Experiment 7

Aim: To implement different clustering algorithms.
Problem statement:
a) Clustering algorithm for unsupervised classification (K-means, density based (DBSCAN))
b) Plot the cluster data and show mathematical steps.

Theory:

CLUSTERING

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.

Applications of Clustering in different fields

1. Marketing: It can be used to characterize & discover customer segments for marketing purposes.

2. Biology: It can be used for classification among different species of plants and animals.

3. Libraries: It is used in clustering different books on the basis of topics and information.

4. Insurance: It is used to acknowledge the customers, their policies and identifying the frauds.

5. City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.

6. Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones

Clustering Algorithms

When choosing a clustering algorithm, you should consider whether the algorithm scales to your dataset. Datasets in machine learning can have millions of examples, but not all clustering algorithms scale efficiently. Many clustering algorithms work by computing the similarity between all pairs of examples. This means their runtime increases as the square of the number of examples n, denoted as O(n2) in complexity notation. O(n2) algorithms are not practical when the number of examples are in millions.

1. Density-Based Methods:

These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.

2. Hierarchical Based Methods:

The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

1. Agglomerative (bottom-up approach) 2.

Divisive (top-down approach)

examples CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies), etc.

3. Partitioning Methods:

These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.

4. Grid-based Methods:

In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects, for example STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest), etc.

Dataset Description: adult.csv

The **Adult Income dataset** contains demographic data of individuals, sourced from the 1994 US Census. It includes features like **age**, **education**, **occupation**, **hours per week**, etc., and aims to predict whether a person earns **more than $50K or not** annually. This is a classic dataset used for **classification tasks** in machine learning.

```
[1] import pandas as pd

    file_path = "adult.csv"
    df = pd.read_csv(file_path)

    df.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-States	<=50K

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

```
[2] df.dtypes
```

	0
age	int64
workclass	object
fnlwgt	int64
education	object
education.num	int64
marital.status	object
occupation	object
relationship	object
race	object
sex	object
capital.gain	int64
capital.loss	int64
hours.per.week	int64
native.country	object

Here we are going to see implementation of K-means and DB-SCAN clustering algorithms.

1) K-means clustering

1. Objective

To group data points into distinct clusters based on feature similarity using the K-Means algorithm.

2. Why the Elbow Method?

The **Elbow Method** helps determine the **optimal number of clusters (k)** by plotting:

- **X-axis**: Number of clusters (k)
- **Y-axis**: Within-Cluster Sum of Squares (WCSS / Inertia)

We select the **"elbow point"** – where the decrease in WCSS slows down – as the best k.

Now lets see the actual implementation.

```
# we are selecting 2 columns....hrs per week and age for kmeans clusterization
data = df[['age', 'hours.per.week']]
```

```
[4] # missing values with the median..not mode or mean
data = data.fillna(data.median())
```

```
[5] from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

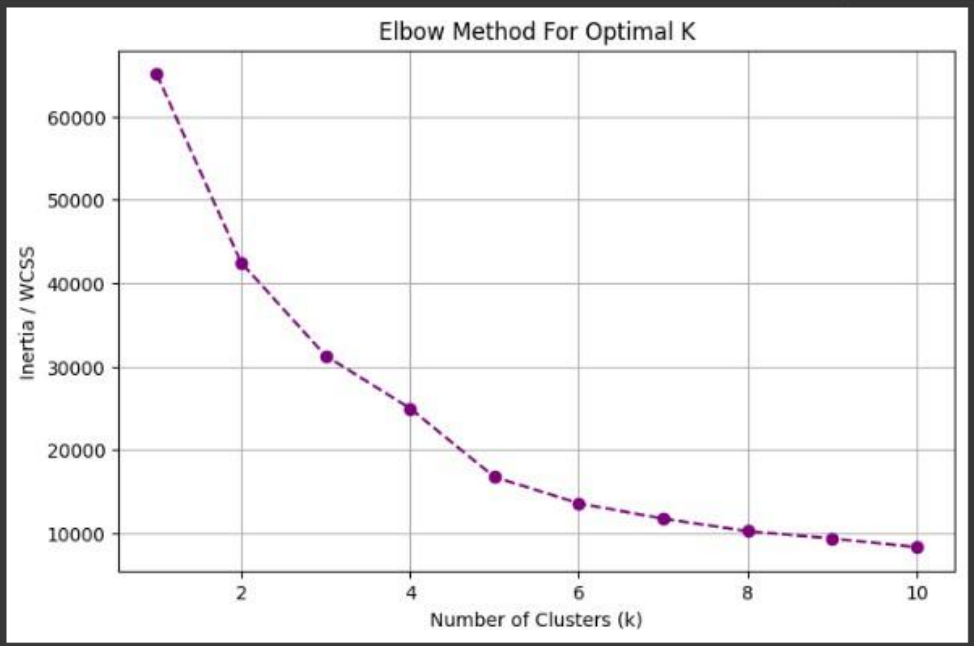
We are primarily selecting 2 columns on which our clustering will be based i.e Age and Working hours per week.Filling the missing values with median here.

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

# Elbow method to find the best value for k
inertia = []  # to store WCSS (Within-Cluster-Sum-of-Squares)

# Try from k=1 to k=10
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)  # use scaled data
    inertia.append(kmeans.inertia_)

# Plotting the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--', color='purple')
plt.title('Elbow Method For Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia / WCSS')
plt.grid(True)
plt.show()
```



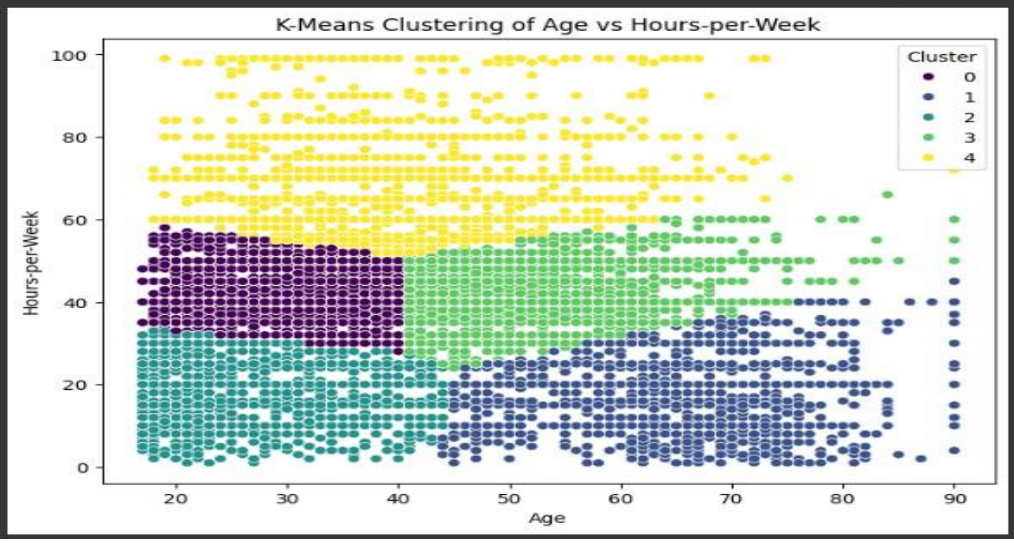
Using the elbow method to find the optimal number clusters(k) that we need . We have chosen k=5.

```python
from sklearn.cluster import KMeans

# using 5 clusters for now
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_data)
```

```python
#visualizing
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,6))
sns.scatterplot(x=df['age'], y=df['hours.per.week'], hue=df['Cluster'], palette='viridis')
plt.title('K-Means Clustering of Age vs Hours-per-Week')
plt.xlabel('Age')
plt.ylabel('Hours-per-Week')
plt.show()
```



The **elbow point** (e.g., at k = 5) indicates that 3 clusters give a good balance between **model accuracy** and **simplicity**.
Each data point is assigned to the nearest cluster based on **Euclidean distance**. The result reveals **natural groupings** or patterns in the dataset.

Silhouette Score

- **Purpose: Measures how well each data point fits within its assigned cluster compared to other clusters.**
- **Range: -1 to +1**
    - **+1 → Perfect clustering**
    - **0 → Overlapping clusters**
    - **Negative → Misclassified points**

```
from sklearn.metrics import silhouette_score

# Assuming `X_scaled` is your feature data and `kmeans.labels_` has your cluster labels
score = silhouette_score(scaled_data, kmeans.labels_)
print("Silhouette Score:", score)
```

```
Silhouette Score: 0.4413614980396684
```

Double-click (or enter) to edit

```
from sklearn.metrics import davies_bouldin_score

# Make sure these are defined:
# X_scaled → Scaled feature data (used to fit KMeans)
# kmeans.labels_ → Labels assigned by KMeans

# Compute Davies-Bouldin Score
db_score_kmeans = davies_bouldin_score(scaled_data, kmeans.labels_)
print("Davies-Bouldin Score (K-Means):", db_score_kmeans)
```

```
Davies-Bouldin Score (K-Means): 0.7348734337435234
```

Double-click (or enter) to edit

The **Davies-Bouldin Score** obtained is **0.73**, which is **fairly low** and indicates that the clusters are **compact** (low intra-cluster distance) and **well-separated** (high inter-cluster distance). This suggests that **K-Means has performed reasonably well** in forming distinct clusters. Additionally, if your **Silhouette Score** is also high (close to 1), it further confirms that the clustering is effective.

2) DB-SCAN

Feature	Description
Type	**Density-based** clustering algorithm
Assumption	Clusters are **dense regions** separated by low-density areas

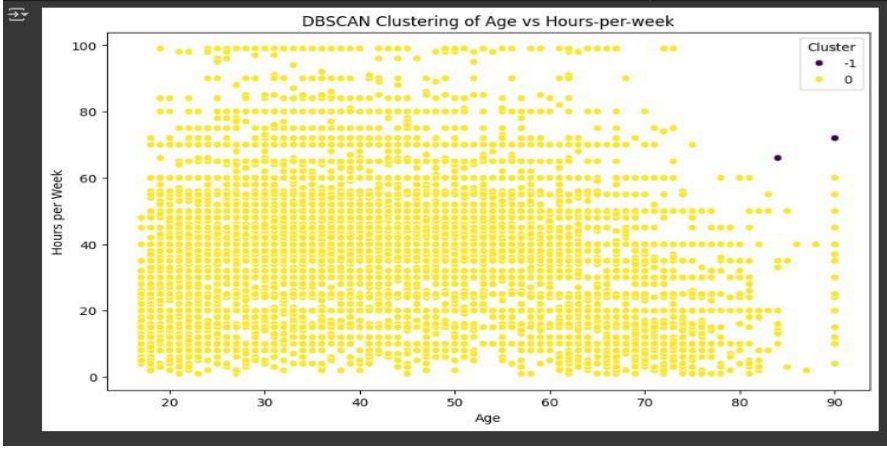
Input Required	eps (radius), min_samples (minimum points in a dense region)
How It Works	Groups points closely packed together (high density), and labels others as **noise**
Performance	Slower for large, high-dimensional data
Handles Noise	☑ Yes (labels outliers as noise)
Shape of Clusters	Can handle **arbitrary shapes** (not just circular)
Scalability	Moderate; better for smaller or medium datasets
Use Case	When clusters have **irregular shapes** or you expect **noise/outliers**

Implementation:

```
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Adjust eps and min_samples based on data
clusters = dbscan.fit_predict(df_selected_scaled)

# Add cluster labels to DataFrame
df_selected["Cluster"] = clusters
```

```
[11] plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_selected["age"], y=df_selected["hours.per.week"], hue=df_selected["Cluster"], palette="viridis")
plt.title("DBSCAN Clustering of Age vs Hours-per-week")
plt.xlabel("Age")
plt.ylabel("Hours per Week")
plt.legend(title="Cluster")
plt.show()
```



Age (X-axis)
Hours-per-week (Y-axis)

Points within **0.5 distance** of each other are considered neighbors
A point needs **at least 5 neighbors** to be a **core point**

Most points belong to Cluster 0 (yellow):

- This means **almost the entire dataset** is treated as **one dense cluster**.
- So people across all ages and working hour ranges generally fall into the same cluster.

A few outliers (Cluster -1, purple dots):

- These are individuals whose Age and Hours-per-week values are **unusual compared to others**.
- Example: A 90-year-old working 70+ hours — rare, hence considered noise.

Conclusion:

In this experiment, we implemented and compared two popular unsupervised clustering algorithms: **K-Means** and **DBSCAN**, using the adult.csv dataset.

- **K-Means** clustering required us to select the number of clusters (k) using the **Elbow Method**, which helped in identifying the optimal k by observing the point where WCSS started to level off.
- **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) automatically detected clusters based on data density, without requiring k. It also identified outliers (labeled as -1), which K-Means cannot do.

This comparison highlighted the strengths of both algorithms — **K-Means** works well for well-separated spherical clusters, while **DBSCAN** is more robust in handling **noise and arbitrary-shaped clusters**, making it suitable for more complex data distributions.