Machine Learning (CS60050) – Assignment 2

Rajat Bachhawat (19CS10073)

Kaushal Banthia (19CS10039)

REPORT

- The code is available on Google Colab using this link.
- The code is also added in the zip file as a .ipynb file.
- We begin with importing the diabetes dataset and dropping any row that has NaN values in it, using the dropna() function. Then we do some exploratory analysis on the dataset using inbuilt pandas functions and some graphs from matplotlib and seaborn.
- Finally we normalise the data. For this we have 2 options available, namely z-normalisation (using mean and std-dev) and min-max normalisation. We first tried out z-normalisation, but that gave really small values for Silhouette Index and Calinski Harabasz Score. So we moved on to min max normalisation. After applying this normalisation, we achieved better scores and trends over varying values of k.

NOTE: Normalisation is required, because of many reasons. Some of them being, removal of anomalies, reducing effect of some columns that have abnormally large values as compared to the others and making convergence easier and reducing the training time, by making all the columns have values in similar ranges.

0. Analysing the Data

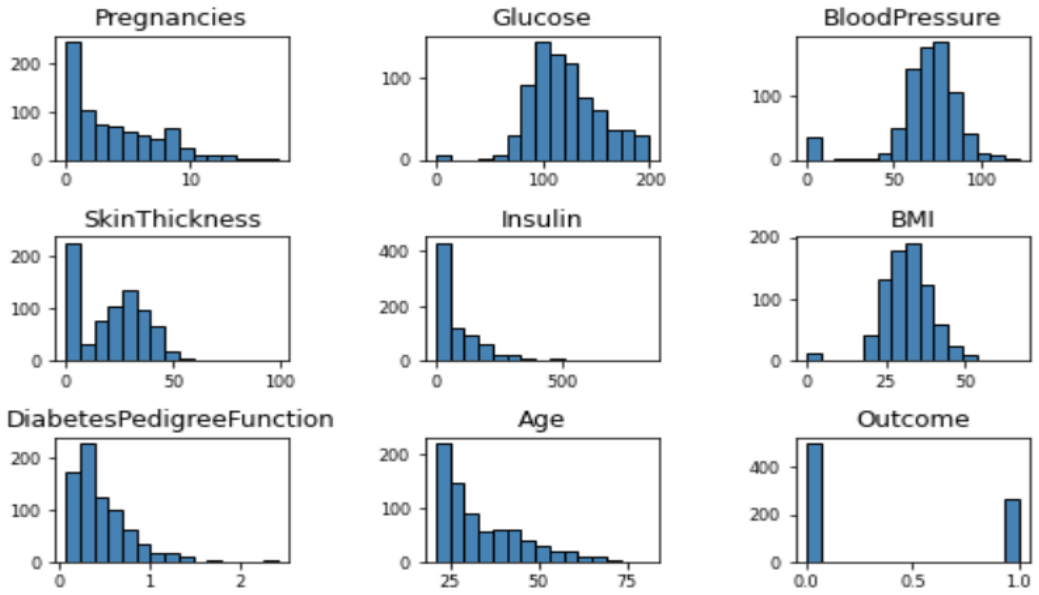
- We start with getting the dataset and dropping any rows that have NaN in them, using the dropna() function
- Then we get the basic information of the dataset, using the info() method.

```
Int64Index: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 60.0 KB
```

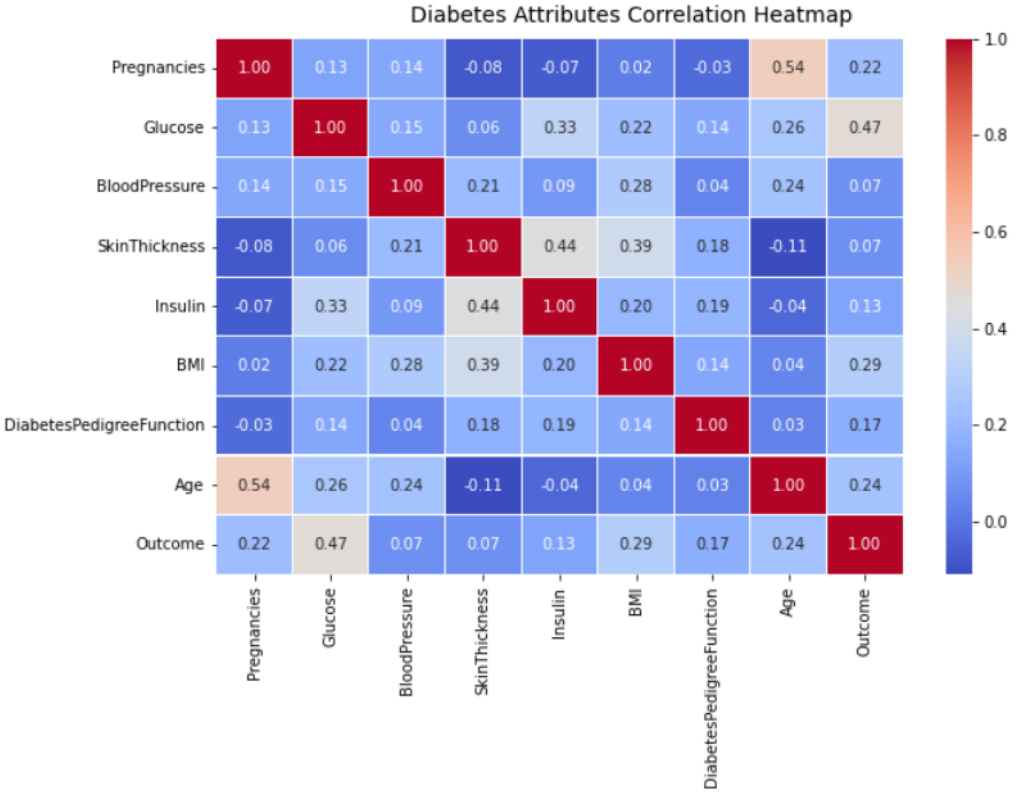
- Then we get some statistics of the data using the .describe() function

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

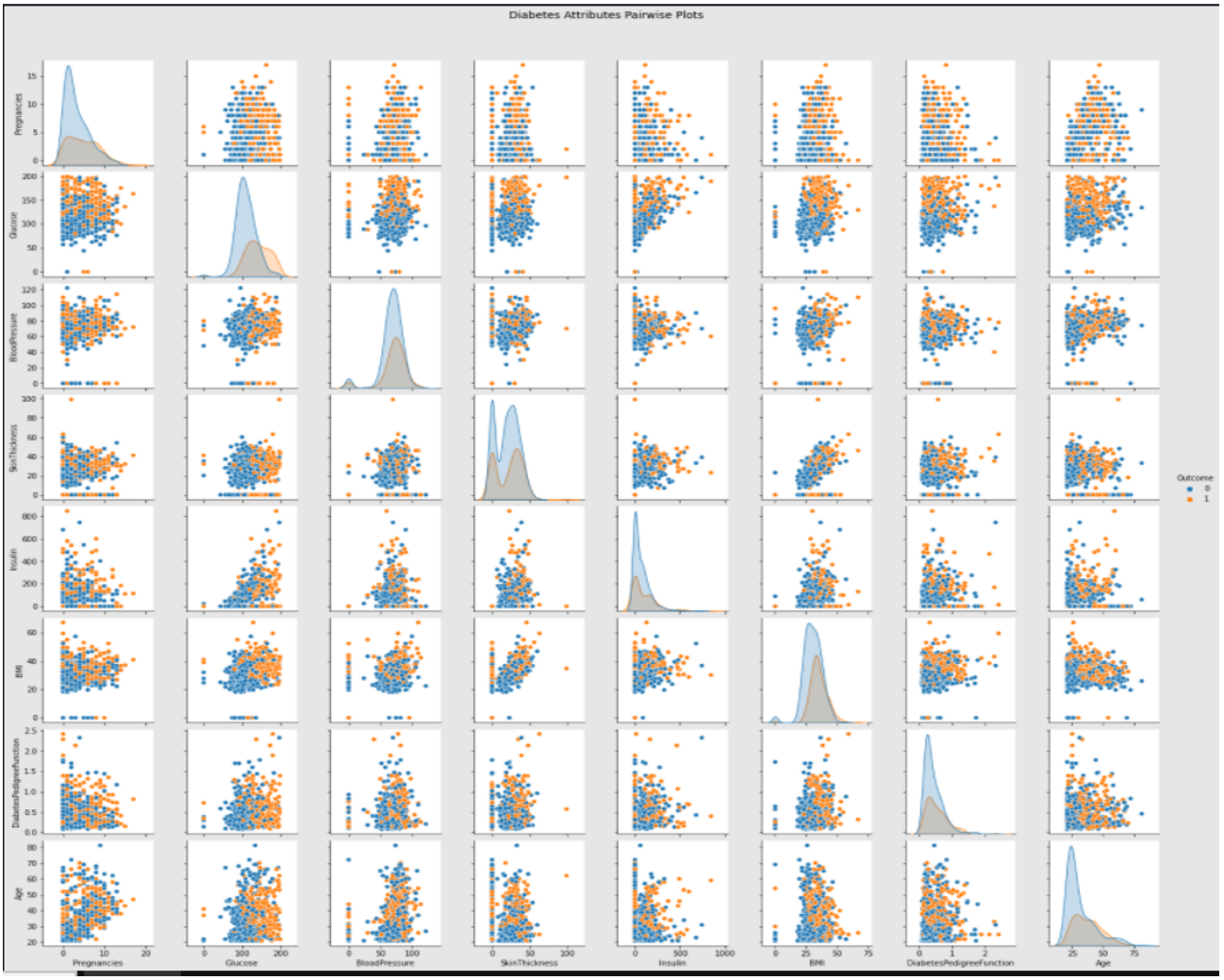
- Then we find out the histogram for the dataset to check out the number of occurrences for each attribute for each value that it takes



- We move on to plot the heatmap for the same dataset, that gives us the correlation between all the columns using colours. Warm colours mean a high correlation (closer to 1), while cool colours mean a low correlation (closer to 0).



- Finally, we plot the pairplot that gives us almost all the information in one plate, by plotting pairwise information for all the columns.



From the obtained pairwise scatter plots of the attributes, it is clear that the dataset that we are dealing with, is not very homogeneous. In most cases, we cannot observe well formed clusters based on the label (“Outcome”). This implies that applying any clustering algorithm to this data may not yield very promising results in terms of performance and accuracy.

1. Implementing K-Means Clustering

PROCEDURE

We define the following helper functions for assisting us in implementing the K-Means Clustering Algorithm.

- `calc_distance()`: Returns the distance between 2 n-dimensional points
- `assign_clusters()`: Assign cluster clusters based on closest centroid
- `calc_centroids()`: Calculate new centroids based on each cluster's mean
- `calc_centroid_variance()`: Calculate variance within each cluster

We then define the KMeans class, in which we have the following functions:

- `__init__()`: Initialises the class
- `train()`: Trains the K-Means model on the given dataset
- `train_predict()`: Trains the K-Means model on the given dataset and then predicts the output on the dataset
- `predict()`: Predicts the output on the dataset passed

We use Lloyd's algorithm to implement the K-Means Algorithm. It is sometimes also referred to as "naïve k-means", because there exist much faster alternatives.
We initialise k centroids by selecting k random samples from the given data. This is the initialisation step.
We assign each data point to that cluster with the nearest mean to that data point. This is the assignment step.
Then we recalculate the means (centroids) assigned to each cluster. This is the update step.

For the completion of this algorithm, we have two ways. The algorithm is set to run for a set number of iterations (`n_iter`). It stops only if the specified number of iterations have been run, or the algorithm has converged. We check for the convergence by comparing the centroids of the current clusters, with the centroids of the previous clusters. If they turn out to be the same, then the algorithm has converged and we break the loop at that iteration.

RESULT

These classes and functions help us to define the K-Means model and then train it on a given dataset for a user-defined value of k (the number of clusters).

2. Finding Clustering Performance

PROCEDURE

We use the method `train_predict` of the `Kmeans` class. This method clusters the given data into k clusters (k = 2 in our submitted notebook) and returns an array that contains the predicted clusters (cluster indices) assigned to each sample. We find clustering performance in two ways:

- Using available ground truth (data labels)
    - Adjusted Rand Index (ARI)

    - Fowlkes Mallows Score (FM)
    - Homogeneity Score
    - Normalised Mutual Info Score (NMI)

- Without using available ground truth
    - Silhouette Index
    - Calinski Harabasz Score

RESULT

```
--- Clustering Performance using available ground truth ---

Adjusted Rand Index: 0.10238136070253465
Fowlkes Mallows Score: 0.5972349586286805
Homogeneity Score: 0.05121587703671173
Normalised Mutual Info Score: 0.051736896303927755

--- Clustering Performance without using available ground truth ---

Silhouette Score: 0.26114611149614975
Calinski Harabasz Score: 237.11040774547635
```

CONCLUSION

- Using available ground truth (data labels)
    - **Adjusted Rand Index (ARI)**: Since we have randomly initialised the clusters, we expect this metric to be close to 0 (which it is).
    - **Fowlkes Mallows Score (FM)**: A high FM Score (as is our case), means that we have a good number of true positives. Thus, our non-homogeneous clusters still give us a good output.
    - **Homogeneity Score**: The value of the Homogeneity Score is very low. This implies that the clusters are not pure. They have differently labelled data points in them as is evident from the pair plots in Section 0.
    - **Normalised Mutual Info Score (NMI)**: This metric is very low, which means that the correlation between the class labels and the clusters is bad. This is because NMI quantifies the "amount of information" obtained about one random variable by observing the other random variable. If it is high, then the correlation is high and vice versa. But here, since the value is small, this means that correlation between the clusters and the class labels is very less.

- Without using available ground truth
    - **Silhouette Index**: The score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. Scores around zero indicate overlapping clusters.
      We have a low-medium silhouette score. This indicates that the clusters formed overlap each other a little bit. This fact may have been strengthened after we normalized the data (which brings the data points closer), as before normalization we obtained a higher score for silhouette index.
    - **Calinski Harabasz Score**: The score is defined as the ratio between the within-cluster dispersion and the between-cluster dispersion. We have got a decently high value for the Calinski Harabasz Score, which means the samples within our clusters are fairly alike.

3. Finding Most Suitable K for the given Data

PROCEDURE

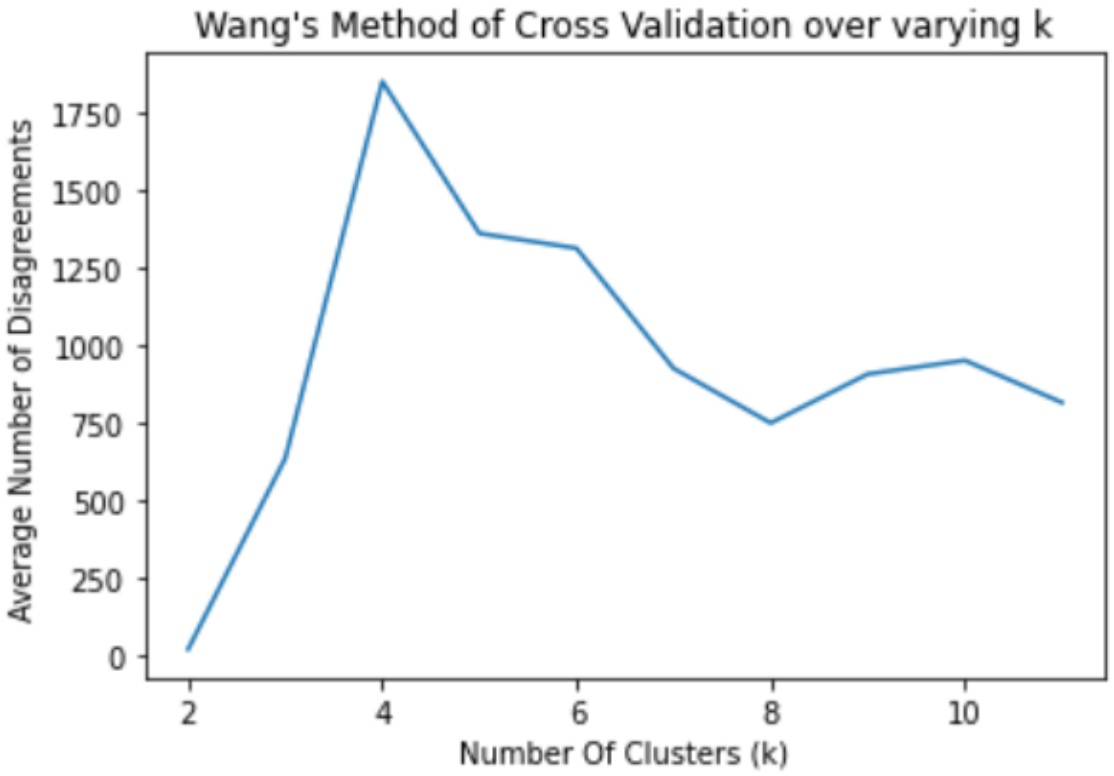
We use 3 methods to find the most suitable K for the given data, namely Wang's Method of Cross-Validation, the Silhouette Index and the Calinski-Harabasz Score for different values of K.

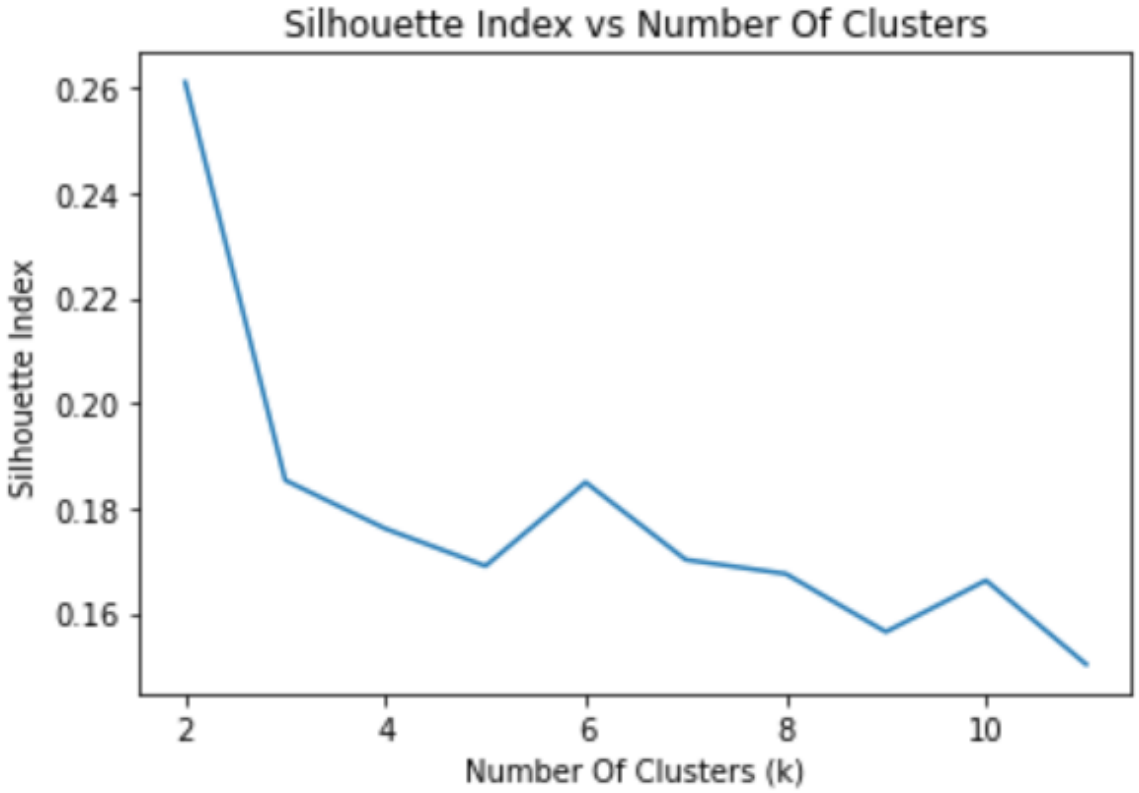
- **Wang's Method of Cross-Validation**: We find the number of average disagreements using this method and try to minimise it while varying K. The value of K that gives us the least number of average disagreements is the best K for our data (here 2, as seen by the graph).

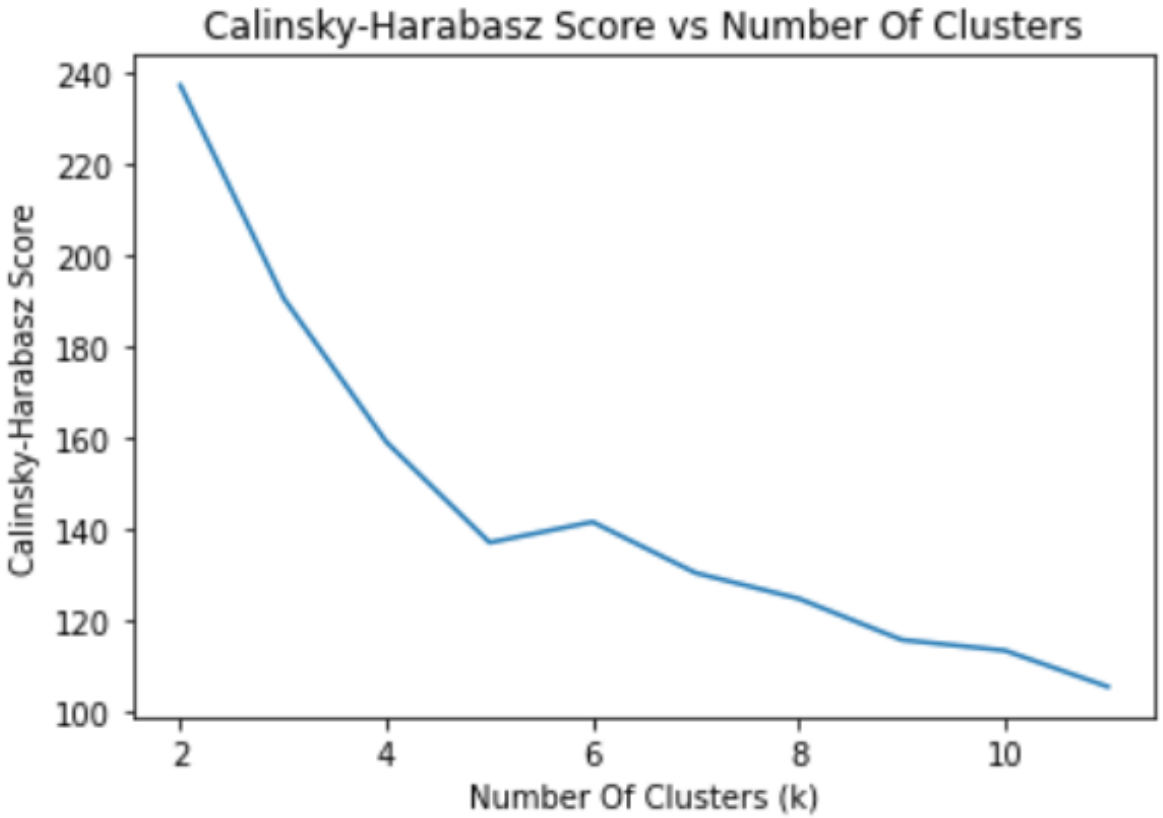
- **Silhouette Index**: We find the Silhouette Index for the predictions using the methods provided by the sklearn library. Since we want the Silhouette Index to be high, we vary K to find the best K for which the Silhouette Index is the highest. The value of K that gives us the highest Silhouette Index is the best K for our data (here 2, as seen by the graph).

- **Calinski-Harabasz Score**: We find the Calinski-Harabasz Score for the predictions using the methods provided by the sklearn library. Since we want this score to be high, we vary K to find the best K for which the Calinski-Harabasz Score is the highest. The value of K that gives us the highest Calinski-Harabasz Score is the best K for our data (here 2, as seen by the graph).

RESULT







CONCLUSION

We choose the value of K to be 2 after referring to all the graphs and the different values of the various scores, over a range of K, as this is best suited for our data (having 2 output labels).

4. Analysing the stability of the clustering over several different initializations

First we define a ‘Test A’ as follows:

```
1. select K points (either randomly or using a heuristic) as the
initial centroids of the *clusters*

2. split remaining data into 2 parts randomly (80:20) ratio

3. apply k-means on training data with K-random points selected in
step-1

4. label test data using k-centroids.

5. use metrics NMI, ARI, etc. in test data to understand clustering
accuracy

6. repeat 2-5 at least 50 times and report average metric
```

Then we use this Test A to determine if there is any change in the clustering outcome if we initialize K cluster centroids with random points in 50 different executions (or iterations). Each of these iterations has a different random initialisation of the K centroids.

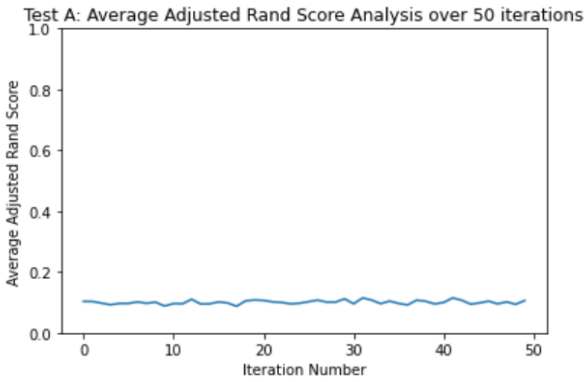
In part B, we try to see if we get more stable clusters (i.e. low standard deviation over the 50 iterations) if we use a heuristic (K-Means++) to choose the initial K centroids.

a. Checking the stability of the clustering over 50 random initializations of the initial K centroids

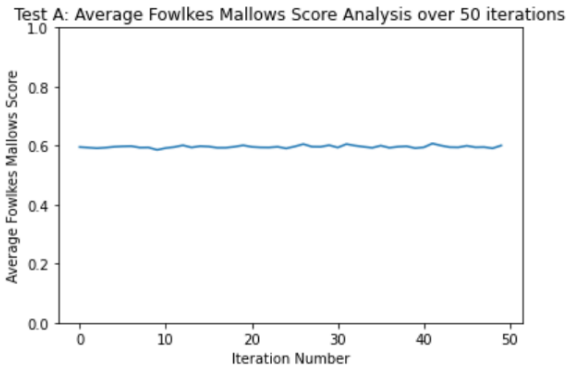
PROCEDURE

We do this without using any heuristic to choose the k initial centroids. We collect 4 scores after running the test A, namely, Average Adjusted Rand Score (ARI), Average Fowlkes Mallows Score (FM), Average Homogeneity Score (HM), Average Normalised Mutual Info Score (NMI). This process is repeated 50 times as suggested and then we plot the graphs of the scores with respect to the iteration number. We find this graph to be nearly linear and the std. deviation (dispersion metric) is very low, thus our model is stable even without an initialization heuristic.

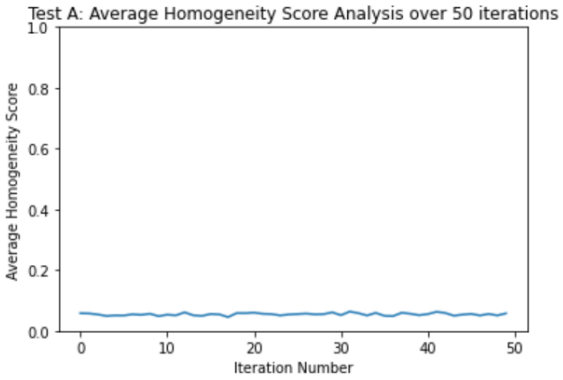
RESULT



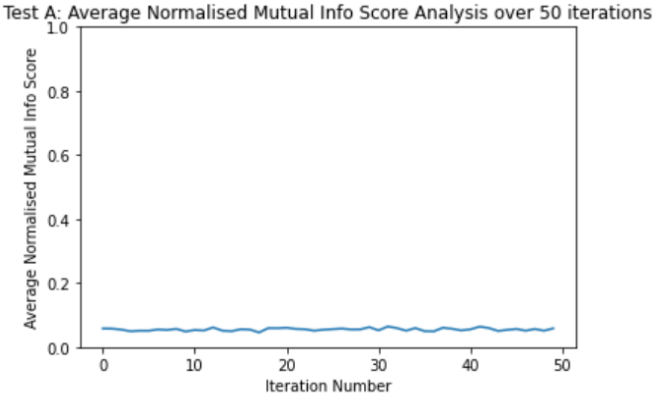

```
Mean over 50 Iterations = 0.1004255833999036
Standard Dev. over 50 Iterations = 0.006164274523057471
```




```
Mean over 50 Iterations = 0.595965424664374
Standard Dev. over 50 Iterations = 0.004014911135917208
```




```
Mean over 50 Iterations = 0.05455174497470611
Standard Dev. over 50 Iterations = 0.0040745908856798544
```




```
Mean over 50 Iterations = 0.055042325142949586
Standard Dev. over 50 Iterations = 0.0041644724736847605
```

CONCLUSION

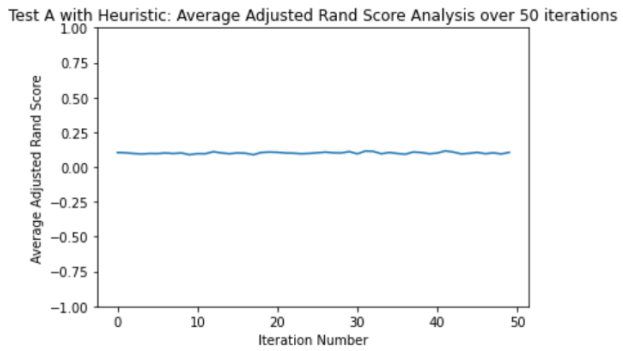
As seen from the almost linear graphs and the 4 scores, our model is stable on this dataset. This means that the formation of clusters is not heavily dependent on the initialisation. But still we give K-Means++ a try, hoping to improve the stability of our model even further.

b. Checking the stability of the clustering over 50 heuristic based (K-Means++) initializations of the initial K centroids

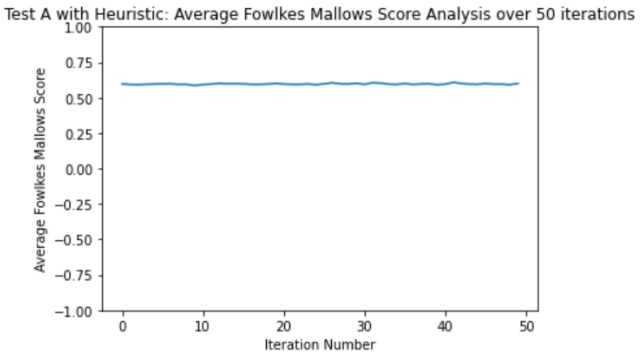
PROCEDURE

We do this using a heuristic (K-Means++) to choose the k initial centroids. This is done to improve the stability of a model. Although our model is already very stable, we still try to improve it by using this heuristic. We collect 4 scores after running the test A, namely, Average Adjusted Rand Score (ARI), Average Fowlkes Mallows Score (FM), Average Homogeneity Score (HM), Average Normalised Mutual Info Score (NMI). This process is repeated 50 times as suggested and then we plot the graphs of the scores with respect to the iteration number. We find this graph to be nearly linear, thus our model is stable.

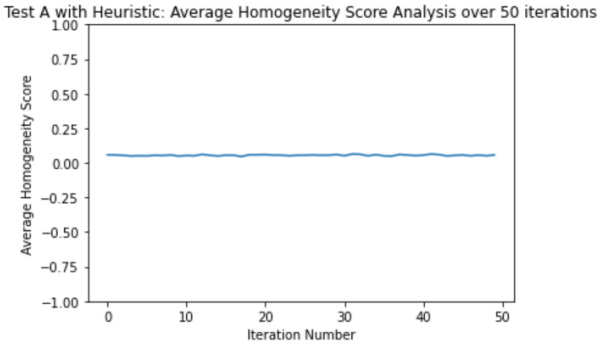
RESULT



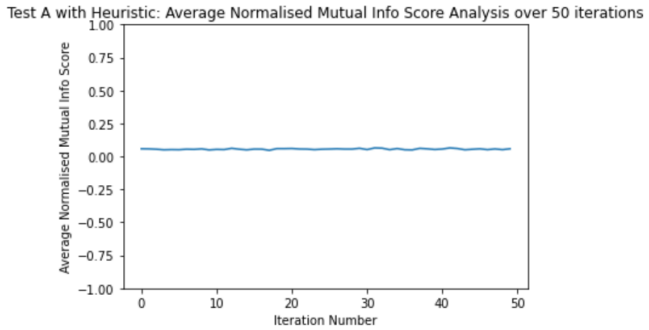

```
Mean over 50 Iterations = 0.10072017486543272
Standard Dev. over 50 Iterations = 0.006168386810223996
```




```
Mean over 50 Iterations = 0.5960867910749468
Standard Dev. over 50 Iterations = 0.004041973536068245
```




```
Mean over 50 Iterations = 0.05472562346872939
Standard Dev. over 50 Iterations = 0.004043816907642185
```




```
Mean over 50 Iterations = 0.05521365421925044
Standard Dev. over 50 Iterations = 0.004137469059202788
```

CONCLUSION

As seen from the almost linear graphs and the 4 scores, our model is stable on this dataset. Thus, we are good to go!

We also formulate the following table to compare the 2 initialisation processes and understand which one performs better.

Metric	**Attribute (over 50 iterations)**	**Random Initialisation**	**K-Means ++**
Adjusted Rand Index (ARI)	*MEAN*	0.1004	0.1007
	STANDARD DEVIATION	0.0062	0.0062
Fowlkes Mallows Score (FM)	*MEAN*	0.5960	0.5961
	STANDARD DEVIATION	0.0040	0.0040
Homogeneity Score (HM)	*MEAN*	0.0546	0.0547
	STANDARD DEVIATION	0.0041	0.0040
Normalised Mutual Info Score (NMI)	*MEAN*	0.0550	0.0552
	STANDARD DEVIATION	0.0042	0.0041

- As we can see from the above table, in all the cases, the mean is greater in case of K-Means++ than in random initialisation. This means that K-Means++ makes better clusters than random initialisation.
- Also, the standard deviation (our dispersion metric) for K-Means++ is less than or equal to that in random initialisation. This means that there is lower spread in the clusters and they have a small variance.

Both these points support the fact that we should prefer the heuristic (K-Means++) over random initialisation, as we get marginally better, more stable outcomes (clusters) in this case.