# Machine Learning Report on Assignment – 2

# K-Means Clustering

**By: Group 32**

Krinal Patel (21CS60R39)

Sarvesh Gupta (21CS60R53)

## Dataset: Pima Indians Diabetes Database

**Details of Dataset:**

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. 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. The datasets consist of several medical predictor variables and one target variable.

**Link of Dataset:** https://www.kaggle.com/uciml/pima-indians-diabetes-database

## Procedure:

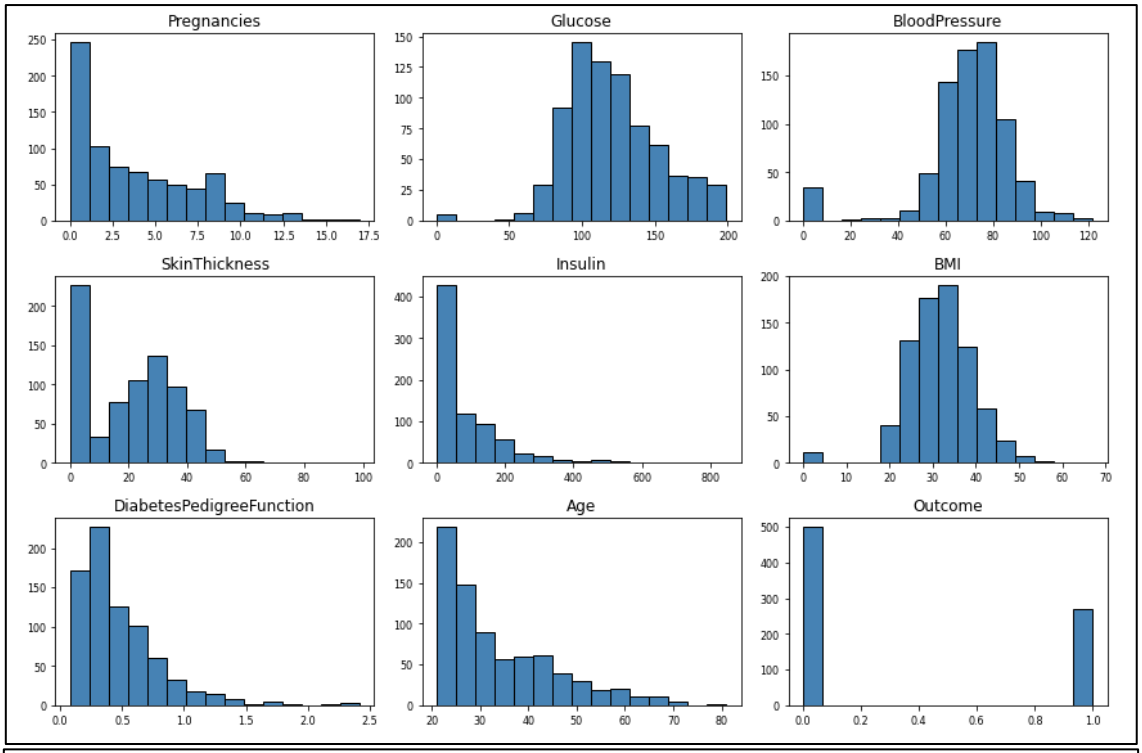
### Data Pre-Processing:

```
df.head()
```

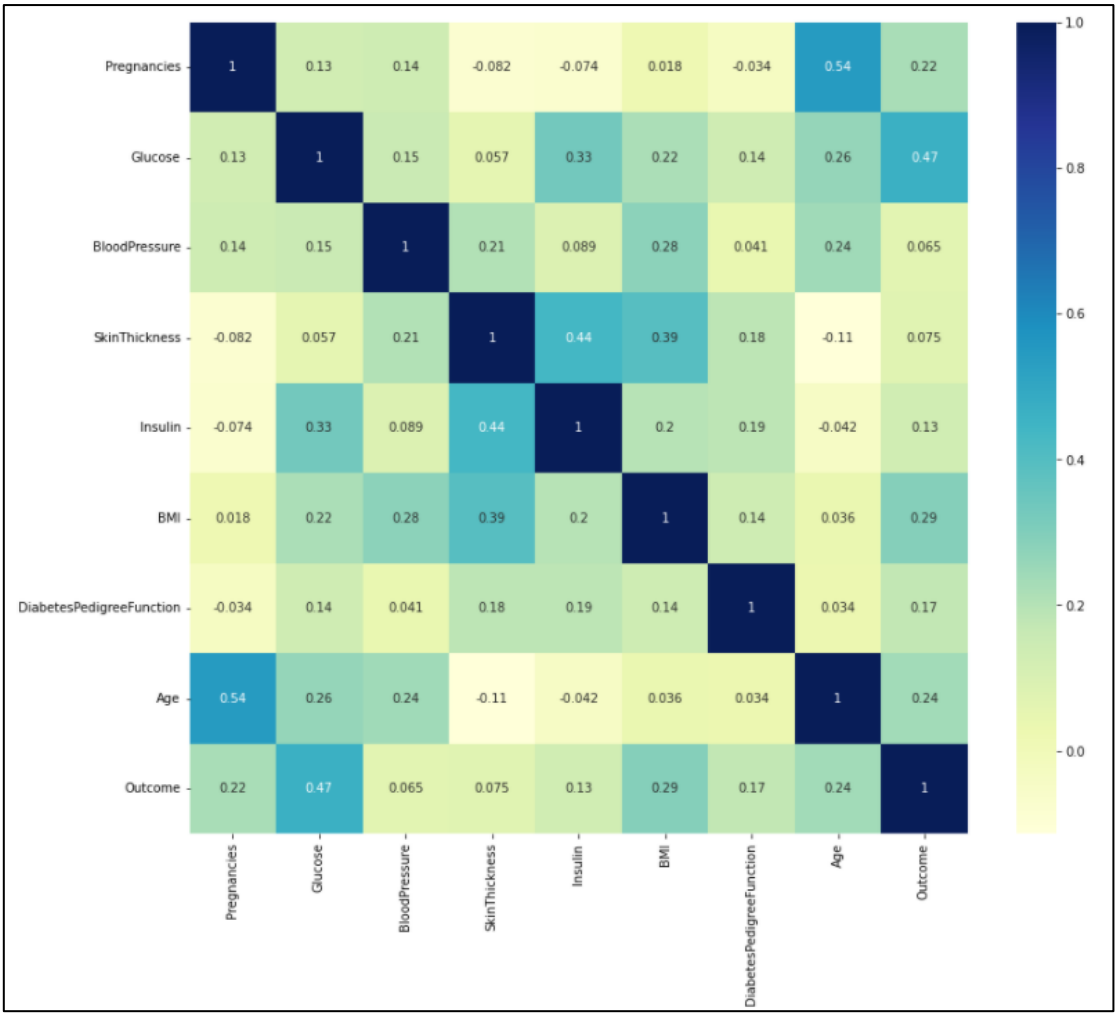
| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 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 |

```
df.describe()
```

| | 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 |


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
0
1

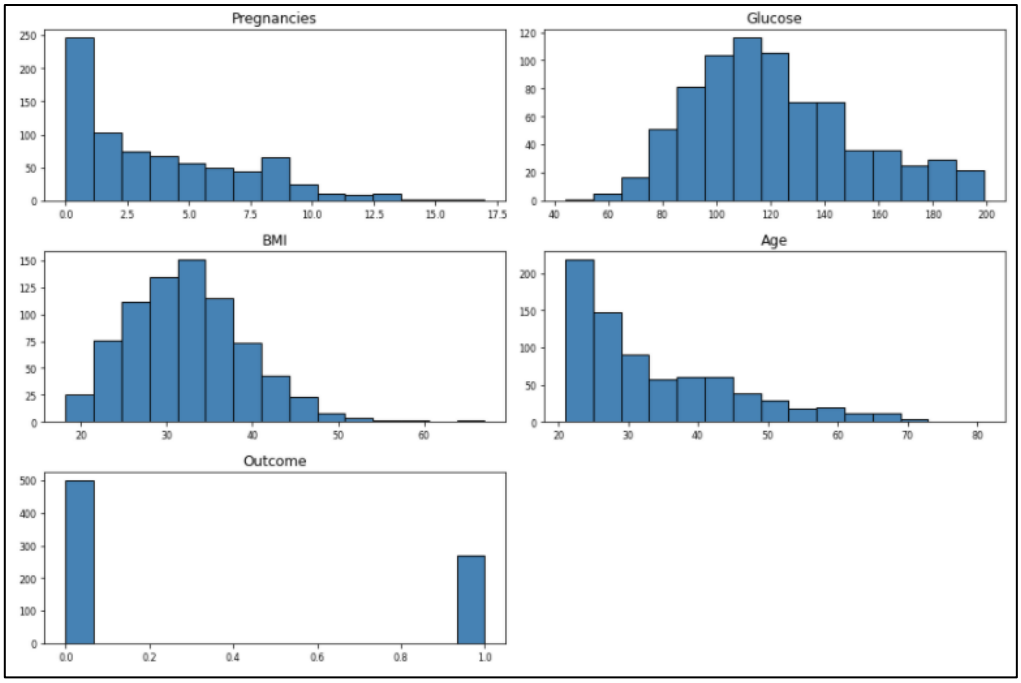


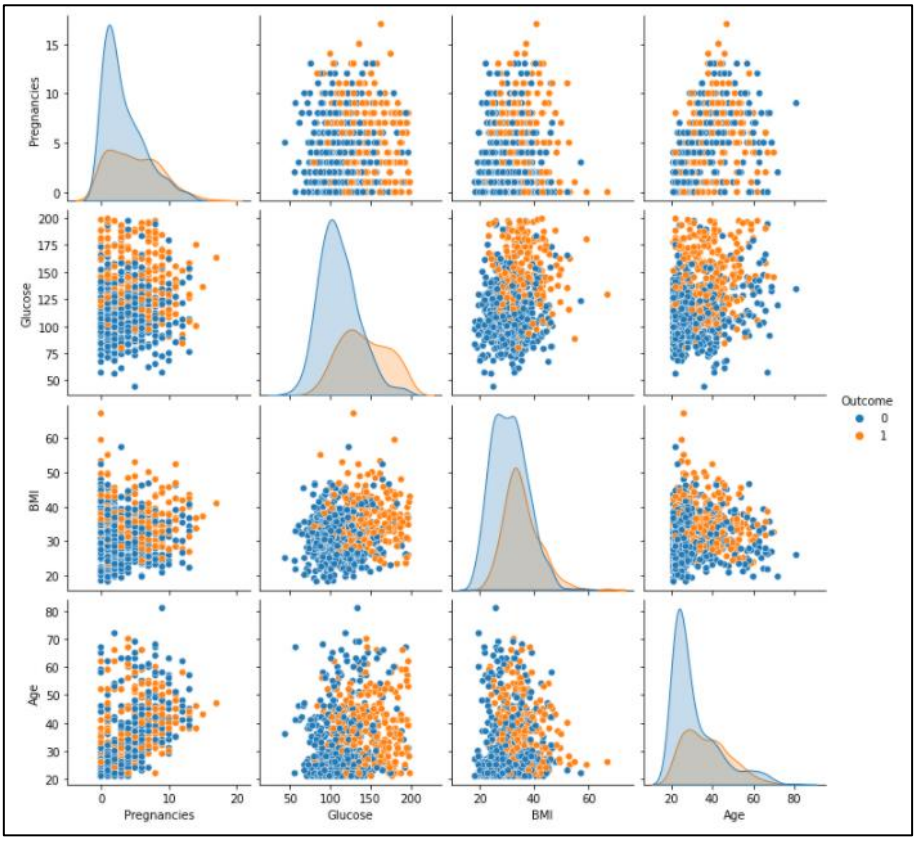

- Now after seeing the following analysis, the question arises that, can minimum value of following columns be 0 (is it practically possible). For the following columns zero value is invalid:
  1. Glucose
  2. BloodPressure
  3. SkinThickness
  4. Insulin
  5. BMI

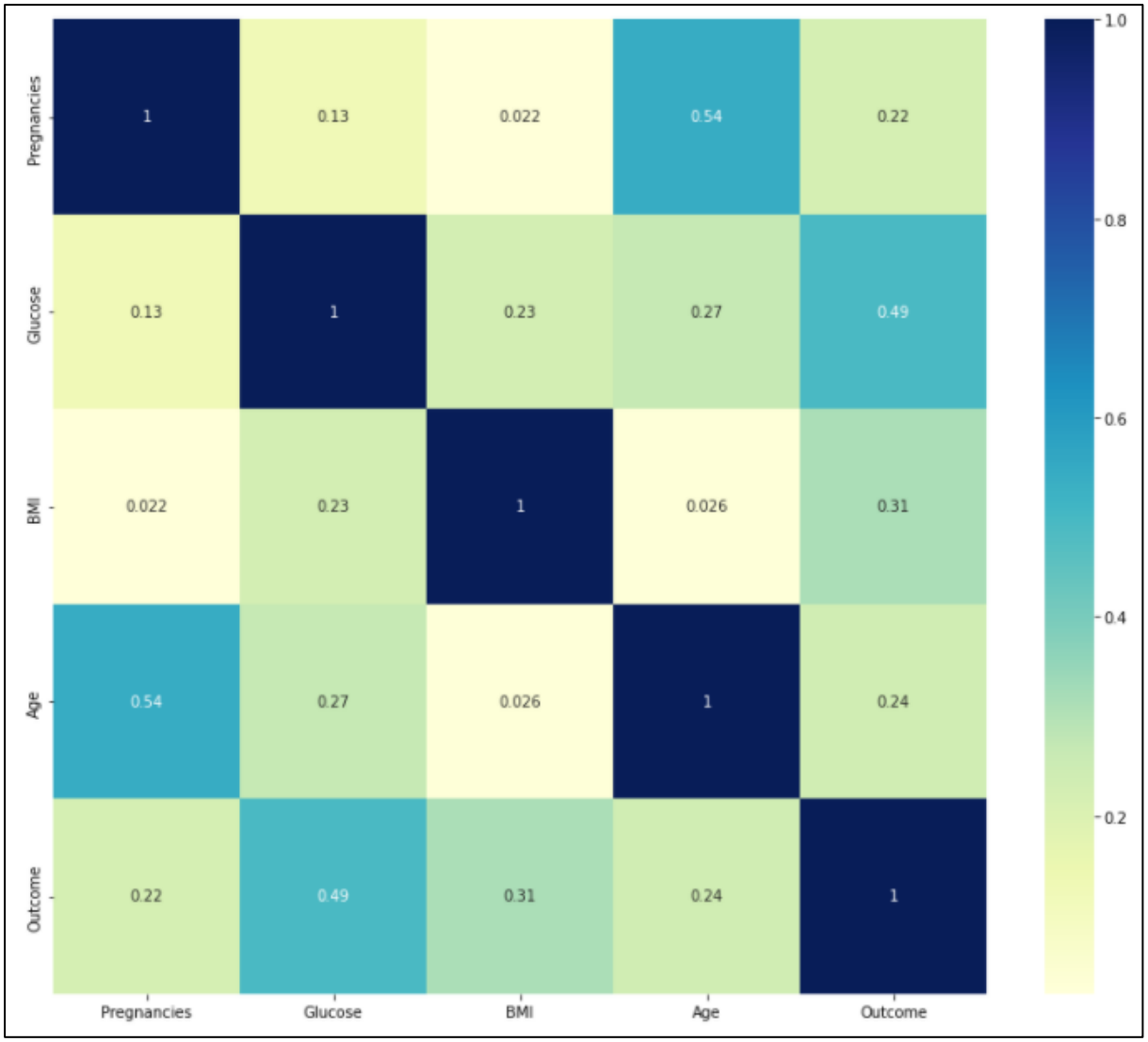
  And for others it is possible:
  1. Pregnancies
  2. Outcome
- As per the analysis of heat map we can see that, some of the features have very less correlation with the goal (outcome), therefore we have decided to take the columns, only if its correlation value is greater than or equal to 0.20.
- Hence Only those features are considered further whose correlation value is greater than 0.20 and have minimal or no invalid value.
- Therefore, we have shortlisted that only [Pregnancies, Glucose, BMI, Age, Outcome] these columns are the part of df_copy dataset, it is copy of the original dataset.
- And it is better to replace zero values of above columns (invalid) with mean or median based on their individual distribution.
- To decide whether data should be replaced with mean or median we considered skewness of the column values for those features.

- As we can see in above histogram that Glucose columns is not skewed, therefore we are replacing it with mean of the column, whereas BMI is right skewed, therefore it replaced with median of the column.
- Basically, now we have 2 datasets, one is original (df given in the question), and the other is modified dataset (df_copy).
- Now we perform all the operations on both the dataset, and compare them at the end.

Analysis of original dataset is shown above, whereas analysis of the modified dataset is shown below.






**Q1. Implement K-means clustering algorithm for a user given K.**

K-means clustering algorithm is an unsupervised learning algorithm. Basically, it works in iterative manner, it divides the unlabelled dataset into k different clusters, such that each element belongs to a group having similar properties. Main aim is to minimize the sum of the distance from the centroid of that cluster.

```
def k_mean(dataset,k,seed=1,given_centroids=[]):
    centroids=given_centroids
    if len(centroids)==0:
        centroids=initial_centroid(dataset,k,state=seed)

    while True:
        clusters=assign_cluster(dataset,centroids)
        new_centroids=update_centroids(clusters,dataset,k)
        if (centroids==new_centroids).all():
            return clusters,centroids
        centroids=new_centroids
```

**Q2. Provide clustering performance (With ground Truth and without ground Truth)**

For Original Dataset:

**Clustering Performance**

```
pos=0
neg=0
labels,centroids=k_mean(np_df,2,seed=42)
for i in range(np_df.shape[0]):
    if labels[i]==np_true[i]:
        pos=pos+1

print("Performance Measures for K=2")
print("Accuracy using Ground Truth: ",round(pos/np_df.shape[0],3))
print("Performance measure using ARI: ",adjusted_rand_score(labels,np_true))
print("Performance measure using Silhouette_Score: ",silhouette_score(np_df, labels, metric='euclidean'))
```

```
Performance Measures for K=2
Accuracy using Ground Truth:  0.66
Performance measure using ARI:  0.07438695547529087
Performance measure using Silhouette_Score:  0.5687788342658853
```

For Modified Dataset:

**Clustering Performance**

```
pos=0
neg=0
labels1,centroids=k_mean(np_df1,2,seed=22)
for i in range(np_df1.shape[0]):
    if labels1[i]==np_true1[i]:
        pos=pos+1
print("Performance Measures for K=2")
print("Accuracy using Ground Truth: ",round(pos/np_df1.shape[0],3))
print("Performance measure using ARI: ",adjusted_rand_score(labels1,np_true1))
print("Performance measure using Silhouette_Score: ",silhouette_score(np_df1, labels1, metric='euclidean'))
```

```
Performance Measures for K=2
Accuracy using Ground Truth:  0.737
Performance measure using ARI:  0.21456938548978738
Performance measure using Silhouette_Score:  0.4899305130082588
```

We have used ARI (Adjusted Rand Index) to check the clustering performance:

**Adjusted Rand Index (ARI):**
It is a similarity measure, used to measure the agreement between the true labels, and the labels predicted by the clustering algorithm. It ranges between 0 and 1.
Higher the value, higher is the correctness.

**Silhouette Score:**
We have also used internal indices called Silhouette Score to measure the performance of clustering. Silhouette score doesn't use labels, it measures how close each point in one cluster is to point in the neighbouring cluster. This measure has range of [-1,1].

**Q3. Provide the most suitable K for your data by varying K using some valid internal indices, (like silhouette index, Calin ski Harabasz, Wang's method of cross-validation.). Use graphical interpretation (K vs. metric) to justify your observation.**

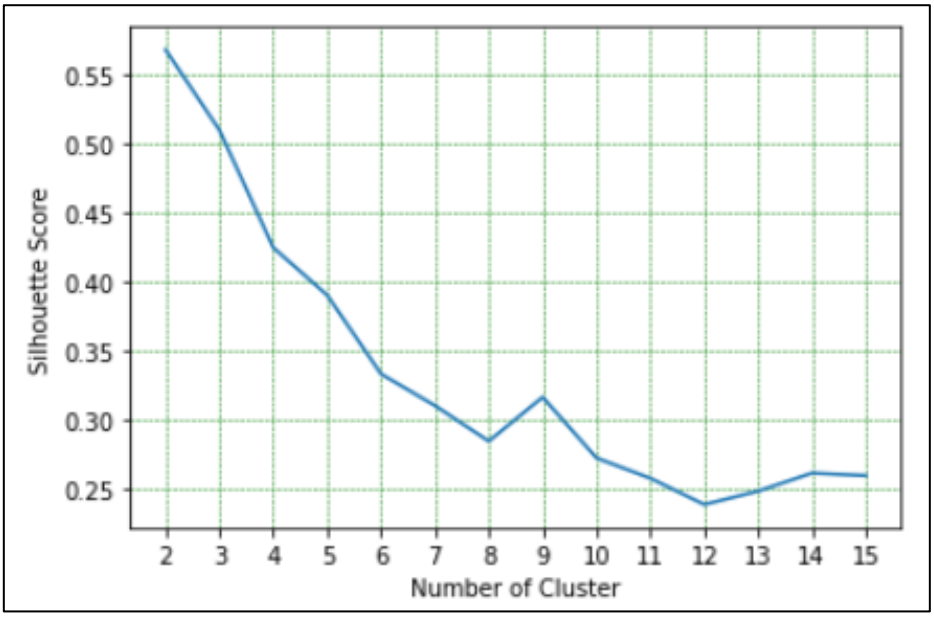
Most suitable value of K is **2**

**Silhouette Score or Coefficient:** It is used to calculate goodness of a clustering technique, or distinguishability of the clusters. It ranges from -1 to 1.
1: Means clusters are at good significant distance from each other.
0: Means distance between the cluster is not significant
-1: Clusters are assigned in wrong way

For Original Dataset:

```
values_se=[]
values_score=[]
x=[]
for i in range(2,16):
    labels,centroids=k_mean(np_df,i,seed=42)
    ari1=adjusted_rand_score(labels,np_true)
    score = silhouette_score(np_df, labels, metric='euclidean')
    print("No of cluster: ",i," ARI:",round(ari1,4)," Silhouette_Score:",round(score,4))
    x.append(i)
    values_score.append(score)
```

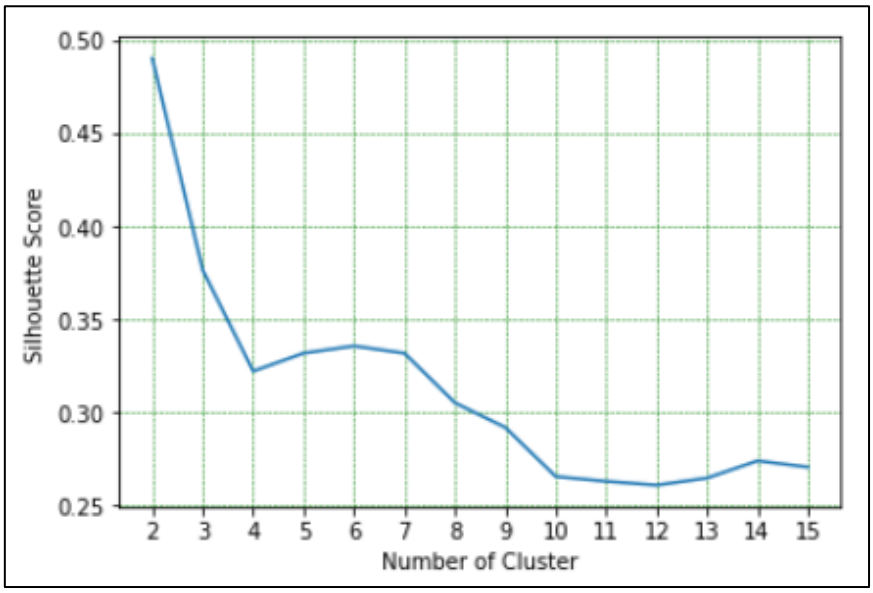
```
No of cluster:  2  ARI: 0.0744  Silhouette_Score: 0.5688
No of cluster:  3  ARI: 0.0452  Silhouette_Score: 0.5104
No of cluster:  4  ARI: 0.0277  Silhouette_Score: 0.4251
No of cluster:  5  ARI: 0.0305  Silhouette_Score: 0.3907
No of cluster:  6  ARI: 0.0686  Silhouette_Score: 0.3335
No of cluster:  7  ARI: 0.0576  Silhouette_Score: 0.3105
No of cluster:  8  ARI: 0.0549  Silhouette_Score: 0.2847
No of cluster:  9  ARI: 0.0537  Silhouette_Score: 0.3167
No of cluster:  10  ARI: 0.0437  Silhouette_Score: 0.2725
No of cluster:  11  ARI: 0.0378  Silhouette_Score: 0.2578
No of cluster:  12  ARI: 0.0328  Silhouette_Score: 0.2388
No of cluster:  13  ARI: 0.0273  Silhouette_Score: 0.2485
No of cluster:  14  ARI: 0.031  Silhouette_Score: 0.2616
No of cluster:  15  ARI: 0.0319  Silhouette_Score: 0.2598
```



For Modified Dataset:

```
values_se1=[]
values_score1=[]
x1=[]
for i in range(2,16):
    labels,centroids=k_mean(np_df1,i,seed=42)

    ari1=adjusted_rand_score(labels,np_true1)
    score = silhouette_score(np_df1, labels, metric='euclidean')

    print("No of cluster:",i," ARI:",round(ari1,2)," Silhouette_Score: ",round(score,2))
    x1.append(i)

    values_score1.append(score)
```

```
No of cluster: 2  ARI: 0.21  Silhouette_Score:  0.49
No of cluster: 3  ARI: 0.16  Silhouette_Score:  0.38
No of cluster: 4  ARI: 0.12  Silhouette_Score:  0.32
No of cluster: 5  ARI: 0.12  Silhouette_Score:  0.33
No of cluster: 6  ARI: 0.12  Silhouette_Score:  0.34
No of cluster: 7  ARI: 0.11  Silhouette_Score:  0.33
No of cluster: 8  ARI: 0.08  Silhouette_Score:  0.31
No of cluster: 9  ARI: 0.07  Silhouette_Score:  0.29
No of cluster: 10  ARI: 0.06  Silhouette_Score:  0.27
No of cluster: 11  ARI: 0.06  Silhouette_Score:  0.26
No of cluster: 12  ARI: 0.05  Silhouette_Score:  0.26
No of cluster: 13  ARI: 0.05  Silhouette_Score:  0.26
No of cluster: 14  ARI: 0.04  Silhouette_Score:  0.27
No of cluster: 15  ARI: 0.04  Silhouette_Score:  0.27
```




**Q4 Is there any change in clustering outcome if you initialize K cluster centroids with random points in different execution? If you observe any change, use any heuristic to initialize K centroids such that observed variation could be minimized? Use the heuristic to choose initial K centroids and perform Test-A.**

Yes, there will be different clustering outcome for different execution, if we initialize centroids with random points. It is one of the major drawbacks of the K-means clustering algorithm, because of that it can generate 2 centroid that are part of same cluster or an outlier as a centroid, so that no point will be near it, and to overcome this, we are using K-means++ as heuristic to initialize K centroids. (K-means++ is K-means algorithm with an initialization algorithm).

For Original Dataset:

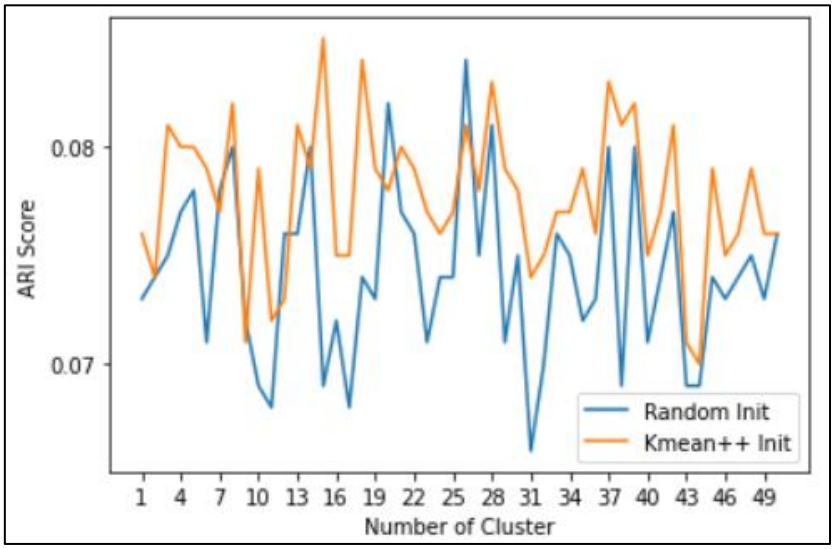
Test A with Random Initialization:

```
no_of_clusters=2
ari_array1=[]
for i in range(1,51):
    ids=random_k(np_df.shape[0],no_of_clusters,r_state=i)

    centroids=np.array([tuple(np_df[id]) for id in ids])
    remaining_df=np.delete(np_df,ids,axis=0)
    remaining_truth=np.delete(np_true,ids,axis=0)
    avg=0
    for j in range(1,51):
        train_x,test_x,train_y,test_y=train_test_split(remaining_df,remaining_truth,seed=j)
        np.append(train_x,np.array([np_df[id] for id in ids]),axis=0)
        np.append(train_y,np.array([np_true[id] for id in ids]),axis=0)
        labels,centroids=k_mean(train_x,no_of_clusters,given_centroids=centroids)
        clusters=assign_cluster(test_x,centroids)
        avg=avg+adjusted_rand_score(clusters,test_y)

    ari_array1.append(round(avg/50,3))
    print("Avg value for iteration ",i,":",round(avg/50,3))
```

Test A with Heuristic for Initialization (K-Means++):

```
no_of_clusters=2
ari_array2=[]

for i in range(1,51):

    centroids=k_mean_plus(np_df,no_of_clusters,r_state=i)

    ids=[]
    for j in range(2):

        x=np.where(np.all(np_df==centroids[j],axis=1))
        ids.append(x[0][0])
    remaining_df=np.delete(np_df,ids,axis=0)
    remaining_truth=np.delete(np_true,ids,axis=0)
    avg=0
    for j in range(1,51):
        train_x,test_x,train_y,test_y=train_test_split(remaining_df,remaining_truth,seed=j)
        np.append(train_x,np.array([np_df[id] for id in ids]),axis=0)
        np.append(train_y,np.array([np_true[id] for id in ids]),axis=0)
        labels,centroids=k_mean(train_x,no_of_clusters,seed=42,given_centroids=centroids)
        clusters=assign_cluster(test_x,centroids)
        avg=avg+adjusted_rand_score(clusters,test_y)
    avgg=round(avg/50,3)
    ari_array2.append(avgg)
    print("Avg value for iteration ",i,":",round(avg/50,3))
```

Graph Comparing ARI values for Random and K-Means++ Initialization:



For Modified Dataset:

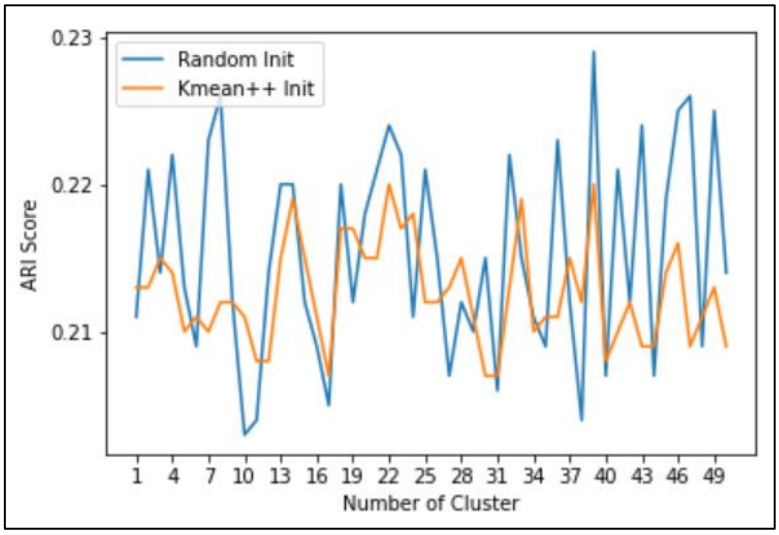
Test A with Random Initialization:

```
no_of_clusters=2
ari_array3=[]
for i in range(1,51):
    ids=random_k(np_df1.shape[0],no_of_clusters,r_state=i)

    centroids=np.array([tuple(np_df1[id]) for id in ids])
    remaining_df=np.delete(np_df1,ids,axis=0)
    remaining_truth=np.delete(np_true1,ids,axis=0)
    avg=0
    for j in range(1,51):
        train_x,test_x,train_y,test_y=train_test_split(remaining_df,remaining_truth,seed=j)
        np.append(train_x,np.array([np_df1[id] for id in ids]),axis=0)
        np.append(train_y,np.array([np_true1[id] for id in ids]),axis=0)
        labels,centroids=k_mean(train_x,no_of_clusters,given_centroids=centroids)
        clusters=assign_cluster(test_x,centroids)
        avg=avg+adjusted_rand_score(clusters,test_y)

    ari_array3.append(round(avg/50,3))
    print("Avg value for iteration ",i,":",round(avg/50,3))
```

Test A with Heuristic for Initialization (K-Means++):

```
no_of_clusters=2
ari_array4=[]

for i in range(1,51):
    centroids=k_mean_plus(np_df1,no_of_clusters,r_state=i)
    ids=[]
    for j in range(2):
        x=np.where(np.all(np_df1==centroids[j],axis=1))
        ids.append(x[0][0])

    remaining_df=np.delete(np_df1,ids,axis=0)
    remaining_truth=np.delete(np_true1,ids,axis=0)
    avg=0
    for j in range(1,51):
        train_x,test_x,train_y,test_y=train_test_split(remaining_df,remaining_truth,seed=j)
        np.append(train_x,np.array([np_df1[id] for id in ids]),axis=0)
        np.append(train_y,np.array([np_true1[id] for id in ids]),axis=0)
        labels,centroids=k_mean(train_x,no_of_clusters,seed=42,given_centroids=centroids)
        clusters=assign_cluster(test_x,centroids)
        avg=avg+adjusted_rand_score(clusters,test_y)

    avgg=round(avg/50,3)
    ari_array4.append(avgg)
    print("Avg value for iteration ",i,":",round(avg/50,3))
```

Graph Comparing ARI values for Random and K-Means++ Initialization:



**K-Means++ Algorithm:**

```
#kmean++ initialization: heuristic function: centroids should be as far as possible from each other
def k_mean_plus(dataset,k,r_state=21):
    rn.seed(r_state)

    c0 = rn.randint(0,dataset.shape[0]-1)
    centroids=[]
    centroids.append(tuple(dataset[c0]))
    dataset=np.delete(dataset,c0,axis=0)

    for i in range(0,k-1):
        distances=np.min(squared_distance_matrix(dataset,np.array(centroids)),axis=1)
        normalized_distances = distances / distances.sum()
        normalized_distance=np.sort(normalized_distances)
        normalized_distance=normalized_distances
        rn.seed(i)
        dice_roll=rn.random()
        min_over_roll = normalized_distance[normalized_distance.cumsum() >= dice_roll].min()
        idx = np.where(normalized_distances==min_over_roll)
        centroids.append(dataset[idx[0][0]])
        dataset=np.delete(dataset,idx[0][0],axis=0)

    return np.array(centroids)
```

As we can see in the graph, that after applying the K-means++ Algorithm as heuristic to initialize the K centroids, the variation is reduced to some extent, as it is giving ARI in particular range only, for both original as well as modified data.

## Conclusion:

- Accuracy using Ground Truth for the Original Data is **0.66**, whereas for the modified data is **0.737**. Also, we are getting better ARI score for modified data.
- Most Suitable value of K is **2** (For both the dataset).
- Our analysis about the dataset is correct, and as per all the above observations and the modified dataset is providing better results, as compared to original dataset in most of the cases.
- For Original Dataset:
    - Actual Cluster based on true labels:

[[ 3.298, 109.98, 68.184, 19.664, 68.792, 30.3042, 0.429734, 31.19, 0]
[4.86567164, 141.25746269, 70.82462687, 22.1641791, 100.3358209, 35.14253731, 0.5505, 37.06716418, 1]]

    - Predicted Cluster using K-means Clustering:

[[ 3.88391376, 115.26699834, 68.09784411, 17.6185738, 32.21227197, 31.17363184, 0.43757048, 33.11442786, 0.30182421]
[ 3.7030303, 141.46060606, 72.78787879, 31.2, 253.70909091, 34.98545455, 0.59724848, 33.7030303, 0.52121212]]

- For Modified Dataset:
    - Actual Cluster based on true labels:

[[ 3.298, 110.71012058, 30.8856, 31.19, 0]
[ 4.86567164, 142.16557286, 35.38358209, 37.06716418, 1]]

    - Predicted Cluster using K-means Clustering:

[[ 3.48850575, 104.51615673, 31.52241379, 30.87164751, 0.21455939]
[ 4.60162602, 158.12195122, 34.43455285, 38.26829268, 0.63414634]]