UCD Michael Smurfit
Graduate Business School

Assignment Topic	Cluster Analysis
Name	Anushka Jain
Student No.	22200246
Programme	MSc. Business Analytics
Module Title	Data Management and Mining - 2022/23 Spring
Module Code	MIS41270
Lecturer	Elayne Ruane
Submission Deadline	10th April 2023
Session	2022/23 Spring
Grade/Mark	

A SIGNED COPY OF THIS FORM MUST ACCOMPANY ALL SUBMISSIONS FOR ASSESSMENT. STUDENTS SHOULD KEEP A COPY OF ALL WORK SUBMITTED.

Procedures for Submission and Late Submission

Ensure that you have checked the school's procedures for the submission of assessments.
Note: There are penalties for the late submission of assessments. For further information please see the University's ***Policy on Late Submission of Coursework***, (http://www.ucd.ie/registrar/)

Plagiarism: the unacknowledged inclusion of another person's writings or ideas or works, in any formally presented work (including essays, examinations, projects, laboratory reports or presentations). The penalties associated with plagiarism are designed to impose sanctions that reflect the seriousness of the University's commitment to academic integrity. Ensure that you have read the University's ***Briefing for Students on Academic Integrity and Plagiarism*** and the UCD ***Plagiarism Statement, Plagiarism Policy and Procedures***, (http://www.ucd.ie/registrar/)

Declaration of Authorship

I declare that all material in this assessment is my own work except where there is clear acknowledgement and appropriate reference to the work of others.

Name : Anushka Jain

Date : 10th April 2023

Table of Contents

Theory & Introduction

Clustering Definition:

Cluster Analysis in Data Mining refers to the process of finding groups of objects that are similar to each other but distinct from the objects in other groups. Clustering is a data analytics procedure that divides data sets into groups or classes based on data similarity (upGrad blog, 2020). Since it is used to find clusters from unlabeled data, it thus falls under the category of unsupervised machine learning (www.javatpoint.com, 2011).

Properties of Clustering:

1. Clustering Scalability: Able to form clusters from any size of data.
2. Algorithm usability with multiple types of data (eg: grouped, discontinuous, videos, continuous).
3. Dealing with unstructured data
4. Interoperability: ability of systems to interact. (Appen, 2020)

There are various algorithms as part of Cluster Analysis (www.youtube.com, n.d.):

- Partitioning Method: K-means
- Hierarchical Method: Agglomerative
- Density Based Method: DBSCAN
- Grid Based Methods
- Model Based Methods
- Constraint Based Methods
- Distribution Based Methods: GMM

Partitioning Methods:

Where n data items are mapped to k partitions, n <= k. The partition should satisfy the following 2 conditions:

1. Each partition should have at least 1 object.
2. Each object should belong to only 1 partition.

Task 1.1: K-means

- **Code:**

```
# KMeans Function
def run_kmeans(k, dataset, x_col='', y_col=''):
    # Making a copy of the dataframe so we can freely add columns
    df = dataset.copy(deep=True) # For our use case
    kmeans = KMeans(n_clusters=k, init='random', n_init=1)
    kmeans.fit(dataset) # Compute k-means clustering
    # Compute cluster centers and predict cluster index for each sample.
    cluster_labels = kmeans.fit_predict(dataset)
    df[f'cluster_labels'] = cluster_labels
    plt.scatter(df[x_col], df[y_col], c=kmeans.labels_.astype(float))
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    # Find the coordinates of the centroid
    centroids = kmeans.cluster_centers_
    # Compute the SSE Value to determine which cluster is the best
    print("SSE: ", kmeans.inertia_)
    # Plot the centroids
    plt.scatter(centroids[:,0] , centroids[:,1] , s = 30, color = 'r')
    plt.show()
    return df
```

- **Dataset 1: Salary_data.csv (50 data points)**

The above function was executed 10 times on salary data with k = 3, i.e., we wanted the number of clusters formed to be equal to 3, because this was the most optimal value when referred from the Elbow Plot (Figure 1 and has been discussed in depth in Task 1.4).

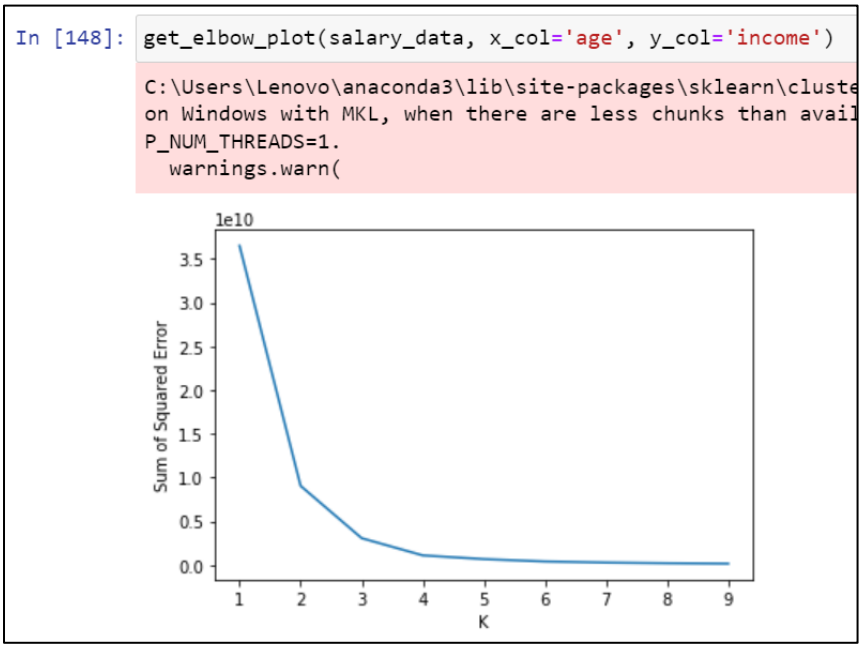
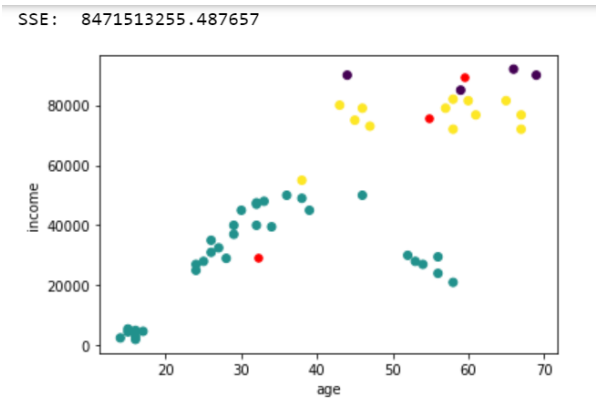


Figure 1: Elbow Plot of Salary Dataset with elbow at k = 3

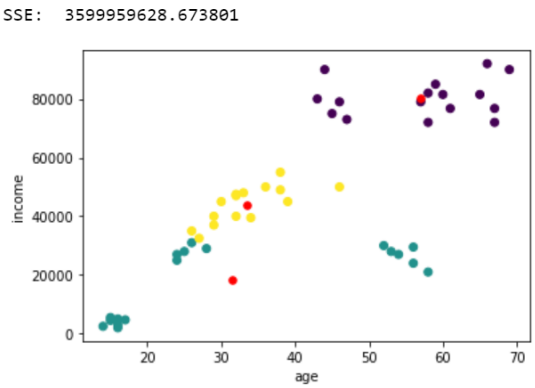
We then get the following graphs with their SSE (sum of squared distance) values and centroids (center of each cluster) highlighted with a red dot. Since the basic idea behind K-Means clustering is defining clusters so that the within-cluster variation (SSE) is minimized (Rao, 2022). In the Elbow Plot, we plot the Within-Cluster-Sum of Squared Errors (WSS) for different number of clusters (k) and, select the k for which change in WSS first starts to diminish (Baruah, 2020).

Significance of SSE: We have used the value of SSE as a measure to differentiate between the clusters obtained by the prediction model as the lower the value of SSE, the more similar are the data points within their cluster (Cluster Analysis 4 Marketing, n.d.).

RUN 1



RUN 2

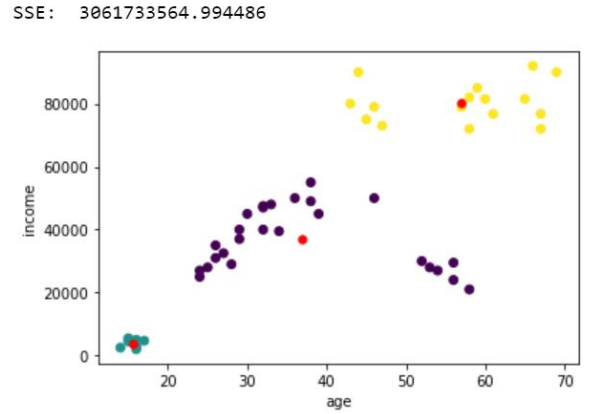


Run 1: Here we can see how randomly 3 data points were assigned as centroids, marked with red dots for each of the 3 clusters (represented in purple, yellow and cyan colors). The SSE with value = 8471513255, is the highest for this run.

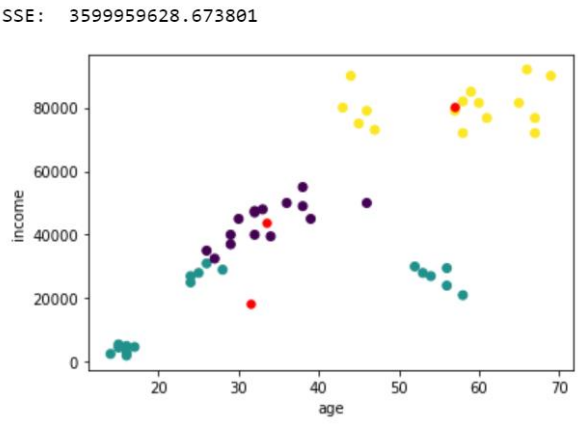
Run 2, 4, 7 and 8: As new centroids are assigned and SSE is calculated, the value has dropped to 3599959628.673801. We have equivalent values of SSE and same centroids for Run 2, 4, 7 and 8.

Run 3, 5, 6 and 9: However, the values of SSE reduce even further to 3061733564.994486 for Run 3, 5, 6 and 9. Runs with similar values of SSE have been color coded.

RUN 3

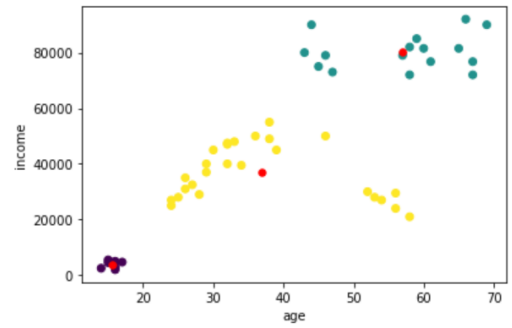


RUN 4



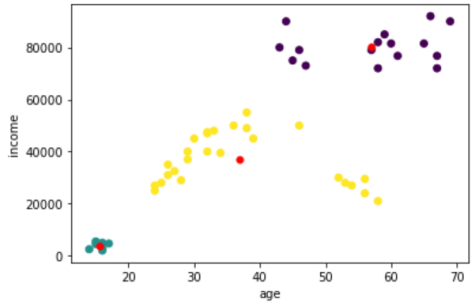
RUN 5

SSE: 3061733564.994486



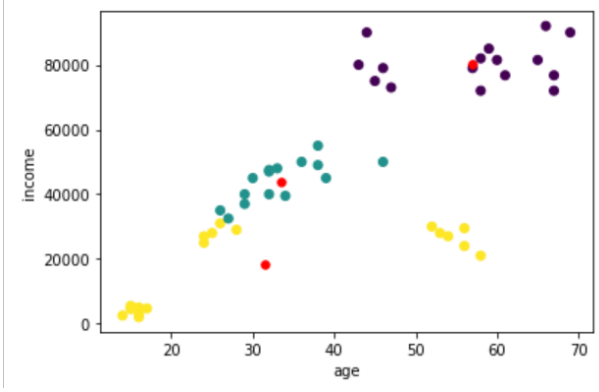
RUN 6

SSE: 3061733564.994486



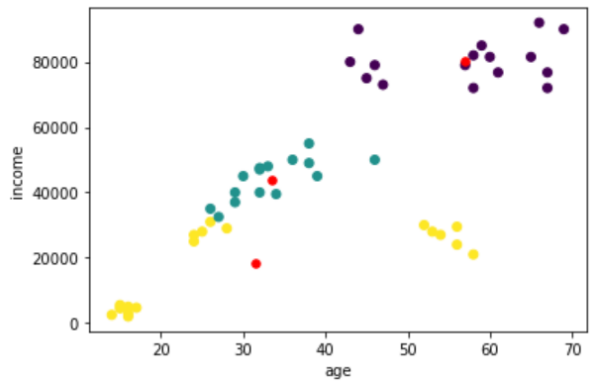
RUN 7

SSE: 3599959628.673801



RUN 8

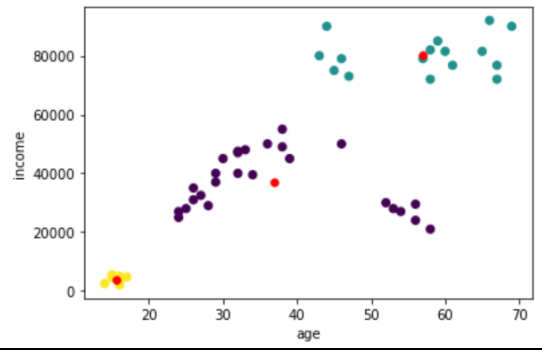
SSE: 3599959628.673801



Run 10: Finally, we see that within 10 runs, the K-Means algorithm has refined and converged to the most suitable partitions or clusters with the lowest SSE value of 3054334948.967023.

RUN 9

SSE: 3061733564.994486



RUN 10

SSE: 3054334948.967023

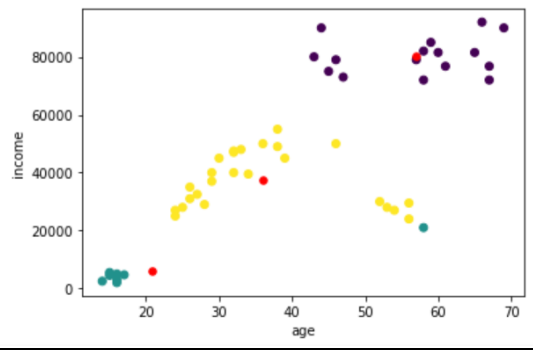


Table 1: Clustering charts from Salary_data.csv

If it weren't for the SSE values, one would assume that the Runs 3, 5, 6 and 9 are the best suitable clusters as they have 3 distinctive clusters created, but from Run 10, it is evident that including one data point with age ~ 58 and income ~ 20K should be included with the cluster of age group of 0-20 years.

- **Dataset 2: Random Data (50 data points).**

For the next part of this task, a random data set with 50 data points was generated using the below code and was then plotted.

```
def generate_random_data(n):
    return pd.DataFrame({'x':np.random.rand(n), 'y':np.random.rand(n)})
# Generating a random data set with 50 data points
rand_data = generate_random_data(50)
plt.scatter(rand_data.x, rand_data.y)
```

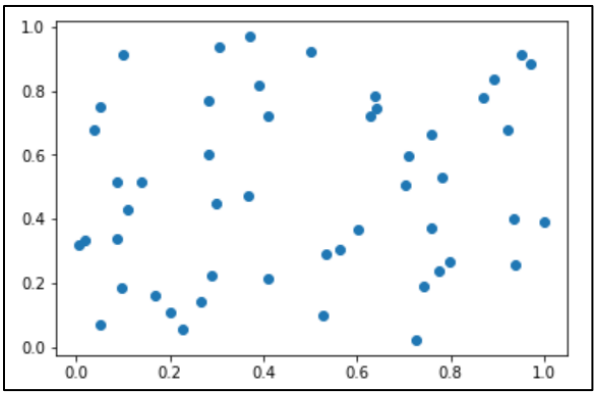


Figure 2: Scatter Plot of random data set

It is evident from the above scatter plot, that the data points for this are spread across a wide area and not clustered as clearly as for the salary dataset. For the next part, we executed the same K-Means function but with value of k as 4. This was done because this value seemed the most optimal from the Elbow-Method (figure 3).

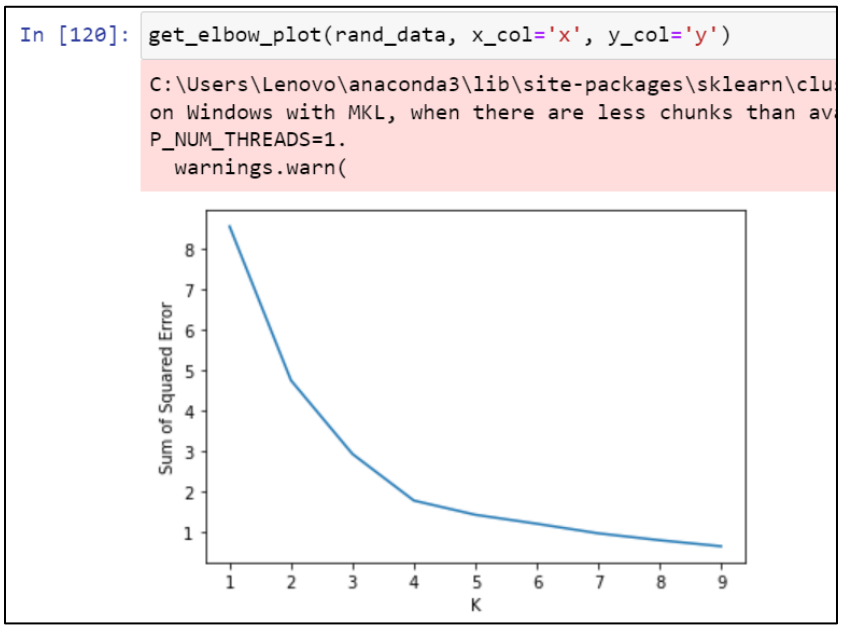


Figure 3: Elbow Plot of Random Dataset with prominent elbow at k = 4

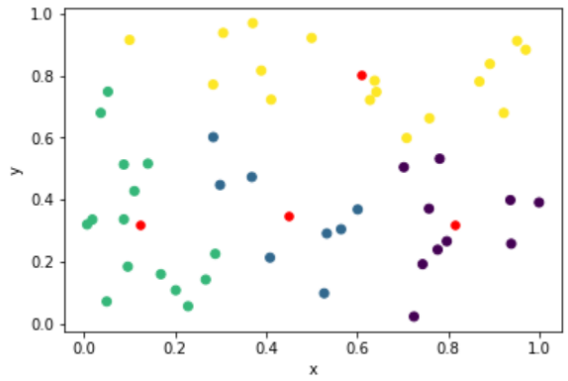
Since the plot is relatively flat, it suggests that there may not be a clear clustered structure in the data.

Run 1: Like our above discussion we see, that for Run 1, when 4 datapoints are assigned as centroids randomly, the value of SSE is quite high as it is equal to 2.6924514936905144.

Run 4 and 9: 2 out of 10 runs formed a partition in such a way that the value of SSE comes out to be 1.7782133968318512.

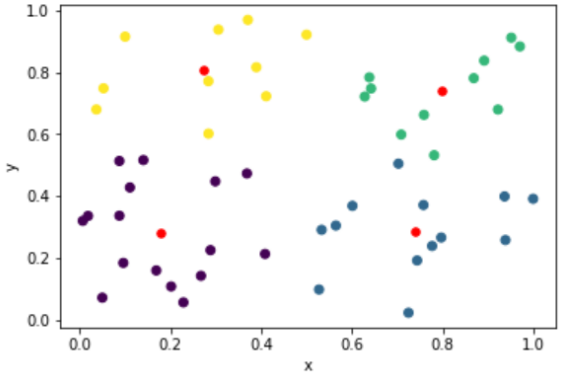
Run 1

SSE: 2.6924514936905144



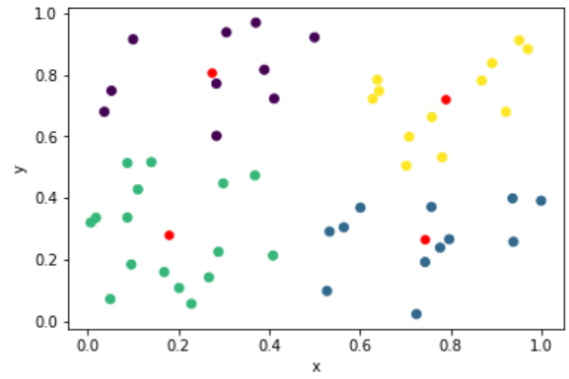
Run 2

SSE: 1.7730999463552877



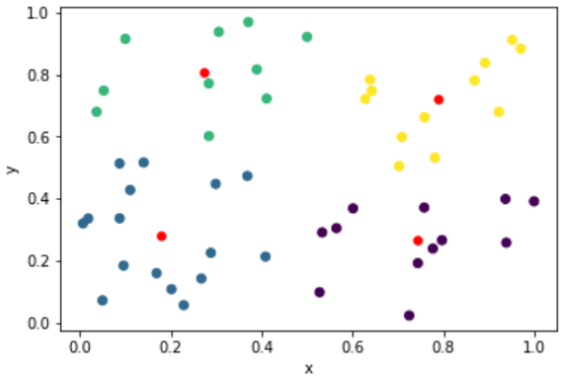
Run 3

SSE: 1.7782133968318512



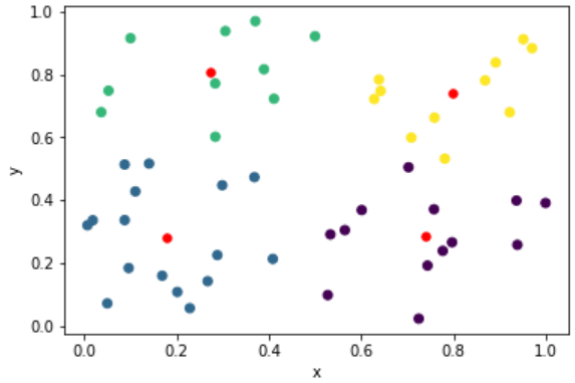
Run 4

SSE: 1.7782133968318512



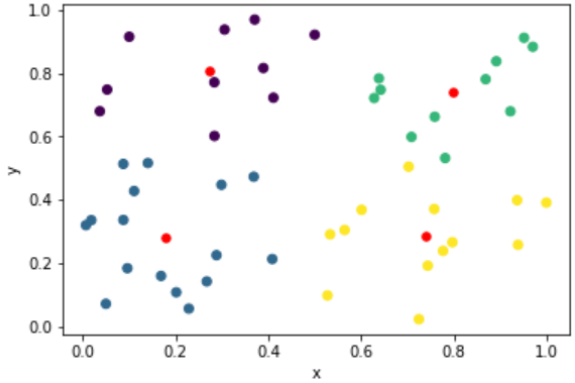
Run 5

SSE: 1.7730999463552877



Run 6

SSE: 1.7730999463552877



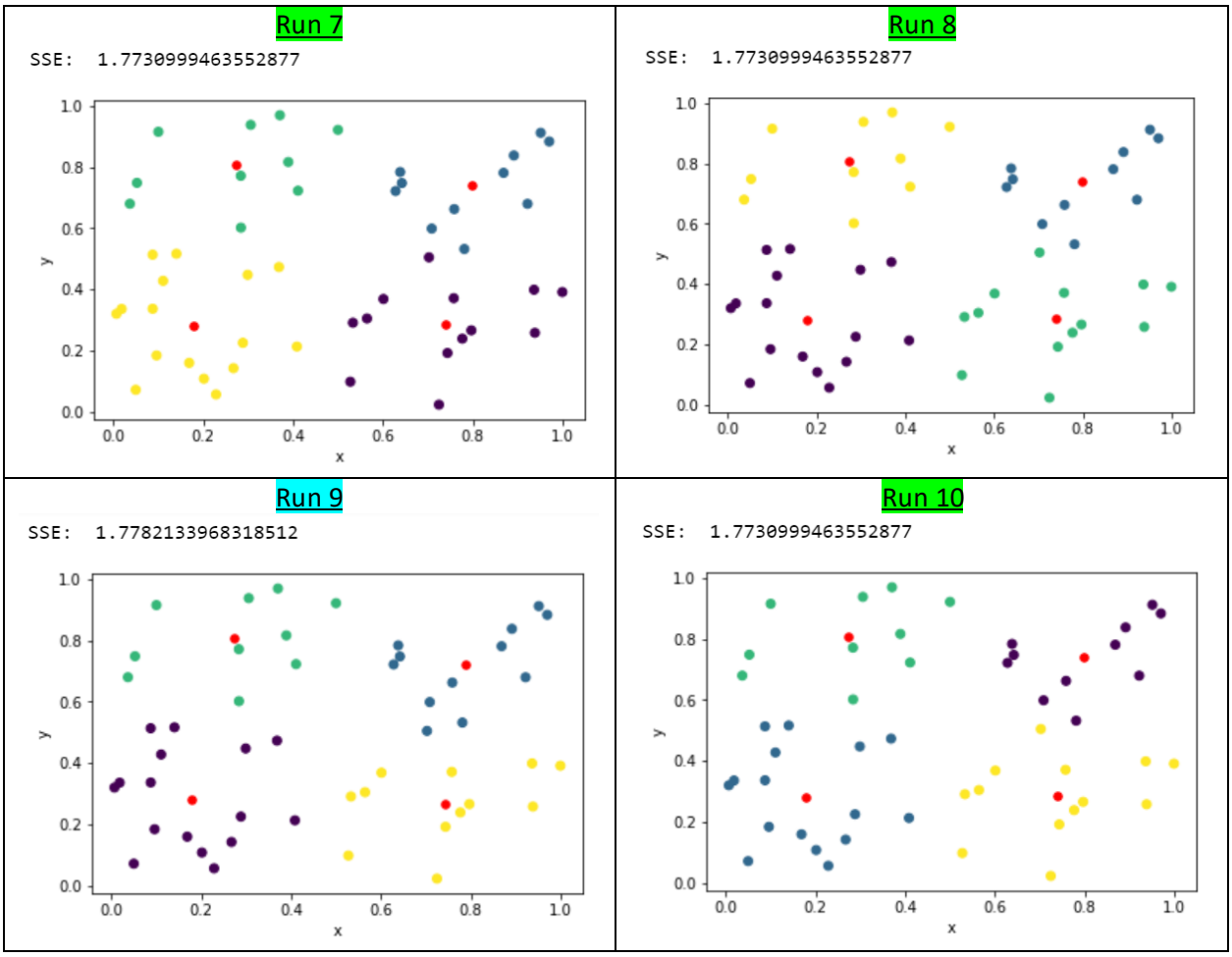


Table 2: Clustering charts from randomly generated dataset of 50 data points

Run 2, 3, 5, 6, 7, 8 and 10: Here, we saw that 7/10 runs gave the most optimal clustering of the random data with the lowest SSE value of 1.7730999463552877. Thus, these clusters are the most optimal representation of partitioning of unlabeled data.

Interpretation: Since the data is too widespread, clusters formed are not very distinct. Not much can be inferred from this data. Moreover, the randomly generated unlabeled data has no characteristics or truth labels, which is why not much can be interpreted from the clusters. Finally, the elbow-plot had also suggested that there may not be clear clustered structure in the data.

Task 1.2: Understanding of Parameters in K-means

```
In [112]: # KMeans Function
          def run_kmeans(k, dataset, x_col='', y_col=''):
              # Making a copy of the dataframe so we can freely add columns
              df = dataset.copy(deep=True) # For our use case
              kmeans = KMeans(n_clusters=k, init='random', n_init=1) # Random because KMeans
              kmeans.fit(dataset) # Compute k-means clustering
              cluster_labels = kmeans.fit_predict(dataset) # Compute cluster centers and predict cluster index for each sample.
              df[f'cluster_labels'] = cluster_labels
              plt.scatter(df[x_col], df[y_col], c=kmeans.labels_.astype(float))
              plt.xlabel(x_col)
              plt.ylabel(y_col)
              centroids = kmeans.cluster_centers_ # Find the coordinates of the centroid
              print("SSE: ", kmeans.inertia_) # Compute the SSE Value to determine which cluster is the best
              plt.scatter(centroids[:,0] , centroids[:,1] , s = 30, color = 'r') # PLot the centroids
              plt.show()
              return df
```

Highlighting the part to be explained:

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

- **n_clusters = k**

This defines the number of clusters to be formed from the given data points. This is an integer type value, and the default value is 8 (scikit-learn.org, n.d.). We can define k as any number. We have already seen the clusters obtained when k was 3. For more understanding, I kept value of k as 4 from salary dataset and the clusters were formed as:

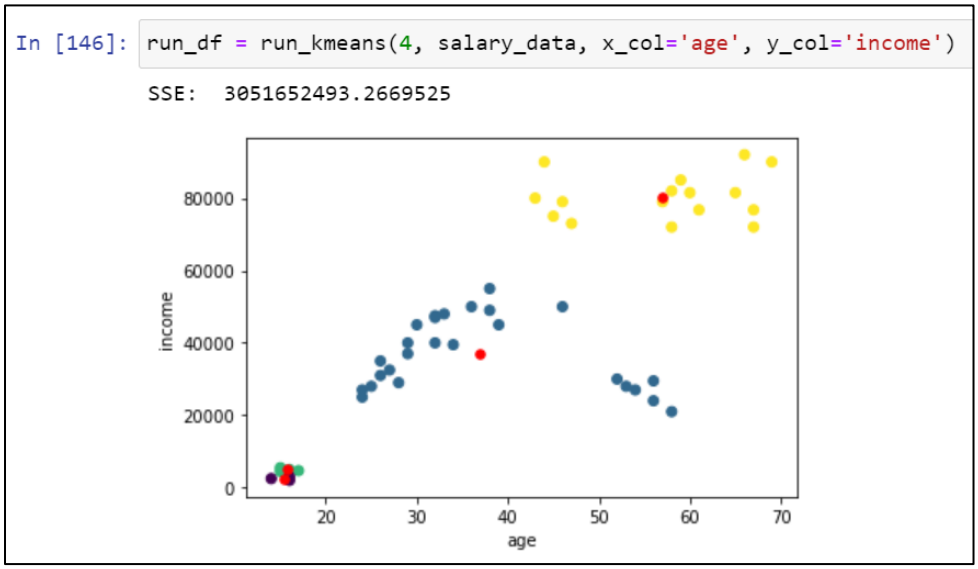


Figure 4: Forming 4 clusters using 'random' init method on Salary Dataset

We see how for the first run, when number of clusters we wanted were 4, how randomly 4 datapoints have been assigned as centroids, with 2 data points lying pretty close to each other.

- **init = 'random' or 'k-means++'**

This is a method which is used for initialization of the centroids of the clusters. If the value is 'random', then n_clusters number of centroids are chosen at random from the given dataset. Else if the value is

'k-means++', then the initial cluster of centroids is chosen via sampling so as to refine the process of clustering. After implementing the 'k-means++' method in task 1.3, we see how we have predicted the optimal partitions of the data in one run (least value of SSE), while it took various runs in the former case. Basically, this method accelerated the convergence (scikit-learn.org, n.d.). 'k-means++' is also the default value of the init method.

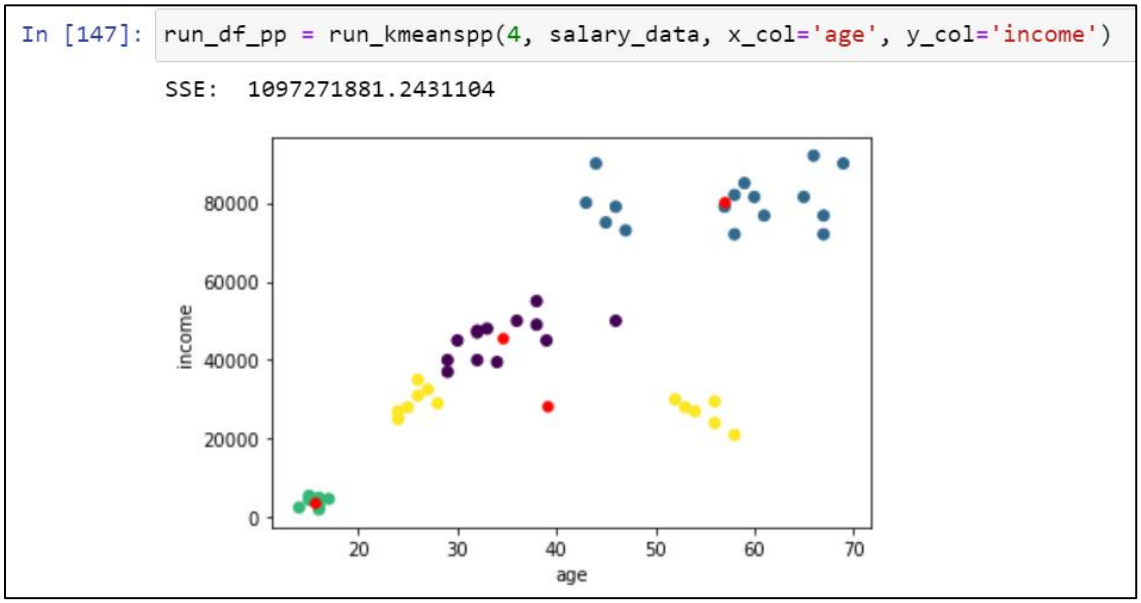


Figure 5: Forming 4 clusters using 'k-means++' init method on Salary Dataset

We see how in just one execution, where we wanted 4 clusters, the greedy k-means++ algorithm has optimized the partitioning or formation of clusters with the lowest possible SSE value.

- **n_init = 1**

This parameter defines the number of times the algorithm will run with different data points as centroids (scikit-learn.org, n.d.). This is an integer value which if not specified depends on the algorithm selected in the init parameter. If the value of init is 'random' then the default value of n_init is 10, since it requires 10 runs to find the optimal partitioning, while if the value of init is 'k-means++' then the value of n_init is 1 as it uses greedy algorithm and given optimal clusters within 1 execution (scikit-learn.org, n.d.).

Most importantly, the value of n_init is set for high values like 10 for 'random' init methods **to avoid finding a sub-optimal local minimum** (scikit-learn, n.d.).

Task 1.3: Clusters with k-means++ algorithm

Code K-Means++

```
# KMeans++ Clustering Algorithm
def run_kmeanspp(k, dataset, x_col='', y_col=''):
    df = dataset.copy(deep=True)
    # Modified the init method
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=1)
    kmeans.fit(dataset)
    cluster_labels = kmeans.fit_predict(dataset)
    df[f'cluster_labels'] = cluster_labels
    plt.scatter(df[x_col], df[y_col], c=kmeans.labels_.astype(float))
    plt.xlabel(x_col)
    plt.ylabel(y_col)
    centroids = kmeans.cluster_centers_
    # Print the SSE Value
    print("SSE: ", kmeans.inertia_)
    # Plotting Centroids
    plt.scatter(centroids[:,0] , centroids[:,1] , s = 30, color = 'r')
    plt.show()
    return df
```

After executing the above code once on our Salary dataset with k = 3, we can see the below clusters with the SSE value as **3054334948.967023.**

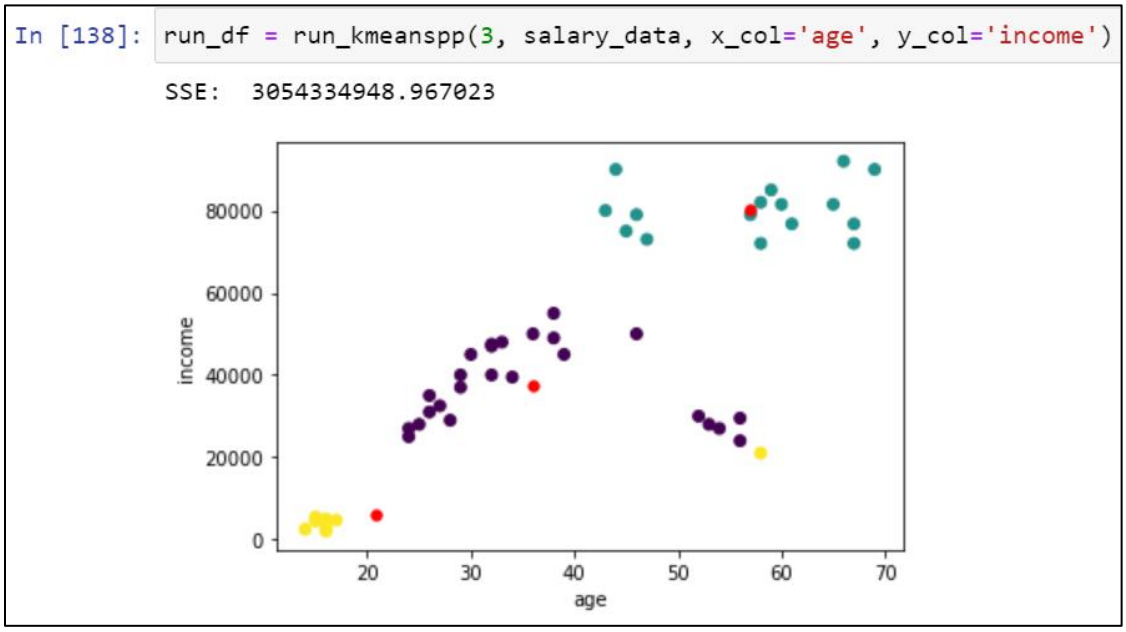


Figure 6: Clusters formed using 'k-means++' init method & k=3 on Salary Dataset

Attaching cluster formation and SSE value from 10th Run of kmeans algorithm without the use of greedy k-means++ algorithm in figure 7:

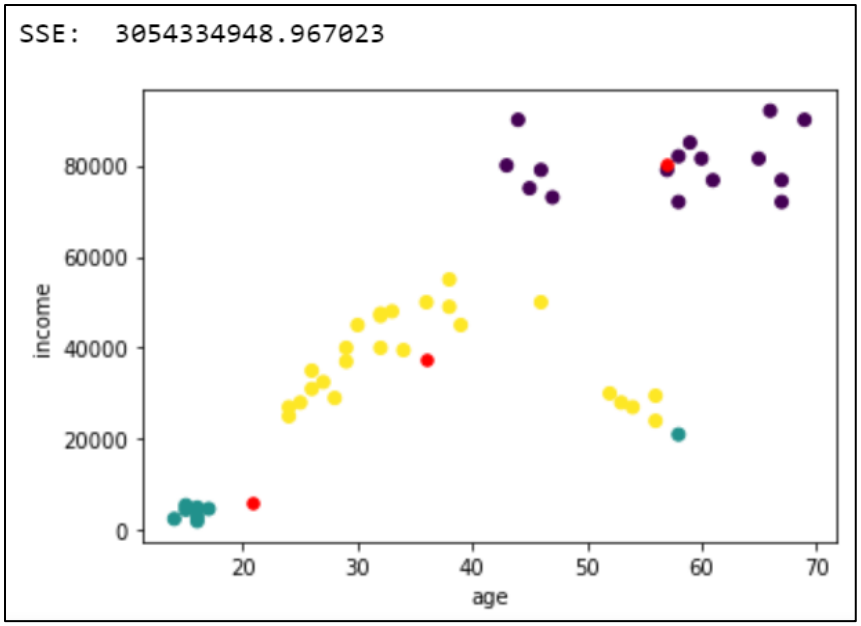


Figure 7: Clusters formed in the 10th run, without 'k-means++' init method on Salary Dataset

By comparing the two images, we can infer that the convergence was accelerated when we implemented the 'k-means++' initialization approach, as we obtained the optimum cluster partitioning in just one execution. The SSE value is the same, which makes sense because the clusters and centroids with the minimum value of sum of squared distance is the one where the data points are the most similar to the ones in their own cluster. Additionally, in the initial execution of the kmeans algorithm with init='random,' we obtained the clusters below with a very high SSE value of 8471513255, nearly 2.5 times the lowest SSE value. As a result, we can observe how the greedy k-means++ method accelerated convergence.

Thus, we can infer that kmeans++ is more efficient than kmeans algorithm.

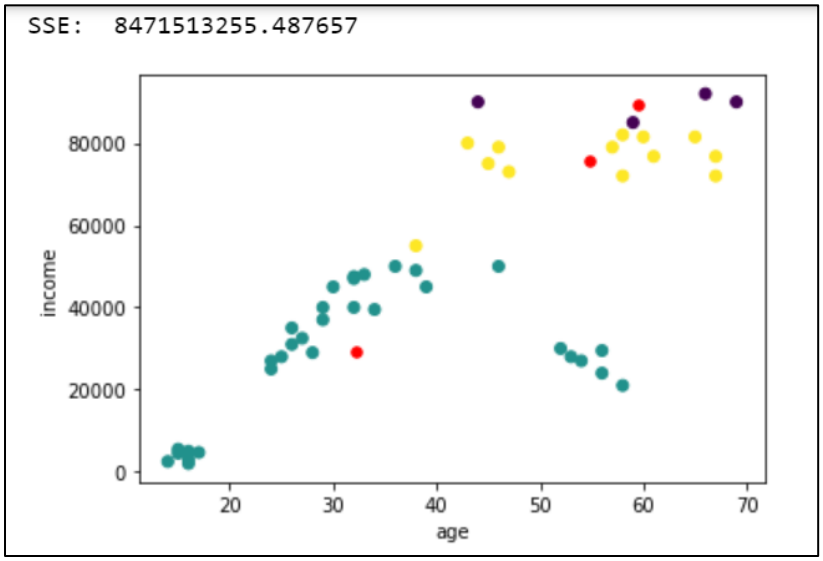


Figure 8: Clusters formed in the first run, without 'k-means++' init method on Salary Dataset

Task 1.4: Elbow Plot of somewhat clearly clustered dataset.

To create a random dataset of 30 data points, I have used the make_blob method which was imported from sklearn.datasets.

```
In [238]: # using make_blobs to generate dataset
          n_samples = 30 # number of data points
          random_state = 150 # to have reproducable data points
          X_varied, y_varied = make_blobs(n_samples=n_samples, cluster_std=[1.0, 1.5, 1.5], random_state=random_state)
          plt.scatter(X_varied[:, 0], X_varied[:, 1])
          plt.show()
```

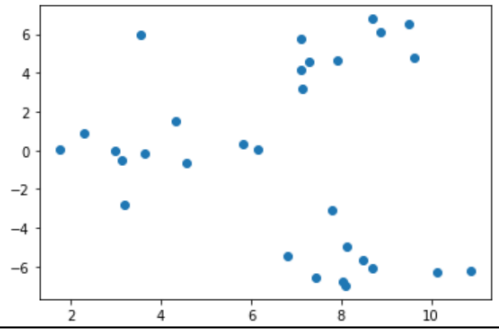


Figure 9: Scatter Plot of 30 random points

The **random_state** parameter in the **make_blobs** function is used to set the random seed. By setting the random seed, you can ensure that the generated data points are reproducible. This means that if you run the **make_blobs** function with the same random seed multiple times, you will get the same set of data points each time (scikit-learn, n.d.). The standard deviation of each cluster is different to create a somewhat clustered dataset. Elbow Plot was then generated for the same:

```
In [239]: # generate an elbow plot for the data
          get_elbow_plot(X_varied, x_col='x', y_col='y')
```

```
C:\Users\Lenovo\anaconda3\lib\site-packages\sklearn\cluster\_kmean
on Windows with MKL, when there are less chunks than available thr
P_NUM_THREADS=1.
  warnings.warn(
```

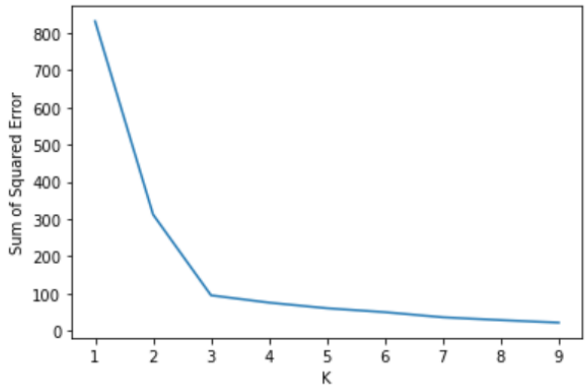


Figure 10: Elbow Plot for the above generated 30 random points

Meaning of Elbow Plot:

An elbow plot is used to calculate the ideal number of clusters in a clustering process. It aids in balancing the trade-off between minimizing the WCSS (within-cluster sum of squares) and minimizing the number of clusters, as well as providing insights into the data's structure (Tomar, 2022). The WCSS is the sum of the squared distances between each data point and its assigned cluster center. It is a measure of the variance within each cluster, and it decreases as the number of clusters increases. The goal of clustering is to minimize the WCSS, while also minimizing the number of clusters.

"elbow point" - the point where the rate of decrease in WCSS starts to slow down.

Interpretation:

Since there is a steep fall in figure 10 and from the above meaning of “elbow-point”, it can be inferred that k = 3 is the optimal number of clusters for this data. The shape also provides insights into the structure of the data like the steep drop suggests that there is a natural clustering in the data. Since, the elbow plot suggests this, I would use this information to use K-Means as the Partition Method for Clustering Analysis.

Task 2: Random Dataset with 3000 samples

I used the method *make_classification*, to generate clustered data of 3000 samples with predicted number of clusters = 4 (n_classes * n_clusters_per_class = 2*2) (scikit-learn, n.d.). Since it is random generated data, we do not have the truth labels of the data.

```
# define dataset with 3000 samples
data_values, class_labels = make_classification(n_samples=3000, n_features=2,
n_informative=2, n_redundant=0, n_clusters_per_class=2, random_state=None)
```

I then plotted the scatter plot and finalized the one which looked the most interesting to me:

```
In [103]: # create scatter plot for samples from each class
          for class_value in range(len(unique(class_labels))):
              # get row indexes for samples with this class
              row_ix = np.where(class_labels == class_value)
              # create scatter of these samples
              plt.scatter(data_values[row_ix, 0], data_values[row_ix, 1])
          # show the plot
          plt.show()
```

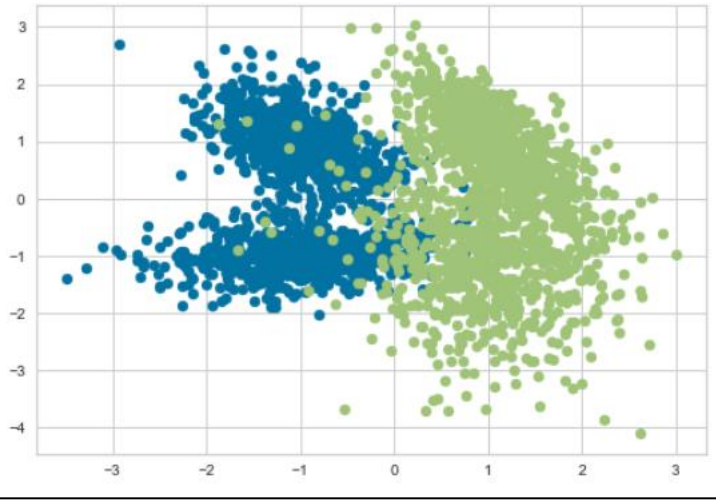


Figure 11: Scatter Plot for the above generated 3000 random points

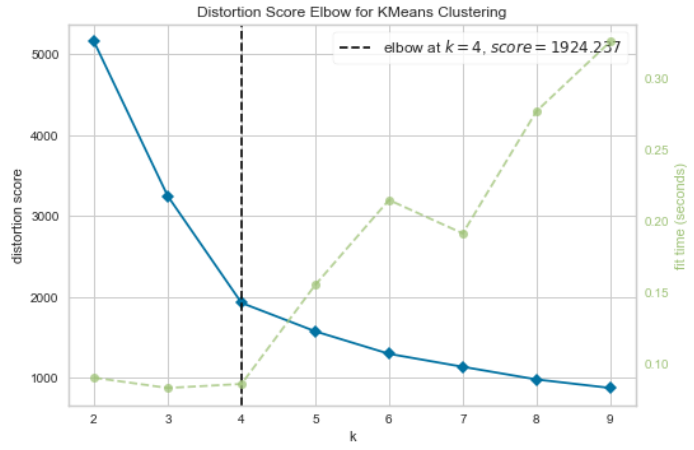
Task 2.1: K-Means on a dataset with 3000 samples

After my learnings from task 1, I implemented the K-Means algorithm using the greedy k-means++ algorithm and *random_state* as 10, because I wanted to fix the randomness with which it was choosing the centroids for initialization (scikit-learn.org, n.d.). I did it in an attempt to be able to run it multiple times and get the same randomness.

But before I implemented the algorithm to view the clusters, I executed the elbow plot to view its score for various values of k. The code has been referred from the source (IDB-FOR-DATASCIENCE, 2020).

```
In [106]: # Elbow Method for K means
          from sklearn.cluster import KMeans
          # Import ElbowVisualizer
          from yellowbrick.cluster import KElbowVisualizer

          cluster_df = data_values
          # model = KMeans(random_state=10)
          model = KMeans(n_clusters=4, init='k-means++', random_state =10)
          # k is range of number of clusters.
          visualizer = KElbowVisualizer(model, k=(2,10), timings= True)
          visualizer.fit(cluster_df)        # Fit the data to the visualizer
          visualizer.show()        # Finalize and render the figure
```




```
Out[106]: <AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>
```

Figure 12: Elbow Plot for the above generated 3000 random points

From figure 12, the elbow point is achieved with 4 clusters (as expected from the *make_classification* generated data). The green line represents how much time was required to plot the model for various values of k (Baruah, 2020). The variation in the values of WSS vary rapidly but then they seemed to slow down, which lead to the formation of “elbow”. This point occurs for us at k=4. Thus, after selecting the value of k, I ran the k-means algorithm as provided in lecture:

```
In [107]: kmeans_model = KMeans(n_clusters=4, init='k-means++', random_state=10) # setting k = 4
kmeans_model.fit(data_values)
kmeans_clusters = kmeans_model.predict(data_values)

# create scatter plot for samples from each cluster
for cluster in unique(kmeans_clusters):
    # get row indexes for samples with this cluster
    row_ix = np.where(kmeans_clusters == cluster)
    # create scatter of these samples
    plt.scatter(data_values[row_ix, 0], data_values[row_ix, 1])
# show the plot
plt.show()
```

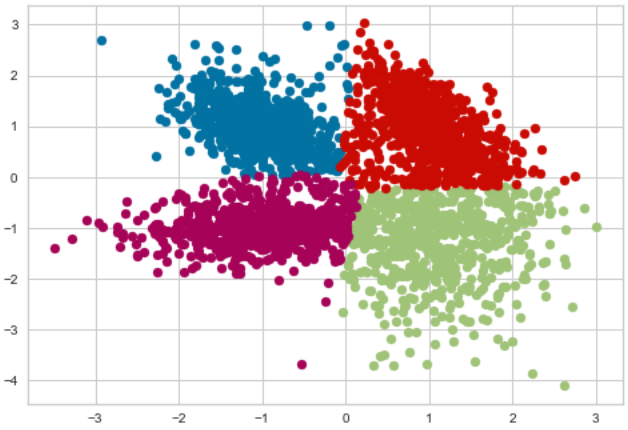
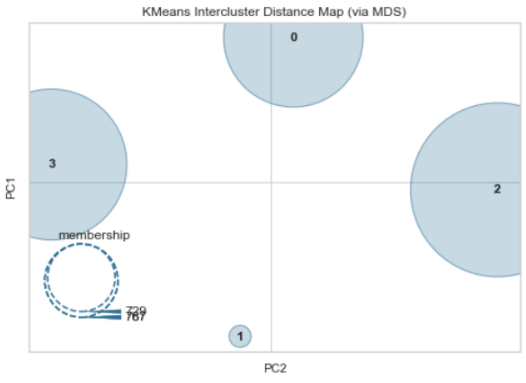


Figure 13: Modelling of K-Means

I then evaluated the relative size and distribution of the clusters using the inter-cluster distance map (Baruah, 2020):

```
In [109]: # Inter cluster distance map
from yellowbrick.cluster import InterclusterDistance
cluster_df = data_values
# Instantiate the clustering model and visualizer
visualizer = InterclusterDistance(kmeans_model)
visualizer.fit(cluster_df)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```




```
Out[109]: <AxesSubplot:title={'center':'KMeans Intercluster Distance Map (via MDS)'}, xlabel='PC2', ylabel='PC1'>
```

Figure 14: Inter-cluster distance map for K-Means Algorithm with k=4

It can be interpreted that 3 out of 4 clusters are quite large when compared to the remaining 1. It appears that Cluster 2 is the biggest. All of the clusters appear to be reasonably separated from one another.

Task 2.2: DBSCAN on a dataset with 3000 samples

DBSCAN: **D**ensity **B**ased **S**patial **C**lustering of **A**pplication with **N**oise.

It is a density-based algorithm where objects are clustered based on density (scikit-learn, n.d.). This means that the data points which are packed closely are grouped together, while the ones lying in low-density areas are termed as outliers. It has 2 inputs:

1. **eps**: radius of circle formed with a data point as center.
2. **min_samples**: minimum number of datapoints that should be inside the circle to form a dense region.

Now, these parameters need to be calculated, before they can be used to run the DBSCAN Algorithm.

- **eps:**

```
In [112]: # parameter tuning for eps
          from sklearn.neighbors import NearestNeighbors
          cluster_df = data_values # Assign Random DataSet value to cluster_df
          nearest_neighbors = NearestNeighbors(n_neighbors=4)
          neighbors = nearest_neighbors.fit(cluster_df)
          distances, indices = neighbors.kneighbors(cluster_df)
          distances = np.sort(distances[:,3], axis=0)

          from kneed import KneeLocator
          i = np.arange(len(distances))
          knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp_method='polynomial')
          fig = plt.figure(figsize=(5, 5))
          knee.plot_knee()
          plt.xlabel("Points")
          plt.ylabel("Distance")

          print(distances[knee.knee])
```

```
0.12356729733404877

<Figure size 360x360 with 0 Axes>
```

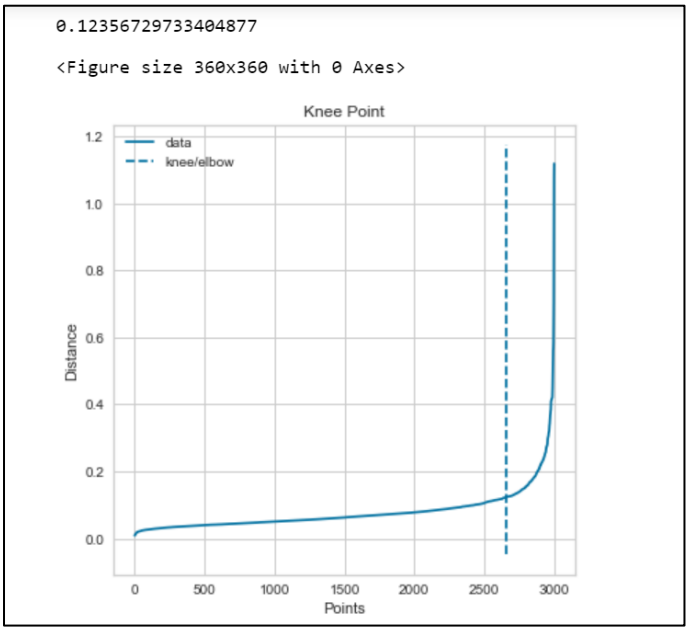


Figure 15: Optimal Value of eps for DBSCAN

I referred to the code from (IDB-FOR-DATASCIENCE, 2020). I have used the knee-plot to estimate the value of the parameter *eps*. From the output highlighted in figure 15, it is evident that the value of eps which will be used to calculate the other parameter *min_samples* is **0.12356729733404877.**

- **min_samples:**

I then used the value of *eps* from above into the Silhouette Score, to estimate the value of *min_samples*.

```
# Silhouette Score for DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.cluster import DBSCAN

def get_dbscan_score(data, center):
    '''
    INPUT:
        data - the dataset you want to fit kmeans to
        center - the number of centers you want (the k value)
    OUTPUT:
        score - the Silhouette Score for DBSCAN
    '''
    #instantiate kmeans
    dbscan = DBSCAN(eps= 0.12356729733404877, min_samples=center)

    # Then fit the model to your data using the fit method
    model = dbscan.fit(cluster_df)

    # Calculate Silhoutte Score

    score = silhouette_score(cluster_df, model.labels_, metric='euclidean')

    return score

scores = []
centers = list(range(2,30))
cluster_df = data_values

for center in centers:
    scores.append(get_dbscan_score(cluster_df, center))

plt.plot(centers, scores, linestyle='--', marker='o', color='b');
plt.xlabel('min_samples');
plt.ylabel('Silhouette Score');
plt.title('Silhouette Score vs. min_samples');

df3 = pd.DataFrame(centers,columns=['min_samples'])
df3['scores'] = scores
df4 = df3[df3.scores == df3.scores.max()]
print('Optimal number of min_samples based on silhouette score:',
df4['min_samples'].tolist())
```

I got the below plot:

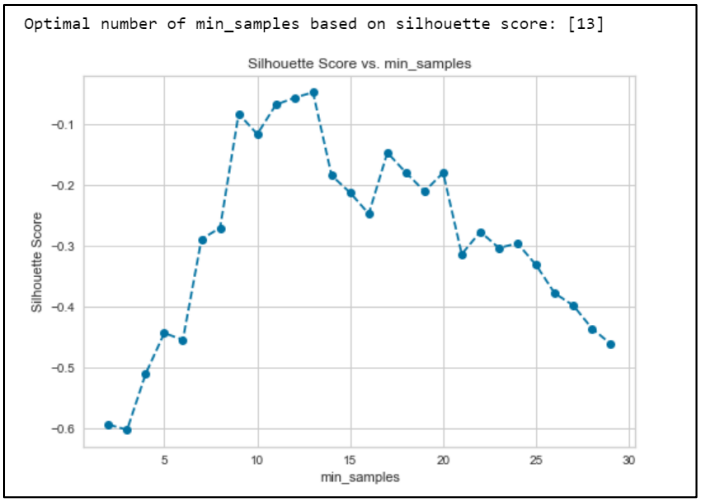


Figure 16: Optimal Value of min_samples for DBSCAN using Silhouette Score

The optimal number of *min_samples* obtained from the Silhouette Score comes out to be 13.

Using above values of the parameters in DBSCAN Model and found 15 clusters.

```
In [141]: dbscan_model = DBSCAN(eps=0.12356729733404877, min_samples = 13)
          dbscan_clusters = dbscan_model.fit_predict(data_values)

          print(f'Found {len(unique(dbscan_clusters))} clusters')
          # create scatter plot for samples from each cluster
          for cluster in unique(dbscan_clusters):
              # get row indexes for samples with this cluster
              row_ix = np.where(dbscan_clusters == cluster)
              # create scatter of these samples
              plt.scatter(data_values[row_ix, 0], data_values[row_ix, 1])
          # show the plot
          plt.show()
```

```
Found 15 clusters
```

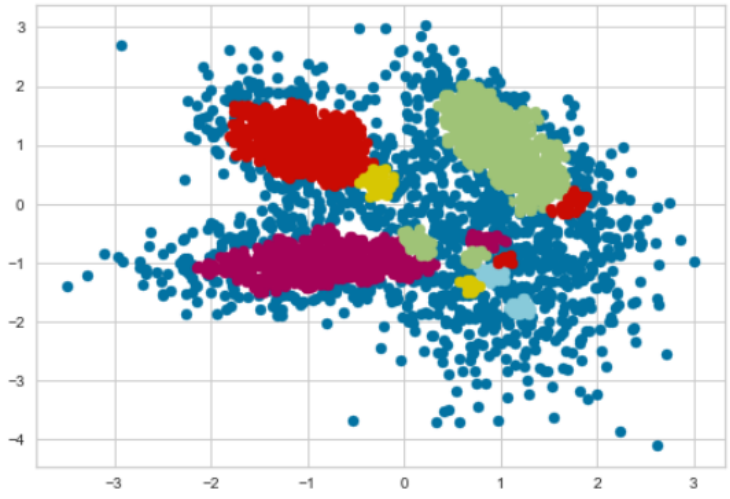


Figure 17: Cluster distribution in DBSCAN Model

Task 2.3: Gaussian Mixture on a dataset with 3000 samples

For my third task, I went with the Gaussian Mixture Modeling. It is a probabilistic model based on distance that makes the assumption that all data points are produced by a linear combination of multivariate Gaussian distributions with unknown parameters (Baruah, 2020). It takes into account the covariance structure along with latent Gaussian Distribution. In this model, we don't need to select the optimal number of clusters, only the value of *n_component*.

I have decided to choose full covariance type i.e., each distribution has its own general covariance matrix. To calculate the value of *n_components,* I have executed the BIC Score when referred to the code from (IDB-FOR-DATASCIENCE, 2020).

In [142]:
```
# BIC for GMM
from sklearn.mixture import GaussianMixture
n_components = range(1, 30)
covariance_type = ['spherical', 'tied', 'diag', 'full']
score=[]
cluster_df = data_values

for cov in covariance_type:
    for n_comp in n_components:
        gmm=GaussianMixture(n_components=n_comp,covariance_type=cov, random_state = 10)
        gmm.fit(cluster_df)
        score.append((cov,n_comp,gmm.bic(cluster_df)))
score_1 = pd.DataFrame(score)
score_1.columns = ['Covariance_Type', 'N_Components','BIC_Score']
score_2 = score_1[score_1.BIC_Score == score_1.BIC_Score.min()]

score_2.head(n=2)
```

Out[142]:

	Covariance_Type	N_Components	BIC_Score
90	full	4	17040.186421

From above, the value of *n_components* come out to be 4. Using this value in the modelling of GMM:

In [143]:
```
from sklearn.mixture import GaussianMixture
# define the model
# model = GaussianMixture(n_components=4,covariance_type= "full", random_state = 10)
model = GaussianMixture(n_components=4)

data = data_values

# fit the model
model.fit(data)
# assign a cluster to each example
clusters = model.predict(data)
# retrieve unique clusters
uniq_clusters = np.unique(clusters)
print(f'Found {len(uniq_clusters)} clusters')

# create scatter plot for samples from each cluster
for cluster in uniq_clusters:
    # get row indexes for samples with this cluster
    row_ix = np.where(clusters == cluster)
    # create scatter of these samples
    plt.scatter(data[row_ix, 0], data[row_ix, 1])

plt.show()
```

```
Found 4 clusters
```

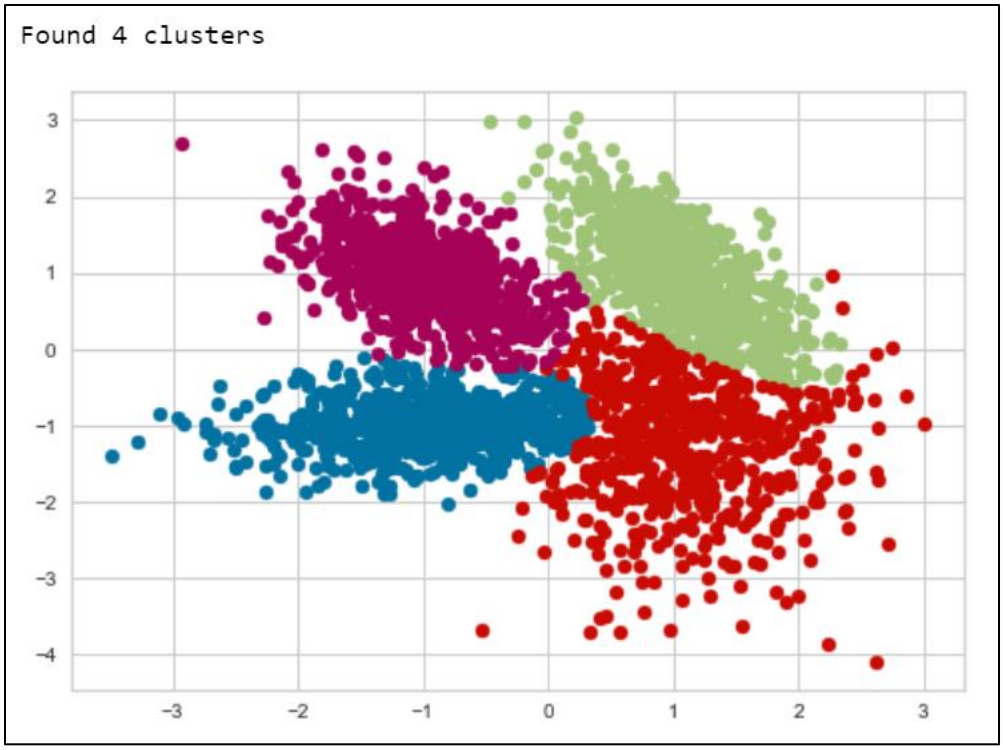


Figure 18: Cluster distribution in Gaussian Mixture Model

4 clusters are found when modelled Gaussian Mixture. This is as anticipated from the *make_classification* method. I could have also used the value of *n_components* from the Elbow-Plot, used in Task 2.1. One thing I noticed while running these algorithms was that GMM took more time to compute and plot the clusters.

Our aim is to find logical segmentation of data. If this were a real dataset with truth labels, we would have been easily able to characterize the data into various segments which would have made some business sense. Moreover, the cluster formation is very similar to that of K-Means where the data is at the outskirts in the scatter plot, while it is different for the highly dense region and has shifted clusters, especially the data points in the middle.

Task 2.4: Comparing the 3 Clustering Algorithms

To compare the results of the 3 algorithms in terms of performance, I have chosen 3 metrics to assess the quality of the clustering results (Baruah, 2020):

1. **Silhouette score**: This metric measures how well each point in a cluster is separated from points in other clusters. Higher values indicate better-defined clusters. This tells us if individual points are correctly assigned to their clusters or not. If the value is close to 0, implies that it exists between 2 clusters. A value closer to 1 defines that it is in the right cluster and a value close to -1 means that the point has been assigned to the wrong cluster.

2. **Calinski-Harabasz index**: This metric computes the ratio of the between-cluster dispersion and the within-cluster dispersion. Higher values indicate better-defined clusters. It is based on the notion that good clusters are defined as the ones which themselves are very compact and well-distanced from other another.

3. **Davies-Bouldin index**: This metric measures the average similarity between each cluster and its most similar cluster. Lower values indicate better-defined clusters. Similar to above indexes, this index also captures both separation and compactness of the clusters.

I have used the estimated values of number of clusters (k), eps, min_samples from task 2.1, 2.2 and 2.3, into the code for calculating above matrix.

- K-Means

```
In [146]: # K-Means Performance Metrics
          from sklearn.cluster import KMeans
          from sklearn.metrics import silhouette_score
          from sklearn.metrics import calinski_harabasz_score
          from sklearn.metrics import davies_bouldin_score
          # Fit K-Means
          kmeans_1 = KMeans(n_clusters=4, random_state = 10)
          cluster_df = data_values
          # Use fit_predict to cluster the dataset
          predictions = kmeans_1.fit_predict(cluster_df)
          # Calculate cluster validation metrics
          score_kmeans_s = silhouette_score(cluster_df, kmeans_1.labels_, metric='euclidean')
          score_kmeans_c = calinski_harabasz_score(cluster_df, kmeans_1.labels_)
          score_kmeans_d = davies_bouldin_score(cluster_df, predictions)


          print('Silhouette Score: %.4f' % score_kmeans_s)
          print('Calinski Harabasz Score: %.4f' % score_kmeans_c)
          print('Davies Bouldin Score: %.4f' % score_kmeans_d)

          Silhouette Score: 0.4777
          Calinski Harabasz Score: 3359.9342
          Davies Bouldin Score: 0.6936
```

- DBSCAN

In [147]:
```
# DBSCAN Performance Metrics
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import DBSCAN
from matplotlib import pyplot
# define dataset
cluster_df = data_values

# define the model
model = DBSCAN(eps=0.12356729733404877, min_samples= 13)

# rule of thumb for min_samples: 2*len(cluster_df.columns)
# fit model and predict clusters
yhat = model.fit_predict(cluster_df)
# retrieve unique clusters
clusters = unique(yhat)

# Calculate cluster validation metrics
score_dbscan_s = silhouette_score(cluster_df, yhat, metric='euclidean')
score_dbscan_c = calinski_harabasz_score(cluster_df, yhat)
score_dbscan_d = davies_bouldin_score(cluster_df, yhat)

print('Silhouette Score: %.4f' % score_dbscan_s)
print('Calinski Harabasz Score: %.4f' % score_dbscan_c)
print('Davies Bouldin Score: %.4f' % score_dbscan_d)
```

```
Silhouette Score: -0.0474
Calinski Harabasz Score: 190.8779
Davies Bouldin Score: 2.4586
```

- GMM

In [149]:
```
# Gaussian Mixture Performance Metrics
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.mixture import GaussianMixture
from matplotlib import pyplot
# define the model
model = GaussianMixture(n_components=4, covariance_type= "full", random_state = 10)
# define dataset
cluster_df = data_values

# fit the model
model.fit(cluster_df)
# assign a cluster to each example
yhat = model.predict(cluster_df)
# retrieve unique clusters
clusters = unique(yhat)

# Calculate cluster validation score
score_gmm_s = silhouette_score(cluster_df, yhat, metric='euclidean')
score_gmm_c = calinski_harabasz_score(cluster_df, yhat)
score_gmm_d = davies_bouldin_score(cluster_df, yhat)

print('Silhouette Score: %.4f' % score_gmm_s)
print('Calinski Harabasz Score: %.4f' % score_gmm_c)
print('Davies Bouldin Score: %.4f' % score_gmm_d)
```

```
Silhouette Score: 0.4588
Calinski Harabasz Score: 3073.8319
Davies Bouldin Score: 0.7278
```

Cumulating the performance metrics:

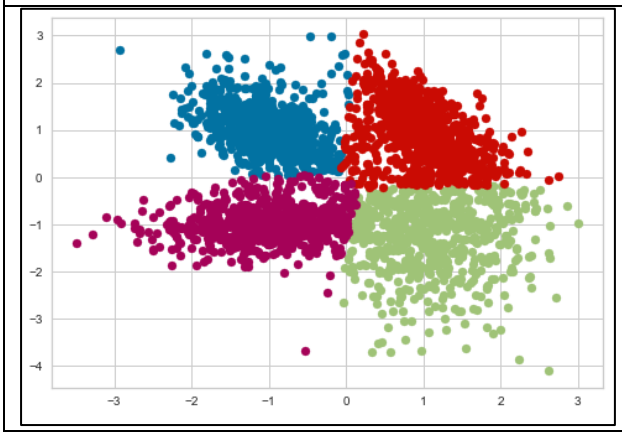
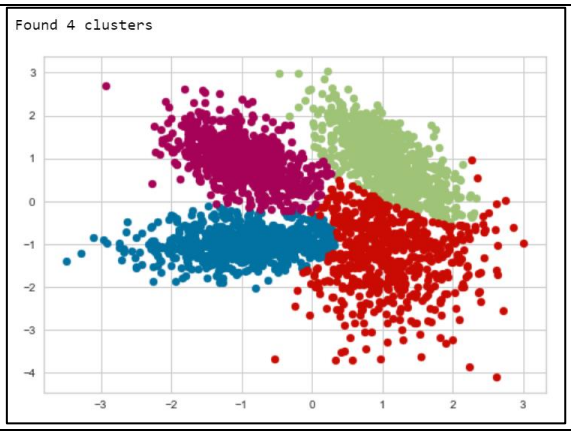
Metric	K-Means	DBSCAN	Gaussian Mixture
Number of Clusters	4	15	4
Silhouette Score	0.4777	- 0.0474	0.4588
Calinski-Harabasz Index	3359.9342	190.8779	3073.8319
Davies-Bouldin Index	0.6936	2.4586	0.7278

In general, higher silhouette scores and Calinski-Harabasz index values and lower Davies-Bouldin index values indicate better clustering performance. Based on these metrics, one can choose the algorithm that performs the best for their dataset.

K-Means vs DBSCAN:
We can see how both Silhouette Score and DB Index is higher for K-Means and Calinski-Harabasz Index is lower than that of DBSCAN. Thus, it is evident that K-Means is a better algorithm than DBSCAN, deployed for the same dataset.

Moreover, the negative value of silhouette score of DBSCAN model implies that the algorithm has failed to identify clusters and is not appropriate to define the clustering of the randomly generated dataset. Despite my attempts to tune in the value of *eps* and *min_samples*, the value of Silhouette Score comes out to be negative, indicates DBSCAN is not suitable for the given dataset and other clustering algorithms should be considered. Thus, DBSCAN is the worst algorithm amongst the 3 for the given dataset.

K-Means vs Gaussian Mixture:
Both algorithms produce 4 clusters and a very similar distribution of the clusters. Attaching images for reference:

K-Means	Gaussian Mixture
	

The only way to compare them is through the performance metrics. Silhouette Score is higher for K-Means, implies that points in the cluster are better separated from points in other clusters than in GMM. The lower value of DB Index in K-Means suggests the clusters are better defined than in GMM. Finally, the higher index value of Calinski-Harabasz in K-Means suggests that the clusters in K-Means are more compact and further-distanced from other clusters than in GMM. Thus, K-Means has outperformed GMM based on all cluster validation metrics.

REFERENCES

1. Amir Masoud Sefidian - Sefidian Academy. (2020). How to determine epsilon and MinPts parameters of DBSCAN clustering. [online] Available at: http://www.sefidian.com/2020/12/18/how-to-determine-epsilon-and-minpts-parameters-of-dbscan-clustering/ [Accessed 7 Apr. 2023].
2. Appen. (2020). What Does Interoperability Mean for the Future of Machine Learning? [online] Available at: https://appen.com/blog/what-does-interoperability-mean-for-the-future-of-machine-learning/ [Accessed 7 Apr. 2023].
3. Baruah, I.D. (2020). Cheat sheet for implementing 7 methods for selecting the optimal number of clusters in Python. [online] Medium. Available at: https://towardsdatascience.com/cheat-sheet-to-implementing-7-methods-for-selecting-optimal-number-of-clusters-in-python-898241e1d6ad.
4. Baruah, I.D. (2020). K-means, DBSCAN, GMM, Agglomerative clustering — Mastering the popular models in a segmentation…. [online] Medium. Available at: https://towardsdatascience.com/k-means-dbscan-gmm-agglomerative-clustering-mastering-the-popular-models-in-a-segmentation-c891a3818e29.
5. Cluster Analysis 4 Marketing. (n.d.). Sum of squared error (SSE). [online] Available at: https://www.clusteranalysis4marketing.com/interpretation/sum-of-squared-error-sse/.
6. IDB-FOR-DATASCIENCE (2020). Unsupervised-ML-Modelling-for-Segmentation/Segmentation Notebook _Final.ipynb at main · IDB-FOR-DATASCIENCE/Unsupervised-ML-Modelling-for-Segmentation. [online] GitHub. Available at: https://github.com/IDB-FOR-DATASCIENCE/Unsupervised-ML-Modelling-for-Segmentation/blob/main/Segmentation%20Notebook%20_Final.ipynb [Accessed 9 Apr. 2023].
7. Rao, S. (2022). K-Means Clustering: Explain It To Me Like I'm 10. [online] Medium. Available at: https://towardsdatascience.com/k-means-clustering-explain-it-to-me-like-im-10-e0badf10734a.
8. scikit-learn. (n.d.). Demonstration of k-means assumptions. [online] Available at: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-autoexamples-cluster-plot-kmeans-assumptions-py [Accessed 9 Apr. 2023].
9. scikit-learn. (n.d.). sklearn.cluster.DBSCAN. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN.
10. scikit-learn.org. (n.d.). sklearn.cluster.KMeans — scikit-learn 0.23.1 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans.
11. scikit-learn. (n.d.). sklearn.datasets.make_blobs. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html#sklearn.datasets.make_blobs.
12. scikit-learn. (n.d.). sklearn.datasets.make_classification. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html#:~:text=Generate%20a%20random%20n%2Dclass [Accessed 9 Apr. 2023].
13. Tomar, A. (2022). Stop Using Elbow Method in K-means Clustering, Instead, Use this! [online] Medium. Available at: https://towardsdatascience.com/elbow-method-is-not-sufficient-to-find-best-k-in-k-means-clustering-fc820da0631d#:~:text=The%20elbow%20method%20is%20a.
14. upGrad blog. (2020). Cluster Analysis in Data Mining: Applications, Methods & Requirements. [online] Available at: https://www.upgrad.com/blog/cluster-analysis-data-mining/#:~:text=Cluster%20Analysis%20in%20Data%20Mining%20means%20that%20to%20find%20out.

15. www.javatpoint.com. (2011). *Data Mining Cluster Analysis - Javatpoint*. [online] Available at: https://www.javatpoint.com/data-mining-cluster-analysis.
16. www.youtube.com. (n.d.). #22 Cluster Analysis - Properties, Categories Of Methods |DM|. [online] Available at: https://www.youtube.com/watch?v=tSXCVvBmOQU&list=PLmAmHQ-_5ySxFoIGmY1MJao-XYvYGxxgj&index=22&ab_channel=Trouble-Free [Accessed 7 Apr. 2023].