**ARYAMAN MISHRA**

**19BCE1027**

**Lab Exercise 6**

**Take any text corpora, apply necessary preprocessing and perform the k-means clustering on the corpora.**

**Proposed Algorithm/Pseudocode:**

**Kmeans** algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster’s centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

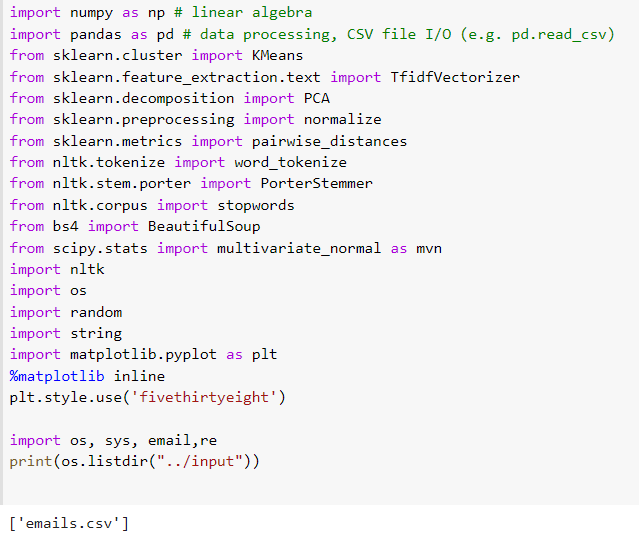
The way kmeans algorithm works is as follows:

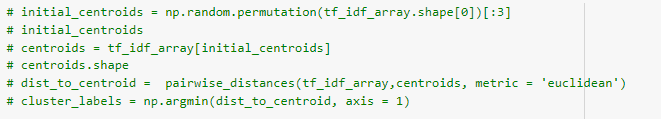
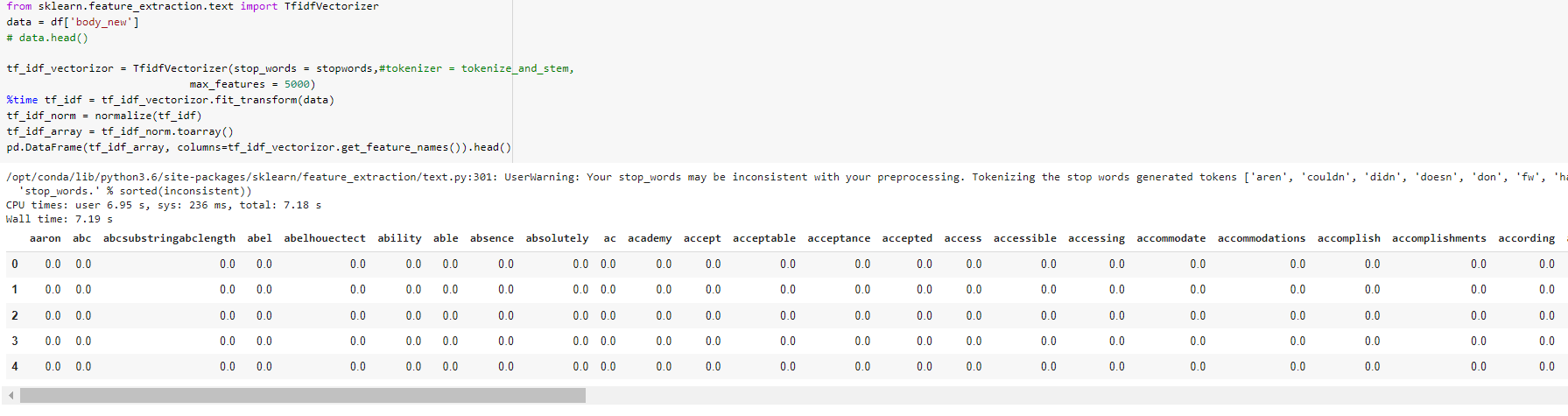
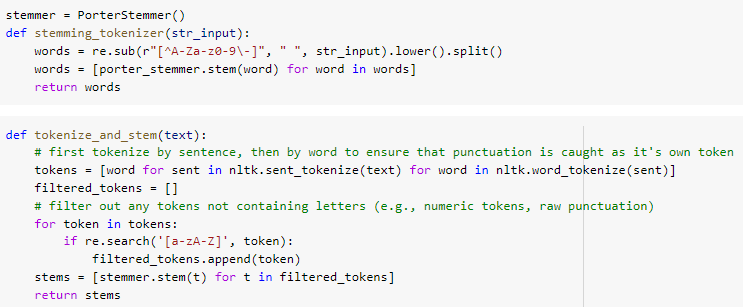
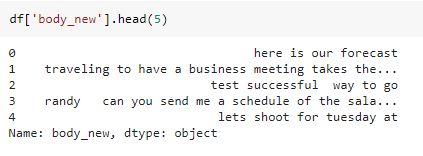
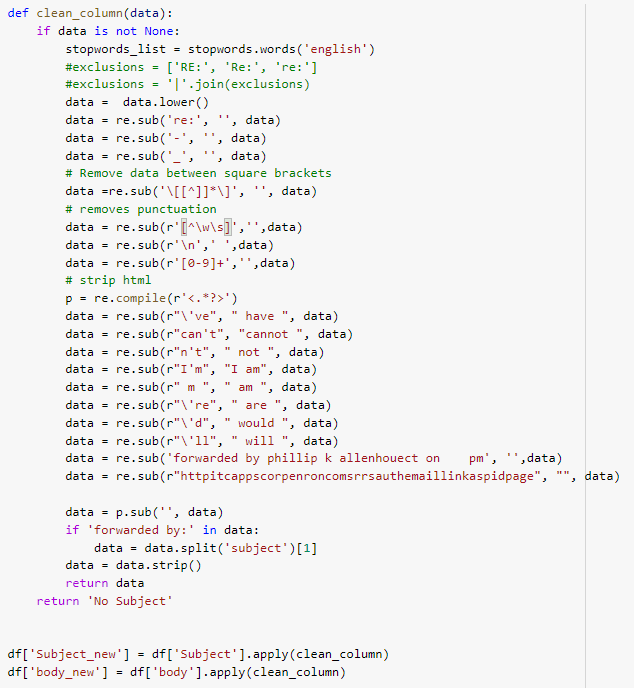
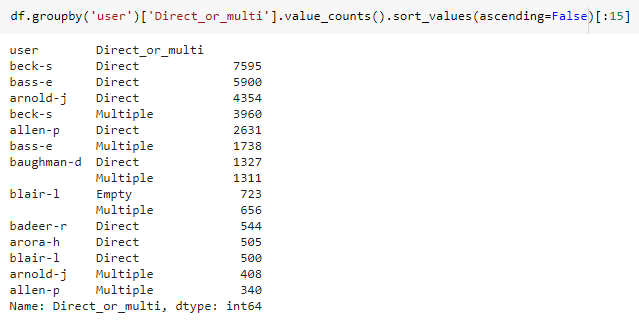
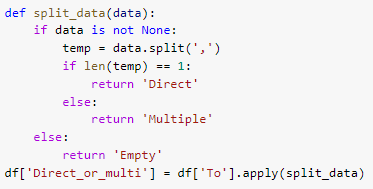
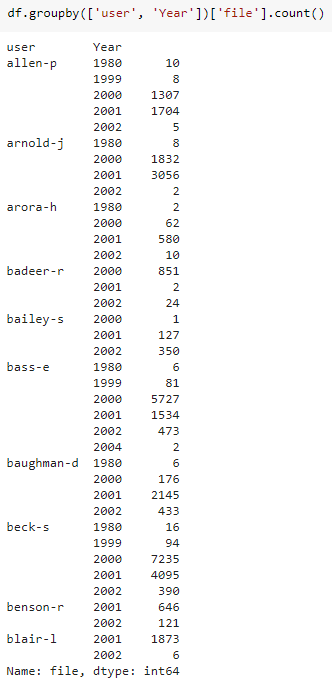
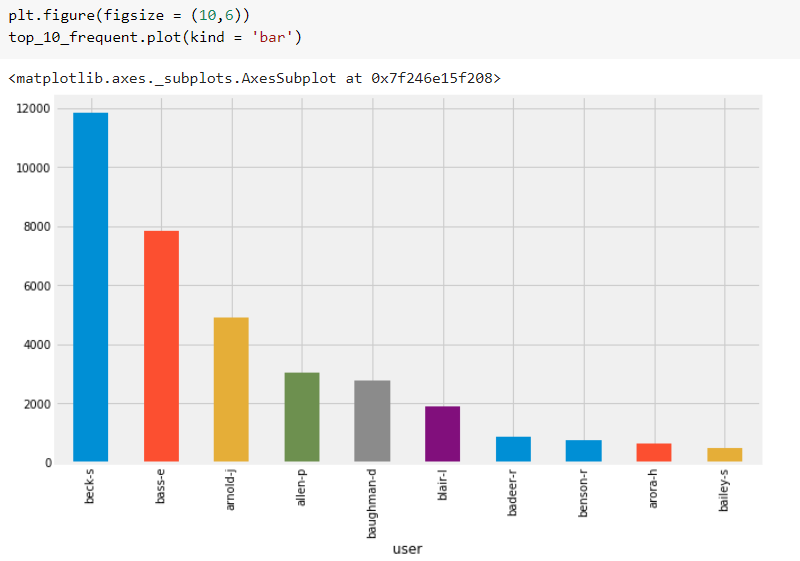
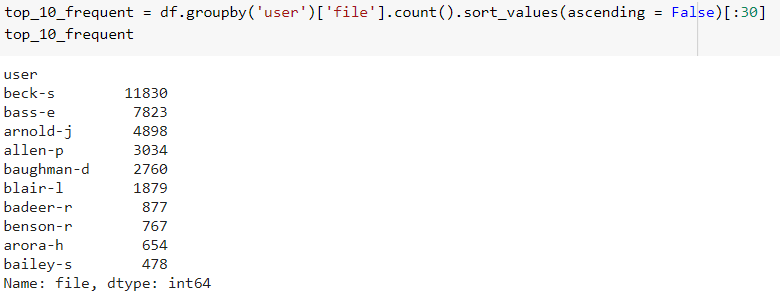
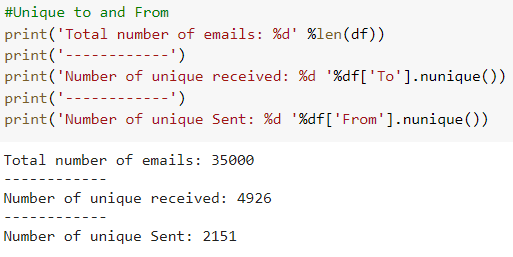
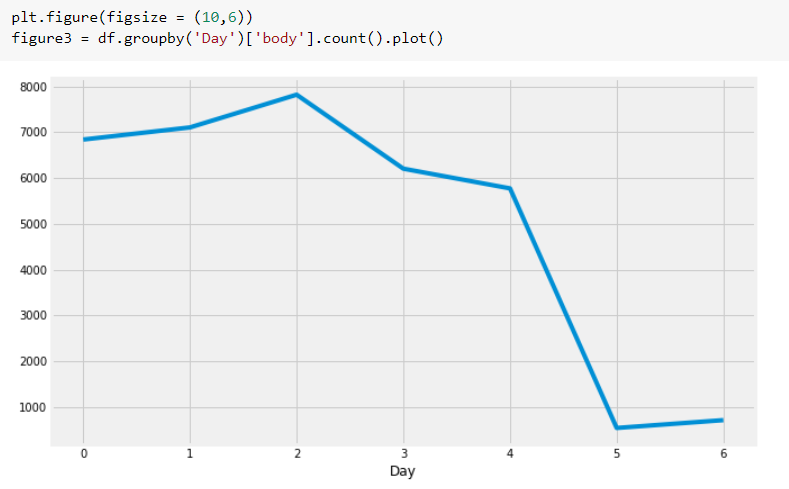
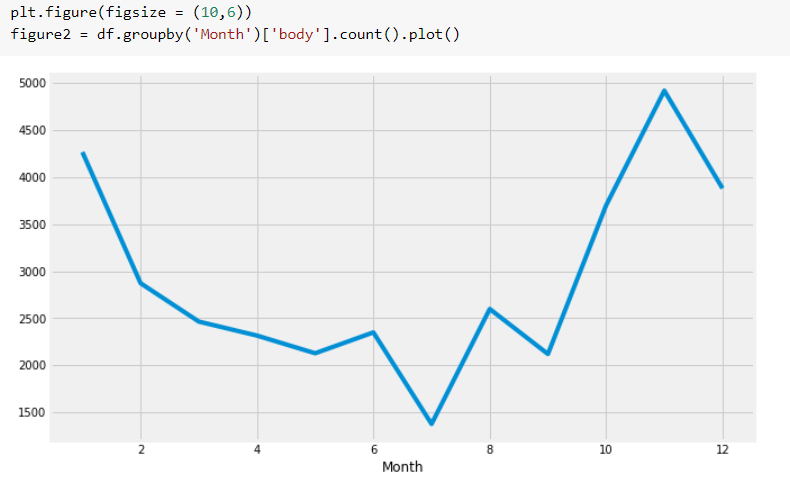
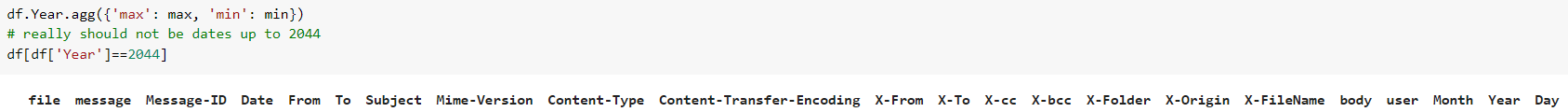
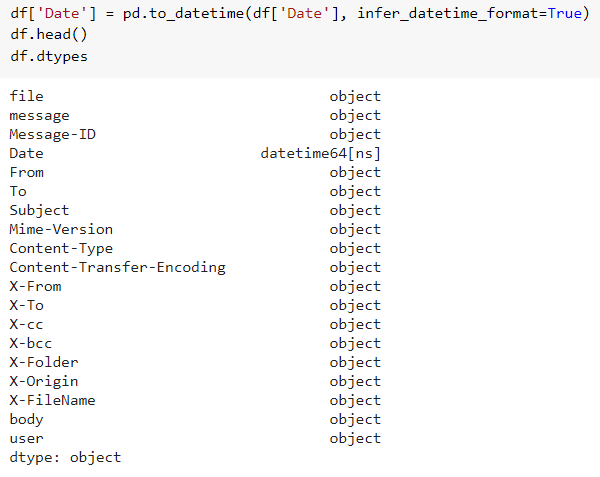
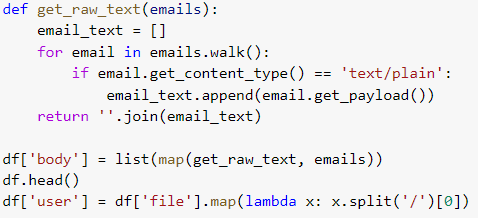
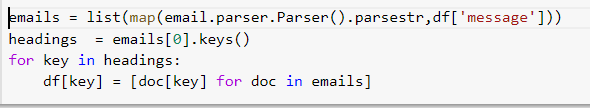
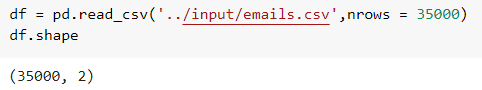
1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn’t changing.

* Compute the sum of the squared distance between data points and all centroids.
* Assign each data point to the closest cluster (centroid).
* Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

**Data Structure Proposed: 2D-Arrays, Dictionaries.**

**IMPLEMENTATION CODE AND RESULTS:**





class Kmeans:

    def \_\_init\_\_(self, k, seed = None, max\_iter = 200):

        self.k = k

        self.seed = seed

        if self.seed is not None:

            np.random.seed(self.seed)

        self.max\_iter = max\_iter

    def initialise\_centroids(self, data):

        initial\_centroids = np.random.permutation(data.shape[0])[:self.k]

        self.centroids = data[initial\_centroids]

        return self.centroids

    def assign\_clusters(self, data):

        if data.ndim == 1:

            data = data.reshape(-1, 1)

        dist\_to\_centroid =  pairwise\_distances(data, self.centroids, metric = 'euclidean')

        self.cluster\_labels = np.argmin(dist\_to\_centroid, axis = 1)

        return  self.cluster\_labels

    def update\_centroids(self, data):

        self.centroids = np.array([data[self.cluster\_labels == i].mean(axis = 0) for i in range(self.k)])

        return self.centroids

    def convergence\_calculation(self):

        pass

    def predict(self, data):

        return self.assign\_clusters(data)

    def fit\_kmeans(self, data):

        """

        This function contains the main loop to fit the algorithm

        Implements initialise centroids and update\_centroids

        according to max\_iter

        self.centroids = self.initialise\_centroids(data)

        # Main kmeans loop

        for iter in range(self.max\_iter):

            self.cluster\_labels = self.assign\_clusters(data)

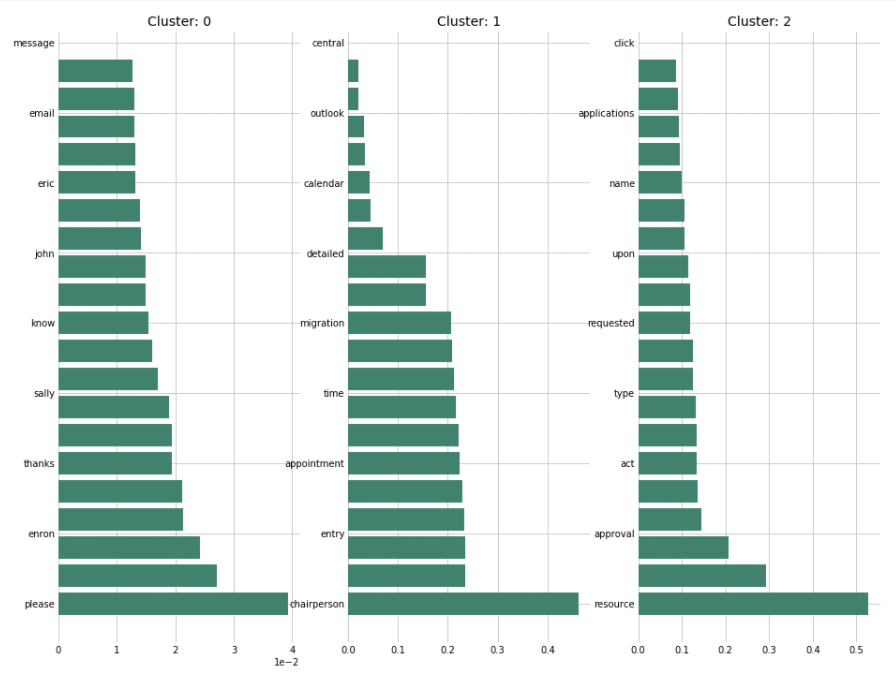
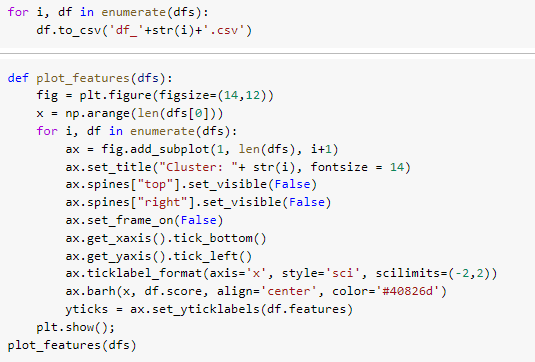
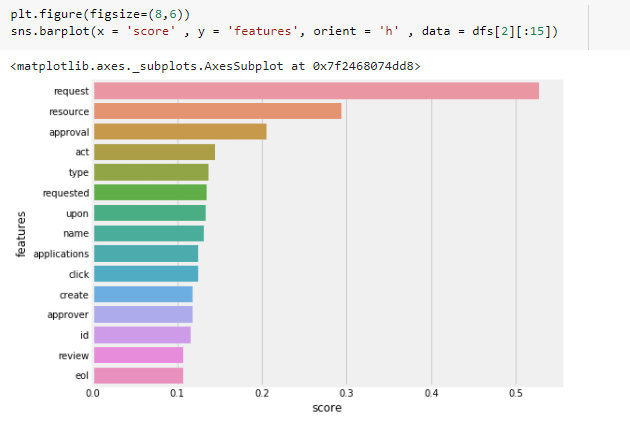
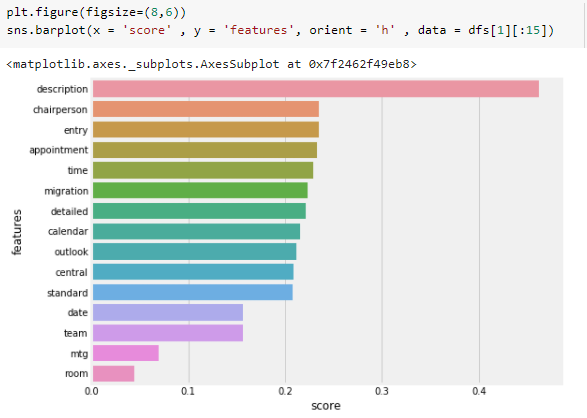
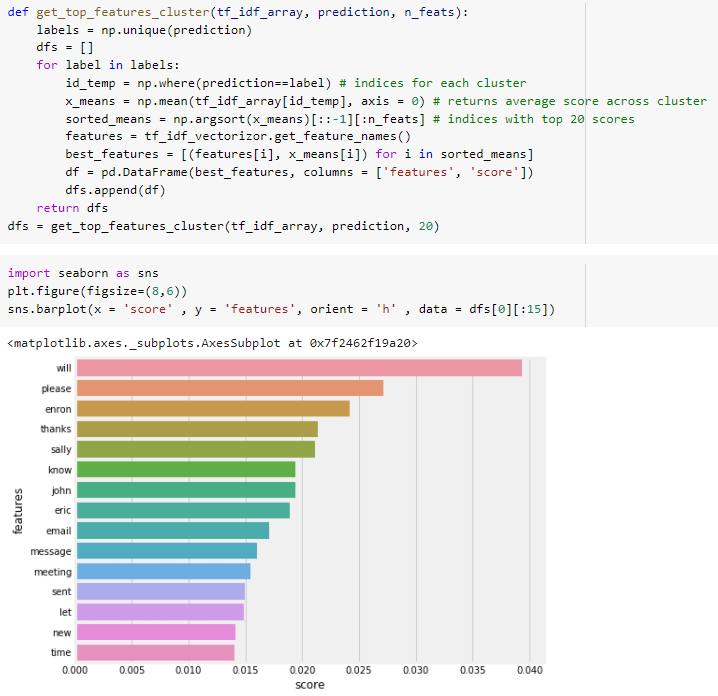
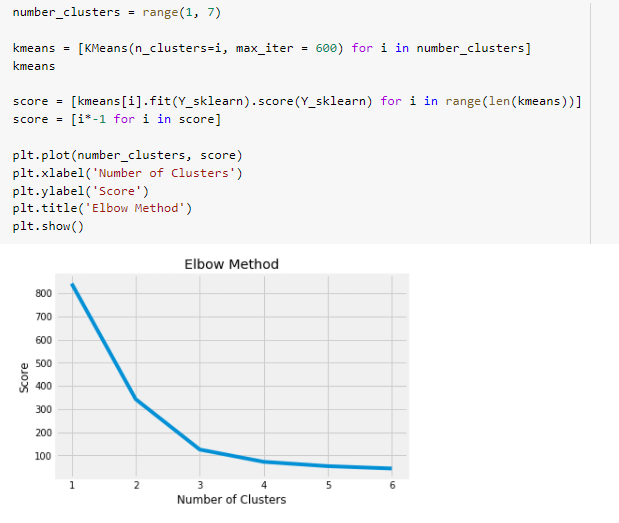
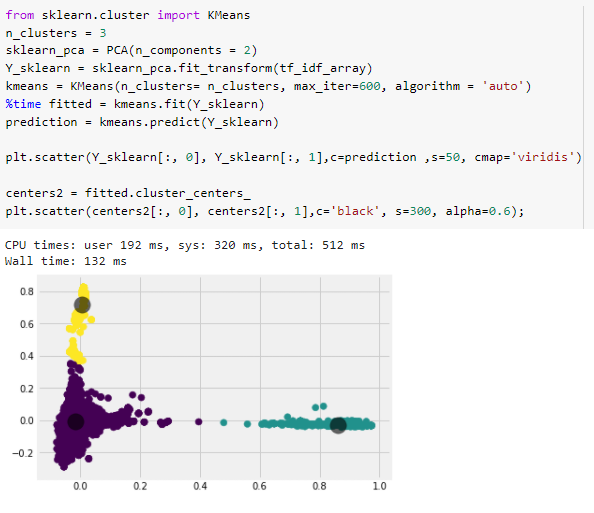
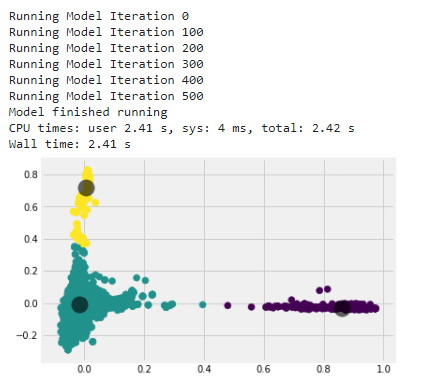
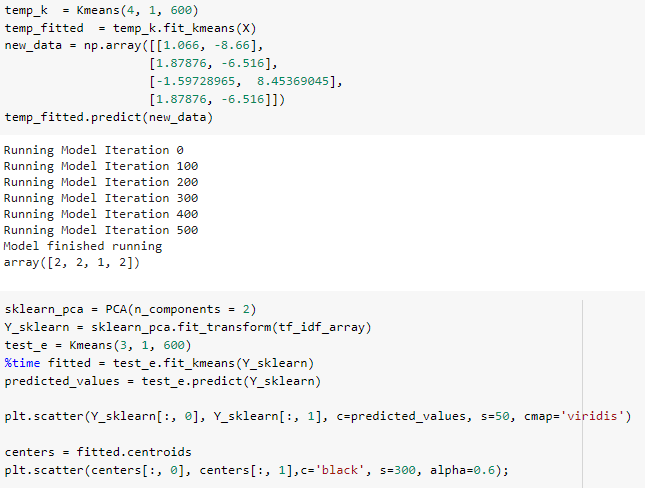
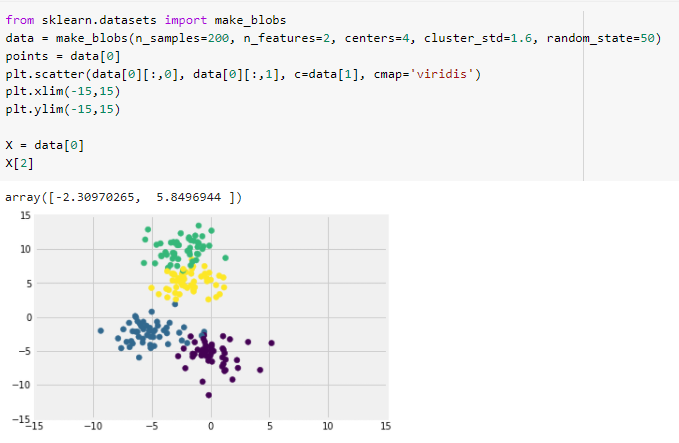
            self.centroids = self.update\_centroids(data)

            if iter % 100 == 0:

                print("Running Model Iteration %d " %iter)

        print("Model finished running")

        return self



**CONCLUSION:**

**K-Means Clustering for email txt corpora has been successfully implemented and executed.**