PRACTICAL NO. 05

Student Name : Samuel	**UID : 24MAI10018**
Branch : ME - CSE (AI & ML)	**Section : 24MAI – 1**
Semester : 2nd	**Date of Perf. :**
Subject : Machine Learning Lab	**Subject Code : 24CSH - 667**

AIM ::

Implementation of K – Means Algorithm using Python.

SOFTWARE REQUIREMENT ::

- Windows 11
- Python IDE (Anaconda Distributor)
- Jupyter Notebook
- Python Libraries (NumPy, Pandas, Matplotlib).

THEORY ::

UNSUPERVISED LEARNING ::

Unsupervised learning is a type of machine learning where the model is trained on data that is neither labelled nor categorized. The system tries to learn the underlying patterns and structure from the input data. Let's break it down further:

KEY CHARACTERISTICS OF UNSUPERVISED LEARNING:

1. No Labelled Data:
   - The training data consists of input features without any associated output labels.
   - The goal is to discover hidden patterns or intrinsic structures within the data.

2. Exploratory:
   - Used for exploratory data analysis to identify patterns, groupings, or anomalies.
   - Commonly applied in clustering, dimensionality reduction, and association problems.

<u>COMMON ALGORITHMS IN UNSUPERVISED LEARNING:</u>

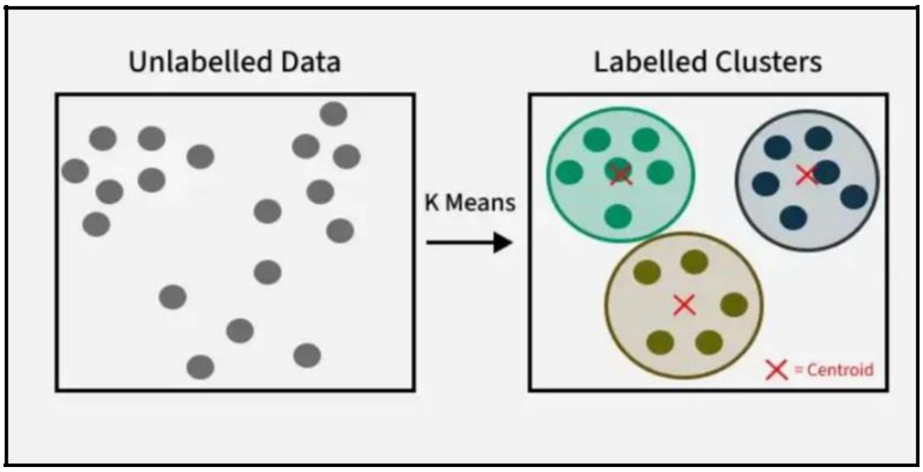
1. Clustering:
   - K-Means Clustering: Partitions the data into K distinct clusters based on similarity.
   - Hierarchical Clustering: Builds a hierarchy of clusters by either merging or splitting them.
   - DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Forms clusters based on the density of points.
2. Dimensionality Reduction:
   - Principal Component Analysis (PCA): Reduces the dimensionality of the data while preserving as much variance as possible.
   - t-SNE (t-Distributed Stochastic Neighbour Embedding): Reduces high-dimensional data to two or three dimensions for visualization.
   - Autoencoders: Neural networks used to learn compressed representations of data.
3. Association Rule Learning:
   - Apriori Algorithm: Identifies frequent item sets and generates association rules.
   - Eclat Algorithm: Uses a depth-first search approach to discover frequent item sets.

K-MEANS CLUSTERING ::

K-Means Clustering is a popular unsupervised machine learning algorithm used for partitioning data into K distinct clusters based on similarity. Here's an overview of how K-Means Clustering works, along with an example:

STEPS OF K-MEANS CLUSTERING

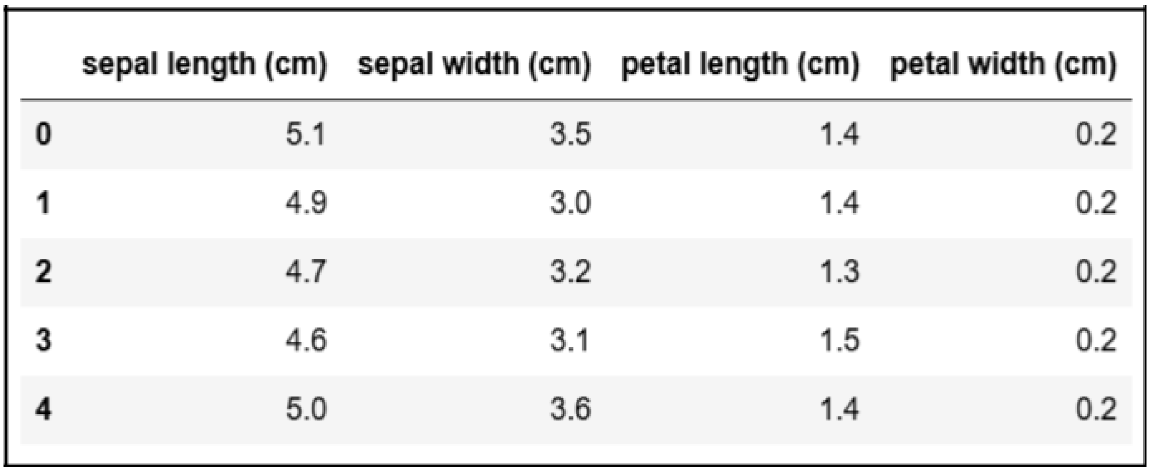
1. Initialization:
   - Choose the number of clusters K.
   - Initialize K centroids randomly. These centroids are the initial center points of the clusters.
2. Assignment Step:
   - Assign each data point to the nearest centroid based on the Euclidean distance.
   - This forms K clusters, with each data point belonging to the cluster with the closest centroid.
3. Update Step:
   - Recalculate the centroids by taking the mean of all data points assigned to each cluster.
   - The new centroids are the updated center points of the clusters.
4. Repeat:
   - Repeat the Assignment and Update steps until the centroids no longer change or change very



little between iterations (i.e., the algorithm converges).

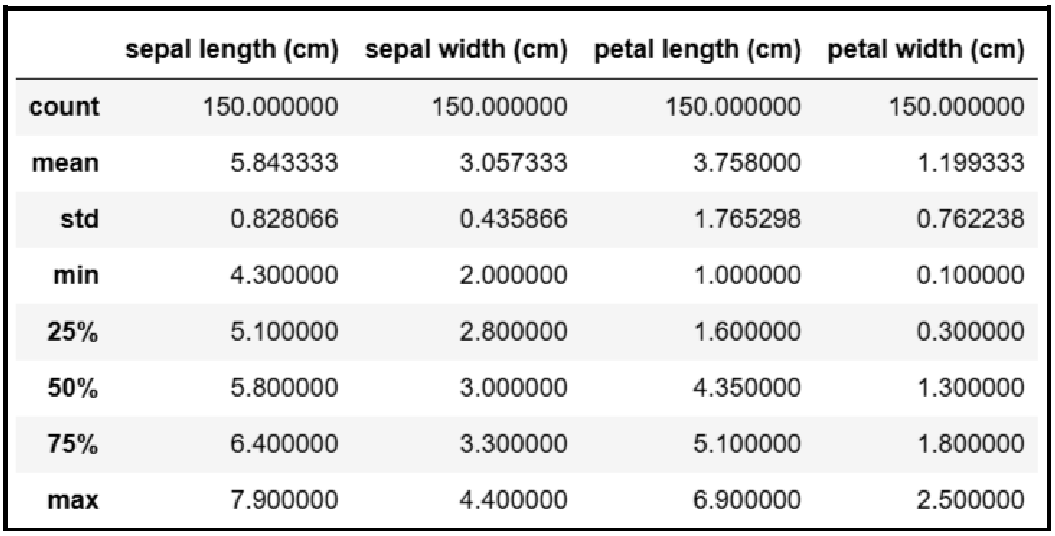
SOURCE CODE ::

```
#import the necessary libraries,
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

#Load the Iris dataset from scikit-learn's built-in datasets.
from sklearn.datasets import load_iris
iris = load_iris()

#Create a dataframe from the iris data
iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)

#Use the head method to see the top of the dataframe.
iris_df.head()
```



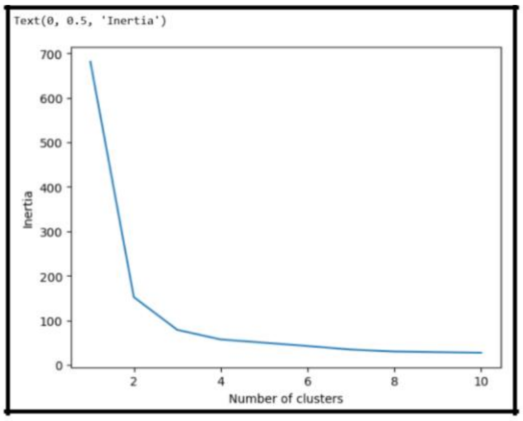
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
#Use the describe method on the DataFrame to get a summary of the data.
iris_df.describe()
```



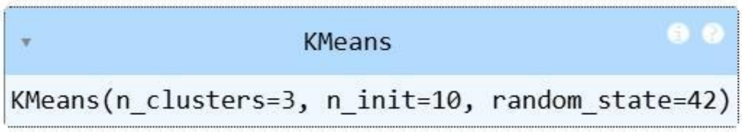
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
#Determine the optimal number of clusters using the elbow method
# Make a list of inertia for each number of clusters (1-
10) inertia = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, random_state=0)
    km.fit(iris_df)
    inertia.append(km.inertia_)
# Plot the inertia for each number of clusters
plt.plot(range(1, 11), inertia)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```



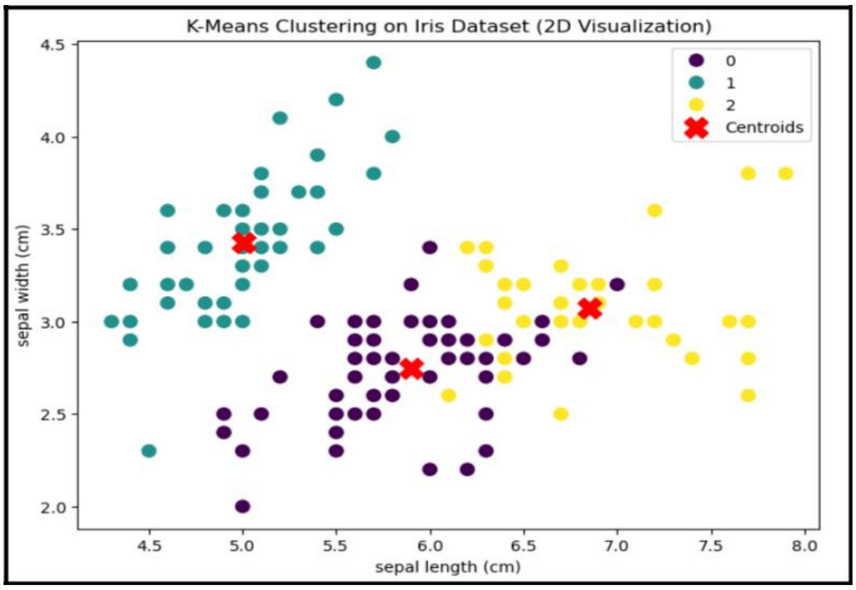
```
Text(0, 0.5, 'Inertia')
```

```
#Fit a k-means model to the data with 3 clusters
km = KMeans(n_clusters=3, random_state=0)
km.fit(iris_df)
```

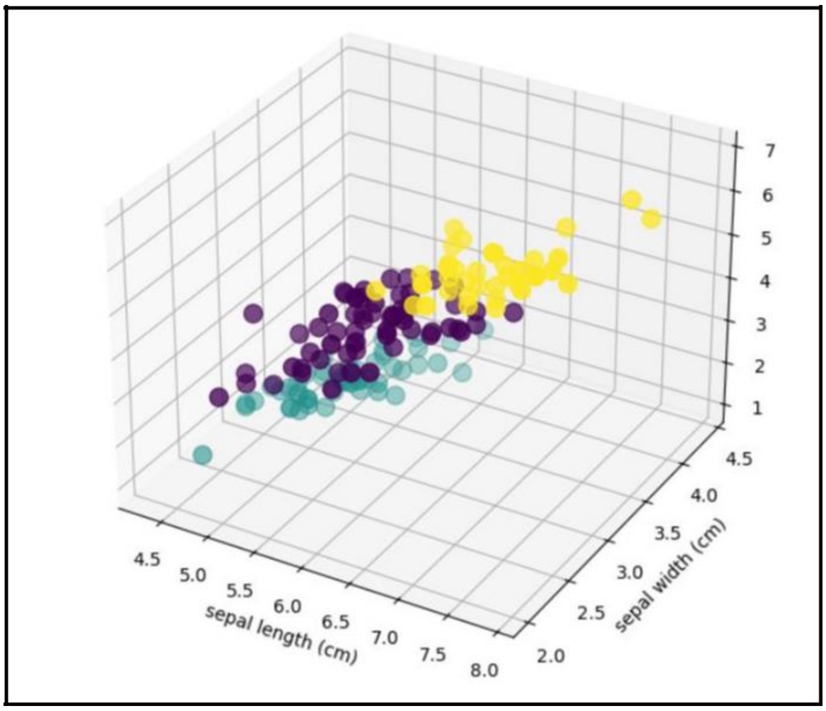


```
KMeans
KMeans(n_clusters=3, n_init=10, random_state=42)
```

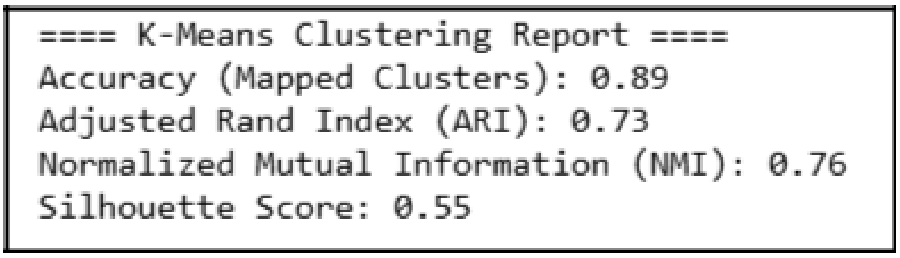
```
# 2D Visualization (First two
features) plt.figure(figsize=(8, 6))
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=kmeans, palette="viridis", s=100)
plt.scatter(kmeans.cluster_centers_[:,  0],  kmeans.cluster_centers_[:,  1],
c='red', marker='X', s=200, label='Centroids')
plt.title('K-Means Clustering on Iris Dataset (2D Visualization)')
plt.xlabel(feature_names[0])
plt.ylabel(feature_names[1])
plt.legend()
plt.show()
```



```
# 3D Visualization (First three features)
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y_kmeans, cmap='viridis',
s=100)
ax.set_title('K-Means Clustering on Iris Dataset (3D
Visualization)') ax.set_xlabel(feature_names[0])
ax.set_ylabel(feature_names[1]) ax.set_zlabel(feature_names[2])

plt.show()
```

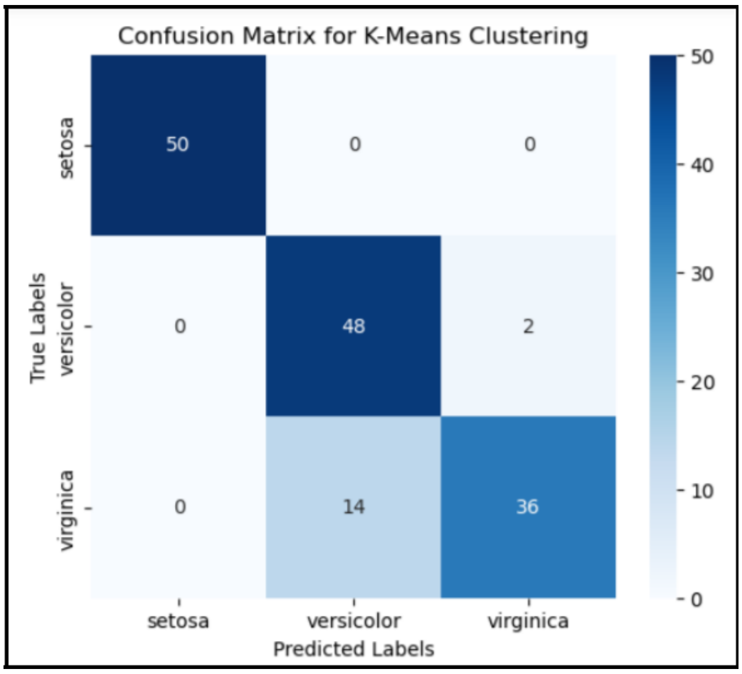


```
# Compute clustering metrics
accuracy = accuracy_score(y_true, labels)
ari = adjusted_rand_score(y_true, y_kmeans)
nmi = normalized_mutual_info_score(y_true, y_kmeans)
silhouette = silhouette_score(X, y_kmeans)
conf_matrix = confusion_matrix(y_true, labels)
```



```
==== K-Means Clustering Report ====
Accuracy (Mapped Clusters): 0.89
Adjusted Rand Index (ARI): 0.73
Normalized Mutual Information (NMI): 0.76
Silhouette Score: 0.55
```

```
# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, cmap="Blues", fmt='d',
xticklabels=iris.target_names, yticklabels=iris.target_names)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix for K-Means
Clustering") plt.show()
```



LEARNING OUTCOMES ::

1. Learned how clustering works and the importance of choosing the right number of clusters.
2. Gained skills in plotting and interpreting cluster separation and centroids.
3. Applied K-Means using Python libraries like sklearn, matplotlib, and seaborn.
4. Understood how clustering helps in pattern discovery in real-world datasets.