Part2: Clustering

Attribute Information:

- (x, y) Points on a plane.

1.1. Load

Below are some basic information about the dataframe.

Shape of each dataframe:

```
(400, 2)
```

Figure 1: First Dataframe

```
(450, 2)
```

Figure 2: Second Dataframe

```
(500, 2)
```

Figure 3: Third Dataframe

I added 'X' and 'Y' headers so each dataframe would look like this:

	X	Y
0	-0.125391	-1.268829
1	0.062522	1.278778
2	-0.048762	0.200549
3	0.105585	-0.496629
4	0.011886	-0.739317

Figure 4: Head of the first dataframe

1. Methods and Models

In this section I defined functions to perform the actions needed for each clustering algorithm. Here are brief descriptions for each:

1.2. Scatter Plots

Each dataset is plotted on a 2 dimension plot.

1.3. Lloyd's k-means Clustering

k-means is a partitioning clustering algorithm and works well with spherical-shaped clusters.
There are various methods to find the ideal number of clusters in a data. One of them is to Use K-means algorithm along with **Silhouette distance**.
Silhouette metric is a distance calculation algorithm using euclidean or Manhattan distance. The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample.
The Silhouette Coefficient for a sample is $\frac{(b-a)}{\max(a,\, b)}$. To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of.

The **best value is 1** and the **worst value is -1**. **Values near 0** indicate **overlapping clusters**. **Negative values** generally indicate that a sample has been assigned to the **wrong cluster**, as a different cluster is more similar.

I tried different number of clusters (k) and computed the SSE as well as the Silhouette coefficient. As per assignment's request, I set *init='random'* and *n_init=200* so that we would have the centroids randomly selected and perform the clustering 200 times and keeping only the best result .

1.3.1. K-Means Elbow Method

The Elbow method is a very popular technique and the idea is to run k-means clustering for a range of clusters k (let's say from 1 to 15) and for each value, we are calculating the sum of squared distances from each point to its assigned center(distortions).

When the distortions are plotted and the plot looks like an arm then the "elbow"(the point of inflection on the curve) is the **best value of k**.

I used the elbow method to find the **number of clusters** that would **minimize the SEE.**

1.4. Fuzzy C-Means Clustering

"fuzzy" here means *"not sure"*, which indicates that it's a **soft clustering** method.
"C-means" means c cluster centers.

In a clustering algorithm, if the probability of one data point belonging to a cluster can only take the value of 1 or 0, it's **hard clustering**. The boundary of a cluster in a hard clustering method can be visualized as a crisp boundary. On the contrary, in a soft clustering method, the probability of one data point belonging to a cluster can take any value between 0 and 1, such as 75%, for which the boundary of a cluster can be visualized as a fuzzy boundary.

Besides the **Silhouette coefficient**, I used two other validation metrics for my model. Which are:

1. The fuzzy partition coefficient (PC): The FPC is defined on the range from 0 to 1, with 1 being best. It is a metric which tells us how cleanly our data is described by a certain model. This is used **to find the optimal number of clusters.**

2. The partition entropy (PEC): Measures the fuzzy degree of final divided clusters by means of the fuzzy partition matrix, and the smaller its value, the better the partition result

Since *fuzzy_c.predict* returns a hard clustering of our model, I used the *fuzzy_c.soft_predict* to label the data myself. ***membership_cutoff*** is a parameter that determines how many clusters is a datapoint a member of. For instance if *membership_cutoff=0.01* and our model tells us that a point d_1 is a member of $C_1,\ C_2,$ and C_3 with a probability of $0.4,\ 0.5,$ and 0.1 respectively, Then $p(C_2) - p(C_1) = 0.1 =$ *membership_cutoff* thus d_1 is a member of both C_1 and C_2.

1.5. DBSCAN Clustering

DBSCAN is a popular **density-based** data clustering algorithm. To cluster data points, this algorithm separates the **high-density** regions of the data from the **low-density** areas. Unlike the K-Means algorithm, the best thing with this algorithm is that we don't need to provide the number of clusters required prior.

DBSCAN algorithm group points based on distance measurement, usually the Euclidean distance and the minimum number of points. An essential property of this algorithm is that it helps us track down the outliers as the points in low-density regions; hence it is not sensitive to outliers as is the case of K-Means clustering.

The parameters for DBSCAN are:

1. Epsilon (Eps): This is the least distance required for two points to be termed as a neighbor. Thus we consider Eps as a threshold for considering two points as neighbors, This is the **most important DBSCAN parameter** to choose appropriately for your data set and distance function.

2. MinPoints: This refers to the minimum number of points needed to construct a cluster. We consider MinPoints as a threshold for considering a cluster as a cluster. A cluster is only recognized if the number of points is greater than or equal to the MinPts.

1.5.1. Nearest Neighbours Method

Since the eps figure is proportional to the expected number of neighbours discovered, we can use the **nearest neighbours to reach a fair estimation for eps.** We find a suitable value for epsilon by calculating the distance to the nearest n points for each point using the **NearestNeighbors**, sorting and plotting the results. Then we look to see where the change is most pronounced (elbow or knee points) and select that as epsilon.

After finding the **best eps** for DBSCAN using **NearestNeighbors**, I then perform a search on a range of numbers and choose the **best value for the min_samples** parameter by **maximizing the silhouette score**.