

**AIDS Lab Exp 07****Aim: To implement different clustering algorithms.****Problem Statement:**

- a) Clustering algorithm for unsupervised classification (K-means, density based(DBSCAN), Hierarchical clustering)
- b) Plot the cluster data and show mathematical steps.

**Theory:**

Clustering is an unsupervised machine learning technique used to group similar data points based on their characteristics. It is widely applied in various fields such as marketing, biology, and social media analytics. In this experiment, we explore three clustering techniques: K-Means, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and Hierarchical Clustering (although not implemented here). We apply these methods to a dataset containing Twitter user metrics to identify distinct groups based on their social influence (followers and friends count).

**Understanding the Dataset**

The dataset used in this experiment, **Twitter Analysis.csv**, consists of Twitter user metrics that describe user activity and influence. The key attributes considered for clustering include:

- **followers\_count**: The number of users following a particular Twitter account. This metric represents a user's influence or reach within the platform.
- **friends\_count**: The number of accounts the user follows. This reflects the user's engagement level and social connectivity.

These attributes exhibit high variance since some accounts have millions of followers, while others have only a few. This imbalance can significantly affect clustering performance, making preprocessing crucial before applying clustering algorithms.

**1. Data Preprocessing**

Before applying clustering algorithms, it is essential to preprocess the dataset to improve performance and accuracy.

**1.1 Loading the Dataset**

The dataset, **Twitter Analysis.csv**, contains attributes such as **followers\_count** and **friends\_count**, which represent a user's influence and connections on Twitter. We load this dataset into a Pandas DataFrame for further processing.

**1.2 Log Transformation**

Social media metrics often exhibit large variations, making it difficult to analyze them directly. To mitigate the impact of extreme values, we apply a log transformation:  $\log(x+1)$  where  $x$  represents the follower or friend count. This transformation reduces skewness and ensures that large values do not dominate clustering.

### 1.3 Data Standardization

Since clustering algorithms rely on distances between points, it is crucial to standardize the data to ensure equal weighting across features. We use StandardScaler from sklearn.preprocessing to normalize log\_followers and log\_friends to have zero mean and unit variance.

#### LOAD FILE ONTO GOOGLE COLAB

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

# Load dataset
df = pd.read_csv('/content/Twitter Analysis.csv')

# Apply log transformation to handle large variations
df['log_followers'] = np.log1p(df['followers_count'])
df['log_friends'] = np.log1p(df['friends_count'])

# Standardize the data to improve clustering performance
scaler = StandardScaler()
df[['scaled_followers', 'scaled_friends']] = scaler.fit_transform(df[['log_followers', 'log_friends']])
```

## 2. K-Means Clustering

K-Means is a centroid-based clustering technique that partitions data into K clusters by minimizing intra-cluster variance. It iteratively assigns points to the nearest cluster center and updates the centroids.

### 2.1 Elbow Method for Optimal K

The Elbow Method helps determine the optimal number of clusters by plotting inertia (sum of squared distances to the nearest centroid) for different values of K. The optimal K is chosen at the point where inertia starts decreasing at a slower rate, forming an "elbow."

#### ELBOW METHOD

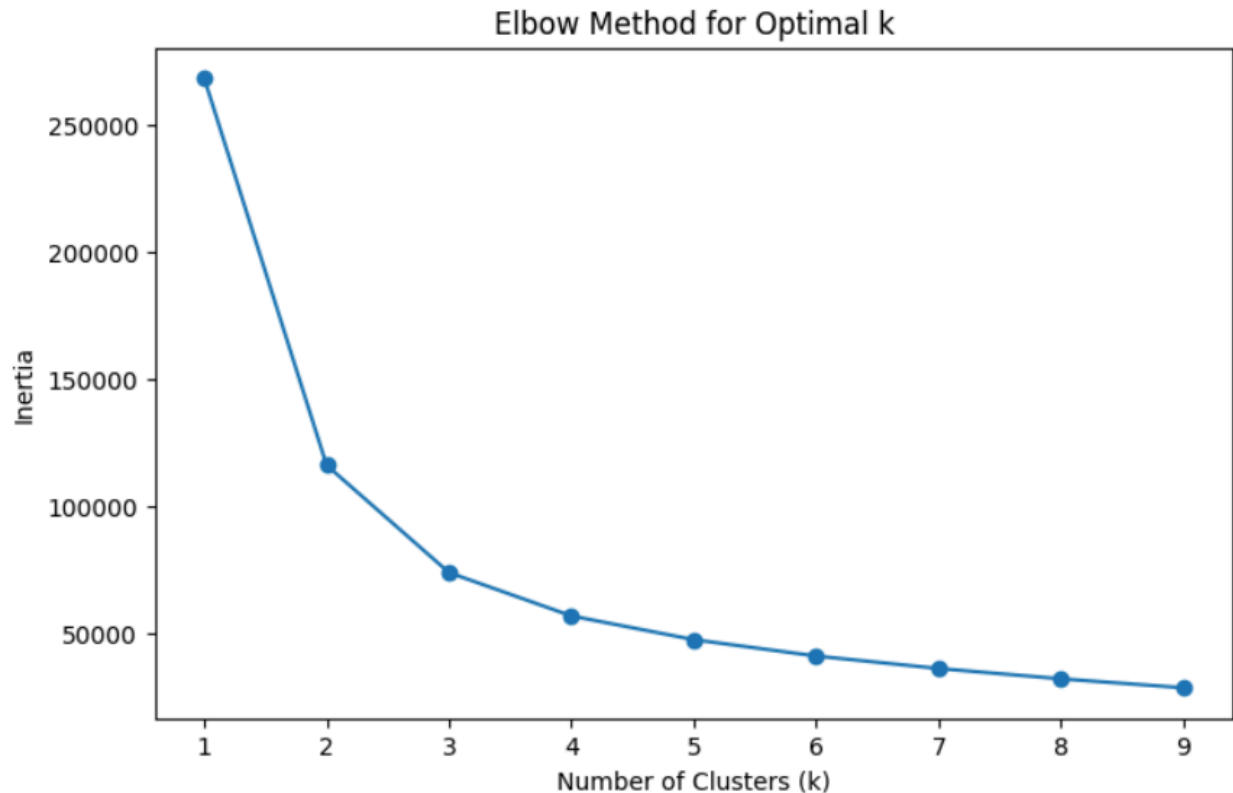
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

distortions = []
K_range = range(1, 10) # Testing for k from 1 to 10

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df[['scaled_followers', 'scaled_friends']])
    distortions.append(kmeans.inertia_)

# Plot Elbow Method
```

```
plt.figure(figsize=(8, 5))
plt.plot(K_range, distortions, marker='o', linestyle='-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```



## 2.2 Applying K-Means Clustering

Once the optimal K is selected, we apply the K-Means algorithm to cluster users based on their scaled follower and friend counts. The algorithm iteratively refines cluster assignments until convergence.

## 2.3 Visualization of K-Means Clustering

We plot the clustered data using a scatter plot, with different colors representing different clusters. This visualization helps interpret how users are grouped based on their Twitter metrics.

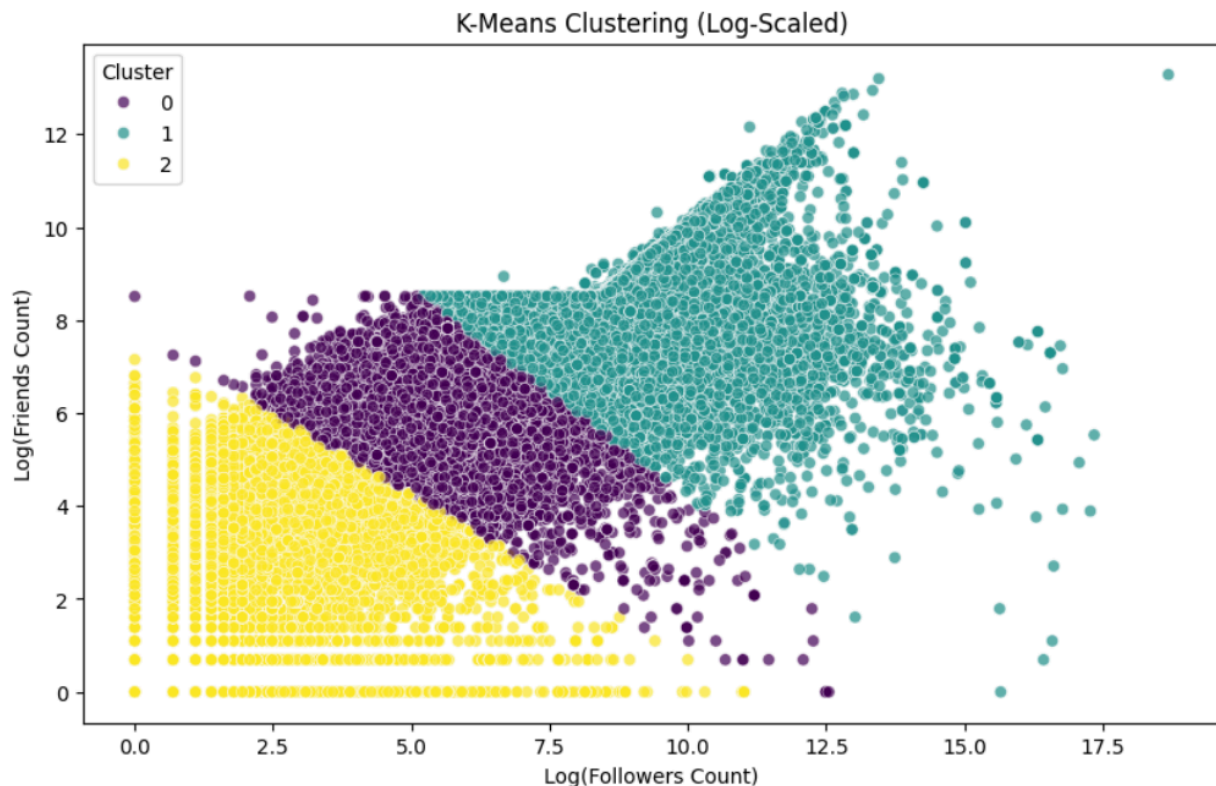
## K MEANS CLUSTERING CODE

```
# Set optimal k (based on the Elbow Method)
optimal_k = 3 # Adjust based on the elbow plot
```

```
# Apply K-Means clustering
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df['kmeans_cluster'] = kmeans.fit_predict(df[['scaled_followers', 'scaled_friends']])

import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['log_followers'],
    y=df['log_friends'],
    hue=df['kmeans_cluster'],
    palette='viridis',
    alpha=0.7
)
plt.xlabel('Log(Followers Count)')
plt.ylabel('Log(Friends Count)')
plt.title('K-Means Clustering (Log-Scaled)')
plt.legend(title="Cluster")
plt.show()
```



### 3. Density-Based Clustering (DBSCAN)

DBSCAN is a clustering algorithm that groups points based on density rather than distance. It is effective for datasets with noise and clusters of irregular shapes.

### 3.1 Applying DBSCAN

DBSCAN requires two key parameters:

- `eps`: Defines the radius of a neighborhood around a point.
- `min_samples`: Specifies the minimum number of points required to form a cluster.

We sample 5000 data points and apply DBSCAN with optimized parameters to identify clusters while filtering out noise points (assigned label -1).

### 3.2 Visualization of DBSCAN Clustering

We plot the clustered data points, excluding noise, to observe the structure of the detected clusters. Unlike K-Means, DBSCAN does not require specifying the number of clusters beforehand.

#### DBSCAN CODE

```
from sklearn.cluster import DBSCAN
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Sample a subset of data (Adjust sample size if needed)
```

```
df_sample = df[['followers_count', 'friends_count']].sample(n=5000, random_state=42)
```

```
# Apply DBSCAN with optimized parameters
```

```
dbscan = DBSCAN(eps=1000, min_samples=5)
```

```
df_sample['dbscan_cluster'] = dbscan.fit_predict(df_sample)
```

```
# Plot DBSCAN Clusters (excluding noise points)
```

```
plt.figure(figsize=(8, 5))
```

```
sns.scatterplot(data=df_sample[df_sample['dbscan_cluster'] != -1],
                x='followers_count', y='friends_count',
                hue='dbscan_cluster', palette='viridis', legend='full')
```

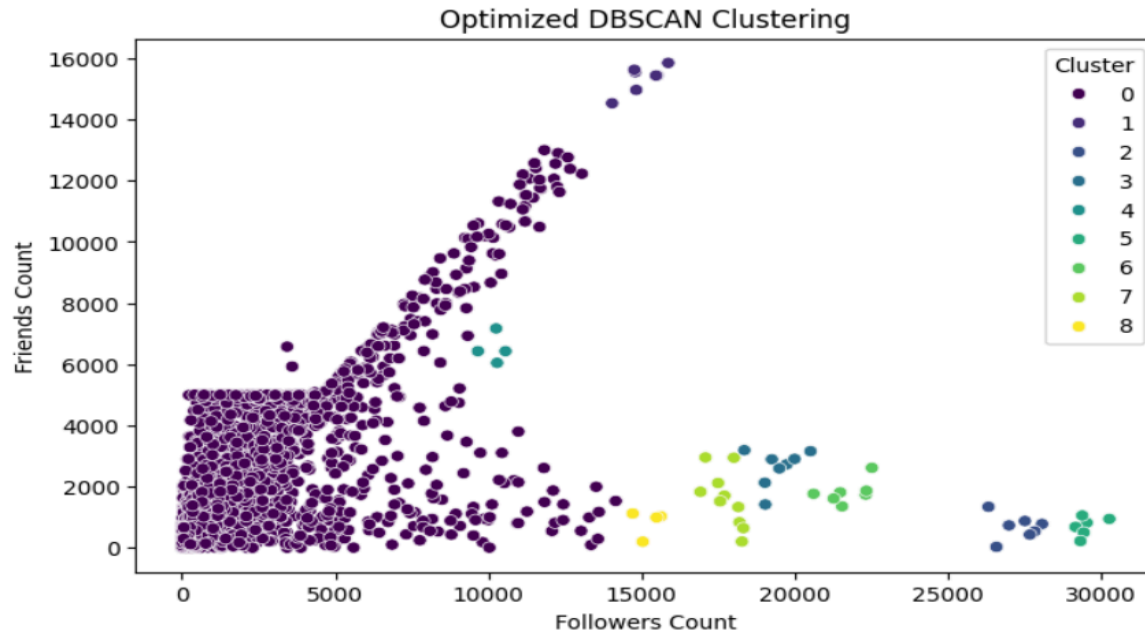
```
plt.xlabel('Followers Count')
```

```
plt.ylabel('Friends Count')
```

```
plt.title('Optimized DBSCAN Clustering')
```

```
plt.legend(title='Cluster')
```

```
plt.show()
```



#### 4. Silhouette Score for Clustering Validation

The Silhouette Score is a metric used to evaluate the quality of clustering. It measures how well-separated clusters are and is defined as:  $S = \frac{b-a}{\max(a,b)}$  where:

- $a$  is the average intra-cluster distance (cohesion)
- $b$  is the average nearest-cluster distance (separation)
- $SS$  ranges from -1 to 1, with higher values indicating better clustering.

##### 4.1 Silhouette Score for K-Means

We compute the silhouette score to assess how well-defined the K-Means clusters are. A higher score suggests distinct, well-separated clusters.

##### 4.2 Silhouette Score for DBSCAN

For DBSCAN, we compute the silhouette score only if more than one valid cluster exists (excluding noise points). If DBSCAN fails to identify meaningful clusters, the silhouette score may be low.

#### SILHOUTTE SCORE FOR K MEANS CLUSTERING

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

```
# Compute silhouette score only if clustering labels exist
```


```
if 'kmeans_cluster' in df.columns:
```

```
    score = silhouette_score(df[['scaled_followers', 'scaled_friends']], df['kmeans_cluster'])
```



```
    print(f"📊 Silhouette Score: {score:.3f}")
```

```
else:
```

```
    print("⚠️ Cluster labels missing! Ensure K-Means clustering was applied correctly.")
```

 Silhouette Score: 0.431

### SILHOUTTE SCORE FOR DBSCAN

```
if valid_clusters['dbscan_cluster'].nunique() > 1:
    score = silhouette_score(valid_clusters[['followers_count', 'friends_count']],
                             valid_clusters['dbscan_cluster'])
    print(f DBSCAN Silhouette Score: {score:.3f}')
else:
    print(f" DBSCAN did not find valid clusters or too many noise points.")
```

 DBSCAN Silhouette Score: 0.717

### Conclusion

This experiment demonstrates the application of clustering techniques to Twitter user data. K-Means clustering effectively groups users into predefined clusters based on their social influence, while DBSCAN identifies dense regions of similar users without requiring a predefined number of clusters. The Silhouette Score helps validate clustering performance. By understanding these methods, we can gain insights into user behaviors and segment social media audiences effectively.