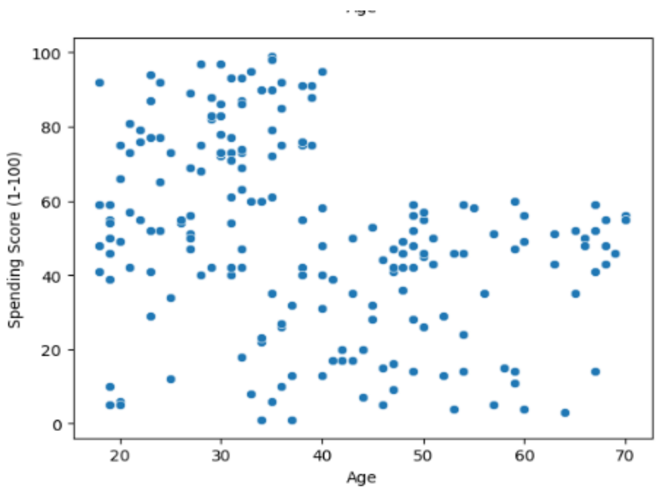
- Clusters are labeled numerically.
- Points labeled −1 are considered **noise/outliers**.

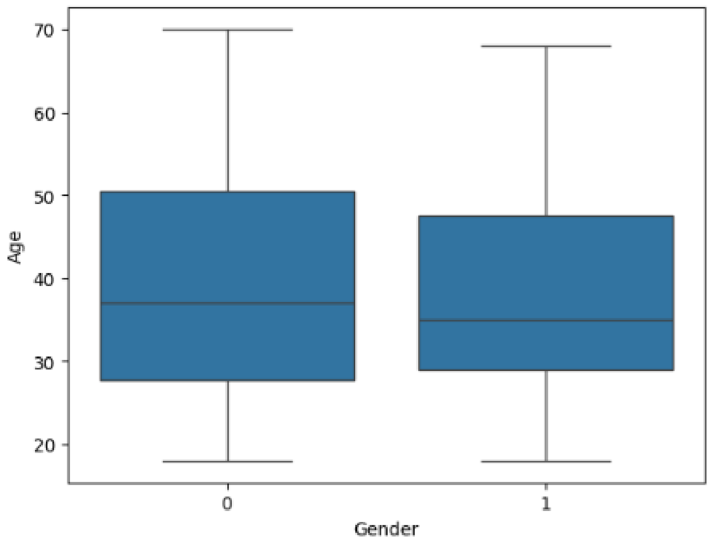
🎨 Visualizing Clusters

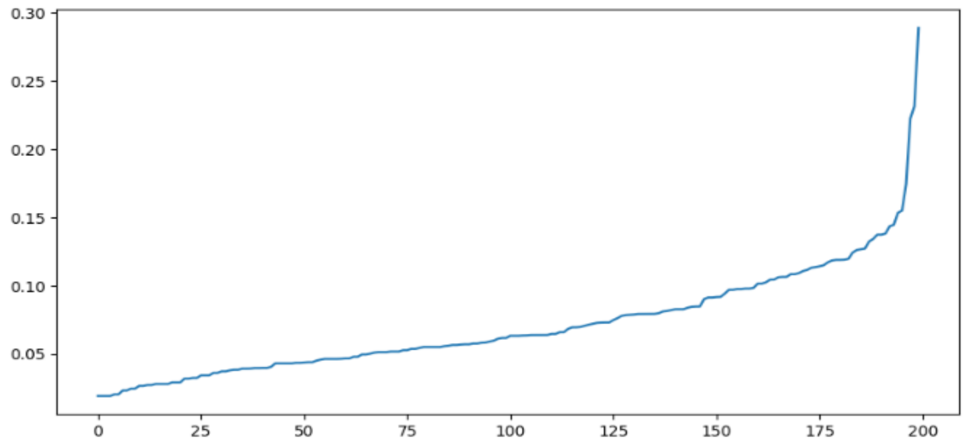
```
df_filtered = df[df['cluster'] != -1]  # Exclude noise

plt.figure(figsize=(10, 8))
sns.scatterplot(
    data=df_filtered,
    x='Annual Income (k$)', y='Spending Score (1-100)',
    hue='cluster', palette='Set2'
)
plt.title("DBSCAN Clustering Result")
plt.show()
```

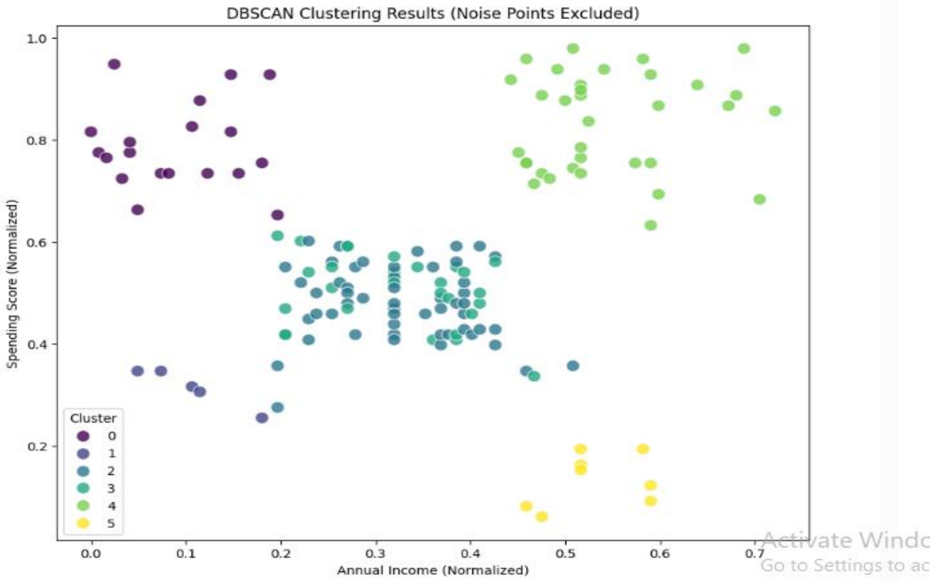
✅ <u>**Outputs**</u>

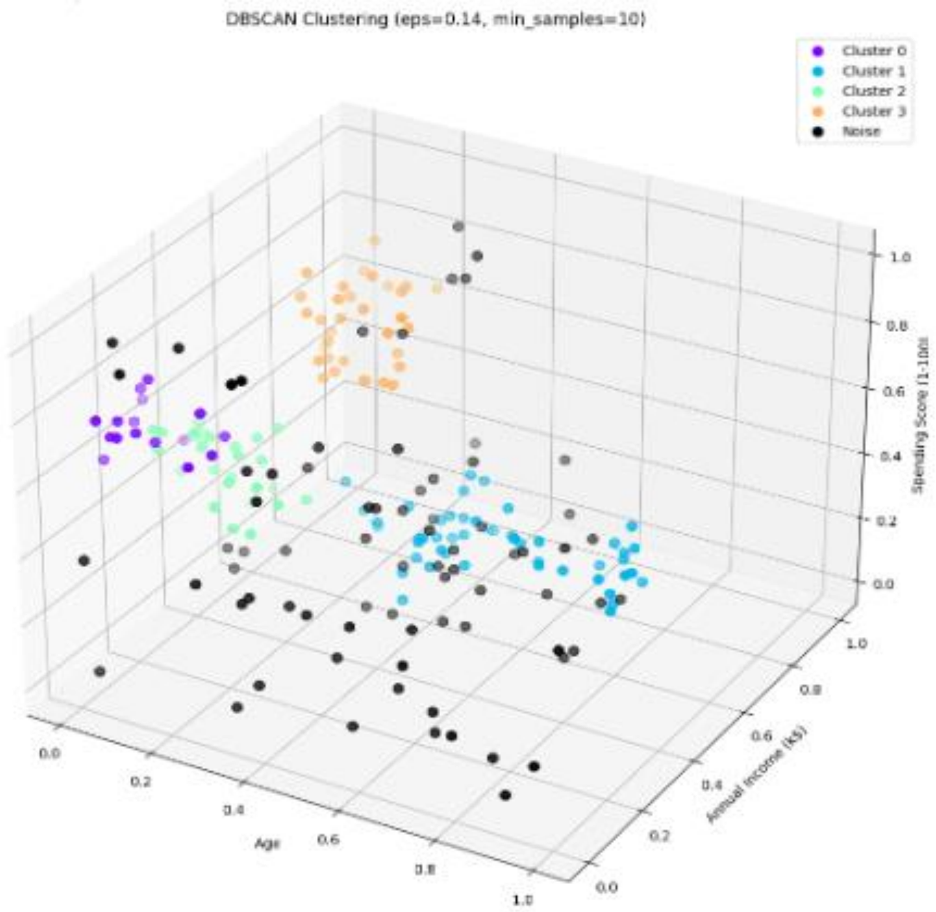







0.30
0.25
0.20
0.15
0.10
0.05
0
25
50
75
100
125
150
175
200


DBSCAN Clustering Results (Noise Points Excluded)
Spending Score (Normalized)
Annual Income (Normalized)
Cluster
0
1
2
3
4
5


DBSCAN Clustering (eps=0.14, min_samples=10)
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Noise
Age
Annual Income (k$)

✅ Results & Observations

- DBSCAN successfully grouped dense regions and excluded noisy points.
- It does not require a pre-defined number of clusters.
- It was able to **detect outliers** that other clustering methods might miss.

🔎 Sample Cluster Labels:

```
df['cluster'].value_counts()
```

📈 Performance Metric – Silhouette Score

```
score = silhouette_score(df_filtered[['Age', 'Annual Income (k$)',
'Spending Score (1-100)']],
                         df_filtered['cluster'])
print("Silhouette Score:", score)
```

A **higher silhouette score** indicates better-defined clusters.

📝 Conclusion

- Implemented **DBSCAN Clustering** using Mall Customer data.
- Used **KNN plot** to identify the best `eps` value for DBSCAN.
- Detected natural groupings and **outliers** without defining cluster count.
- Learned how **density-based clustering** is different from centroid or tree-based methods.
- Gained hands-on experience in **advanced unsupervised learning** techniques.