Name: Shivpratik Hande Class:D15C/14

Experiment No 7

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 Algorithms for Unsupervised Classification

Clustering is an unsupervised machine learning technique used to group similar data points based on certain features. Below are three widely used clustering algorithms:

1. K-Means Clustering

K-Means is a centroid-based clustering algorithm that partitions data into k clusters.

Steps of K-Means Algorithm:

1. Choose the number of clusters k.
2. Initialize k cluster centroids randomly.
3. Assign each data point to the nearest centroid based on Euclidean distance.
4. Compute the new centroids as the mean of all points in each cluster.
5. Repeat steps 3 and 4 until centroids no longer change or a stopping criterion is met.

Mathematical Steps:

- Compute the distance between a point x_i and centroid C_j:

$$d(x_i, C_j) = \sqrt{\sum_{d=1}^{n} (x_{id} - C_{jd})^2}$$

- Update centroid:

$$C_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

where S_j is the set of points assigned to cluster

2. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is a density-based clustering algorithm that groups points that are closely packed together while marking outliers as noise.

Steps of DBSCAN Algorithm:

1. Select a random point P and check if it has at least MinPts neighbors within radius ε.
2. If yes, create a new cluster and expand it by adding density-reachable points.
3. If no, mark P as noise.
4. Repeat until all points are processed.

Mathematical Concepts:

- A point P is a **core point** if it has at least MinPts neighbors within ε.
- A point Q is **density-reachable** from P if $d(P,Q) \leq \varepsilon$.
- A point is **noise** if it does not belong to any cluster.

3. Hierarchical Clustering

Hierarchical clustering builds a hierarchy of clusters using either **Agglomerative (bottom-up)** or **Divisive (top-down)** approaches.

Steps of Agglomerative Clustering (Bottom-Up Approach):

1. Treat each data point as its own cluster.
2. Compute the distance between all pairs of clusters.
3. Merge the two closest clusters.
4. Repeat steps 2-3 until one cluster remains.

Mathematical Concepts:

- **Single linkage:**

$$d(A,B) = \min_{a \in A, b \in B} d(a,b)$$

- **Complete linkage:**

$$d(A,B) = \max_{a \in A, b \in B} d(a,b)$$

- **Average linkage:**

$$d(A,B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a,b)$$

Steps :

Step 1: Load and Explore Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
```

```
[ ] file_path="/content/Battery_RUL.csv"
    df=pd.read_csv(file_path)
```

```
[3] df.head()
```

	Cycle_Index	Discharge Time (s)	Decrement 3.6-3.4V (s)	Max. Voltage Dischar. (V)	Min. Voltage Charg. (V)	Time at 4.15V (s)	Time constant current (s)	Charging time (s)	RUL
0	1.0	2595.30	1151.488500	3.670	3.211	5460.001	6755.01	10777.82	1112
1	2.0	7408.64	1172.512500	4.246	3.220	5508.992	6762.02	10500.35	1111
2	3.0	7393.76	1112.992000	4.249	3.224	5508.993	6762.02	10420.38	1110
3	4.0	7385.50	1080.320667	4.250	3.225	5502.016	6762.02	10322.81	1109
4	6.0	65022.75	29813.487000	4.290	3.398	5480.992	53213.54	56699.65	1107

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15064 entries, 0 to 15063
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Cycle_Index               15064 non-null  float64
 1   Discharge Time (s)        15064 non-null  float64
 2   Decrement 3.6-3.4V (s)    15064 non-null  float64
 3   Max. Voltage Dischar. (V) 15064 non-null  float64
 4   Min. Voltage Charg. (V)   15064 non-null  float64
 5   Time at 4.15V (s)         15064 non-null  float64
 6   Time constant current (s) 15064 non-null  float64
 7   Charging time (s)         15064 non-null  float64
 8   RUL                       15064 non-null  int64
dtypes: float64(8), int64(1)
memory usage: 1.0 MB
```

In this step, the cleaned **Battery RUL dataset** containing **15,064 entries** was loaded and explored. Each entry represents a battery usage cycle and includes features such as:

- **Cycle number** (charging/discharging cycle)
- **Discharge and charging times**
- **Voltage metrics** during charge and discharge
- **Time spent at critical voltage levels**
- **Constant current duration**
- and the target variable — **Remaining Useful Life (RUL)**.

Feature Extraction

```
[ ]  features = ["Discharge Time (s)", "Decrement 3.6-3.4V (s)", "Charging time (s)"]
     X = df[features]
```

A subset of features Discharge Time, Voltage Decrement (3.6–3.4V), and Charging Time was selected to form the input dataset X. These features were chosen to group battery cycles based on similar performance patterns using clustering algorithms.

Printing Missing Values

```
X.isnull().sum()
```

	0
Discharge Time (s)	0
Decrement 3.6-3.4V (s)	0
Charging time (s)	0

dtype: int64

No missing values were found across Features.

Step 2: Normalizing using StandardScaler

normalization using standardscaler

```
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
```

To prepare the data for clustering, features were normalized using `StandardScaler`. This standardization scales the features to have a **mean of 0** and **standard deviation of 1**, ensuring all features contribute equally to distance-based clustering algorithms.

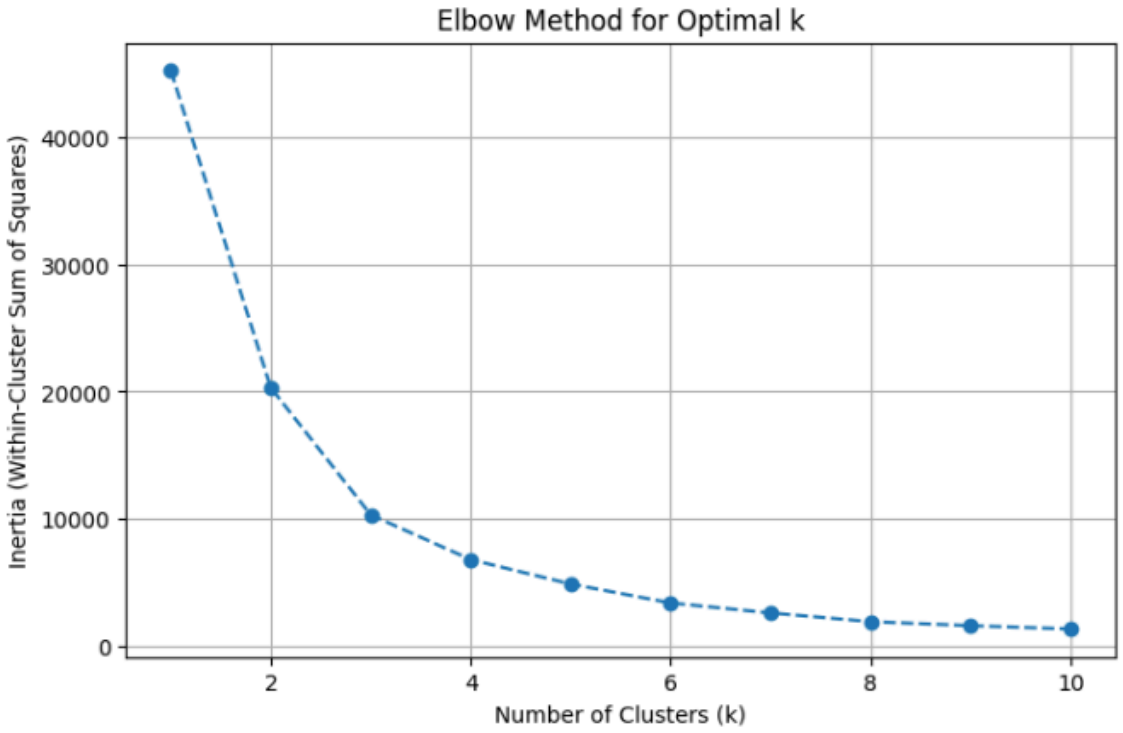
Step 3: K-Means Clustering

Elbow Curve

```
inertia =[]
k_values=range(1,11)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Within-Cluster Sum of Squares)')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.show()
```

The Elbow Method was used to determine the optimal number of clusters (k) for K-Means. The model was trained for values of k from 1 to 10, and the inertia (sum of squared distances to the nearest cluster center) was recorded for each. A plot was generated to visualize the "elbow point," where adding more clusters yields diminishing returns helping to identify the best k. **(Here k =3, since we are classifying into healthy, moderately, worn-out)**



Before clustering we have applied PCA (Principal Component Analysis)

pca

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

To visualize clusters effectively, the feature space was reduced from 3 dimensions to 2 using Principal Component Analysis (PCA). This transformation retains the most significant variance while allowing easy plotting of clusters in 2D.

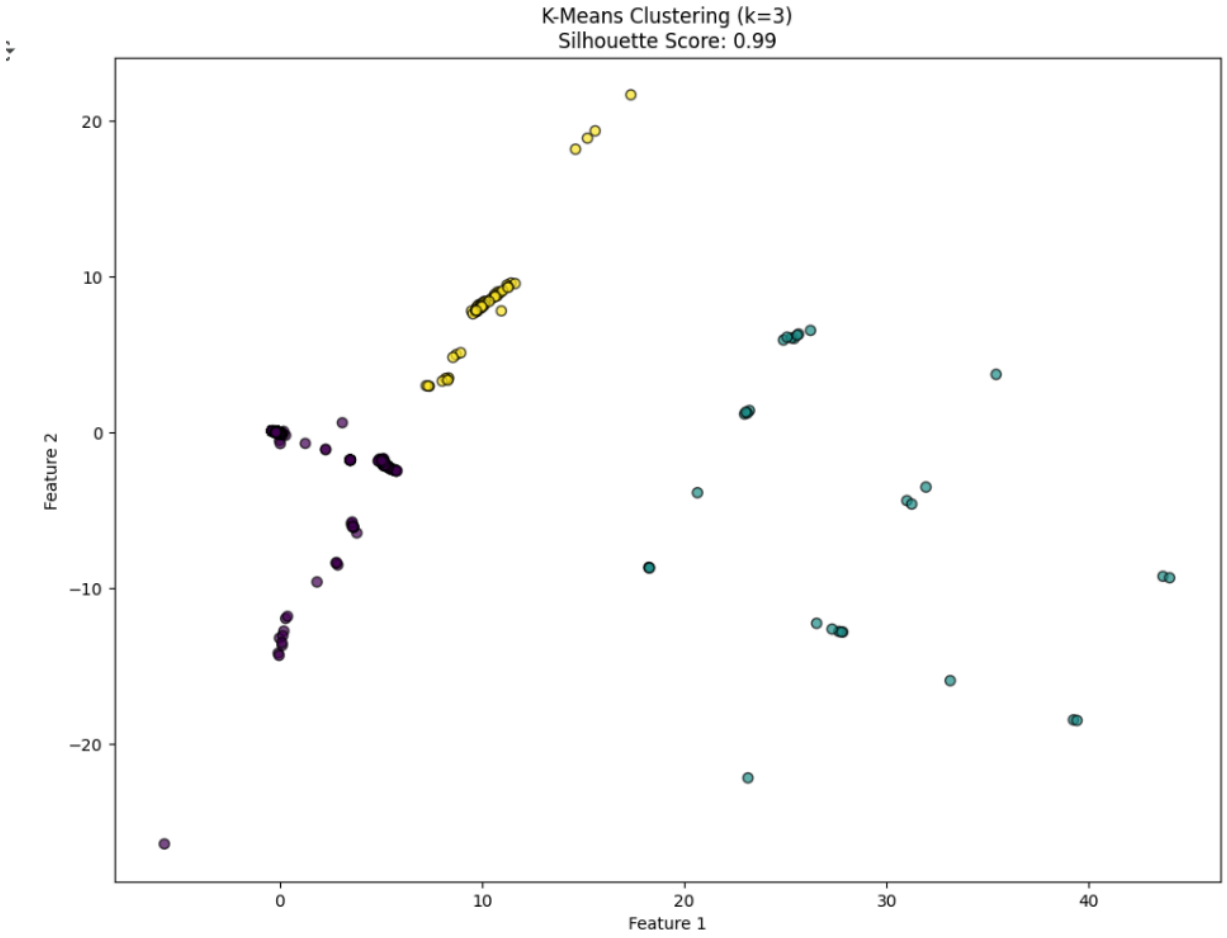
Code to Apply K-Means

```
optimal_k=3
kmeans= KMeans(n_clusters=optimal_k,random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(X_scaled)

kmeans_silhouette = silhouette_score(X_scaled, kmeans_labels)

#plot graph
plt.figure(figsize=(12, 9))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis', edgecolors='k', alpha=0.7)
plt.title(f'K-Means Clustering (k={optimal_k})\nSilhouette Score: {kmeans_silhouette:.2f}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Name: Shivpratik Hande Class:D15C/14



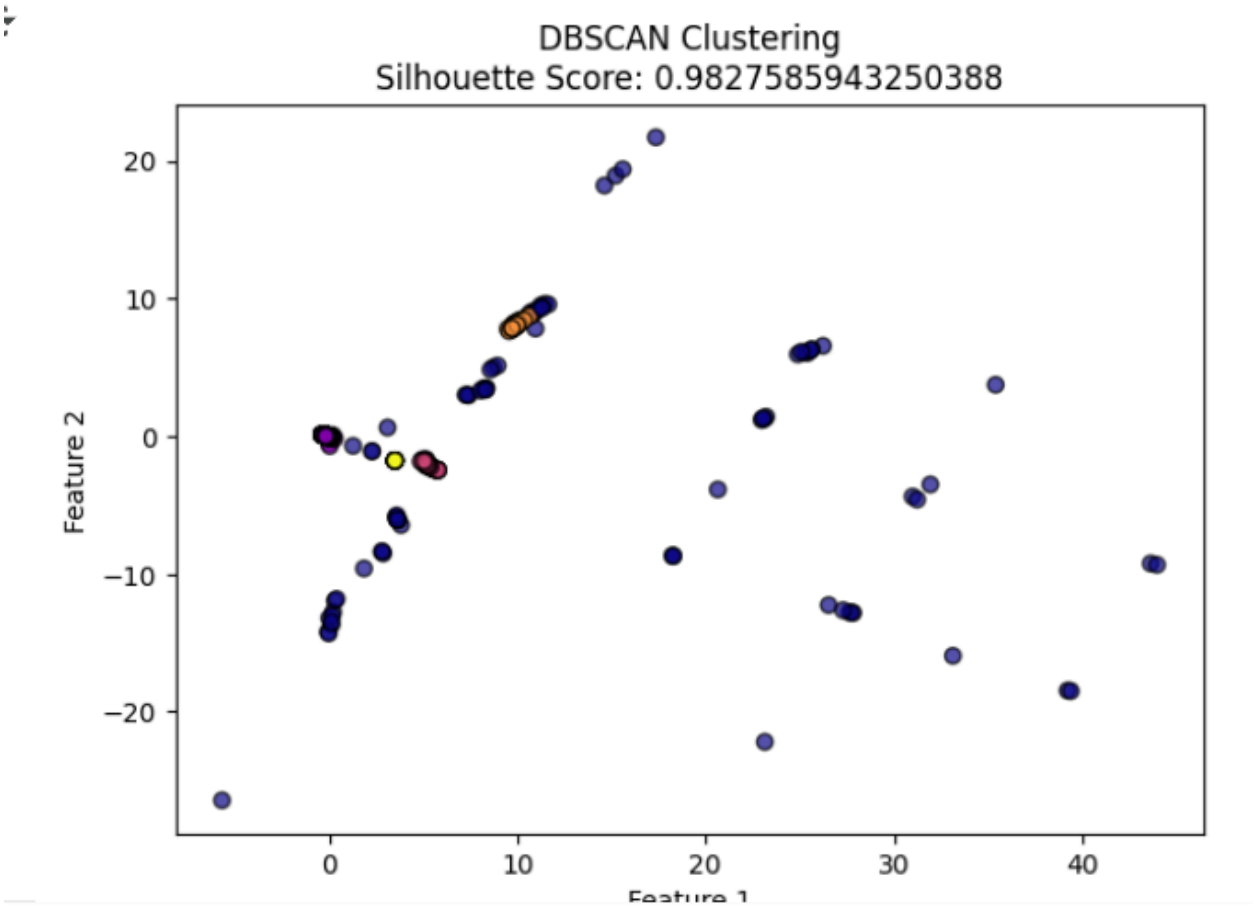
K-Means was applied with K=3 (from the elbow method) to segment Battery. The data points are grouped into three well-defined clusters, as indicated by the distinct color separation. The high Silhouette Score of 0.99 signifies excellent clustering quality, with strong intra-cluster similarity and clear inter-cluster separation. This suggests the underlying battery data forms three distinct behavioral or health patterns that are effectively captured through PCA and K-Means.

Step 4: DBSCAN Clustering

```
dbscan = DBSCAN(eps=0.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)

dbscan_silhouette = silhouette_score(X_scaled, dbscan_labels) if len(set(dbscan_labels)) > 1 else "N/A"

plt.figure(figsize=(7, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels, cmap='plasma', edgecolors='k', alpha=0.7)
plt.title(f'DBSCAN Clustering\nSilhouette Score: {dbscan_silhouette}')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



he DBSCAN clustering result demonstrates highly effective separation of data points into distinct clusters, as reflected by the impressive Silhouette Score of 0.9828. This score, which approaches the maximum value of 1, indicates that the clusters are both compact and well-separated, meaning the chosen eps (epsilon) and min_samples parameters are well-tuned for this dataset. The scatter plot, which visualizes the PCA-reduced data in two dimensions, clearly shows multiple dense groupings of points with minimal overlap, further confirming the clustering quality. Some points that do not belong to any dense region are likely treated as noise by DBSCAN, which helps maintain the integrity of clusters.

```
print(f'K-Means Silhouette Score: {kmeans_silhouette}')
print(f'DBSCAN Silhouette Score: {dbscan_silhouette}')
```

```
K-Means Silhouette Score: 0.9884934288567938
DBSCAN Silhouette Score: 0.9827585943250388
```

Both K-Means and DBSCAN achieved high Silhouette Scores, with K-Means slightly outperforming at 0.9885 compared to DBSCAN's 0.9828. While K-Means forms well-defined spherical clusters, DBSCAN effectively handles noise and non-linear cluster shapes. Despite the slight difference, both methods show excellent clustering performance on the dataset.

Conclusion:

In this project, we successfully implemented and analyzed **three clustering algorithms—K-Means**, and **DBSCAN** to perform **unsupervised classification**. Each algorithm was applied to a cleaned dataset and visualized using scatter plots and dendrograms.

- **K-Means** grouped data based on centroid distance, requiring the number of clusters as input.
- **DBSCAN** automatically identified clusters based on data density and detected outliers effectively.

These methods demonstrated different strengths: K-Means is simple and fast, Hierarchical provides insight into data hierarchy, and DBSCAN handles noise and varying densities well. Overall, clustering proved to be a powerful tool for discovering hidden patterns in unlabeled data.