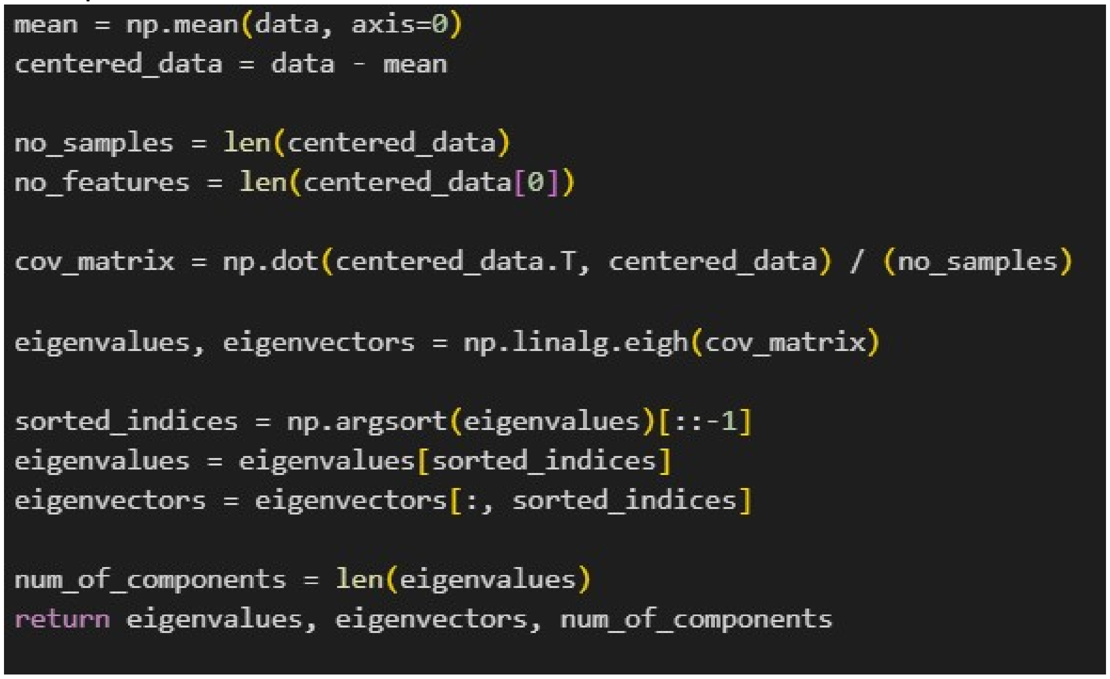
FML ASSIGNMENT – 3 REPORT

Q1. The MNIST dataset is loaded into our workspace using datasets library method.

(i) We need to run the PCA algorithm on this, and visualize the images of the principal components that we have obtained. Also, we need to report the variance explained by each of the principal components in the dataset.

- Initially, we acquired the pixel values from the bytes format, and then sampled 100 images(10 images for each class), and also we have shown the images of each class. Took the numpy form of the pixel dataset and then flattened them to implement PCA.



```
mean = np.mean(data, axis=0)
centered_data = data - mean

no_samples = len(centered_data)
no_features = len(centered_data[0])

cov_matrix = np.dot(centered_data.T, centered_data) / (no_samples)

eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

num_of_components = len(eigenvalues)
return eigenvalues, eigenvectors, num_of_components
```

We then implemented PCA by first centering the pixel values, and found the covariance matrix, and obtained the eigen vectors and eigen values.
Also, we found that 132 components contribute to the 95% variance in the dataset.
We visualized the top 132 components as well, and plotted the variance distribution of each principal components(i.e., eigen vectors) both individually and cumulatively.

(ii) We need to reconstruct the dataset using the different dimensional representations, and also to describe on how it looks. Also, we need to pick a dimension 'd' that can be used for a downstream task where we need to classify the digits correctly.
We reconstructed the image using the top 132 components(since for downstream task, top 132 components are the best as they capture 95% variance).
And d can be taken as 132 for downstream tasks, but we can also choose values less than 132, but reasonably make sure that we are able to see the images but trying multiple iteration of values).

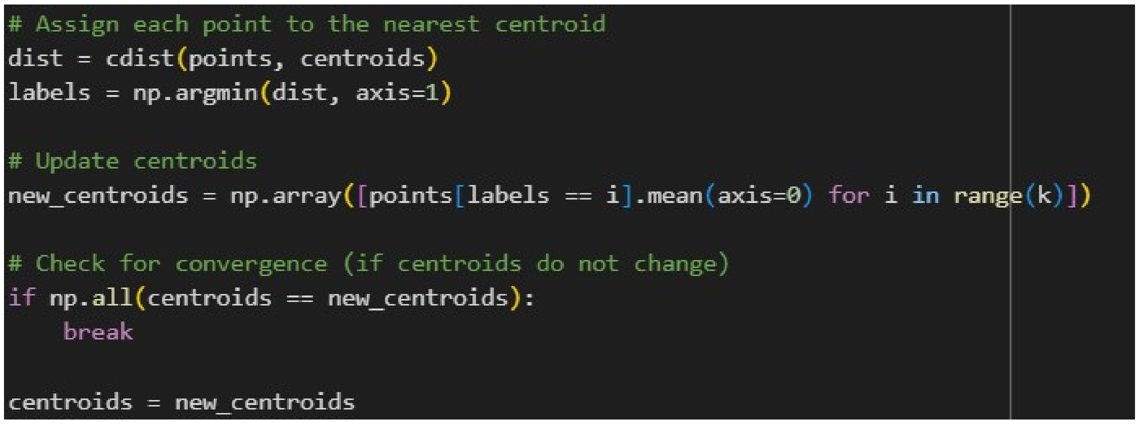
Q2. The given dataset is of two dimension, and has 1000 datapoints. I have converted the points to numpy format so as to implement K-Means.

Now, I implemented Lloyd's algorithm as follows:
Based on the initialization points as centroids, the Euclidean distance from each of the points to these centroids are calculated and the clusters to which these points belong to, are determined by choosing the centroid to which it is close to, when compared to all other centroids.

In this way, clusters are formed. So, now the centroids of each cluster is determined by calculating the mean of the datapoints present in that cluster.

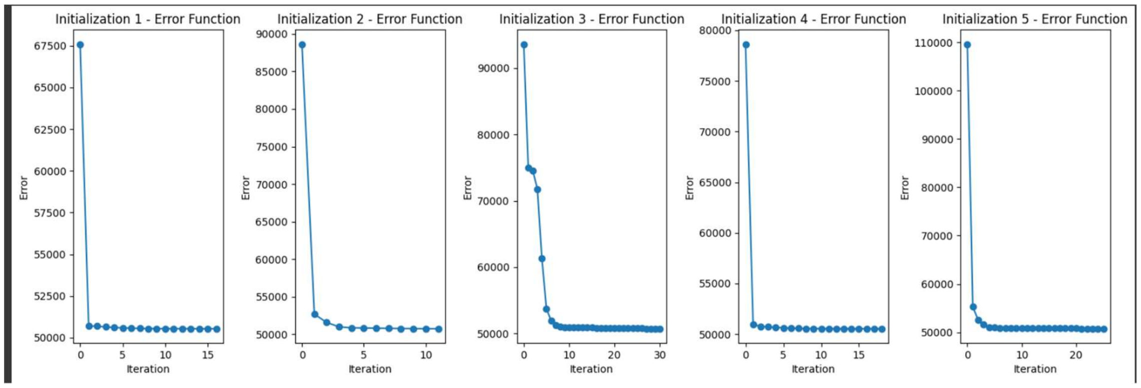
And again, we calculate the distances of each data point to each of these centroids, and assign them to the cluster to which they have the shortest distance. If a tie happens, the points remain in its old cluster.

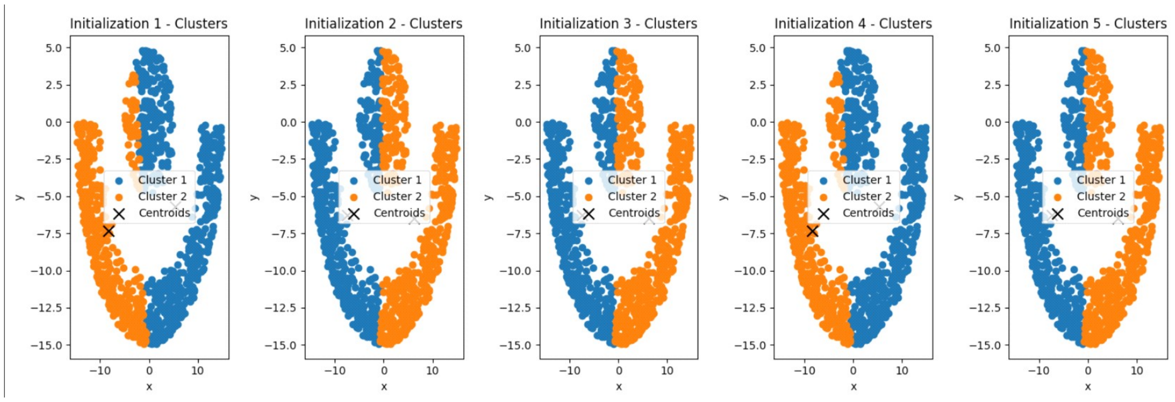
Like this way, I ran the Lloyd's algorithm for 100 iterations, one can modify the iterations as they wish until it converges. But, that might take indefinite time as well depending on the initialization. So, I ran the algorithm for finite number of iterations considering the quantity and dimension of the dataset.



```
# Assign each point to the nearest centroid
dist = cdist(points, centroids)
labels = np.argmin(dist, axis=1)

# Update centroids
new_centroids = np.array([points[labels == i].mean(axis=0) for i in range(k)])

# Check for convergence (if centroids do not change)
if np.all(centroids == new_centroids):
    break

centroids = new_centroids
```

(i) Coming to part (i), I took the five different initializations for two clusters using seed method(by taking 1,2,3,4 and 5 as seed values in order) and took two points at random accordingly and ran the Lloyd's algorithm and got the following result:



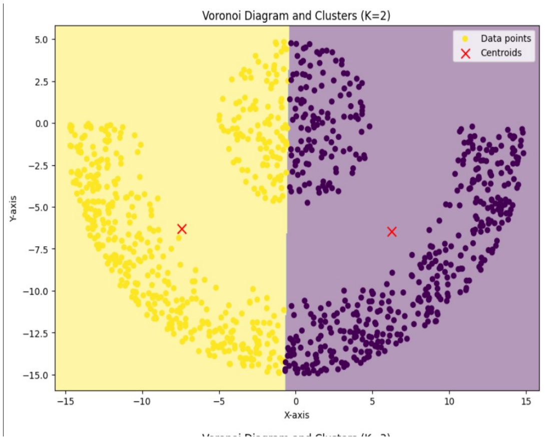


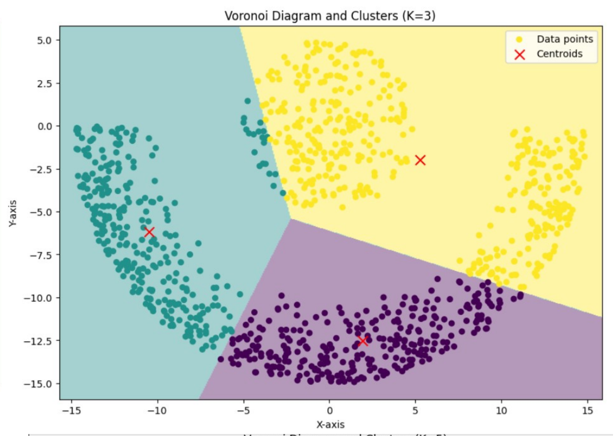
We can observe that for the five different initialization that we took, the clusters have almost converged to the same shape(same form). And there might be some difference in error graph in the initial iterations, but approach almost the same (near convergence in further iterations).

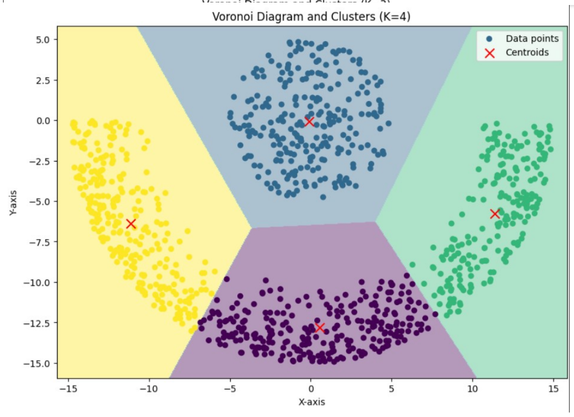
(ii) In part (ii), we are asked to take K values to be 2,3,4,5 one after the other, and run Lloyd's algorithm for the dataset taking these K-values and plot the voronoi regions associated to each cluster.
The process is done as follows:
In order to ease the process, I have written K-means function again but for the case of fixed initialization (anyhow previous code and current code do the same job, only difference is k-means in this case takes the prefixed initiliazation, while k-means in case (i) takes the initialization points in the function itself).

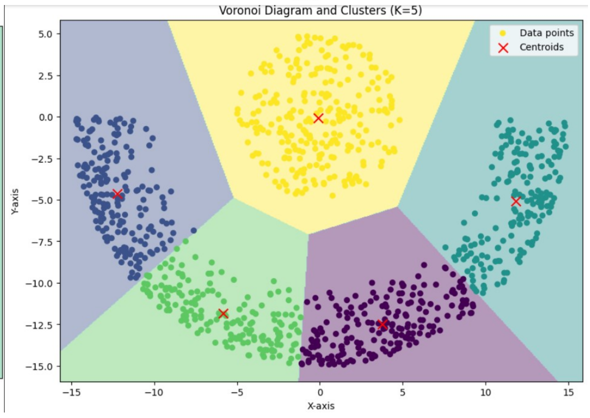
And Lloyd's algorithm is run for each of the K-values, and voroni regions have been plotted. Note that we have coded for the voronoi regions from scratch, taking the idea that perpendicular bisector for the line joining two points is the separating boundary for every pair of centroids in each iteration, and voronoi regions are the intersection of half spaces formed by each of the perpendicular bisectors.

Hence, we have got the voronoi regions as follows:









Hence, we have obtained the following voronoi regions for K values {2,3,4,5}.

(iii) We can observe that the shapes are in the two curve features(circles) appearing to be concentric, and can be captured in higher dimensions, to get them in the form of linear separable regions of two clusters, by using the method of Kernels.
Hence, Lloyd's algorithm is not a good way for this dataset, but it will be of use when it is used for a kernelized version of this dataset in higher dimensions.
Since, the answer is no. The other procedure is to use Kernel-Kmeans wherein you apply kernel for this dataset such that you obtain a linearly separable form (mostly of degree 2) and then implement Lloyd's algorithm to that dataset.