Experiment-7: Implementing K-Means Clustering from Scratch

Objective

The goal of this assignment is to implement the K-Means clustering algorithm **from scratch using NumPy**, apply it to a customer segmentation dataset, and interpret the clusters formed.

1. Dataset and Setup

- Use a customer dataset containing features such as `Age`, `Annual Income (k$)`, and `Spending Score (1-100)`.
- Perform basic **data exploration and preprocessing:**
  - Handle missing values if any.
  - Normalize or standardize the numerical features (justify your choice).
  - Provide relevant summary statistics and visualizations(distribution plots (histograms, scatter matrix) of features.).

2. Tasks

2.1. Part A: Implementing K-Means from Scratch

Write your own functions for each step of the K-Means algorithm using **NumPy only** (no `scikit-learn`).

1. **Initialization**
   - Write a function `initialize_centroids(X, k, random_state=None)` that randomly selects k points as initial centroids.
   - Implement the **K-Means++ initialization scheme:**
     1.1. Choose one centroid uniformly at random from the data points.
     1.2. For each data point x, compute the squared distance $D(x)$ to the nearest chosen centroid.
     1.3. Choose the next centroid with probability proportional to $D(x)$.
     1.4. Repeat until k centroids are chosen.

     Compare convergence speed and quality with random initialization.
2. **Cluster Assignment**
   - Implement `assign_clusters(X, centroids)` that assigns each data point to the nearest centroid (using Euclidean distance).
3. **Centroid Update**

- Implement `update_centroids(X, labels, k)` that recalculates the centroids as the mean of all points belonging to each cluster.

4. **Main Algorithm Loop**
   - Combine the above steps in a function `kmeans(X, k, max_iters=100, tol=1e-4, random_state=None)`.
   - The algorithm should stop when:
     - The centroid positions change less than `tol` (tolerance), or
     - The maximum number of iterations is reached.
   - The function should return:
     - Final centroids
     - Cluster labels
     - Number of iterations
     - Inertia (sum of squared distances to centroids)

2.2. Part B: Choosing the Number of Clusters

- Use the **Elbow Method** and the **Silhouette Score** to determine an appropriate k. Compare both
- Plot inertia vs. k and justify your chosen value.

2.3. Part C: Applying the Algorithm

- Run your K-Means implementation on the dataset using your chosen k.
- Display:
  - Final centroids and cluster sizes.
  - 2D and 3D scatter plots of clusters (color by label).
- For each cluster, compute the average of each feature (e.g., Age, Income, Spending Score). Compare these averages across clusters to identify distinguishing characteristics. Based on these statistics, interpret each cluster in business terms (e.g., "Cluster 0: Young, high-income, high-spending customers" or "Cluster 2: Older, low-income, conservative spenders").

2.4. Part D: Comparison with scikit-learn

- Run `sklearn.cluster.KMeans` on the same dataset.
- Compare:
  - Final centroids and inertia.
  - Number of iterations to converge.
  - Runtime performance.