AKHIL RAJESH SHARMA
22201271

CLUSTERING ANALYSIS REPORT

Assignment 2 Cluster Analysis

Q1.1 : Report the results of these runs and discuss the clusters that are found. Include screenshots of the outputs to aid your discussion.

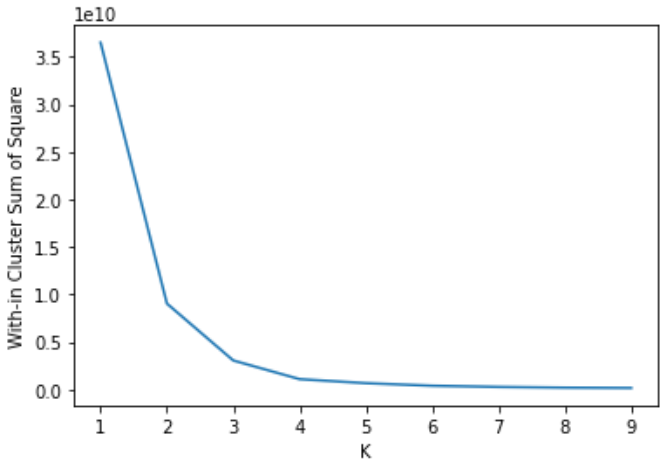
Common Concepts used to analyze the runs:

1. Inertia – Inertia is a term used in clustering which gives the sum of square distances of each sample from their cluster centers. This method is used to check how accurate a cluster is. A low inertia score is considered better as the samples are more tightly packed toward their cluster centers. Inertia also however causes cases of overfitting which is why it is always better to couple it with other methods.

2. Silhouette Score – The silhouette score is also a metric used to evaluate the quality of a clustering result. It assigns a value between -1 to 1 where a higher value indicates that the clusters are well separated and do not have overlaps with each other. Along with this a high score also indicates that the samples within the cluster are similar to each other and the cluster rather than the other cluster. It is often used for the comparison of cluster results to find the optimal clusters.

3. Elbow Plot - The elbow plot is a visual aid that can be used to determine the optimal number of clusters for a dataset in clustering analysis. The plot shows the relationship between the number of clusters and the sum of squared distances (inertia) between each data point and its assigned cluster center (Arvai, 2023). The plot typically displays the number of clusters on the x-axis and the inertia on the y-axis.

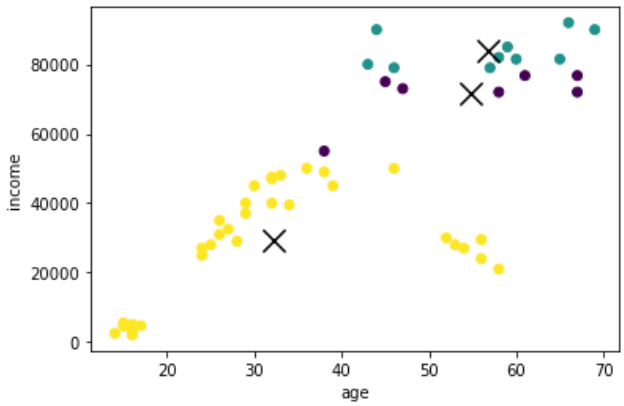
Clusters formed by K-means run on Salary data:



Elbow plot for salary data

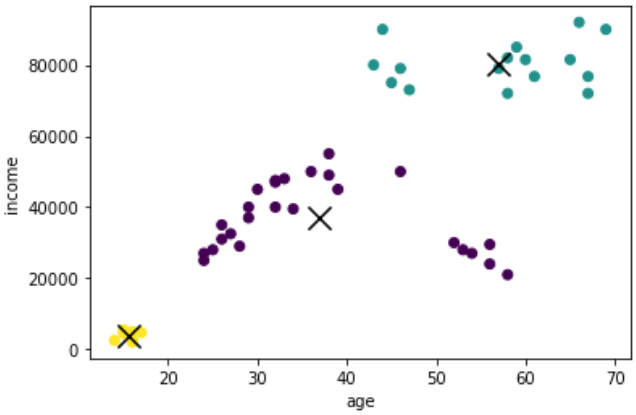
The initial step followed was to accurately choose the number of clusters (k), to do this I used the elbow plot to get a value of k=4 as the most stable and correct value. It was also observed that the highest silhouette score for clustering with k=3 and k=4 for the 10 runs on salary data came out to be 0.711 and 0.742. This shows that in multiple runs, there is a higher chance of us getting a better silhouette score if we use the value of k=4. While observing the 10 runs it was strongly observed that a lot of runs were similar with optimal silhouette scores ranging above 0.7.

Silhouette score for k=3 for salary data



Inertia Score =:8393570985.028571

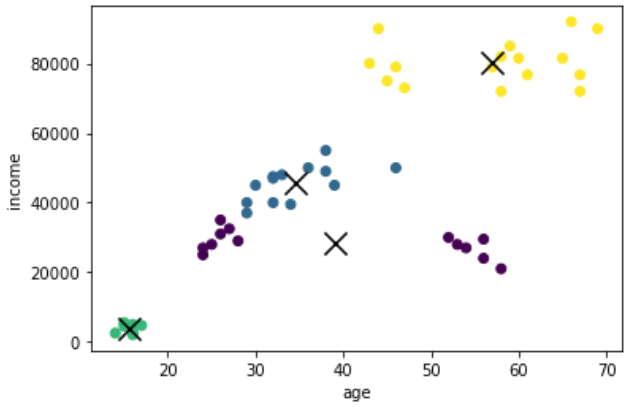
Silhouette Score = :0.49928403009252137



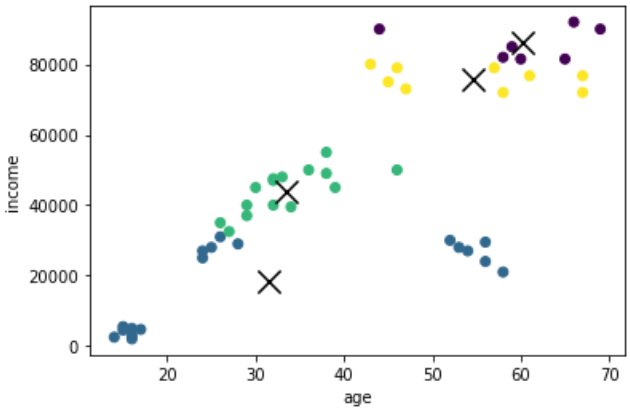
Inertia Score =:3061733558.0274725

Silhouette Score = :0.7112533134788821

Silhouette Score for k=4 for salary data




```
Inertia Score =:1097271876.4505494
Silhouette Score = :0.7420615682179896
```




```
Inertia Score =:3201822344.512699
Silhouette Score = :0.5134823256479996
```

Here the more suitable clusters are selected by viewing metrics such as inertia and silhouette score. There needs to be a balance between the two. Selection criteria should focus on maximizing the silhouette score as well as minimizing the inertia.

The clusters formed in this data are categorized by income range and bounded by age as well. We use the following code to get the minimum and maximum values of each feature in each cluster –

```
# calculate cluster ranges
ranges = df.groupby('cluster_labels')[[x_col, y_col]].agg(['min', 'max'])

# print cluster ranges
print('Cluster Ranges:')
for i in range(k):
    print(f'Cluster {i}: {ranges.loc[i]}')
```

With this, we make the following analysis of our clusters: -

Cluster 1: Age Bracket: 14 to 17 || Income Bracket: 2000 to 5500

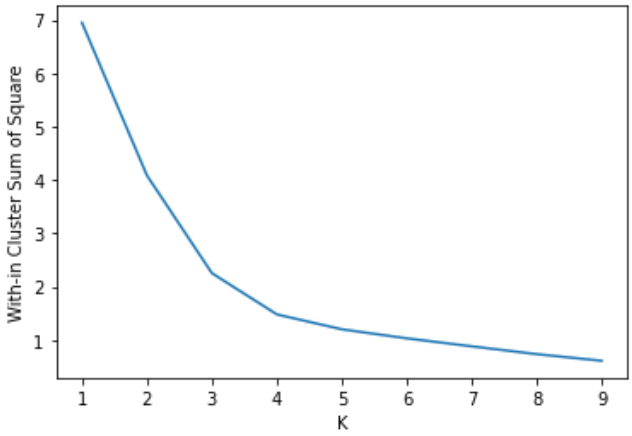
Cluster 2: Age Bracket: 24 to 58 || Income Bracket: 21000 to 37000

Cluster 3: Age Bracket: 29 to 46 || Income Bracket: 39500 to 55000

Cluster 4: Age Bracket: 43 to 69 || Income Bracket: 72000 to 92000

Clusters formed by K-means run on Random data:

In the case of random data, the random generator function was used to create a set of 50 random data points which were projected on a scatter plot. The first initial step was to again choose the value of k which was found by initially plotting the elbow plot. The elbow plot shows clear change near the values of k=3 and k=4.

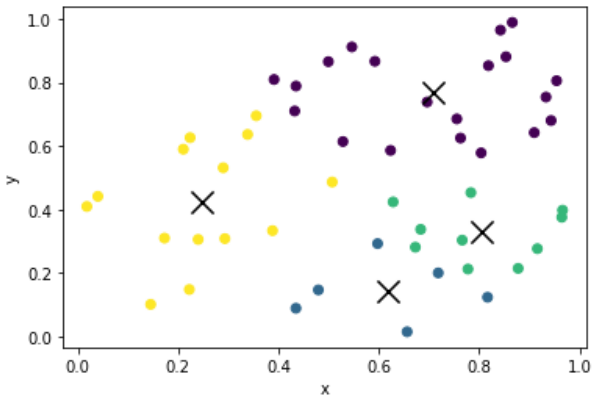


Elbow plot for random data

After deciding the optimal k value as 4, and 10 runs of k-means, we found the following observations for the random data:

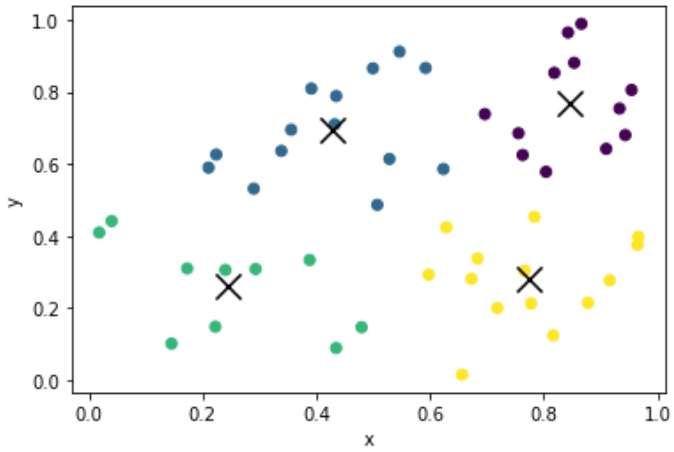
Minimum Score: The following cluster gave the worse results as it had the highest value of inertia and the lowest Silhouette score. A contribution to this poor performance can be seen

as the clusters are not uniformly spread out and hence can lead to more overlapping clustering.




```
Inertia Score =:1.9777640590986685
Silhouette Score = :0.31687610638361846
```

Most Frequent Score: It was observed that out of the 10 runs, a similar silhouette score and inertia were seen in 5 of the samples strongly suggesting that with the current configuration, this is the highest and most optimal clustering.




```
Inertia Score =:1.481868688210445
Silhouette Score = :0.4483035637722795
```

Extra observations:

Similar to the most frequent observation in random data, within the salary data too, there were 4 out of 10 clusters that were similar and most optimal suggesting that different runs give different results which can be taken as best, worst, average, and optimal.

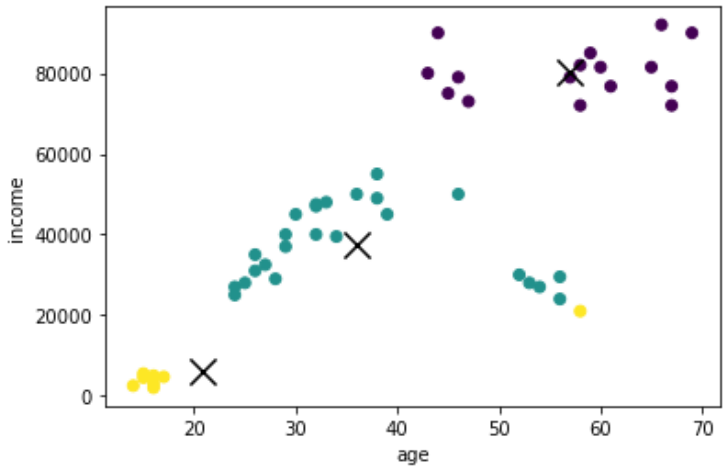
Also, the scores of clustering of random data of k=3 and k=4 were also checked with the outcome of better silhouette score being seen for k=4 for random data

Q1.2: Describe your understanding of these three parameters in the run_kmeans function and discuss how they impact the results:

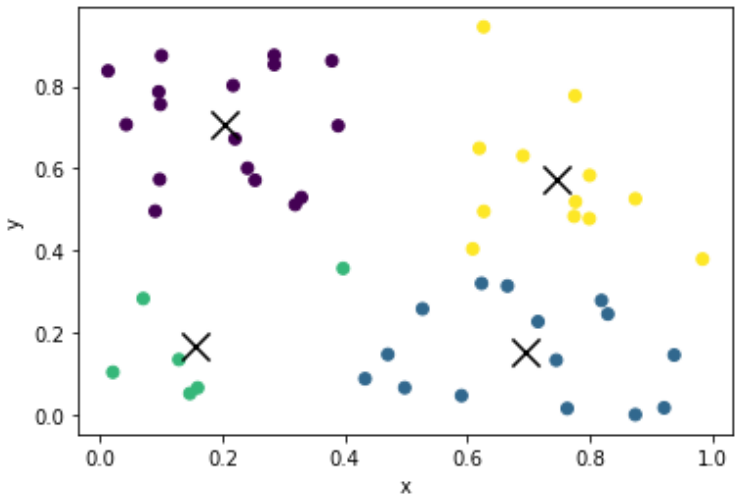
kmeans = KMeans(n_clusters=k, init='random', n_init=1)

The K-means function has the following parameters which specify the way the K means method is used (*Sklearn.cluster.kmeans*) :-

1.**n_clusters=k:** Here the value of K defines how many clusters are we looking to form by specifying the number of centroids used. In the abovementioned example, we see that k=3 which means that the data is clustered into 3 groups with 3 centroids. As shown here now if we change the value of k to 4 the number of clusters changes. Here the number of clusters (k) is a key factor as we need to find a value that gives us stable clusters where all the points are accounted for and there are the least weak points and the clusters do not change.



When the value of k = 3



When the value of k = 4

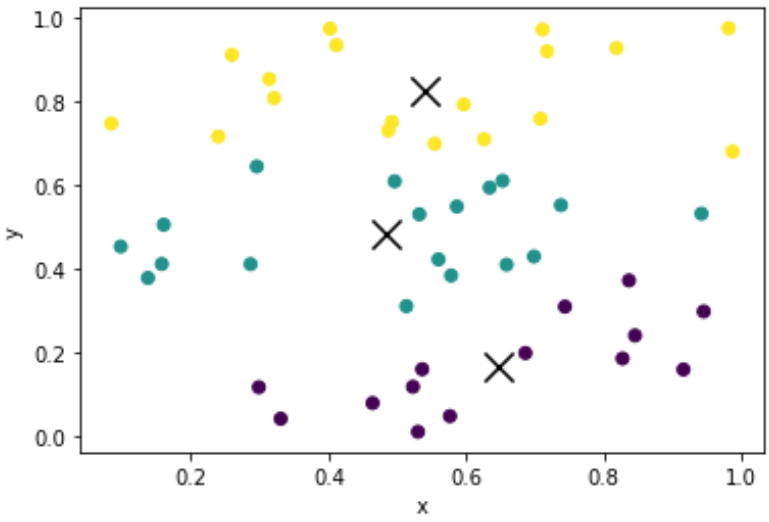
2.**Init=random:** This part of the parameter decides the location of the initial placement of the centroids. The value of init can be of different types like random, k-means++, callable, etc. In the above example, the value of init=random specifies that the initial position of the centroids is random. After each iteration of the k-means run, the position of centroids is readjusted towards the cluster center, and the clusters are recalculated until the clusters are stable and do not change. Another use of the init parameter as init=k-means++ is used often in the K-Means function, in this case, the initial position of the centroids is taken furthest away from each other. This way there is maximum convergence and better clustering clarity. We will discuss the difference between init=random

3.**n_init=1:** This is used to specify the number of times the algorithm should run with a different initial centroid. It is used to avoid local maxima/minima and break free with the use of different centroids. Usually, the results are highly dependent on the initial value of the centroids, hence it is highly likely that a single run with an initial fixed centroid value wouldn't give an ideal result. Therefore, usually, a value of greater than 1 is used to ensure a better optimal result.

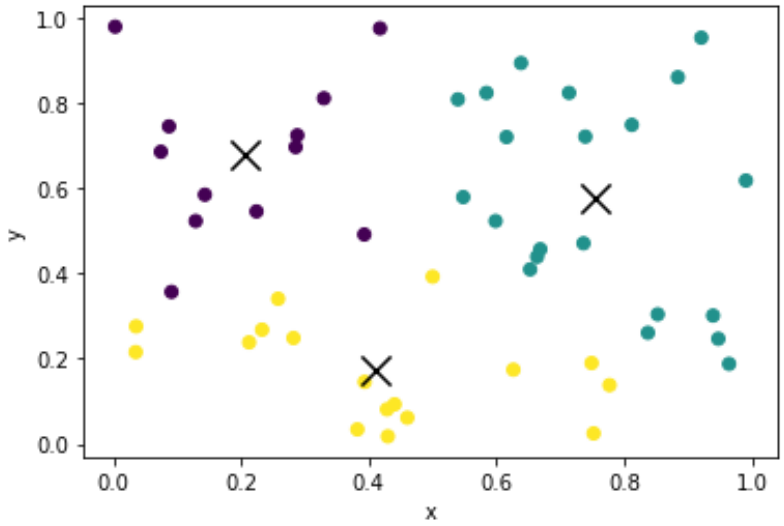
1.3 Report and discuss the clusters that are found. Compare them to the previous output where kmeans++ was not used.

Contrast and Comparison (Mishra, 2020):

	KMeans	KMeans++
Centroids	Initial centroids are chosen at random.	There is computation to choose the initial centroids which are furthest away from each other.
Convergence	Lower convergence	Higher convergence
Computation	Takes lesser computation time as it doesn't involve any calculation and is random.	Takes greater computational time as it required initial analysis to choose the correct centroid positions.
Initial Seed	It is sensitive to the initial seed as it is chosen at random and is more prone to generating suboptimal result.	Lesser sensitive to initial seed as it is selected on a calculation and less prone to generating suboptimal result.
Dataset	Better when used on a smaller database as it needs less time and produces the similar result.	Better when used at larger data as it produces better results due to centroid placement.



When init = random



When init = k-means++

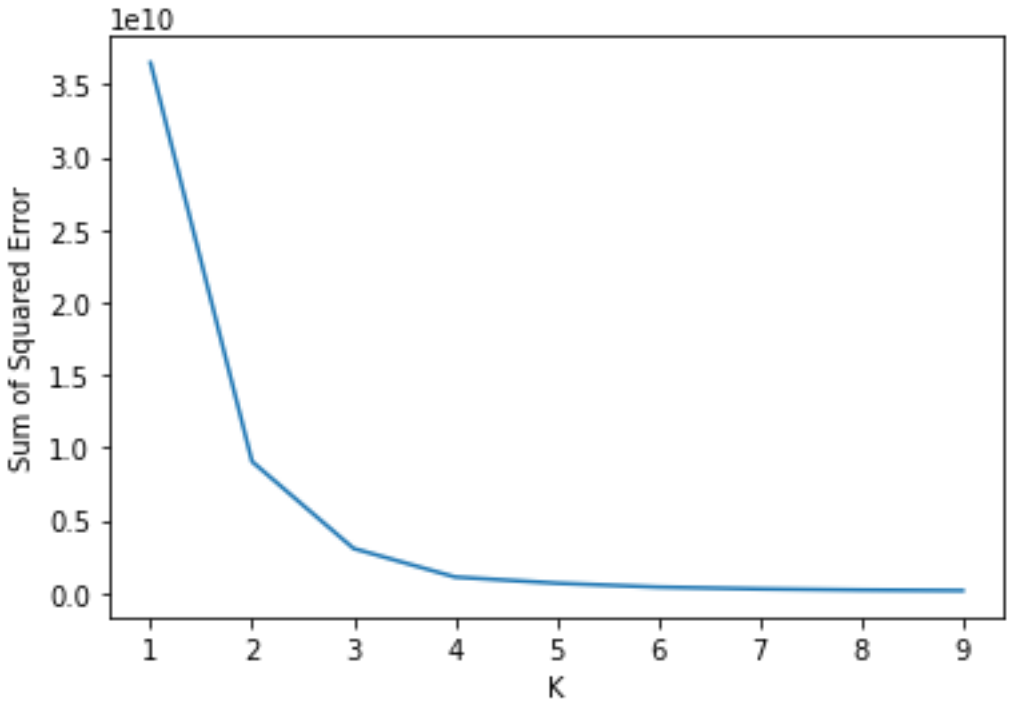
1.4 Report the Elbow plot and discuss how you would interpret and use such information.

In clustering, the main objective is to minimize the distance between the points inside the cluster and maximize the distance between clusters. The distance between points in a cluster is counted as the within-cluster sum of squares (WCSS). If we minimize WCSS, we get an optimal clustering solution.

The elbow chart is used to analyze the best value of k by measuring the value of WCSS for each value of k. The value of k beyond which increasing the number of clusters does not contribute significantly to the decrease in WCSS is called the elbow of the graph. This value of k is taken as a suitable value for the number of clusters as beyond this the value of WCSS does not decrease much and remains more or less the same (Mehta, 2022).

A simple example can be seen in the below graph of salary data, the value of WCSS keeps falling until the value of k=4, this signifies that till the value of 4, the value of WCSS is minimized which means that it is the optimal solution, and it is not required to increase the number of clusters.

Hence the steady drop in the WCSS values signifies that more clusters are needed to reach a better optimal value and the point beyond which the value doesn't significantly decrease as per the increase in clusters is taken as a suitable value for k which is the number of clusters



Elbow plot for Salary data

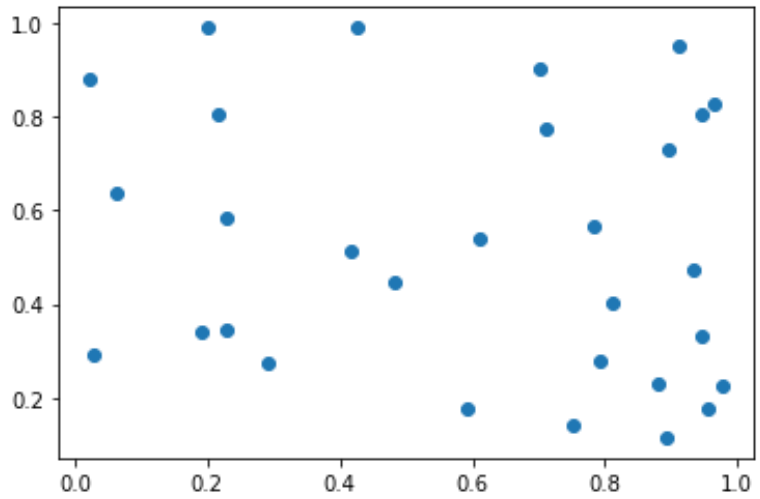
Here the sum of squared error is taken as the WCSS or inertia value and plotted against each value of no. of clusters. The data is seen for its inertia against different clustering values and

plotted against the above graph. The value of K where the slope of the graph approaches zero is taken as the optimal value.

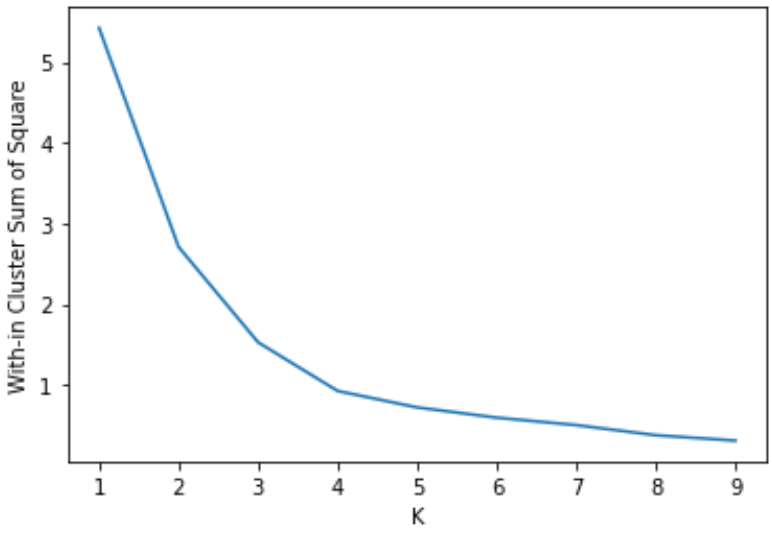
Now plotting an elbow for randomly generated 30 points data.

```
rand_data2 = generate_random_data(30)
plt.scatter(rand_data2.x, rand_data2.y)
```

```
get_elbow_plot(rand_data, x_col='x', y_col='y')
```



Random data scatter plot for 30 data points



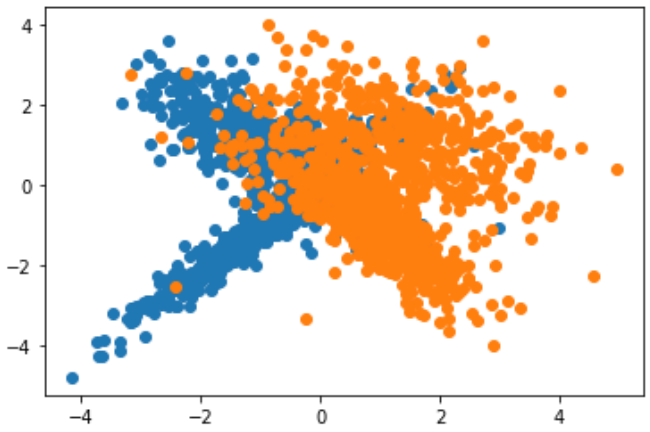
Elbow plot for random data with 30 points

Here following the same explanation, we can see that there is observable change in the solve at k=4 which suggests that a value of k=4 can be used to cluster the random data.

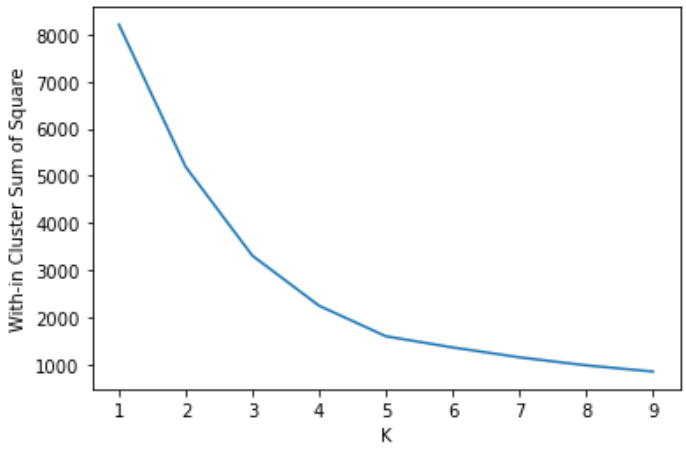
2.1 Run the k-means algorithm on this dataset. Report and discuss your results. Also discuss how you selected a value of k and any other relevant parameters.

```
# define dataset
data_values, class_labels = make_classification(n_samples=2000, n_features=2, n_informative=2,
n_redundant=0, n_clusters_per_class=2, random_state=None)
```

After the implementation of the above code, the following was the plot formed for the data.

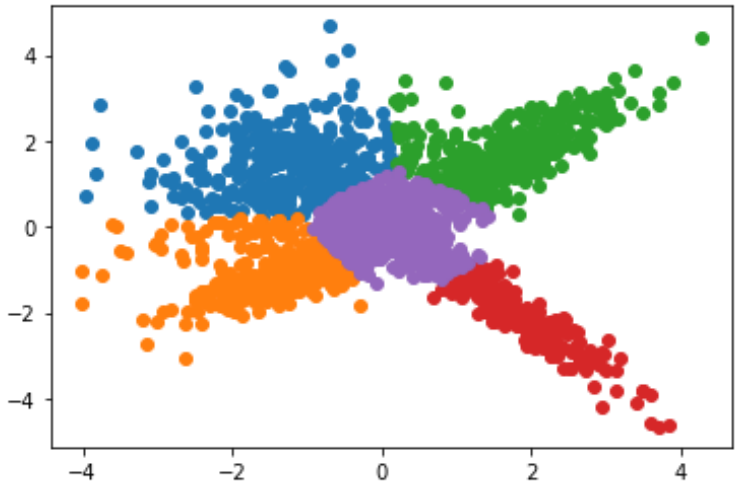


A good example of parameter tuning can be seen here. Initially while plotting the elbow plot for this data, we get the following :



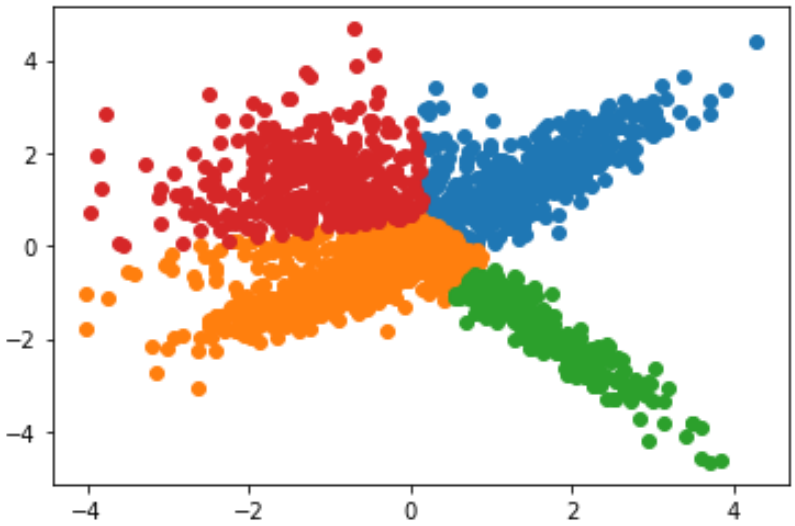
Here we are indicated that the value of k would be more ideal for a value of k=5.

For the value of k=5 the following clustering results are seen for k-means over a large 3000 points dataset, we get the following as the result.




```
The silhouette_score is : 0.4406945697226211
The calsinki harabasz score is : 2064.779680199025
```

This should ideally suggest an optimal score as the number of clusters decided by the elbow plot was given as 5. However, while checking the solution, after implementing the same run on the same code with the value of k=4, we find that the values are better.



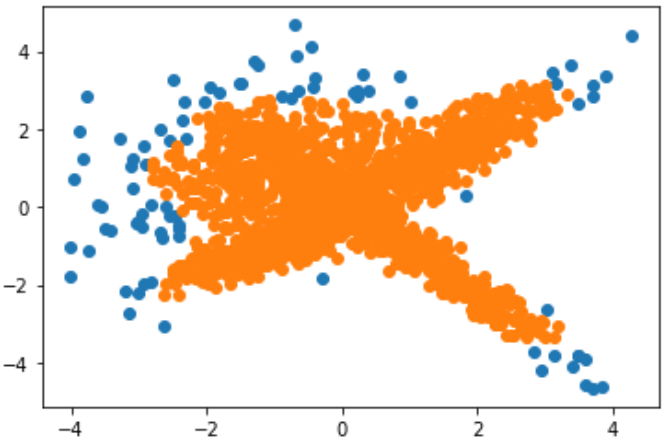

```
The silhouette_score is: 0.4601632363121615
The calsinki harabasz score is: 1768.339293648760
```

The Calsinki Harabasz metric is also used to evaluate the quality of the cluster. It is fairly useful if we want to balance the cluster separation as well as the compactness of the clusters formed. A higher Calsinki Harabasz score is considered better (Mehta, 2022).

2.2 Run the DBSCAN algorithm on this dataset. Report and discuss your results. Discuss your approach to parameter estimation in this case.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised clustering algorithm used in machine learning to group similar data points together in a dataset (*Sklearn.cluster.DBSCAN*). Unlike other clustering algorithms, DBSCAN does not require the number of clusters to be defined beforehand, as it is able to identify clusters based on the density of data points in the given dataset. The algorithm assigns data points to a cluster if they are within a certain distance of each other and if they satisfy a minimum density criterion. Here the value of eps(epsilon) is used to choose the distance and min_sample is used to choose the minimum point density used in the DBSCAN method. Data points that are not part of any cluster are labelled as noise or outliers.

To implement this DBSCAN algorithm, initially the value of parameters was taken as eps=0.3 and min_samples=8, to which the result was seen as follows:




```
The silhouette_score is: 0.3898018745019377
The calsinki harabasz score is: 25.252896290234112
```

Here only 2 clusters are formed, out of which the blue outliers are marked as -1 and are considered as the noise and the densely populated orange region is the primary cluster which is separated because of the density of the sample. A silhouette score of 0.38 shows the positive quality of clustering however it is not considered as a very high value.

Parameter Tuning:

Now we shall try to find a better clustering result by parameter tuning using a grid search.

In grid search, we use all the possible combinations of the eps value and the min_samples value and compare the results we get to find an optimal solution.

We use the following values of epsilons as 0.1,0.2 …..0.6 and we use the values of min_samples as 5,6,7….15. And for each result, we make a grid for the value of silhouette score we get as the primary indicator for the quality of clustering.

The following were the silhouette value results of the grid search:

Minimum Samples -> Epsilon	5	6	7	8	9	10
0.1	-0.38	-0.42	-0.42	-0.34	-0.43	-0.44
0.2	-0.22	-0.05	-0.31	-0.32	-0.35	-0.14
0.3	0.42	0.10	0.39	0.38	0.13	0.36
0.4	0.37	0.44	0.43	0.42	0.41	0.41
0.5	0.45	0.45	0.44	0.44	0.43	0.45
0.6	0.47	0.47	0.47	0.48	0.47	0.46

However, the Silhouette score itself isn't sufficient enough as it increases with increasing values of epsilon so we look at the Calsinki Harabasz score too which gives us the following grid after eliminating all the poor values we found above for the Silhouette scores, we can remove eps=0.1 and eps=0.2:

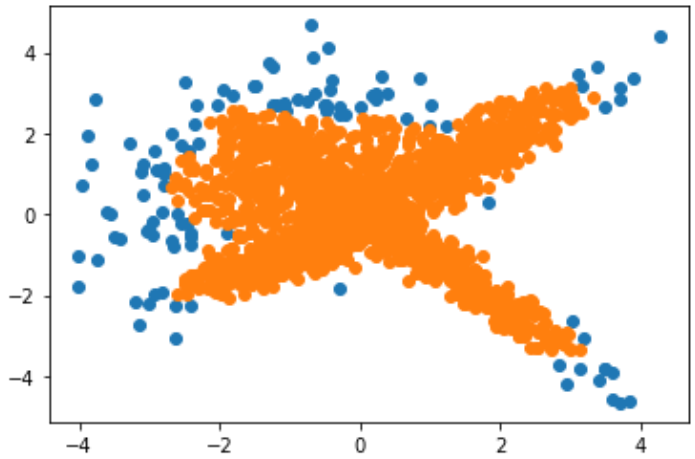
Minimum Samples -> Epsilon	5	6	7	8	9	10
0.3	8	11	25	39	20	41
0.4	22	4	5	8	12	13
0.5	13.5	13.5	15	15	16	5
0.6	12	12	9.5	5.75	6.5	11

Here we see a trend that over higher epsilon values, the value of Calsinki Harabasz score decreases.

Therefore by this grid analysis, we tuned the parameters to be eps=0.3 and min_samples=10.

After tunning, the following graph was seen which clearly portrays the noise and the outliers, and the single cluster in the middle as it is densely populated.

```
Found 2 clusters
```




```
The silhouette_score is: 0.36283007665362516
The calsinki harabasz score is: 40.9676797535679
```

3.3 Implement one other clustering algorithm on this dataset. Report and discuss your results.

BIRCH or Balanced Iterative Reducing and Clustering hierarchies is an algorithm that is designed to work on large data (*Sklearn.cluster.birch*). In its initial stage, it roughly clusters the data to save up on time and then refines the formed clusters to get a better result. The BIRCH Algorithm forms a tree structure to represent the entire dataset. The algorithm works in two stages the initial scan stage and the refinement stage. The two stages are: -

1. Clustering Feature Extraction – In this stage, an initial scan of the data is taken to build an overview of the data which contains several clustering features that represent sub-clusters of the data. This is the tree data that is stored.

2. Clustering Refinement – In this stage the tree data is recursively traversed and split into smaller sub-trees and refined into more similar clusters until a good clustering result is reached.

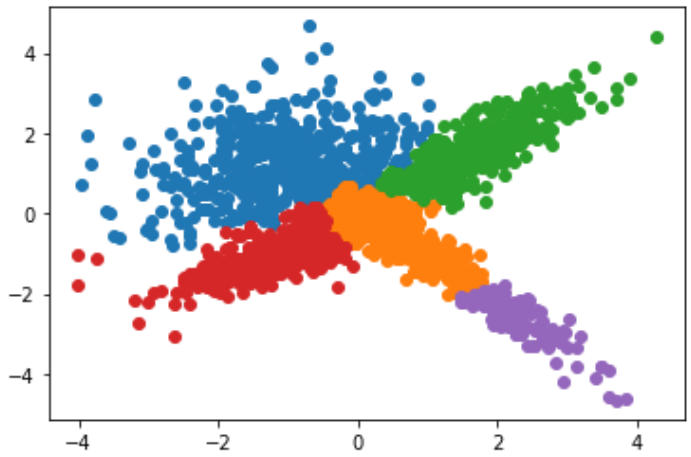
The implementation of BIRCH is as follows: -

```
birch_model = Birch(threshold=0.01, n_clusters=5)
birch_model.fit(data_values)
birch_clusters = birch_model.predict(data_values)

print(f'Found {len(unique(birch_clusters))} clusters')

# create scatter plot for samples from each cluster
for cluster in unique(birch_clusters):
    # get row indexes for samples with this cluster
    row_ix = np.where(birch_clusters == cluster)
    # create scatter of these samples
    plt.scatter(data_values[row_ix, 0], data_values[row_ix, 1])
# show the plot
plt.show()
ss_avg = ss(data_values,birch_clusters)
print("The silhouette_score is :", ss_avg)

ch_avg = ch(data_values,birch_clusters)
print("The calsinki harabasz score is :", ch_avg)
```

Here the threshold parameter is determined by understanding the data. The current random data is within the range of -4 to 4 and the average difference between the data was 0.11 hence the selection of threshold is taken in that range. To choose the value of the branching factor the formula is given by Branching factor = Sqrt (Samples/Threshold).

With the following parameters the results are seen as:

```
Found 5 clusters
```




```
The silhouette_score is: 0.40444847267241035
The calsinki harabasz score is: 1635.2712825630358
```

There are 5 clusters formed as seen in the image. We can see how many points lie in each cluster using the following code:

```python
cluster_counts = Counter(birch_clusters)
for cluster in cluster_counts.items():
    print(f"Cluster {cluster}")
```

```
Cluster (4, 117)
Cluster (0, 479)
Cluster (3, 512)
Cluster (1, 516)
Cluster (2, 376)
```

3.4 Compare and contrast the results of all three algorithms (k-means, DBSCAN, your choice) on the same dataset.

The following differences are seen between the three algorithms (Tomar & Sharma, 1970) (*What is the difference between K-means and DBSCAN*) :-

Parameter	KMEANS	DBSCAN	BIRCH
Type	Centroid-based	Density-based	Hierarchical
Parameters	Number of clusters to be made(k), initialization method (random/k-means++), maximum number of iterations to avoid local maxima	Eps – minimum distance between points, min_samples (total number of points in the vicinity)	Threshold (maximum radius of a sub cluster), branching factor, number of clusters
Noise Handling	Assigns all the points to the nearest cluster and does not handle outliers	Handles outliers and assigns them all to a cluster assigned by -1	Requires tuning of parameters to handle noise and outliers
Data Size	Can be used with large datasets	Difficult for the large dataset as it is density scanned so larger data needs more resources	Can easily work with large data as tree structures are used to traverse through the clusters.
Predetermined Clusters	Requires a k value for the number of clusters needed	As per the data and the input parameters, the number of clusters is decided automatically.	Can work without mentioning the value of k but needs other parameters to do that.
Computation Time	Faster as compared to others but slower when more data is introduced as it traverses manually.	Slow for large datasets.	Fastest as it uses a tree to traverse through the data rather than linearly.

Resources

Arvai, K. (2023) *K-means clustering in Python: A practical guide*, *Real Python*. Real Python. Available at: https://realpython.com/k-means-clustering-python/ (Accessed: April 10, 2023).

Intro to machine learning: Clustering: K-means Cheatsheet (no date) *Codecademy*. Available at: https://www.codecademy.com/learn/machine-learning/modules/dspath-clustering/cheatsheet (Accessed: April 10, 2023).

Mehta, S. (2022) *A tutorial on various clustering evaluation metrics*, *Analytics India Magazine*. Available at: https://analyticsindiamag.com/a-tutorial-on-various-clustering-evaluation-metrics/#:~:text=The%20evaluation%20metrics%20which%20do,index%2C%20Davies%2DBouldin%20Index. (Accessed: April 10, 2023).

Mishra, S. (2020) *K means vs K means++*, *OpenGenus IQ: Computing Expertise & Legacy*. OpenGenus IQ: Computing Expertise & Legacy. Available at: https://iq.opengenus.org/k-means-vs-k-means-p/#:~:text=Both%20K%2Dmeans%20and%20K,dependent%20on%20initialization%20of%20centroid (Accessed: April 10, 2023).

Sklearn.cluster.birch (no date) *scikit*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html (Accessed: April 10, 2023).

Sklearn.cluster.DBSCAN (no date) *scikit*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html (Accessed: April 10, 2023).

Sklearn.cluster.kmeans (no date) *scikit*. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (Accessed: April 10, 2023).

Tomar, R. and Sharma, A. (1970) *K-means and birch: A comparative analysis study*, *SpringerLink*. Springer Nature Singapore. Available at: https://link.springer.com/chapter/10.1007/978-981-19-4960-9_23#:~:text=K%2DMeans%20is%20a%20technique,a%20total%20of%20five%20datasets. (Accessed: April 10, 2023).

What is the difference between K-means and DBSCAN (no date) *Tutorials Point*. Available at: https://www.tutorialspoint.com/what-is-the-difference-between-k-means-and-dbscan#:~:text=K%2Dmeans%20needs%20a%20prototype,influenced%20by%20noise%20or%20outliers. (Accessed: April 10, 2023).