**Assignment 2**

**Haniel Sceva Manda**

**1.** Over here we first converted the .data files into .csv file and then replaced the all the ‘?’ to ‘NaN’ as there was a direct fill method to replace the ‘NaN’ in python. Here we performed normalization by finding the mean of each column and then replaced the ‘NaN’ with the mean values. Later on we performed Z-score normalization which uses the formula **(value-(mean of the column))/(standard-deviation).**

**2.** The basic ideology behind k-mean has us defining the k cluster before so that the total within-clusters variation is minimum. It is an iterative process. It will keep on running until the centroids of newly formed clusters do not change or the maximum number of iterations are reached. But in this method, we need to mention the number of clusters even before we execute it. To find the right K value we have couple of methods

* Elbow Method
* Silhouette Method

Over here we’ve used the elbow method to find the k value. At first, we calculate the Within-Cluster-Sum of Squared Errors (WSS) for different values of k. The Squared Error for each point is the square of the distance of the point from its representation. The WSS score is the sum of these Squared Errors for all the points. And then choose the k for which WSS becomes first starts to diminish. In the plot of WSS-versus-k, this is visible as an elbow.

**Code**

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

kmeans.fit(df)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss)

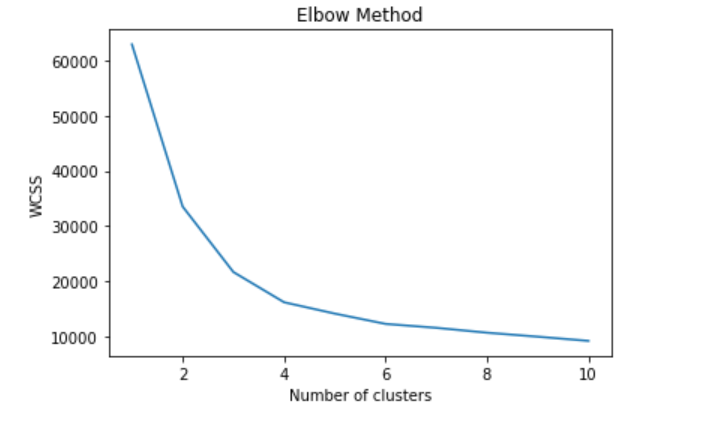
plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

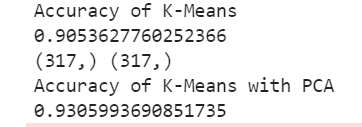
plt.show()

Here we run the number of clusters from 1 to 11 where and plot the elbow graph which ended up like



From the graph we deduced that the right number of clusters is 4 where the perfect elbow is formed.

**5.** PCAis used to reduce the dimensionality of dataset consisting of many variables correlated with each other, while retaining the variation present in the dataset, up to the maximum extent. By performing the K-mean on the dataset which went through PCA the accuracy has been increased from 90% to 93%.



6.

* For dimensionality reduction to be effective, there needs to be underlying low dimensional structure in the feature space. I.e the features should have some relationship with each other.
* If there is non-linearity or curvature in low dim structure than autoencoders can encode more information using less dimensions. So, they are a better dimensionality reduction technique in these scenarios.
* There are no guidelines to choose the size of the bottleneck layer in the autoencoder unlike PCA. With PCA, the top k components can be chosen to factor in x% of the variation. Often PCA can be used as a guide to choose k.
* The autoencoder tends to perform better when k is small when compared to PCA, meaning the same accuracy can be achieved with less components and hence a smaller data set. This is important when dealing with very large data sets.
* Autoencoders require more computation than PCA. Although, for very large data sets that can’t be stored in memory, PCA will not be able to be performed.

**Reference:** <https://www.r-bloggers.com/pca-vs-autoencoders-for-dimensionality-reduction/>