Experiment: Implementing the K Means Clustering algorithm on the Iris dataset to visualize and analyze diverse clusters.

Title:

Implement K Means Clustering Algorithm on the Iris Dataset and Visualize Diverse Clusters

Aim:

To apply and evaluate the K Means clustering algorithm on the Iris dataset for clear visualization of distinct clusters.

Objective:

Students will learn:

- How to implement the K Means Clustering algorithm on a real-world dataset (Iris).
- Methods to visualize the resulting clusters effectively.
- Techniques to interpret the clustering outcomes and assess the performance of the algorithm.

Problem Statement

Apply and implement the k-means clustering algorithm on the iris dataset and visualize the diverse clusters to identify patterns and distinctions among the iris species

---

Explanation / Stepwise Procedure / Algorithm

1. Data Acquisition and Preprocessing

- Dataset Overview:
    - The Iris dataset consists of 150 samples with four features: sepal length, sepal width, petal length, and petal width. It is widely used for clustering and classification tasks.
- Preprocessing Steps:
    - Loading Data: Import the Iris dataset using libraries like scikit-learn.

2. Initialization of the K Means Algorithm

- Choosing the Number of Clusters (k):
    - Since the Iris dataset contains three species, k is typically set to 3.
    - Initial Centroids:

        - Randomly select k data points from the dataset to serve as the initial centroids.

3. Cluster Assignment

  - Distance Calculation:
    - For each data point, calculate the Euclidean distance to each centroid.
  - Assignment:
    - Assign each data point to the cluster whose centroid is closest, forming initial clusters.

4. Centroid Recalculation

  - Recomputing Centroids:
    - For each cluster, calculate the mean (average) of all data points assigned to that cluster.
    - Update the centroid positions with these new mean values.
    - Repeat the assignment and centroid update steps until the centroids stabilize (i.e., changes become negligible) or a predefined maximum number of iterations is reached.

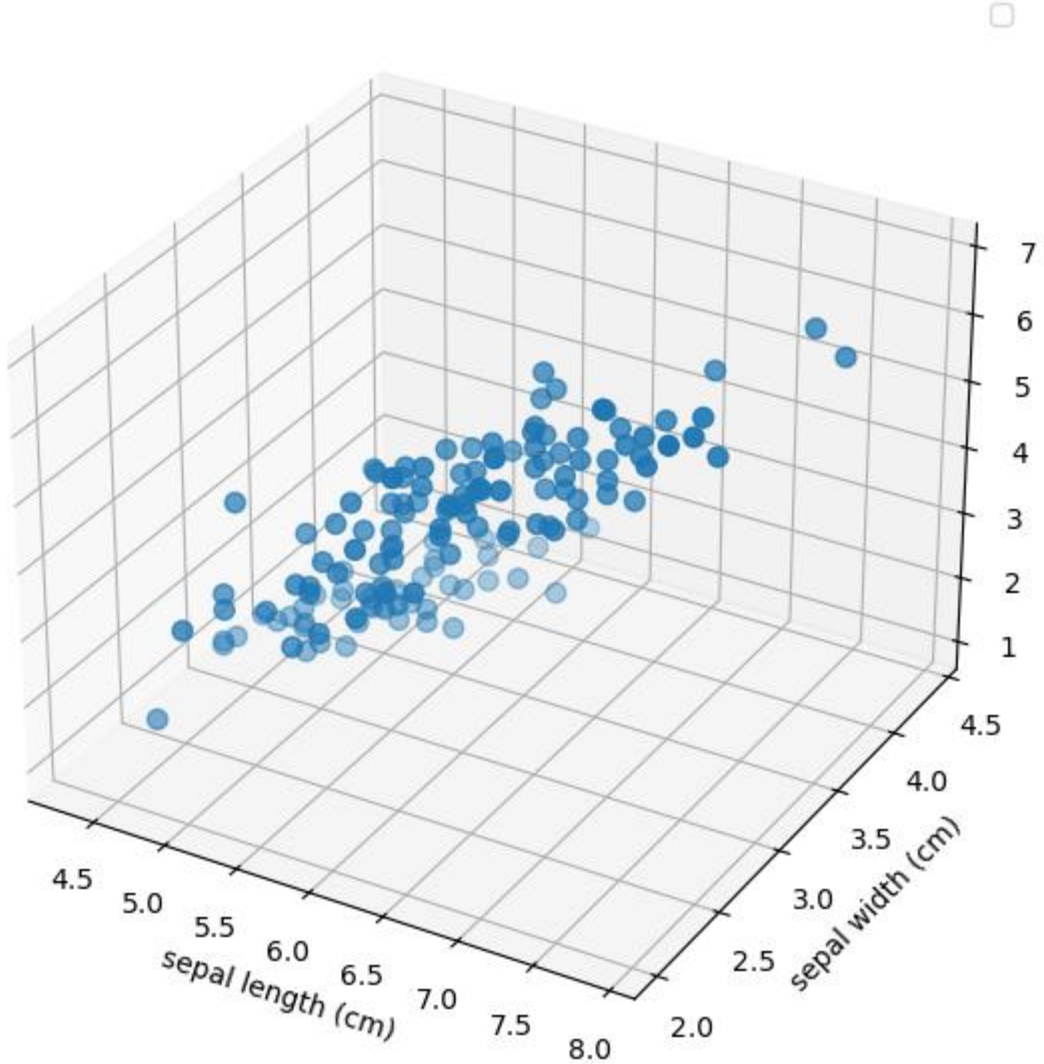
6. Visualization

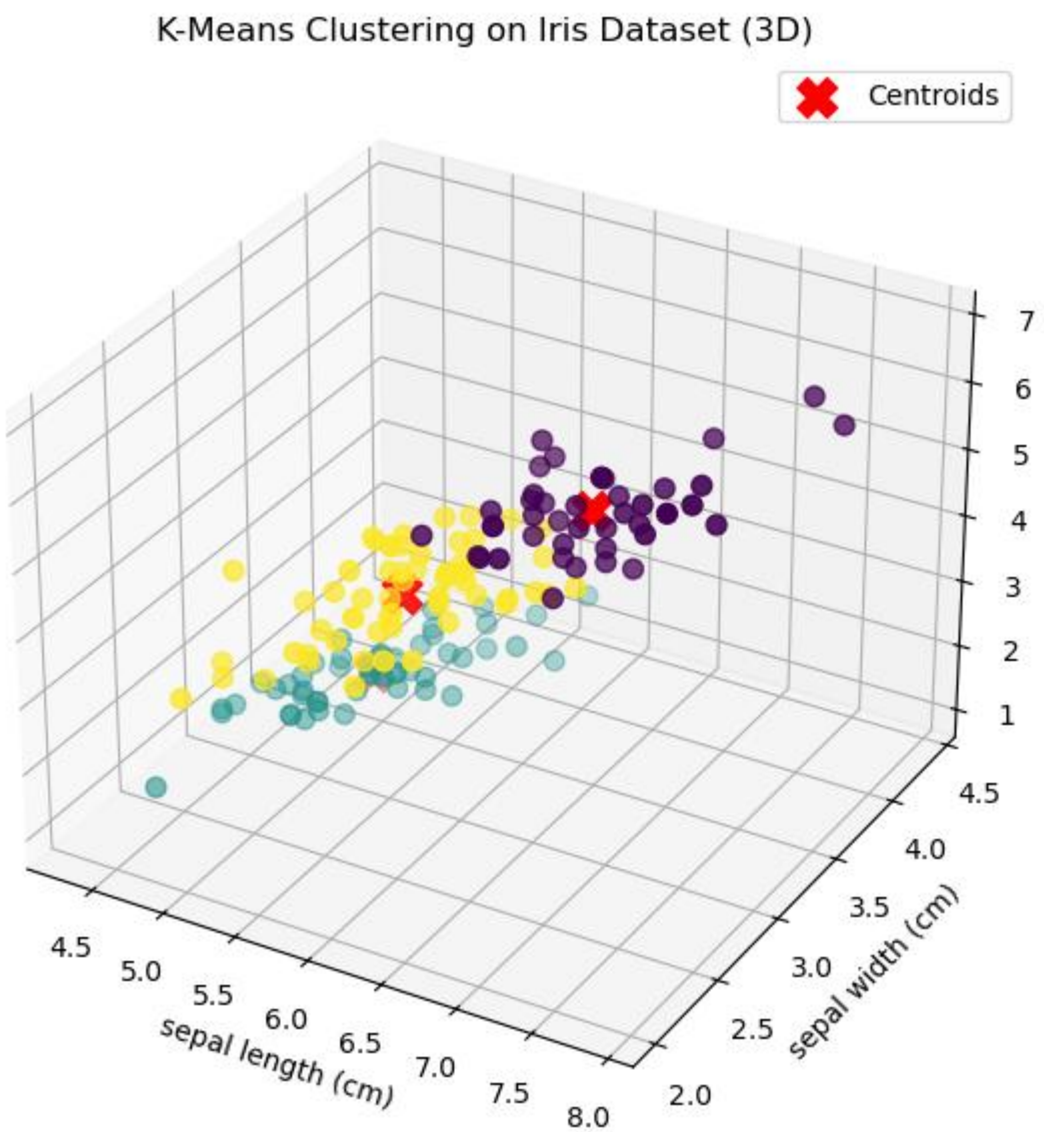
  - Plotting:
    - Create scatter plots where each data point is colored based on its cluster assignment.
    - Highlight the final centroid positions using distinct markers (e.g., larger or different colored points).

Figures/Diagrams

- Iris-dataset look before the clustering of the data.
- After Clustering comparison between the unclustered and the clustered data.

K-Means Clustering on Iris Dataset (3D)





Challenges Encountered

1. Choosing the right value of k to cluster the data
2. Decide the right feature to drop so that we could cluster the data in 3D.
3. Plotting the different graphs so that it looks easy to understand and interpret.

Conclusion

- K Means successfully groups the Iris dataset into distinct clusters.
- PCA-based visualization clearly illustrates these natural groupings.
- Sensitivity to initialization and cluster shape assumptions suggest exploring alternative approaches for improved results.

USL_KNN_IPYNB

March 2, 2025

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from mpl_toolkits.mplot3d import Axes3D
```

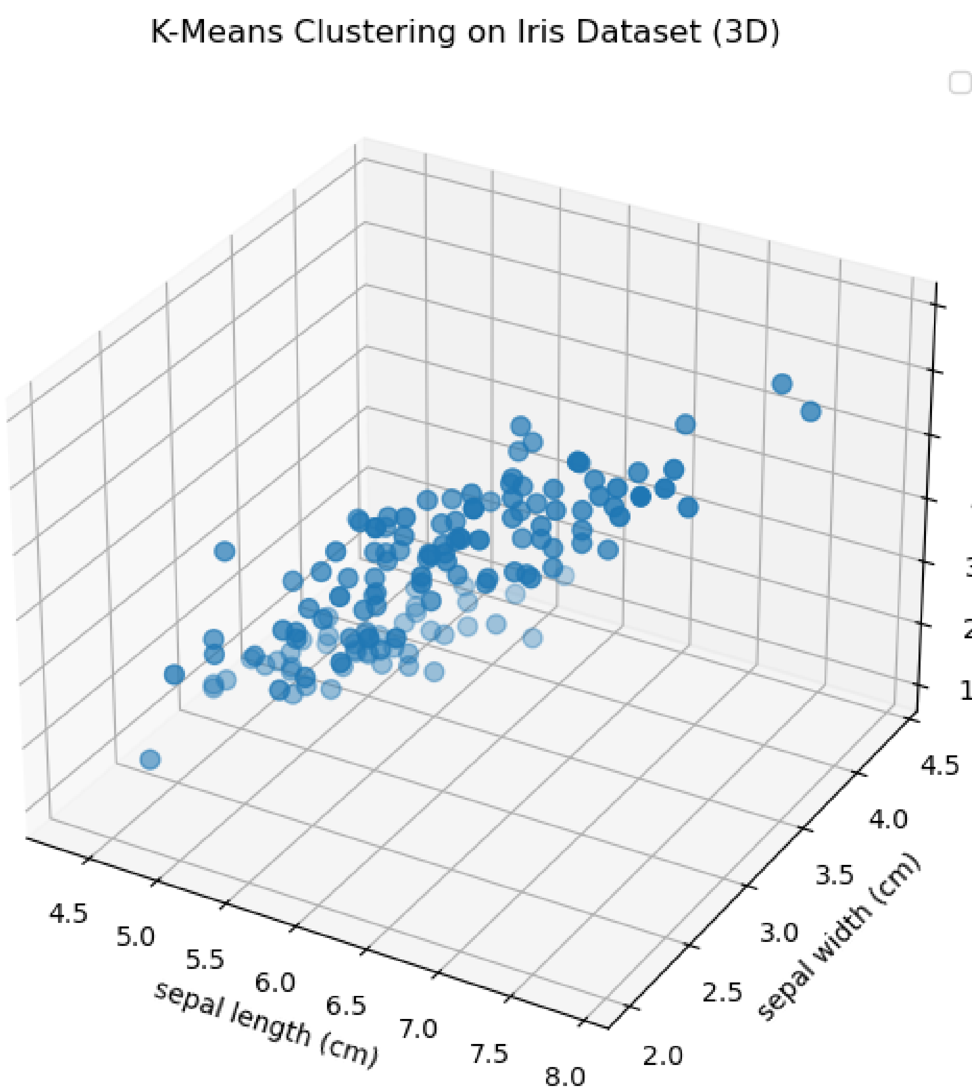
[ ]:
```
# Load the iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

[ ]:
```
# Select three features for 3D visualization
features = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)']
data = df[features]
```

[ ]:
```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data.iloc[:, 0], data.iloc[:, 1], data.iloc[:, 2], cmap='viridis',␣
 ↪marker='o', s=50)
ax.set_xlabel(features[0])
ax.set_ylabel(features[1])
ax.set_zlabel(features[2])
ax.set_title('K-Means Clustering on Iris Dataset (3D)')
ax.legend()
```

```
C:\Users\Neil\AppData\Local\Temp\ipykernel_38364\4258469181.py:3: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  ax.scatter(data.iloc[:, 0], data.iloc[:, 1], data.iloc[:, 2], cmap='viridis',
marker='o', s=50)
C:\Users\Neil\AppData\Local\Temp\ipykernel_38364\4258469181.py:8: UserWarning:
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
  ax.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x29ad4a8b4a0>
```



[ ]:
```python
# Perform K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(data)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

```
c:\Users\Neil\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

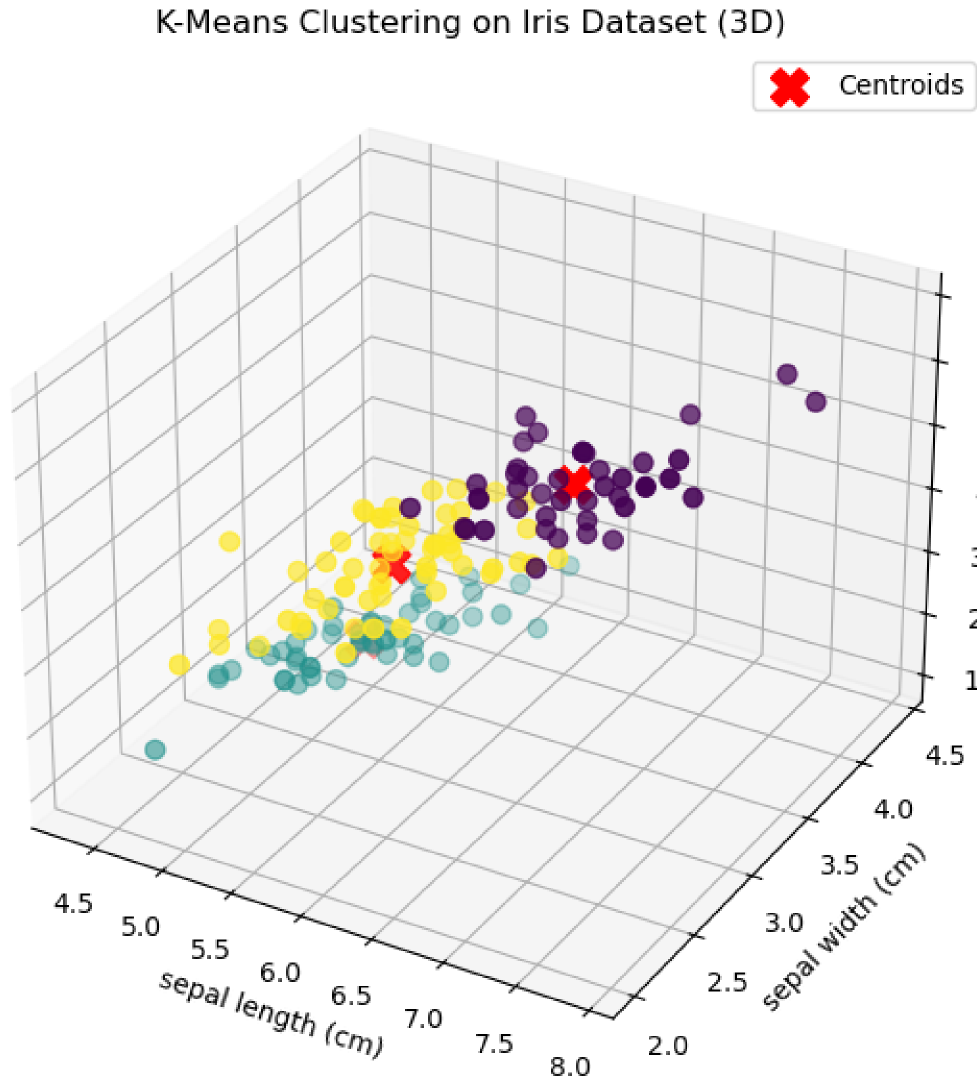
[ ]:

```
# Plot the clusters in 3D
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data.iloc[:, 0], data.iloc[:, 1], data.iloc[:, 2], c=labels,␣
↪cmap='viridis', marker='o', s=50)

# Plot centroids
ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], c='red',␣
↪marker='X', s=200, label='Centroids')

# Labels and title
ax.set_xlabel(features[0])
ax.set_ylabel(features[1])
ax.set_zlabel(features[2])
ax.set_title('K-Means Clustering on Iris Dataset (3D)')
ax.legend()

plt.show()
```


K-Means Clustering on Iris Dataset (3D)
Centroids
sepal length (cm)
sepal width (cm)
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
2.0
2.5
3.0
3.5
4.0
4.5
1
2
3
4
5
6
7