Report for the ML Assignment-2

Implementing K-Means Clustering in Python

Prepared by

Shobhit Gupta
19EC10058

Shreyash Vaish
19EE30030

Dataset - Pima Indians Diabetes Database

url - https://www.kaggle.com/uciml/pima-indians-diabetes-database

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females, at least 21 years old of Pima Indian heritage.

Sample from the data-

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Attributes

1. *Pregnancies* Number of times pregnant
2. *Glucose* Plasma glucose concentration in oral glucose test
3. *BloodPressure* Diastolic blood pressure (mm Hg)
4. *SkinThickness* Triceps skinfold thickness (mm)
5. *Insulin* 2-Hour serum insulin (mu U/ml)
6. *BMI* Body mass index (weight in kg/(height in m)^2)
7. *DiabetesPedigreeFunction* Diabetes pedigree function
8. *Age* Age (years)
9. *Outcome* Class variable (0 or 1)

Description of the data

	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

All the attributes of the dataset are continuous-valued, except the outcome.
Number of unique values of each attribute are given below -

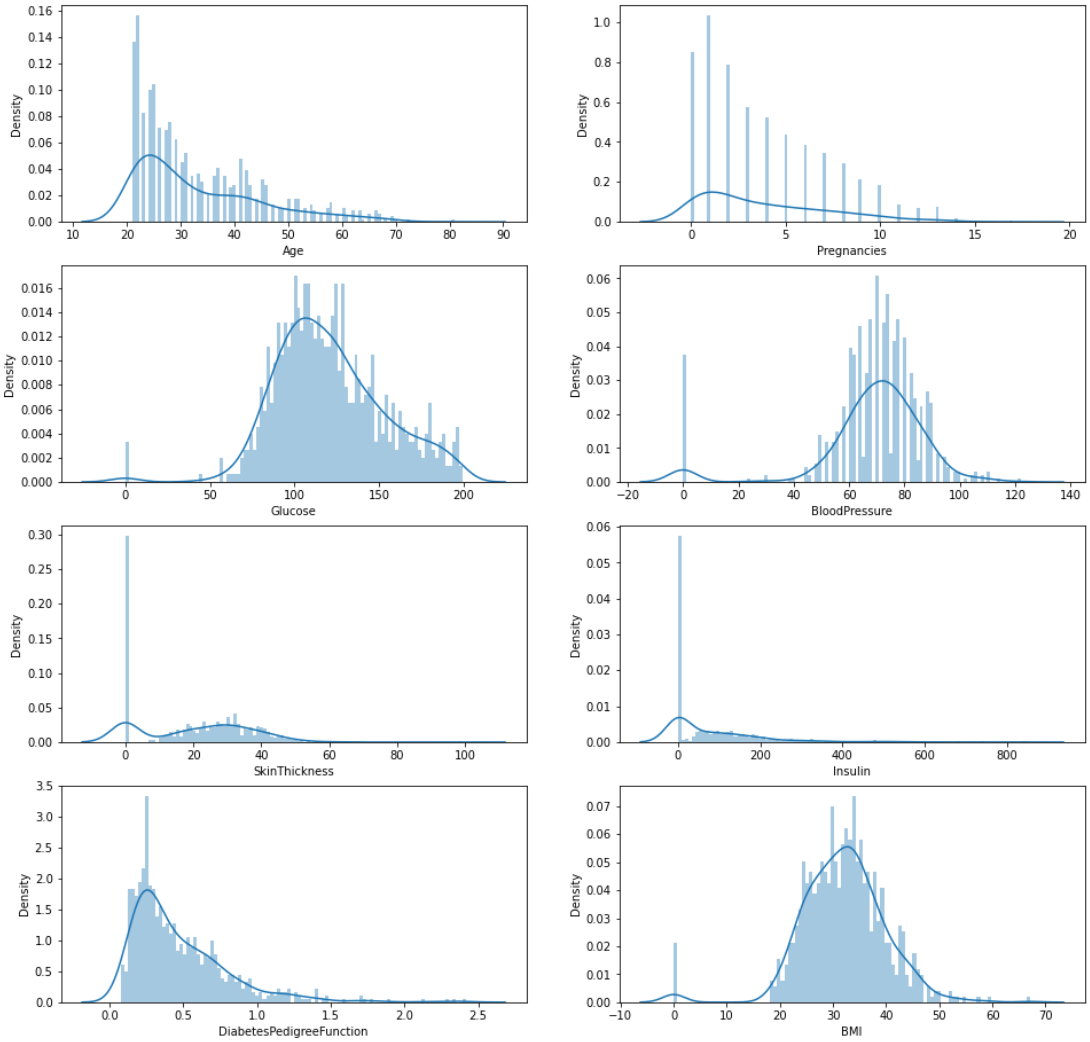
```
Pregnancies                 17
Glucose                    136
BloodPressure               47
SkinThickness               51
Insulin                    186
BMI                        248
DiabetesPedigreeFunction   517
Age                         52
Outcome                      2
dtype: int64
```

Correlation between features is shown below -

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

From the correlation matrix, we can observe the following things -
(i) “Outcome” has the largest correlation with “Glucose” although it is still not big enough(<0.5) to predict the outcome with sufficient confidence.
(ii) “Pregnancies” is highly correlated with “Age”.
(iii) All the attributes have positive correlation with Outcome, that is, increase in any attribute(pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function and age) increases the chance of the patient having diabetes.
(iv) Attributes SkinThickness and BloodPressure have very less correlation with Outcome, that is, SkinThickness and BloodPressure attributes have a smaller role in determining the target value.

What are the distributions of various features?



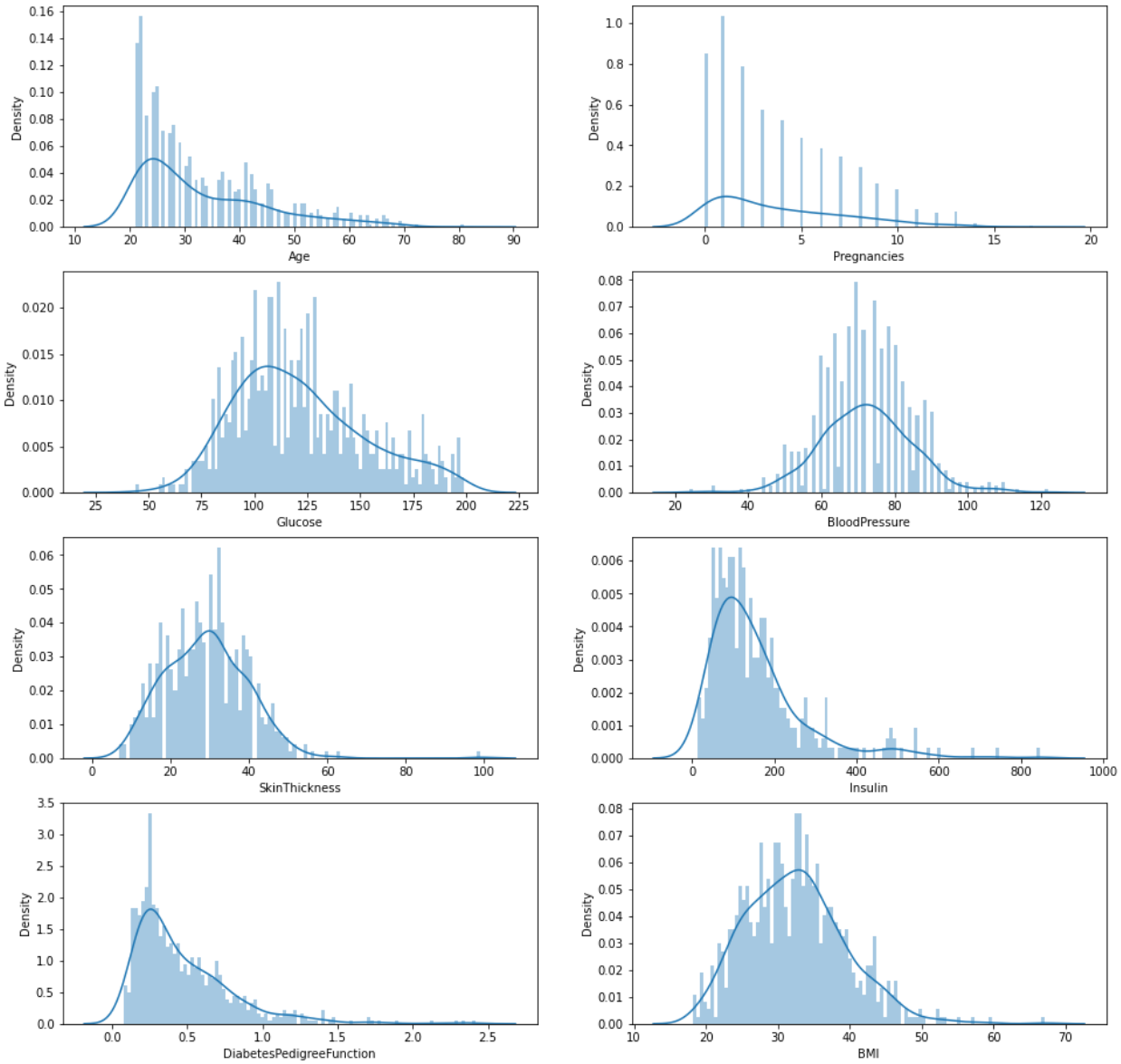
For some variables, 0 value is unrealistic and hence, it should be treated as a missing value and suitable arrangements to substitute these values.
These unrealistic 0 values exist in **BMI, Insulin, SkinThickness, Glucose and BloodPressure attributes**.
Number of zeros existing in the above features -

```
Pregnancies                111
Glucose                      5
BloodPressure               35
SkinThickness              227
Insulin                    374
BMI                         11
DiabetesPedigreeFunction     0
Age                          0
Outcome                    500
dtype: int64
```

After replacing the zeros in the columns - BMI, Insulin, SkinThickness, Glucose and BloodPressure - we obtain the number of NaN values as follows -

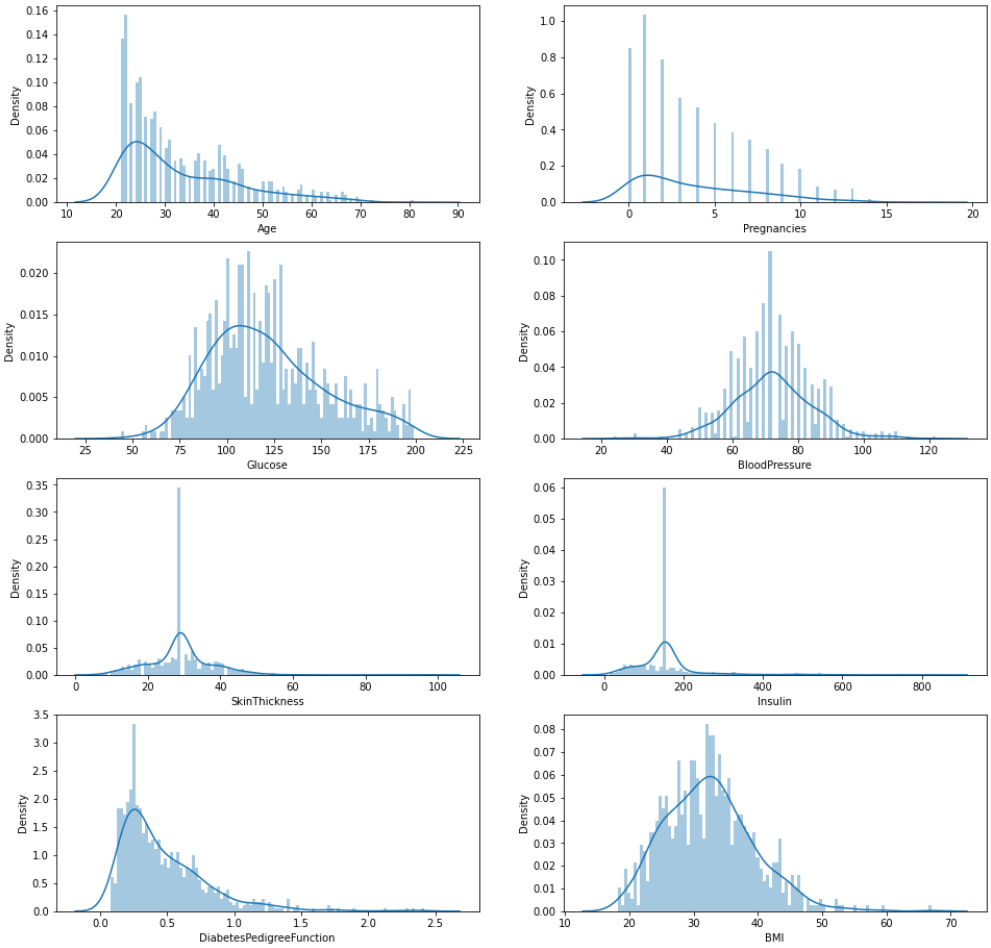
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

After replacing with the NaN values, we obtain the following distributions -

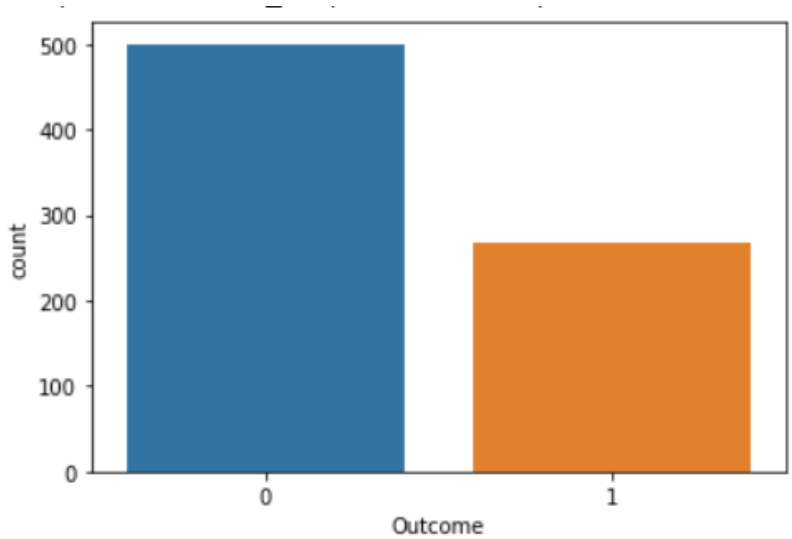


We can see that after replacing zeros with NaN values, the data is approximately gaussian distributed. Hence, the mean of that attribute can be a good measuring for the missing values.

After replacing the values with mean in the selected columns(BMI, Insulin, SkinThickness, Glucose and BloodPressure), we get the following distributions -



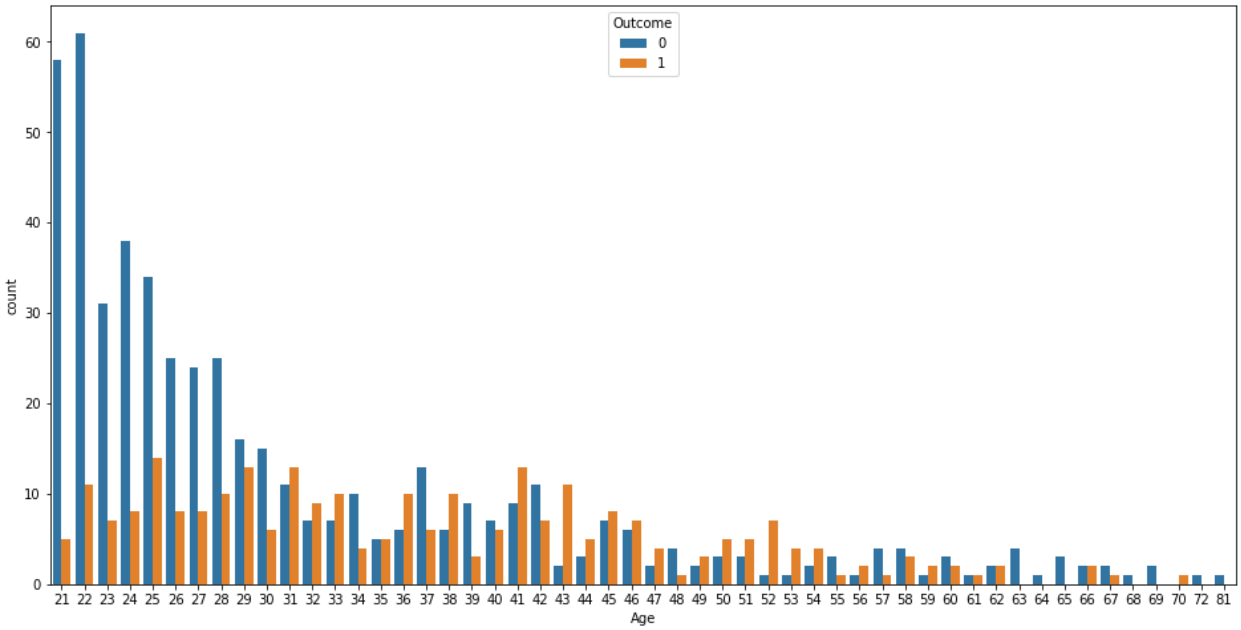
Number of positive and negative patients -



Total number of patients = 500

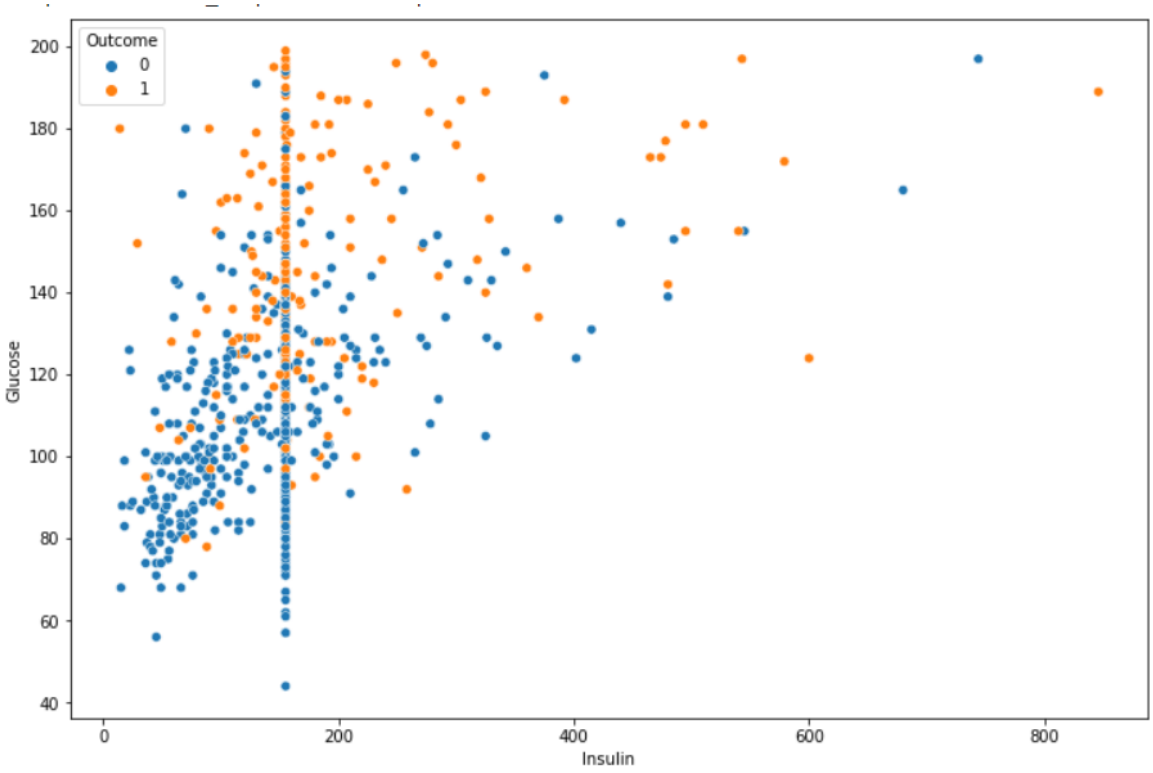
Total number of non-patients = 268

Variation of Outcome wrt Age -



We can see that chances of having diabetes increases with age and after age 30, chances are much higher.

The following plot shows the variation of insulin and glucose, along with the class labels -



The plot shows that more glucose causes an increase in insulin production. But at the same time, high levels of glucose and insulin in blood suggest that the person is diabetic.

How to compile and run the code?

FOR WINDOWS & LINUX-

1. Extract the "Group31_Assg2.zip".
2. Make sure Python3 is installed on the system.
3. Install the dependencies stored in the "requirements.txt" file using the following command - **pip install -r requirements.txt**
4. For running any python file, use the following command - **python filename.py**
    i) For viewing the data analysis part, run the "analyse.py" file.
    ii) For each sub task i = 1,2,3,4,5., run the corresponding "task_i.py" file.

Description of files in the root folder of assignment

1. k_means.py:
   This file contains the complete implementation of k-means clustering and its associated functions.

2. analyse.py:
   This file deals with the data analysis part. All the tables and plots shown previously can be generated by executing this file.

3. task_q.py (q = 1, 2, 3, 4) contains the code for sub task q (= 1, 2, 3, 4).

4. original_data.csv: Data before pre-processing.

5. mean_data.csv: Data after pre-processing.

Pre-processing the data

There are a lot of zeros in the following columns- BMI, Insulin, SkinThickness, Glucose and BloodPressure, which is unrealistic, since these values practically cannot be 0. So, we treat these values as missing values. So, we first replace these zeros with NaN. After replacing it with NaN, we can see that the distribution of these attributes is close to gaussian. Hence, mean can be a good approximation for the missing values. So, we replace these NaN values with the mean of the corresponding columns.

i) Implementing K-means algorithm

with the user given “K”

Approach -

At first, the instances are normalized. This is done by dividing the value of each attribute for every instance by the difference between the maximum and the minimum value of that attribute overall.

$$X(instance,\ attribute)\ /= (max(attribute) - min(attribute))$$

Then, we take the user input for the value of K and then do the following -

Firstly, select K data points at random from the dataset as the initial centers of the clusters.

Then, assign the data points to the clusters as explained below -

Assigning data points to the clusters -

We begin with a data point and compute its **euclidean distance** from each cluster and then assign it to the cluster whose centre is closest to that data point. Then, move to the next data point. Continue this till all the data points have been covered.

Euclidean distance is given by -

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}$$

where p and q are two data points and p_i, q_i represent the value of the i-th attribute for the points p and q respectively.

Note : Target attribute is not considered when computing the euclidean distance.

Next step is to update the centers of the clusters as explained below -

Updation of cluster centers -

After each data point has been assigned a cluster, the centres of the clusters need to be updated. The cluster centres have all the attributes as any other data point, and the value of these attributes is equal to the average of the attribute values of all the data points assigned to that cluster. In this way, we compute the new centres of the clusters.

When to stop?

We iterate the above process till the cluster centres stop changing much after each iteration. For this purpose, we compute the maximum change in the centre position before and after updation of centres. We stop the iteration if this maximum change is not more than a tolerance value (by default it is taken as 0). The tolerance is maximum error which can be tolerated before the convergence can be said to occur, as centres might take many iterations to become fixed. With a larger tolerance, the convergence occurs faster, but may result in less accuracy.

Dealing with empty clusters: We can sometimes face a situation where we encounter any empty clusters after assigning the data points. To deal with that, we have re-initialized the cluster centers and then performed k-means again.

Results -

On executing “task_1.py”, we can see the number of iterations required for convergence.

Output for some runs are given below with user given K=10 :-

For tolerance=0,

```
Iteration 1 done
Iteration 2 done
Iteration 3 done
Iteration 4 done
Iteration 5 done
Iteration 6 done
Iteration 7 done
Iteration 8 done
Iteration 9 done
Iteration 10 done
Iteration 11 done
Iteration 12 done
Iteration 13 done
Iteration 14 done
Iteration 15 done
Iteration 16 done
The number of iterations required for convergence: 16
```

For tolerance=0.005,

```
Iteration 1 done
Iteration 2 done
Iteration 3 done
Iteration 4 done
Iteration 5 done
Iteration 6 done
Iteration 7 done
Iteration 8 done
Iteration 9 done
Iteration 10 done
Iteration 11 done
The number of iterations required for convergence: 11
```

Here, we can see for the same set of initial centres, the convergence occurs faster with a tolerance.

ii) Providing the clustering performance

with and without ground truth

Approach -

After implementing the k-means clustering algorithm, we can obtain the predictions from our model for the target attribute(“Outcome”). For this purpose, we assign the label to a cluster as the target value which occurs the maximum number of times in the cluster closest to the point.

After that, we provide the measures of performance of our clustering.
There can be two cases for this based on availability or non-availability of the ground truth values.

Case (i) - Ground truth values are available

We can use metrics like homogeneity score, normalized mutual information score(NMI) and Adjusted Rand Index score(ARI).

Homogeneity score - This score is useful to check whether the clustering algorithm returns clusters which contain samples belonging to only one class.
It's defined as:

$$h = 1 - \frac{H(Y_{true}|Y_{pred})}{H(Y_{true})}$$

It's bounded between 0 and 1, with low values indicating a low homogeneity and 1 indicating complete homogeneity, that is, each cluster contains data points from only one class.

Normalized mutual information (NMI) score - Normalized mutual information (NMI) gives us the reduction in entropy of class labels when we are given the cluster labels.
Where the entropy is given by -

$$H(p) = -\sum_i p_i \log_2 (p_i)$$

Adjusted Rand Index (ARI) score - Adjusted Rand index is a measure of similarity between two clusterings.

Case (ii) - Ground truth values are not available

We can use metrics like silhouette score and calinski-harabasz score.

Silhouette score - It is the measure of goodness of clustering when ground truth is not available. It shows how distinguished or indifferent the clusters are from each other.
It is given by :

$$\text{Silhouette score} = \frac{(b-a)}{max(b-a)}$$

where b is the average inter-cluster distance, that is, average distance between cluster centres and a is the average intra-cluster distance, that is, average distance between the cluster centre and the data points in that cluster.

Calinski-Harabasz index - The CH Index (also known as Variance ratio criterion) is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). CH index has a form of (a.Separation)/(b.Cohesion) , where a and b are scalars, and cohesion is estimated based on the distances from the data points in a cluster to its cluster centroid, and separation is based on the distance of the cluster centroids from the global centroid.
It is given by :

$$CH = \left[\frac{\sum_{k=1}^{K} n_k \left\| c_k - c \right\|^2}{K-1} \right] / \frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} \left\| d_i - c_k \right\|^2}{N-K}$$

where n_k and c_k are the number of points and the centroids of the k-th cluster, c is the global centroid, N is the total no. of data points.
Higher value of CH index means the clusters are dense and well separated.

Results -
We can obtain the values of above described performance metrics by executing the “task_2.py” file.
Output for a sample run of the above file for user given K=5 :-

```
The metrics with ground truth are:
The homogeneity score is: 0.06677516320310888
NMI is: 0.08167715983594863
ARI is: 0.13281621705034463

The metrics without ground truth are:
The silhouette score is: 0.146205260397704 46
The calinski harabasz score is: 76.24145660674166
```

iii) Finding the most suitable K

using some valid internal indices

Approach -
First we select a suitable range of K values. In our case, we choose [2, 70]. We vary K in this range and for each K, we perform k-means clustering on the dataset and get predictions from this model as we did previously. After getting the predictions, we record the performance of clustering on the following metrics :-

(i) Total variance- Total variance is calculated as the summation of squared distances of all the data points from its associated cluster centre. Here, the distance is the ‘euclidean distance’ which we also used previously.

(ii) Silhouette score- Explained in part(ii).

(iii) Wang’s cross validation method- The data is divided in three parts S_1, S_2, S_3 with $|S_1| = |S_2|$. Then S_3 is divided into different clusters using the k-means outputs on S_1 and S_2. Then, derangements are calculated as the number of pairs where clustering results of S_1 and S_2 contradict i.e in one they are in the same cluster and in the other they are in a different cluster.

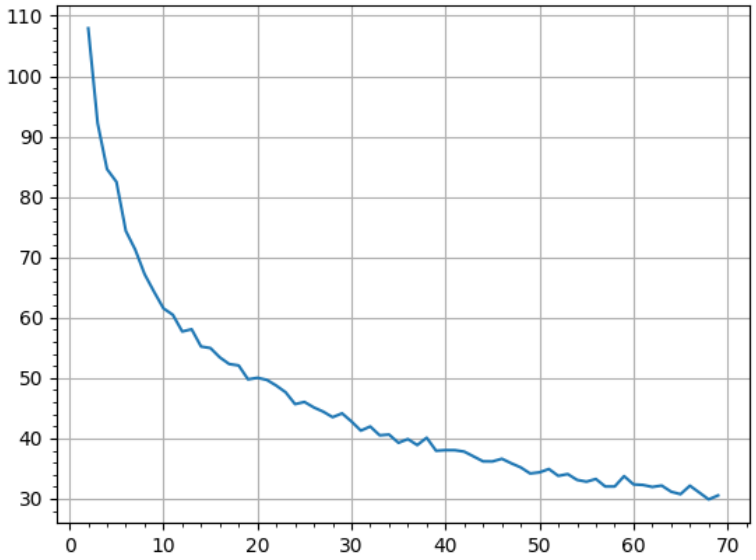
(iv) Accuracy score- Accuracy score represents the ratio of number of correct predictions made by our model to the total number of data points. Its value lies between 0 and 1 where 0 signifies all predictions are wrong and 1 represents all predictions are correct.

After computing these metrics for each k in the chosen range, we plot these metrics with respect to k. k value is selected such that the change in metric is significantly large. On further increase in k, the change in metric is not significant compared to the change at selected value. For silhouette, the higher score is better.

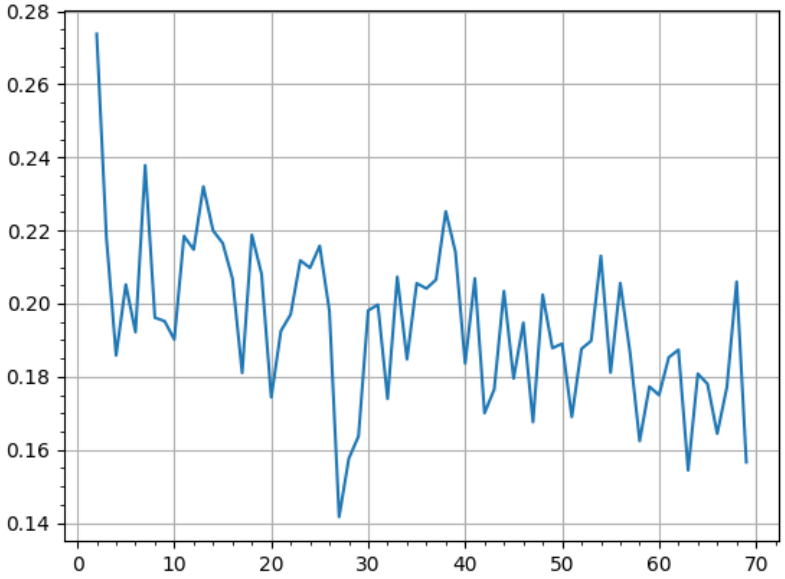
Results -

For plotting purposes, we used the python “matplotlib” library. By running the “task_3.py” file, we get the following plots in the files “tot_variance.png”, “Sill_Sc.png”, “wang_method.png” and “acc_sc.png”-

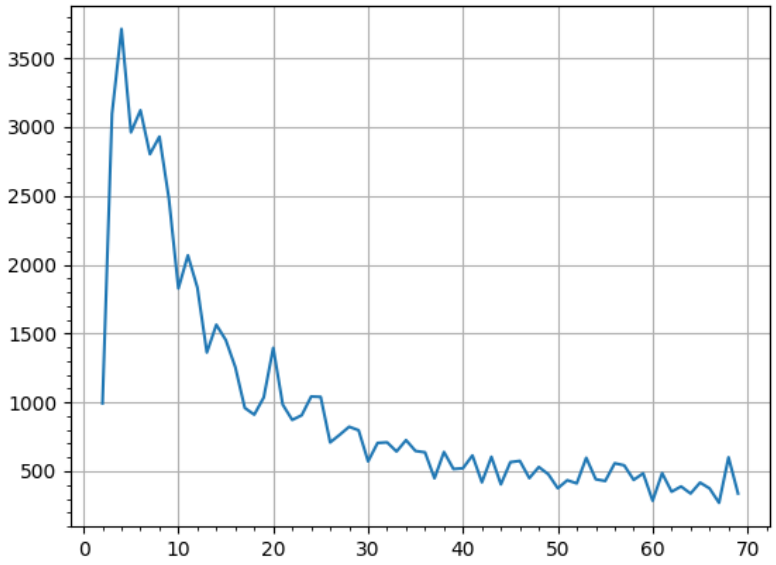
Plot for total variance vs k:



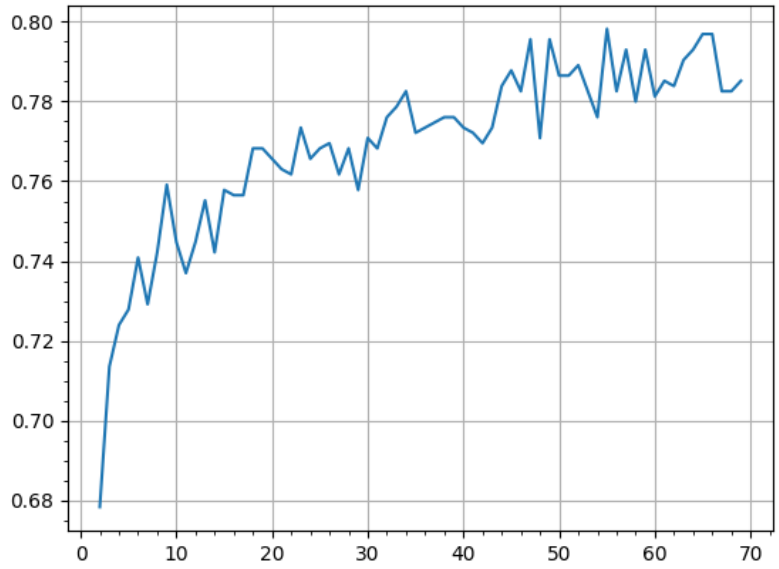
Plot for Silhouette score vs k:



Plot for wang's method vs k:



Plot for accuracy score vs k:



We can observe from the above plots that at k = 10, there is a steep change in the above metrics, but on further increase in k, the change in metrics is less. So, k = 10 is our selected value.

iv) Heuristic-based initialization

for minimizing metric variations

Approach -

In the previous parts, we used the random-initialization technique for initializing cluster centres. For this initialization method, we observe the variation in metrics- NMI, ARI and homogeneity. For observing the variations, we use the following Test-A as given in the problem statement -

Test-A

1. select K random points from the original data
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 some metric(NMI,ARI, etc.) in test data to understand clustering accuracy
6. repeat 2-5 at least 50 times and report average metric

Repeat the above procedure for 50 different K-random points. Report the dispersion of the metric.

To reduce the above variation, we try to use some heuristic based initialization instead of random initialization. For a smaller dataset, kmeans++ gives good results. So, it is suitable for our case. We implement kmeans++ as described below :

kmeans++ -

The first center c_1 is chosen randomly. Then, the i-th (i=2,3,..,k) center c_i is chosen as x' with a probability proportional to the square of the minimum distance from the selected i-1 centers.

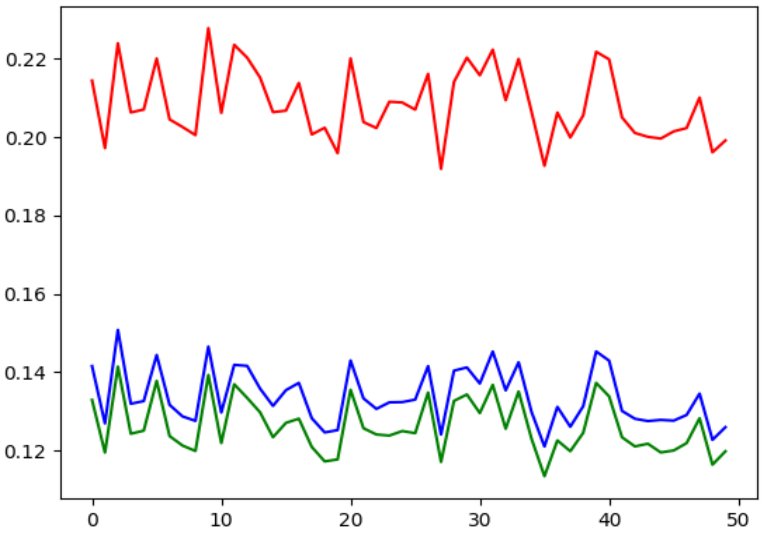
$$p(x') = \frac{\min_{j=1,2,..,i-1} \|x' - c_j\|^2}{\sum_x \min_{j=1,2,..,i-1} \|x - c_j\|^2}$$

After initializing the centers(using random or heuristic based initialization), we perform the usual k-means with k selected from part (iii).

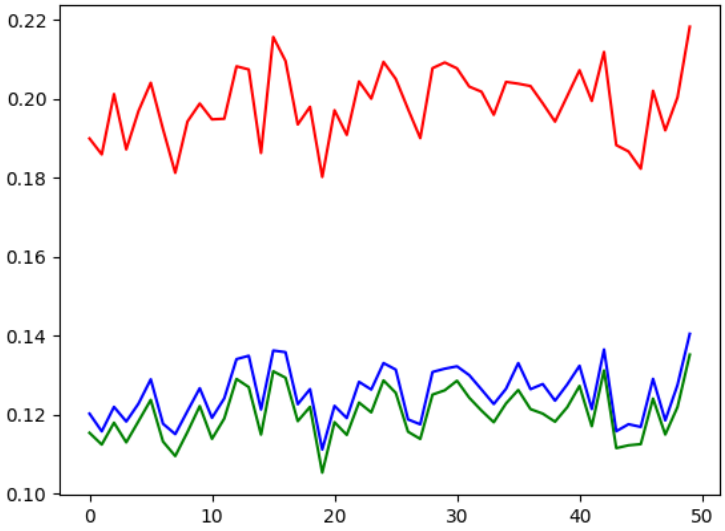
Results -

On running the "task_4.py" file, we obtain the plots of variation of the metrics with random initialization in "random_center_init.png" and with heuristic-based initialization in "heuristic_based_init.png". For a sample run with the selected k = 10(as seen in part(iii)), the obtained plots are shown below -

Random initialization -



Heuristic Based -



With random_center_init, the variance of metrics were:
The variance of NMI: 5.014487399753603e-05 and the mean of NMI: 0.1337291256114543
The variance of ARI: 8.022760402594947e-05 and the mean of ARI: 0.20846762397428772
The variance of homogeneity: 4.735821589487224e-05 and the mean of homogeneity: 0.12623716792950565

With heuritics_based_init, the variance of metrics were:
The variance of NMI: 4.4592392536978165e-05 and the mean of NMI: 0.12539196020695953
The variance of ARI: 7.694850835078537e-05 and the mean of ARI: 0.1986744568642369
The variance of homogeneity: 4.0351673042064023e-05 and the mean of homogeneity: 0.1203455547640338

Hence, we can see that variance in NMI, ARI, and homogeneity is decreased when we use the heuristic-based approach.

Conclusion

With this assignment on implementation of k-means clustering in python, we went through each and every step of building this machine learning model in detail. The dataset we used is “Pima Indians Diabetes Database”. To get insights and to understand the data, we start off with analyzing the data and the relationships among its various attributes and how our target attribute - “Outcome” is affected by these individual attributes.

The next step is to pre-process the data, that is, to use an effective strategy to remove the missing values in the data, since some columns include zero values which is not practically possible for that column. For dealing with these values, we replace them with the mean for the corresponding attribute.

Then, the next step is to build the k-means clustering model. We begin with k initial centres and then assign a cluster to each data point based on minimum euclidean distances between that point and the cluster centres. After that, we update the cluster centre position as the average of the attribute values of the data points for that cluster. In order to achieve faster convergence, we can also specify a small tolerance such that the clustering stops as soon as the maximum change in centre position is less than that tolerance. By default, we keep this tolerance as 0, so that we have maximum accuracy although it may take more time for convergence.

Next step is to obtain the predictions from our clustering. For that purpose, the class which occurs maximum time in a cluster is assigned as predicted target value for all the data points closest to that cluster. Then, we provide the performance measure of the clustering based on some metrics with respect to the presence or absence of ground truth - Homogeneity score, NMI score and ARI score when ground truth available, Silhouette score and Calinski Harabasz score when ground truth is not available.

Since k is the hyperparameter for k-means clustering, the choice of k is very important in determining the maximum performance that we can achieve from our clustering algorithm. For this purpose, we vary k in a suitable range and measure its performance on some metrics, namely, total variance, silhouette score, and accuracy score, and using Wang's method of cross validation.

After making the choice of k, we showed how random initialization can result in significant variations in the metrics, suggesting that our results are not very stable. In order to overcome this, we showed how we can use some heuristic-based approach for initialization of centres, like kmeans++ and we also showed how it compared to the random-initialization.

After completing the assignment, we now have a better understanding of how k-means clustering works and the various techniques with which we can make the model robust and give a better performance with stable results. K-means clustering can typically be applied to any dataset that is numeric, continuous and is not necessarily labelled. K-means is very suitable for such scenarios where we need to identify groups of similar objects from a randomly distributed set of objects. It always tries to construct a nice spherical shape around the centroid. But at the same time, it fails to perform well if the pattern in the dataset follows a complex geometric shape. Despite these drawbacks, k-means is still very popular in identifying patterns from the data with many practical applications including customer segregation, fraud detection, and medical diagnosis among others.