MIS41270-Data Management and Data Mining

Professor: Elayne Ruane

Spring Trimester 2022/23

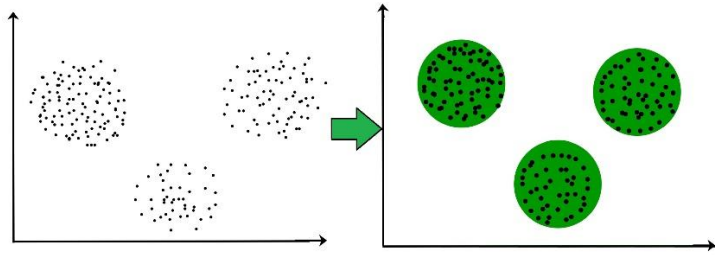
#Assignment_2: Clustering Analysis

Name	Student Number
Siddhant Bajpai	22200242

Table of Contents

What is Clustering?

Using the data mining method of clustering, similar objects are grouped together based on a variety of factors, including proximity, density, graphs, and statistical distributions. Numerous disciplines, including unsupervised machine learning, data mining, statistics, graph analytics and image processing can benefit from the use of this technique. By grouping data points into clusters, it is possible to make items within a cluster more like one another than those in other clusters. To analyse and interpret complicated data, clustering is a potent tool for finding patterns, relationships, and structures in big datasets (Sehgal, 2022).

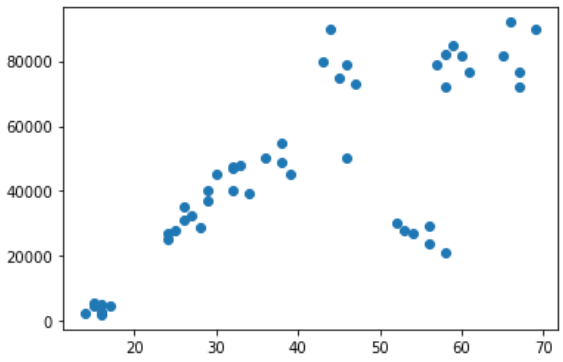


TASK 1: K-means

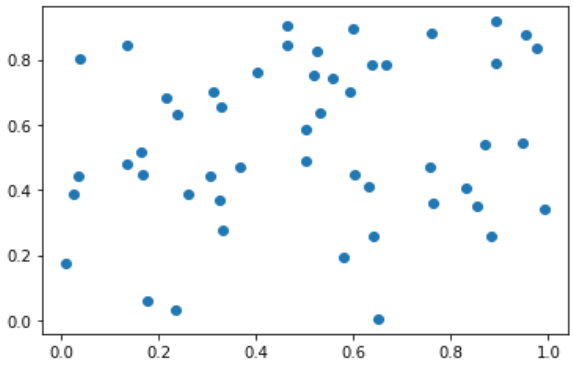
Methodology

To start with the K-means clustering to our raw data, we've done the following in our code:

1. **Importing Libraries:** The code begins by importing the required tools, which include NumPy for numerical computations, Pandas for data manipulation, Matplotlib for data visualization, and scikit-learn for machine learning functions like KMeans, MinMaxScaler, DBSCAN, and AgglomerativeClustering.
2. **Loading Data:** It then reads and loads data from a CSV file named 'salary_data.csv' using Pandas and stores it in a DataFrame called '**salary-data'**
3. **Data Visualisation:** For basic visual representation, using the **plt.scatter()** function, the code creates a scatter plot of 'age' verses 'income' from the 'salary_data' DataFrame.
4. **Generating Random Data**: The code defines a function named '**generate_random_data()**', which generates random 2D data points from 'salary_data.csv' with the features 'age' and 'income'. The function accepts an input 'n' indicating the amount of data points to generate and returns a DataFrame with 'n' rows and 2 columns (age and income). The resulting data is then plotted with the **plt.scatter()** tool.



Plot 1: Scatter plot for salary_data.csv



Plot 2: Scatter plot using random function.

1.1 Using K-means function

```
# use this function to run kmeans on your dataframe
def run_kmeans(k, dataset, x_col='', y_col=''):
    df = dataset.copy(deep=True)
    kmeans = KMeans(n_clusters=k, init='random', 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))
```

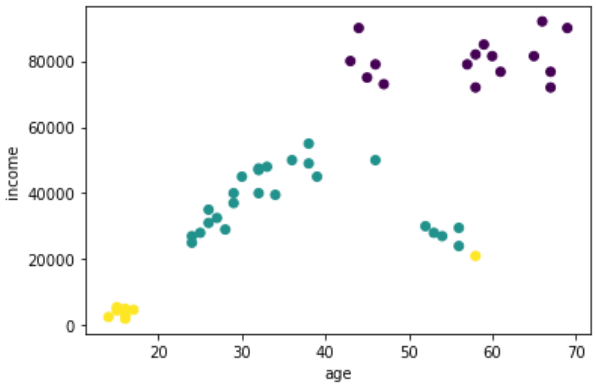
1.1 Running K-means function on 'salary_data.csv' dataframe:

```
sal_data_run_1_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
```

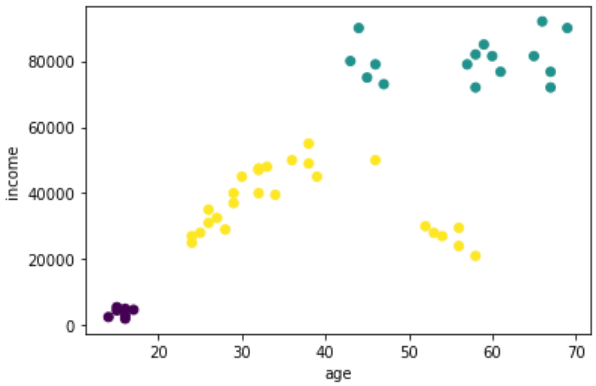
The above code calls the 'run_kmeans()' function with parameters 'k=3', 'dataset=salary_data', 'x_col='age'', and 'y_col='income''. The resulting DataFrame with predicted cluster labels is assigned to first variable called '**sal_data_run_1_df**'.
The dataset provided consists of 50 data points. Using the same code and run_kmeans() function to run and generate 10 different clusters by assigning 10 different variables from 1 to 10.

```
sal_data_run_1_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_2_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_3_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_4_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_5_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_6_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_7_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_8_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_9_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
sal_data_run_10_df = run_kmeans(3, salary_data, x_col='age', y_col='income')
```

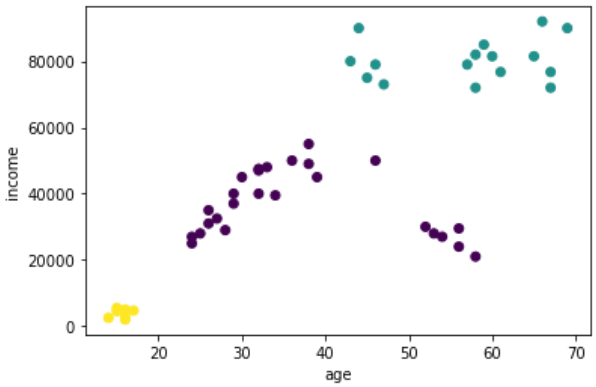
OUTPUT: K-means Clusters (salary-data.csv)



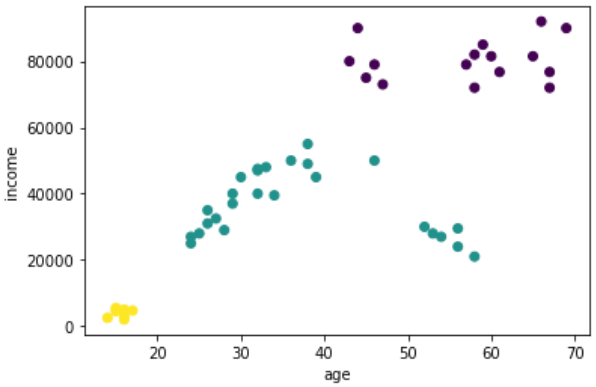
Plot-1



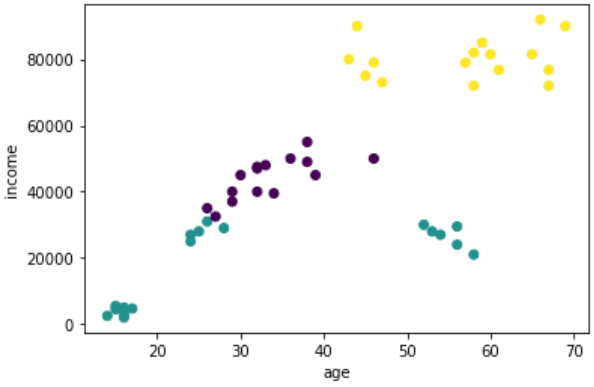
Plot-2



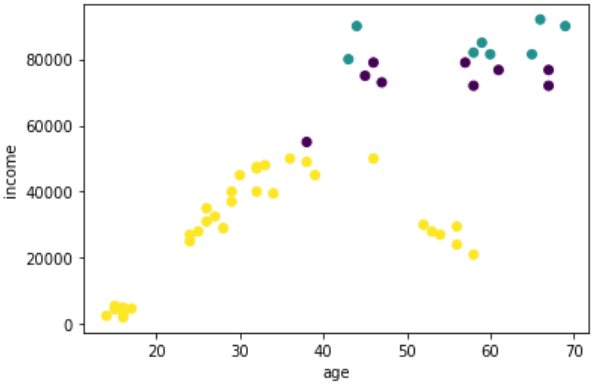
Plot-3



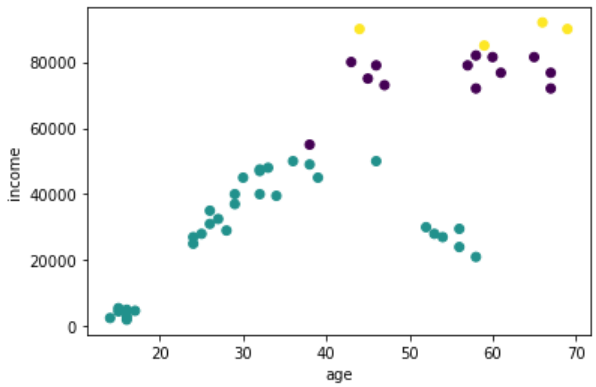
Plot-4



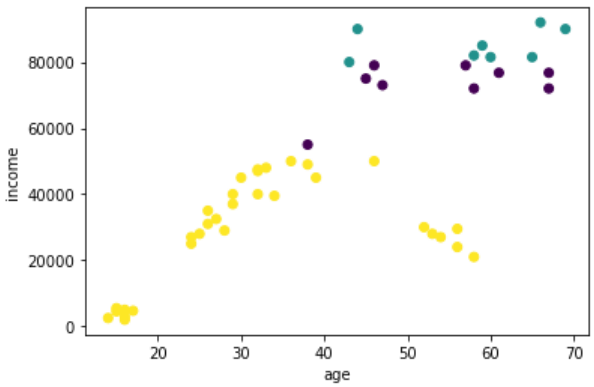
Plot-5



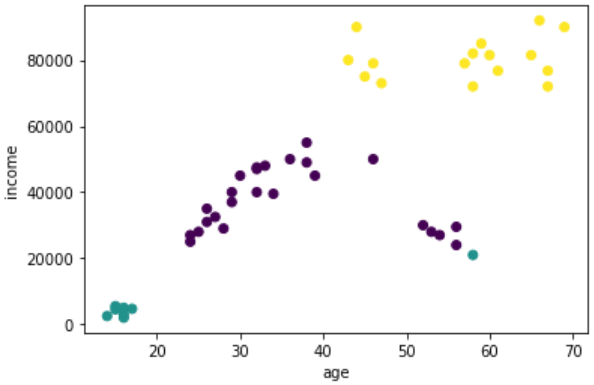
Plot-6



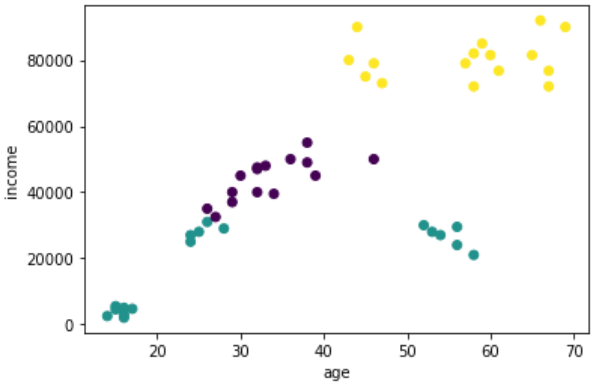
Plot-7



Plot-8



Plot-9



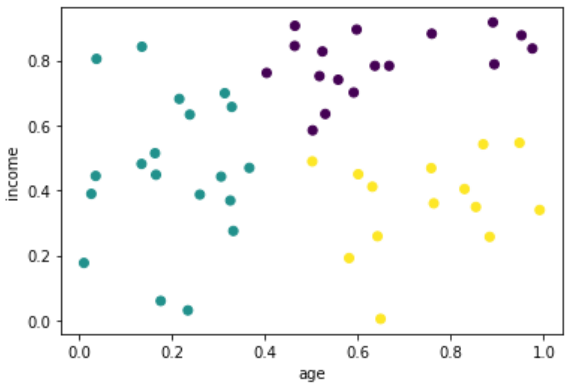
Plot-10

1.1 Running K-means function on 'randomly generated' dataframe:

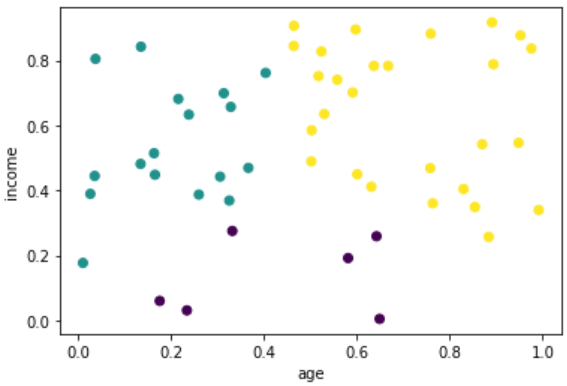
```
#generate 50 random datasets

def generate_random_data(n):
return pd.DataFrame({'age':np.random.rand(n),'income':np.random.rand(n)})
rand_data = generate_random_data(50)

#k-means of 50 points on randomly generated dataset
rand_run_1_df = run_kmeans(3, rand_data, x_col='age', y_col='income')
```

Using above function "**generate_random_data(n)**" generating n=50 random 2D points and declaring a new variable called "**rand_data**" to store the resulting DataFrame with random 'age' and 'income' values. The resulting DataFrame with predicted cluster labels is now assigned to first variable called '**rand_run_1_df**' to generate 10 new clusters with n=50 random points.

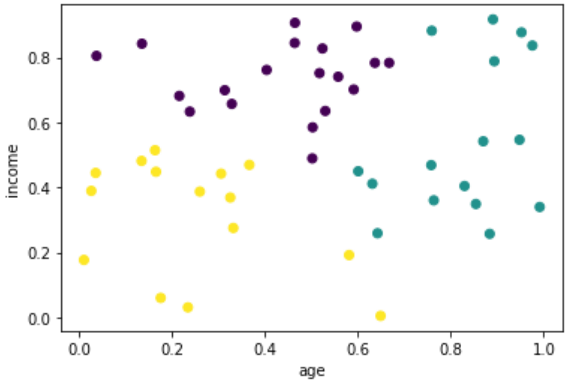
OUTPUT: K-means Clusters (random dataset)



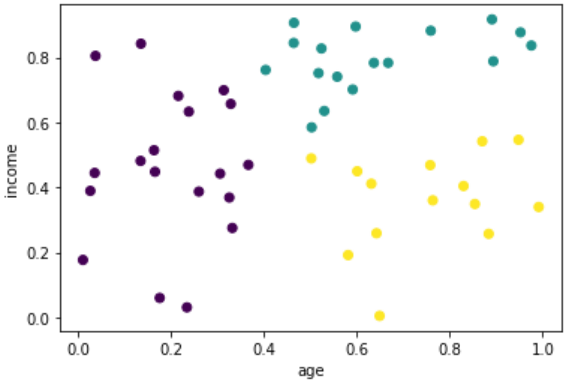
Plot-11



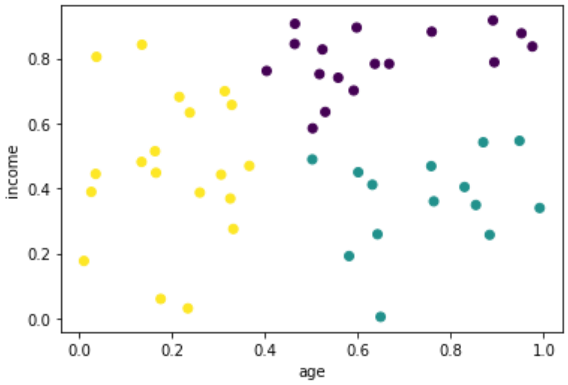
Plot-12



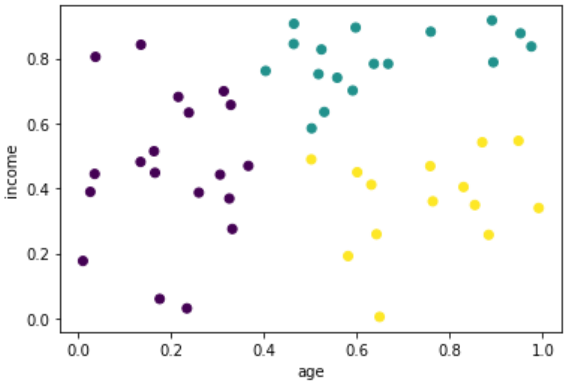
Plot-13



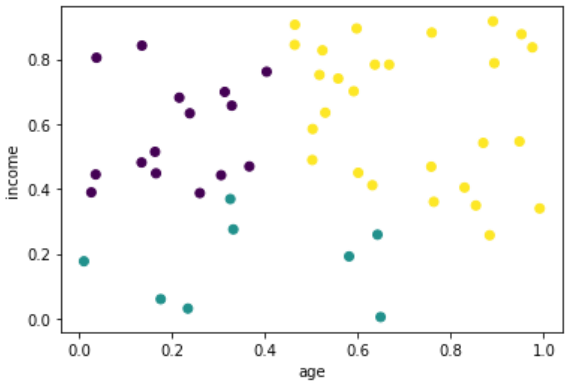
Plot-14



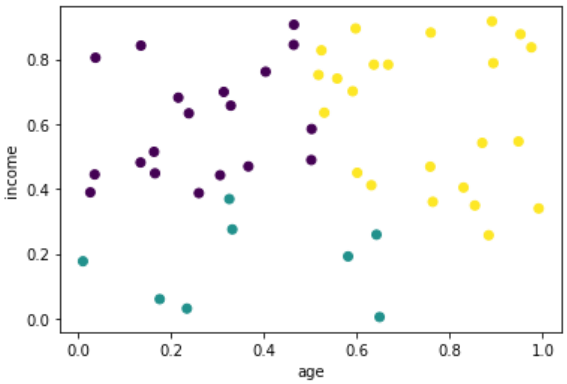
Plot-15



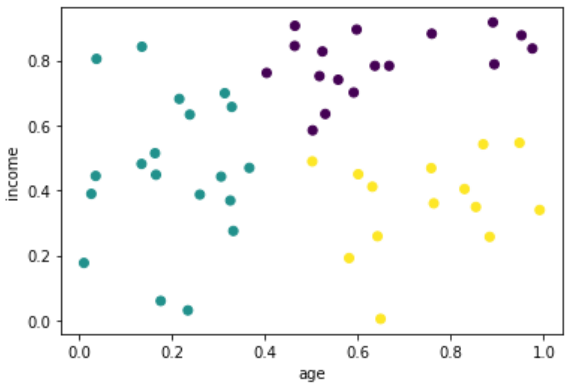
Plot-16



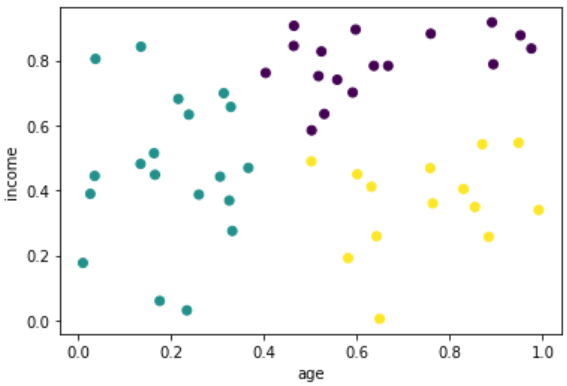
Plot-17



Plot-18



Plot-19



Plot-20

1.1 Analysis of the results

Analysing the results of various clusters obtained from running K-means on 'salary_data.csv' and 'randomly generated' dataset.

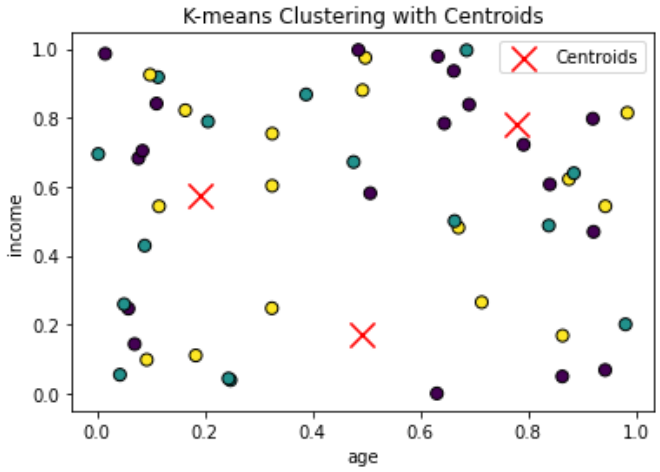
Based on the various clusters formed in the above plots run on 'salary-data.csv' and 'randomly generated' data, it appears that the k-means algorithm with k=3 cluster value has successfully identified three distinct clusters in each plot, represented by different colours (green, purple, and yellow) (Sharma, 2023).

1. On the X-axis, we have the age of the customer and the y-axis represent the relative income. Here we, can clearly visualise that these customers have been segmented into 3 different clusters, which can further be used by some organisation to make strategies and offer various income packages to its working staff based on age.
2. **Cluster Assignments:** Based on how close the data points are to the **centroid points**, they are divided up into various clusters. Points are assigned to a cluster based on their proximity to a specific centroid. The same colour is often used to identify points in the scatter plot that belong to the same cluster (Sharma, 2023).
3. **Cluster Sizes:** The size of each cluster represented by different colours in different plots may vary. Like we see in Plot-1, 2 and 5, people with high income and higher age are clustered by different colours based on centroid changing its place after every **iteration** of k-means algorithm run. The same is happening with people with lower income and lower age range. This indicates differences in the **density or spread of data points** in different regions of the dataset (Sharma, 2023).
4. **Cluster separation:** The 3 clusters generated are separated from each other, with distinct boundaries or gaps between them. This indicates that K-means algorithm has successfully grouped data points into distinct clusters based on their similarity or **proximity** (Sharma, 2023).
5. Based on my understanding, If we notice Plot-2, 3 and 4 we see a better groupism of datapoints in 3 different clusters, with less spread of different colour datapoints in different clusters. Based on the concept of k-means run 'when the centroids stop changing its position, the position of various clusters formed becomes saturated and the algorithm stops to give the best of clustering'. These 3 plots have the best generic distribution of datapoints in their specific cluster which shows the distribution between age and income of various individuals.
6. By using the value of k=3, based on our dataset it becomes a bit easy to segment 3 different type of workforce in an organisation.
    a. **First Cluster** on the bottom left corner of the plot denotes the workforce which have low age (<20) and earning low income(<10000). It contains 7 data points that are closely clustered and overlapping with each other. This cluster could potentially represent a group of younger individuals with lower incomes.
    b. **Second cluster** is in the centre of the plot and has the maximum number of datapoints. Some of the datapoints are overlapping with each other along age vs income axis. Additionally, there are roughly 6 datapoints which are relatively far away from the main 2nd cluster, potentially indicating some outliers or distinct sub-group within this cluster. This cluster represents a mixed group of individuals with varying ages and incomes (Sharma, 2023).
        i. **Mixed Cluster** is another part of the 2nd cluster, if we see plot 5, 6 and 10 generated after multiple iterations, there appears to be a mixture of different coloured datapoints. For example, there is a cluster where green and purple coloured datapoints are intermingled with each other.
        The presence of a mixed cluster in the clustering plots indicates that there may be a **subgroup** of data points with attributes that are in between the characteristics of the other clusters. **Outliers** here as data points may deviate significantly from the general pattern of the data and may not fit well into any identified clusters. Sub-groups in clustering refers to smaller groups of data points within a larger cluster that may have distinct characteristics. In above plots, a sub-group of datapoints with characteristics that are different from the main green and purple clusters but are still close enough to be included in the same cluster by the k-means algorithm (Sharma, 2023).

c. **Third Cluster** is at the top right corner of the plots and represent datapoints/individuals with high age and high income. This covers people who are aged > 50 and have income > 60000.

7. On **Random generated dataset**, the same 3 clusters were identified with different colours, denoted by green, purple, and yellow. The clusters formed on salary_data dataset likely represents specific patterns, trends, and distributions that are inherent to the context of the provided data. On the other hand, the randomly generated dataset may not have such characteristics, as it is generated randomly without any specific patterns or distributions.
8. **Distribution of Datapoints:** We see that, it is possible that the distribution of data points in the primary dataset, "salary-data.csv," differs from the distribution of data points in the dataset that was generated at random. While the randomly generated dataset may have a more uniform distribution of data points, the primary dataset might have a larger variety of ages and incomes with various levels of concentration or density in different sections of the plot.
9. We see that, there are 3 clusters formed on random data, but they are randomly distributed on age vs income axis, giving a mixed concentration on the plot. After multiple iterations, the scatter of datapoints on various plots remains the same, keeping the centroid same.

K-means clustering with centroid on random generated data.



There are 3 centroids marked on scatter plot for 3 different clusters formed between age and income.

Overall, the analysis suggests that the randomly generated dataset of 50 data points also exhibits similar patterns of clusters as observed in the salary_data, with the presence of potential mixed clusters, outliers, and sub-groups that can further be used to make industry level decisions.

1.2 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)
```

In the above function, 'run_kmeans' function uses the 'Kmeans' algorithm from the scikit-learn library in python to perform clustering. The 3 parameters being used in the run_kmeans function and their impact on the cluster results:

a. **'n_clusters'**: This basic parameter determines the number of clusters that the algorithm will attempt to identify in the provided dataset. It is commonly referred to as "**k**" value in k-means clustering. Clustering outcomes can be considerably influenced by the selection of n_clusters. Smaller and more closely clustered clusters may come from a greater number of n_clusters, whereas bigger and more loosely grouped clusters may result from a lower value of n_clusters.
Too few clusters may result in oversimplification, while too many clusters may result in inappropriate or noisy clusters. For our dataset with 50 datapoints, choosing a range of 3~5 clusters are an optimal range (Pedregosa et al., 2011).

b. **'init'**: The process for initializing the cluster centroids is specified by this option. Since it is now set to "**random**," the initial centroids will be generated at random. The initialization strategy selected can also affect the clustering outcomes because it impacts the centroids' initial positions and the algorithm's convergence and stability (Pedregosa et al., 2011).

c. **'n_init':** This option determines how many alternative centroid initializations will be used when running the method. It is set to **1** in this instance, indicating that the procedure will only be applied once with the centroids initialized at random. The problem of the approach getting caught in local

maxima, when the process converges to inferior clustering outcomes, can be reduced by running the algorithm numerous times with various initializations. Better clustering outcomes may be more likely with a greater value of n_init, but the computing cost will also rise (Pedregosa et al., 2011).

1.3 Kmeans++

Using K-means function

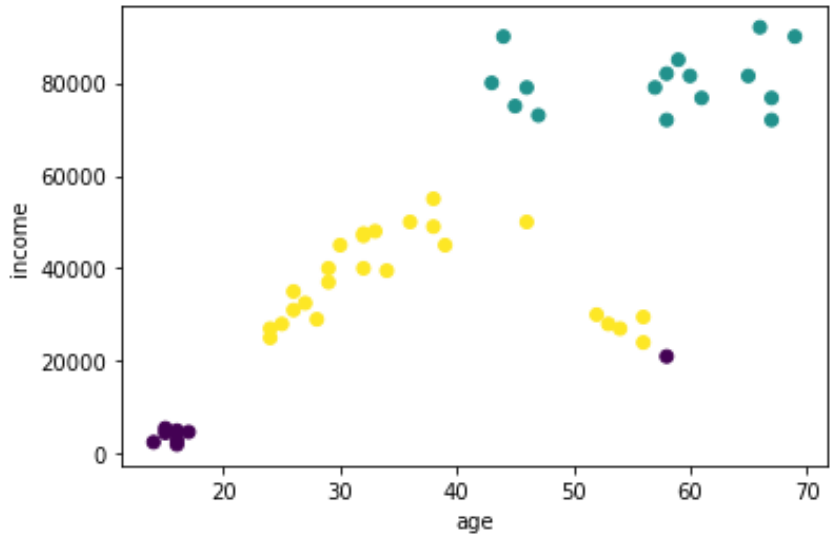
```
#creating a new function to run k-means on the datasets but this time using k-means++
def run_kmeans_plus(k, dataset, x_col='', y_col=''):
    kmeans_plus = KMeans(n_clusters=k, init='k-means++')
    kmeans_plus.fit(dataset)
    cluster_labels = kmeans_plus.fit_predict(dataset)
    df[f'cluster_labels'] = cluster_labels
    plt.scatter(df[x_col], df[y_col], c=kmeans_plus.labels_.astype(float))
```

Running K-means++ function on 'salary_data.csv' dataframe:

```
sal_data_run_plus_df = run_kmeans_plus(3, salary_data, x_col='age', y_col='income')
```

The above code calls the **'run_kmeans_plus()'** function in place of 'run_kmeans()' with parameters 'k=3', 'dataset=salary_data', 'x_col='age'', and 'y_col='income''. The resulting DataFrame with predicted cluster labels is assigned to first variable called **'sal_data_run_plus_df'**.
The dataset provided consists of 50 data points.

OUTPUT: K-means++ Clusters (salary-data.csv)



1.3 Analysis of the Clusters found on Kmeans++

Running the K-means++ algorithm has now provided insights into the clustering structure of the dataset salary_data.csv.

a. The K-means++ algorithm have identified clusters based on similarities in the 'age' and 'income' features. If we see the above diagram, the top right datapoints plotted clearly define the clusters with higher average income and older age represents high-income earners or retirees, while datapoints on bottom left of the plot indicates clusters with lower income and younger age, which means early career professionals or students.

b. **Cluster Composition:** The Kmeans++ identified 3 clusters with varying sizes, with cluster 1 on the top right having 16 datapoints, cluster 2 in the middle of the plot comprising of ~26 datapoints and the cluster 3 on the bottom left having 7 datapoints. We can conclude the cluster 2 with higher number of datapoints lying on the plane represents the largest cluster, meaning that maximum individuals are having 25 < age < 60 and have a median salary in the range of 20000 to 60000.

c. **Cluster Separation:** The extent of separation between 3 clusters is uniform and looks well-separated denoting the clusters formed are distinct in nature. Although there is a point where a few datapoints from

cluster-3 overlaps with the cluster-2, indicating that these segments of individuals or datapoints are sharing some similar characteristics.

d. **Cluster Stability:** I tried running the Kmeans++ with different random initialisations ('n_init' parameter) and upon comparing the cluster results, the results look almost similar indicating that the clusters formed are highly stable and the segmentation done by Kmeans++ is reliable (Rakhlin and Caponnetto, 2006).

Comparing Cluster results between Kmeans++ and Kmeans

The clustering outcomes on the dataset may be impacted in the following ways if the KMeans algorithm is initialized using the kmeans++ initialization method rather than the 'random' initialization method:

a. Improved Convergence: Compared to random initialization (Kmeans), the kmeans++ initialization method typically led to faster convergence. This is because it carefully chose initial centroids that were further away from one another, which resulted in a faster convergence to a clustering solution that sounded more advantageous. With fewer iterations, this produced clustering results that were more precise and reliable (Aubaidan and Mohd, 2014).

b. More Accurate Clustering: This is because it made sure the initial centroids were placed in a way that captures the underlying structure of the data, producing clusters that were more closely related to the actual underlying patterns in the data (StackExchange, 2018).

c. Enhanced Robustness: The technique may be more resistant to outliers or noisy data points if the kmeans++ initialization approach is used. By placing the initial centroids farther apart, kmeans++ proved by lessening the impact of outliers on the placement of the centroid, producing more reliable clustering results that are less influenced by noisy data points, as compared to kmeans cluster plot (Aubaidan and Mohd, 2014).

1.4 Elbow Plot

Created my own dataset with 30 datapoints as X and Y named 'elbow_data.csv' which are nearly clustered in 3 ranges.

Defining elbow plot function:

```
def get_elbow_plot(dataset, x_col='', y_col='', min_k=2, max_k=10):
k_range = range(min_k, max_k)
sse = []
for k in k_range:
      km = KMeans(n_clusters=k)
      km.fit(dataset)
      sse.append(km.inertia_)
```

Creating an Elbow Plot:

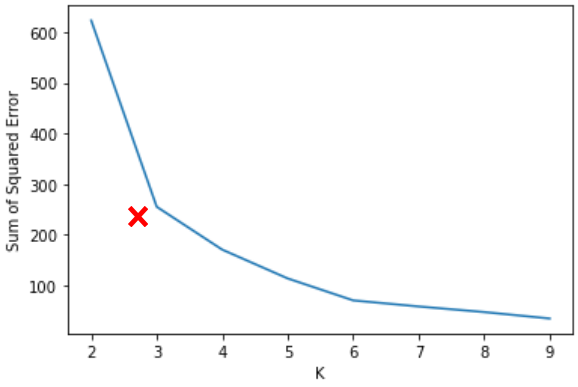
```
elbow_data = pd.read_csv('elbow_data.csv')
print(elbow_data.shape)
elbow_data.head()
get_elbow_plot(elbow_data, x_col='x', y_col='y')
```

```
(30, 2)
Out[150]:
```

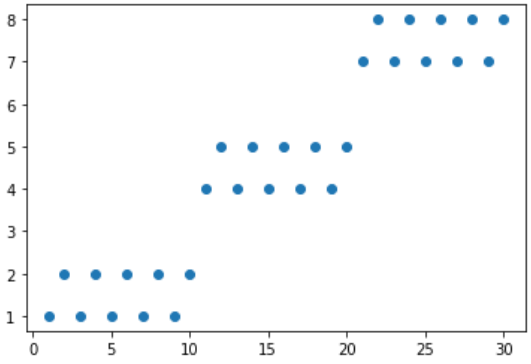
	X	Y
0	1	1
1	2	2
2	3	1
3	4	2
4	5	1

From the own dataset used, the following table is generated with 30 values.

Elbow Plot Curve:



Clearly clustered datapoints (n=30)



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

For various values of k (number of clusters) in K-means clustering, the elbow plot is a visual depiction of the sum of squared distances (SSE) of data points to their respective cluster centroids. It aids in figuring out the ideal value of k that results in generating the best of clusters satisfying the datasets (Sampaio, 2022).

From the dataset of 30 datapoints used, the elbow plot analysis can be interpreted as follows:

a. Based on the dataset provided, which contains 30 datapoints divided into 3 clusters (Cluster 1: X=[1-10], Y=[1-2], Cluster 2: X=[11-20], Y=[4-5], Cluster 3: X=[21-30], Y=[7-8]), the elbow plot would show the SSE value for different values of K, ranging from 1 to a maximum value.

b. When k=1, the SSE would be the highest, as all data points would be assigned to a single cluster, resulting in a high within-cluster variance.

c. As the K increases, the SSE would generally decrease, as the algorithm is able to create more clusters and reduce the within-cluster variance.

d. At a certain point, the rate of decrease in SSE would start to slow down. This point, where the elbow-like bend occurs in the plot, is known as the "elbow point".

e. The elbow plot suggests that **K** should be set to **3**, as the SSE begins to level off at this figure. This shows that grouping the dataset into three groups is a good clustering approach that captures the data patterns without being overfit (Sampaio, 2022).

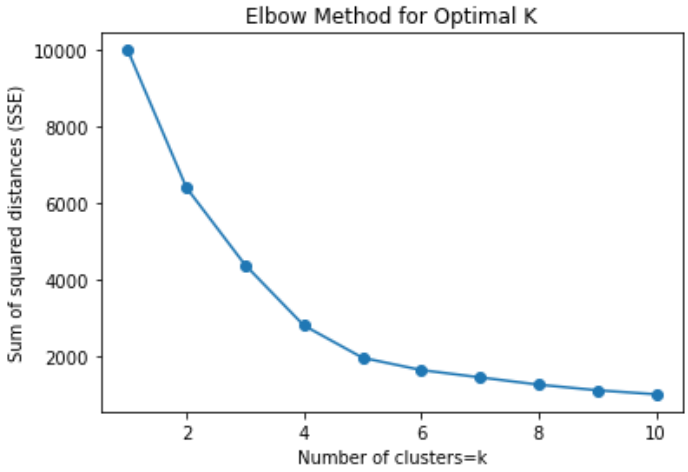
Task 2: Compare Algorithms

Generating best dataset to cluster with 3000 samples by optimising parameter values.

Elbow Plotting Analysis:
Trial 1:

```
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)
```

In the above code, we defined the dataset to be generated with 3000 samples and 'n_features' is set to 2, 'n_informative' is set to 2, 'n_redundant' is set to 0, and 'n_clusters_per_class' is set to 2. This means that the generated dataset will have 2 informative features, no redundant features, and 2 clusters per class. Based on these values we generated an elbow plot which shows near to decent value of K (number of clusters) for the dataset with 3000 samples. The optimal 'K' value obtained lies between "**3 and 5"**. In the next part I've explained as to why this value of 'K' has been used to run K-means and other clustering algorithms.

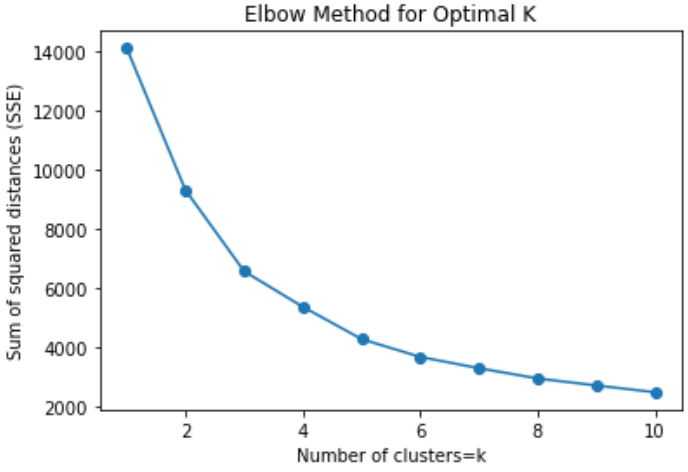


Elbow Plot for above parameter values

Trial 2:

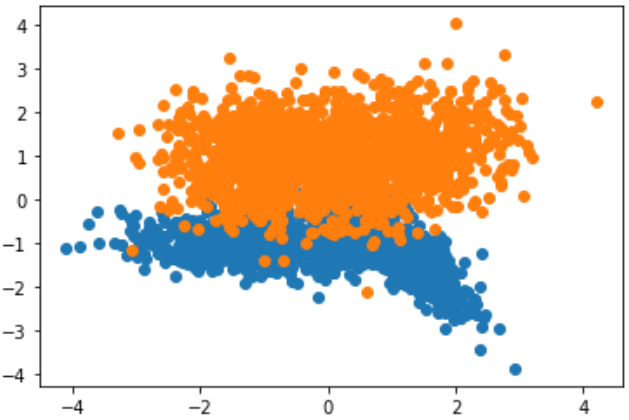
```
data_values, class_labels = make_classification(n_samples=3000, n_features=3, n_informative=3, n_redundant=0, n_clusters_per_class=2, random_state=None)
```

Tried increasing the values of 'n_features' and 'n_informative' to **3**, it somewhat generated a dataset with more informative features and potentially more clusters per class. Below is the Elbow plot generated with above values.



Based on the above trials performed, we see that in the 2nd elbow plot we see more clusters per class, but the optimal value of "**K**" lies between **"3 and 5"** from both the Elbow trials.

Based on the above parameters set for generating 3000 samples, we get the below Clusters formed.



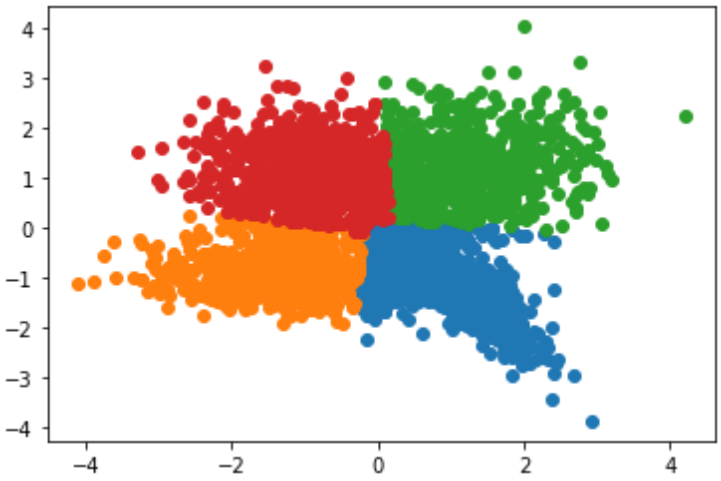
The reason we got 2 clusters in the scatter plot, is because of the parameter 'n_clusters_per-class' is set to 2. So, only 2 clusters are generated for each class where n_samples is 3000, which will result in a total of 2 clusters per class, resulting in a total of 4 clusters in the dataset.

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.

```
kmeans_model = KMeans(n_clusters=4)
kmeans_model.fit(data_values)
kmeans_clusters = kmeans_model.predict(data_values)
```

Using the Kmeans function we're now generating the Kmeans scatter plot for clustering of the 3000 samples generated in the previous plot. Based on the Elbow plot generated, we determined the Optimal value of K will lie between 3 and 5 (curve starts to get flattened post this value) and based on multiple trials we have set the '**n_clusters**' value to **4**, for running the Kmeans algorithm.

Generating the Clusters using Kmeans



Kmeans Algo Scatter Plot(k=4)

Analysis of the Plot:
The scatter plot shows the results of K-means clustering with 4 clusters (green, red, orange, and blue) for the provided dataset.

a. Cluster in Quadrant-1 (Green): The scatter plot reveals a few (~6) green data points on the cluster's periphery that are dispersed in a top-right manner. These dispersed data points, which are not closely linked with the rest of the green cluster, could be outliers or noise in the data. These scattered data points could also be the result of measurement errors, sampling errors, or intrinsic data inconsistency (VanderPlas, 2016)

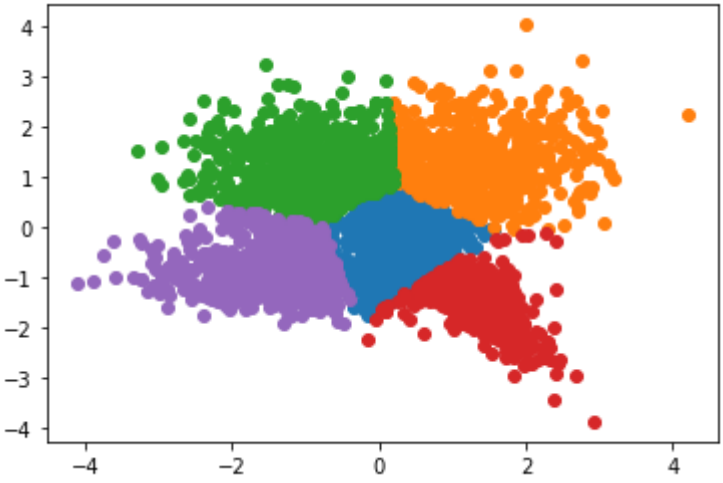
b. Cluster in Quadrant-2 (Red): The scatter plot reveals that there are roughly 12–15 red data points on the cluster's periphery that do not overlap with other red data points. These isolated data points within the red cluster can be a sub-cluster or a separate group. These data points might have distinct feature values that set them apart from the remainder of the red cluster and led to their separation from it (VanderPlas, 2016).

c. Cluster in Quadrant-3 (Orange): The scatter plot demonstrates that there are some orange data points on the periphery of the cluster that are not closely associated with the remainder of the orange cluster, much like the red cluster. These roughly distributed data points could be sub-clusters or outliers inside the orange cluster, indicating some variability or variation in the data (VanderPlas, 2016).

d. Cluster in Quadrant-4 (Blue): The scatter plot reveals that unlike the other three clusters, the blue data points are not densely grouped in a circular pattern. In contrast, they appear to be fleeing in the direction of the x-axis value in the bottom right corner. This would suggest that the blue cluster isn't particularly defined or compact, and that the K-means algorithm is having trouble correctly clustering these data points. The K-means algorithm may have difficulty capturing the complicated shape or structure of the blue cluster, which could explain the scattered or elongated data points (VanderPlas, 2016).

e. Cluster Centre: The four colours that correspond to the clusters that are closely clustered in the scatter plot's centre could suggest that the K-means algorithm was successful in classifying dense areas of data points as clusters. To reduce the within-cluster sum of squared distances, the K-means method iteratively updates the cluster centroids. The algorithm assigns data points to the nearest centroid throughout each iteration, updating the centroids as a result. Up until convergence is obtained and the centroids stabilize, this process continues.
The densest sections of data points, which are often found near the core of the clusters, are where the centroids tend to converge as a result (VanderPlas, 2016).

Tried generating the Clusters with Kmeans algorithm for K=5 value:



Analysis on the K-value and parameters selected:

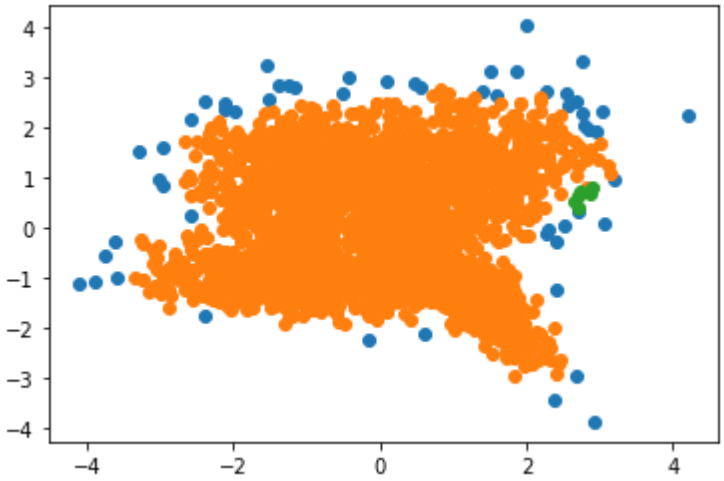
Based on the colors used to depict the various clusters in the provided K-means cluster plot, the value of K (i.e., the number of clusters) appears to be set at 4. The "Elbow Method," which involved plotting the sum of squared distances (SSE) against various values of K and choosing the K value where the SSE starts to level off or form an "elbow" shape, have been used to choose K. This method shows diminishing returns in terms of lowering SSE with increasing K.
'n_features' – A value of 2 is chosen indicating each sample has 2 features and used for generating a 2-D dataset that can be easily visualised in the scatter plot.
'n_informative' – The value of 2 in this parameter, indicates that only 2 out of the total 2 features are informative and contribute to the structure of the dataset. It was used to create a dataset with a specific complexity and variability for clustering analysis.
'n_redundant' = 0, means no redundant features to keep the dataset simple and avoid unnecessary noise that could impact the clustering results.
'n_clusters_per_class' - 2 clusters are generated for each class where n_samples is 3000, which will result in a total of 2 clusters per class, resulting in a total of 4 clusters in the dataset.
'random_state'- A value of 'none' in this parameter is ensuring that the dataset of 3000 samples generated randomly and to allow reproducibility of the results in every iteration.

2.2 DBSCAN

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

```
dbscan_model = DBSCAN(eps=0.3, min_samples=8)
dbscan_clusters = dbscan_model.fit_predict(data_values)
labels = dbscan_model.labels_

print("DBSCAN Clustering Results:")
print(f'Found {len(unique(dbscan_clusters))} clusters')
print("Cluster labels: ", labels)
```

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that can identify clusters of arbitrary shapes and sizes, while also detecting noise points that do not belong to any cluster (Sharma, 2022).



DBSCAN Scatter Plot (3000 samples)

Analysis of Clusters formed by DBSCAN Algorithm

We applied the DBSCAN algorithm on a dataset consisting of 3000 samples and using an epsilon (eps) value of 0.3 and a minimum number of samples (min_samples) requirement of 8 to define the neighbourhood of each data point and identify clusters.

The DBSCAN algorithm resulted in the identification of 3 clusters in the dataset.

a. Orange Cluster: A high density of data points are dispersed throughout the orange cluster, which occupies the largest region of the plot. A dense area at the center of the plot is shown by the fact that the bulk of orange datapoints are concentrated in the range on both the x and y axes. This implies that the orange cluster is a huge and dense collection of closely spaced data points. This implies that there may be a sizable subgroup or subpopulation of data points within the dataset that share comparable traits or attribute values, resulting in a tightly packed cluster (Sharma, 2022).

b. Blue Cluster: In comparison to the orange cluster, the datapoints lying in the blue cluster is smaller and is situated on the orange cluster's periphery. As opposed to the orange cluster, the blue datapoints are less closely spaced apart from one another. This implies that, in comparison to the orange cluster, the blue cluster reflects a less dense region with more dispersed data points (Sharma, 2022).

c. Green Cluster: The green cluster has the fewest data points of the three clusters and is situated on the orange cluster's right outer edge. Because the green data points are clustered closely together in a narrow area, it is likely that they have comparable traits or attribute values. The data points within the green cluster are more like one another than the data points in other clusters, as seen by the high intra-cluster similarity. [9]

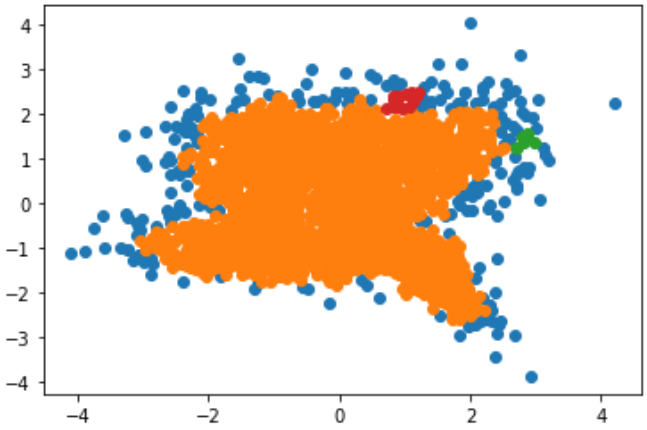
d. Clear separation from Clusters: The orange and blue clusters are clearly separated from the green cluster in the figure, indicating that the green cluster's data points have unique qualities or attribute values when compared to the data points in the other clusters. This implies that the green cluster indicates a distinctive subpopulation or discrete group within the dataset (Sharma, 2022).

e. The Cluster label values obtained: **[0 0 0 … 0 0 0],** indicates that all the data points in the dataset have been assigned to the same cluster, and the cluster label for all the data points is 0 (Sharma, 2022).

f. Overall running the DBSCAN algorithm to obtain the Clusters, indicates that there are distinct groups of data points with different densities and spatial distributions in the dataset, and it can help researchers provide insights into the underlying structure and patterns in the data.

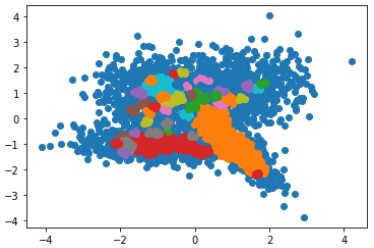
Trial on DBSCAN Algorithm with eps=0.2 and min_samples=8

By reducing the eps value to 0.2 from 0.3, we now get 4 clusters and the Cluster label value obtained is: **[0 0 -1 … 0 0 0]**. Data points with the label 0 are part of a cluster, while data points with the label -1 are considered outliers or noise points that do not belong to any cluster.

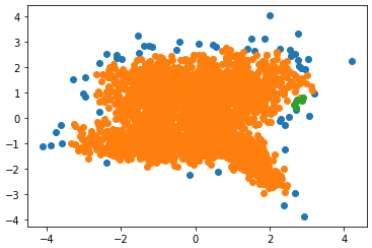


Approach to parameter estimation in DBSCAN:

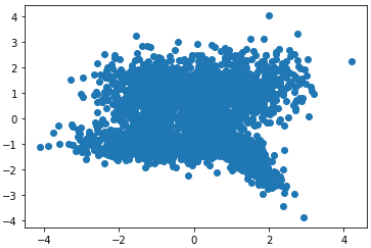
a. '**eps**' **epsilon**: After multiple trials of changing the eps value and visually analysing the scatter plot, choosing eps=0.3 yielded the best cluster results. The 'eps' value determines the radius around each data point within which neighbours are considered.
Each data point formed a bigger neighbourhood as the "eps" value increased, which resulted in larger clusters with more data points and the merger of nearby clusters. In contrast after lowering the eps number, produced a smaller neighbourhood around each data point, possibly leading to the classification of smaller, more densely packed clusters or even individual data points as noise. According on the features of the random data generated of 3000 samples, such as the anticipated density of data points and the desired size of the clusters, the "eps" value of 0.3 have been chosen the optimal value (Mullin, 2020).



eps= 0.1, clusters=45(more noise)



eps= 0.3, clusters=3



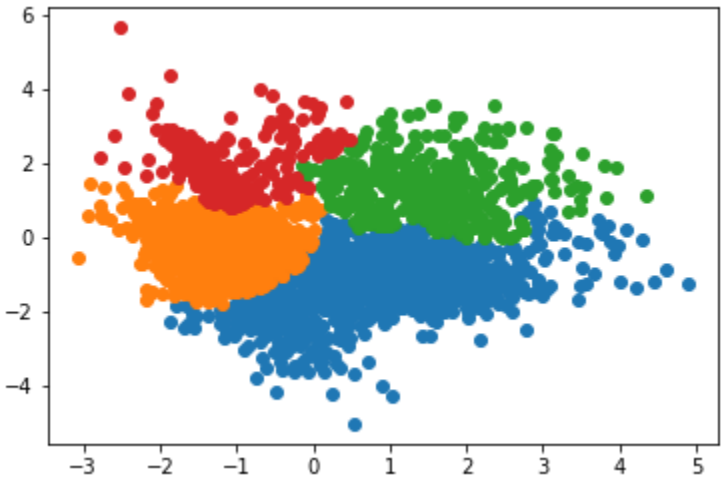
eps= 1.2, clusters=1

b. '**min_samples**': The minimum number of neighbors a data point must have in order to be considered a core point is specified by the "min_samples" setting. DBSCAN's core points are essential since they serve as the basis for locating clusters. A larger "min_samples" number led to smaller clusters with higher densities because it needed more neighbors for a point to be regarded as a core point. On the other hand, upon lowering the "min_samples" number led to larger clusters with lower density because fewer neighbors were needed for a point to be considered a core point. The desired cluster density and the trade-off between noise reduction and cluster identification have led to the selection of 8 as the "min_samples" number in our analysis (Mullin, 2020).

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

Using **Agglomerative Clustering** method for further analysis on clustering.

We used the 'AgglomerativeClustering' class from the Python'scikit-learn' library, which is a library that provides Agglomerative Clustering capabilities, to construct Agglomerative Clustering on the dataset. The number of clusters to be produced, the linkage mechanism for merging clusters, and the distance metric to be applied are all parameters that the "AgglomerativeClustering" class accepts (Ackermann et al., 2014).



a. Quadrant-1 (Green Colour): The green colour datapoints towards the periphery of the Quadrant-1 cluster looks dispersed and isolated from the main cluster in the upper right. This is because of the agglomerative clustering method's tendency to cluster datapoints according to their proximity, where these dispersed datapoints might be close enough to be included in the same cluster but are still somewhat separated from the main cluster due to their positioning in the top right area (Ackermann et al., 2014).

b. Quadrant-2 (Red Colour): There are a few datapoints in the red cluster in quadrant 2 that are not overlapping with other datapoints and are spaced out on the cluster's periphery. These datapoints are less numerous as compared to other clusters, which indicates that the agglomerative clustering method classifies them as outliers or noise. Since the algorithm clusters them together based on their proximity to other datapoints in that location, they are nevertheless incorporated into the red cluster.

c. Quadrant-3 (Orange Colour): The orange datapoints in Quadrant 3 are densely grouped and exhibit some separation from one another, like the red cluster. This indicates that the orange cluster is a dense collection of data points with a few small changes or sub-clusters inside the main cluster. Given their general proximity and the ability to account for certain local variances, the agglomerative clustering algorithm tried to include them all in the same cluster (Ackermann et al., 2014).

d. Quadrant-4 (Blue Colour): In Quadrant 4, the blue datapoints form the largest cluster with the most datapoints and are closely concentrated in the middle. A density gradient or change inside the blue cluster have been observed in that area, yet some datapoints in this cluster appear to be moving in the direction of the x-axis value on the middle right side. Despite being spread out towards the middle x-axis values, the agglomerative clustering approach tried to include these datapoints in the same cluster due to their proximity to the central dense zone.

2.4 Compare and contrast the results of all three algorithms (k-means, DBSCAN, Agglomerative Clustering) on the same dataset.

In summary, the analysis of the plots produced by the three different clustering algorithms (K-means, DBSCAN, and Agglomerative) shows that each approach has its own advantages and disadvantages when it comes to locating clusters in the given dataset of 3000 samples.

1. **K-Means Clustering:**
    a. Similarities with DBSCAN: Both K-means and DBSCAN were able to identify different clusters in the data and assigned data points to particular clusters.

    b. Disparities with DBSCAN: DBSCAN found three clusters whereas K-means produced four clusters. Additionally, DBSCAN places some data points in the noise or outlier group while K-means assigns all data points to one of the four clusters. (label 0).

    c. Similarities with Agglomerative Clustering: Like Agglomerative Clustering, K-means creates clusters that are generally compact and have their data points clustered closely around the cluster centroids.

    d. Disparities with Agglomerative Clustering: K-means requires the pre-specification of the number of clusters, and the resulting clusters might not capture the complex shape or structure of the data points, as seen in the case of the blue cluster. Agglomerative Clustering, on the other hand, is a hierarchical clustering algorithm.

2. **DBSCAN Clustering:**
    a. Similarities with K-means: DBSCAN, like K-means, was able to identify distinct clusters in the data and assigned data points to specific clusters.

    b. Disparities with K-Means: While K-means found four clusters, DBSCAN produced just three. Additionally, DBSCAN allocates some data points to a noise or outlier category (label 0), whereas K-means places all data points in one of the four clusters.

    c. Similarities with Agglomerative Clustering: DBSCAN was able to locate clusters of various densities and shapes based on the local density of data points, just like Agglomerative Clustering.

    d. Disparities with Agglomerative Clustering: DBSCAN has a density-based approach to find clusters, hence it is not necessary to pre-specify the number of clusters and that's why we manipulated the value of 'eps' and 'min_samples'. Additionally, unlike agglomerative clustering was not able to produce clusters with unusual forms or different densities, DBSCAN did.

3. **Agglomerative Clustering:**

    a. Similarities with K-means and DBSCAN: Agglomerative Clustering, like K-means and DBSCAN, was able to identify distinct clusters in the data and assigned data points to specific clusters.

    b. Disparities with K-means and DBSCAN: A hierarchical clustering algorithm called agglomerative clustering joins or aggregates data points or clusters based on a distance or similarity criteria to create clusters from the bottom up. It is not necessary to pre-specify the number of clusters because the dendrogram is used to calculate the number of clusters (Kumar, 2020). In the Agglomerative Clustering plot generated it also produced clusters of different sizes and forms since it took local data point densities into account and merged clusters based on a distance criterion.

In conclusion to the algorithms used (Kmeans, DBSCAN and Agglomerative) to perform efficient clustering on the given dataset, all three algorithms were able to locate distinct clusters in the data, but there were differences in the number of clusters found, how outliers were handled, whether the number of clusters needed to be pre-specified, whether complex cluster shapes or structures could be captured, and how local data point densities were taken into account. In general, the specific qualities of the dataset and the goals of the research will determine which clustering algorithm is best suited.

Reference List

Sehgal, A. (2022) *Clustering data mining techniques: 5 critical algorithms 2023*, *Hevo*. Available at: https://hevodata.com/learn/clustering-data-mining-techniques/ (Accessed 9 April 2023).

Sharma, P. (2023) *The Ultimate Guide to K-means clustering: Definition, methods and applications*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/ (Accessed 9 April 2023).

Pedregosa, F. et al., (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct), pp.2825–2830.

Rakhlin, A. and Caponnetto, A. (2006) *Stability of K -means clustering*, *Mit.edu*. Available at: http://www.mit.edu/~rakhlin/papers/stability_clustering.pdf (Accessed 09 April 2023).

Aubaidan, B. and Mohd, M. (2014) 'Comparative study of k-means and k-means++ clustering algorithms on crime domain', *Journal of computer science*, 10(7), pp. 1197–1206. doi: 10.3844/jcssp.2014.1197.1206.

StackExchange (2018) *K-means vs k-means++*. Available at: https://stats.stackexchange.com/questions/130888/k-means-vs-k-means (Accessed 9 April 2023).

Sampaio, C. (2022) *K-means clustering with the elbow method*, *Stack Abuse*. Stack Abuse. Available at: https://stackabuse.com/k-means-clustering-with-the-elbow-method/ (Accessed 9 April 2023).

VanderPlas, J. (2016) *Python Data Science Handbook*, Available at: https://www.oreilly.com/library/view/python-data-science/9781491912126/ (Downloaded: 06 April 2023).

Sharma, A. (2022) *How to master the popular DBSCAN Clustering Algorithm for machine learning*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/ (Accessed 9 April 2023).

Mullin, T. (2020) *DBSCAN parameter estimation using Python*, *Medium*. Medium. Available at: https://medium.com/@tarammullin/dbscan-parameter-estimation-ff8330e3a3bd (Accessed 9 April 2023).

Ackermann, M. R. *et al.* (2014) 'Analysis of agglomerative clustering', *Algorithmica. An International Journal in Computer Science*, 69(1), pp. 184–215. doi: 10.1007/s00453-012-9717-4.

Kumar, S. (2020) *Agglomerative clustering and dendrograms-explained*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/agglomerative-clustering-and-dendrograms-explained-29fc12b85f23 (Accessed 9 April 2023).