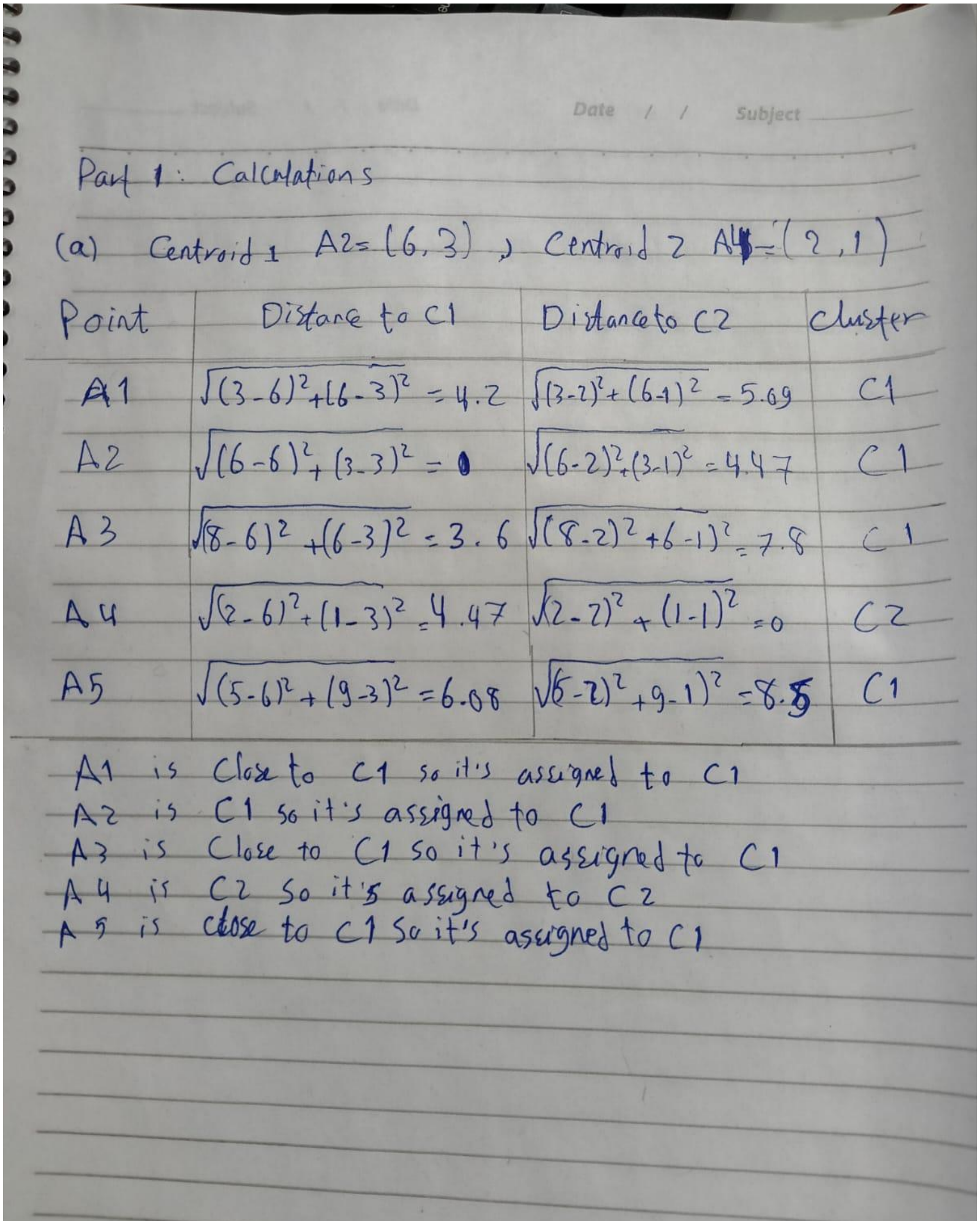
Part 1: Calculations:

1. Use the k-means algorithm and Euclidean distance to cluster the following 5 data points into 2 clusters: A1=(3,6), A2=(6,3), A3=(8,6), A4=(2,1), A5=(5,9). Suppose that the initial centroids (centers of each cluster) are A2 and A4. Using k-means, cluster the 5 points and show the followings for one iteration only:

(a) Show step-by-step the performed calculations to cluster the 5 points



Date / / Subject

Part 1: Calculations

(a) Centroid 1 A2 = (6, 3) , Centroid 2 A4 = (2, 1)

Point	Distance to C1	Distance to C2	Cluster
A1	$\sqrt{(3-6)^2+(6-3)^2} = 4.2$	$\sqrt{(3-2)^2+(6-1)^2} = 5.09$	C1
A2	$\sqrt{(6-6)^2+(3-3)^2} = 0$	$\sqrt{(6-2)^2+(3-1)^2} = 4.47$	C1
A3	$\sqrt{(8-6)^2+(6-3)^2} = 3.6$	$\sqrt{(8-2)^2+6-1)^2} = 7.8$	C1
A4	$\sqrt{(2-6)^2+(1-3)^2} = 4.47$	$\sqrt{(2-2)^2+(1-1)^2} = 0$	C2
A5	$\sqrt{(5-6)^2+(9-3)^2} = 6.08$	$\sqrt{(5-2)^2+9-1)^2} = 8.5$	C1

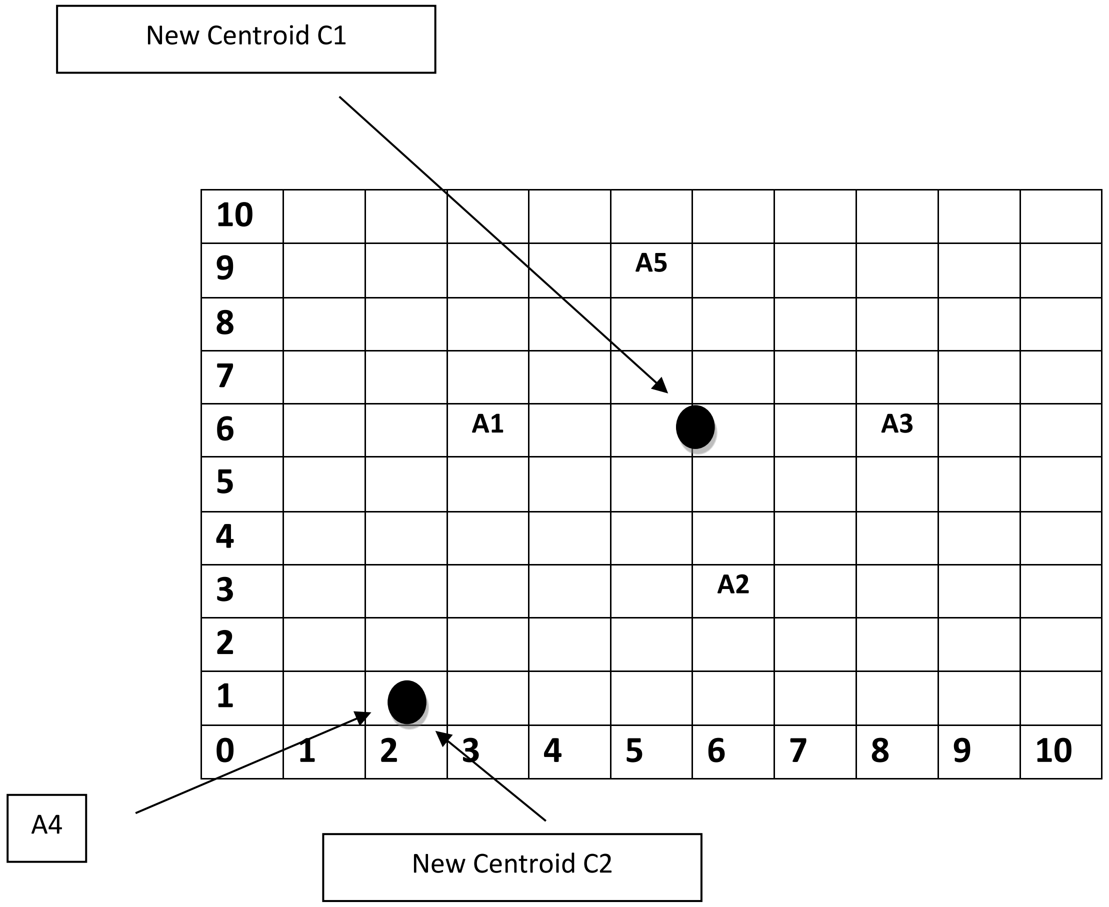
A1 is Close to C1 so it's assigned to C1
A2 is C1 so it's assigned to C1
A3 is Close to C1 so it's assigned to C1
A4 is C2 so it's assigned to C2
A5 is close to C1 so it's assigned to C1

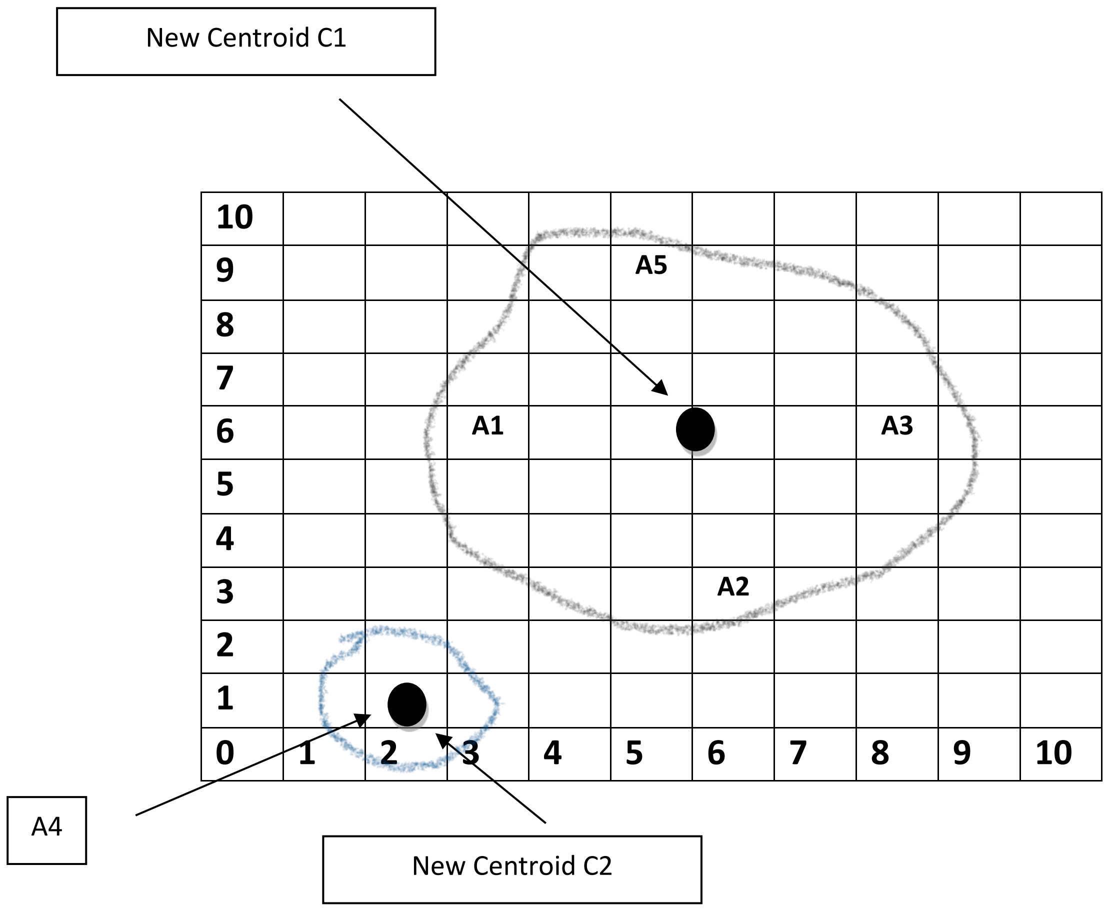
(b) Draw a 10 by 10 space with all the clustered 5 points and the coordinates of the new centroids.

New Centroids:

Centroid 1 = (3+6+8+5 / 4 , 6+3+6+9 / 4) = (5.5 , 6)

Centroid 2 = (2 , 1)




New Centroid C1
A5
A1
A3
A2
A4
New Centroid C2
10
9
8
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8
9
10

(c) Calculate the silhouette score and WSS score

Dissimilarity matrix :

Point	**A1**	**A2**	**A3**	**A4**	**A5**
A1	0	4.2	5	5.1	3.6
A2	4.2	0	3.6	4.5	6.1
A3	5	3.6	0	7.8	4.2
A4	5.1	4.5	7.8	0	8.5
A5	3.6	6.1	4.2	8.5	0

A1: a = (4.2+5+3.6) / 3 = 4.3, b = 5.1

SC = b - a / max(b,a) = (5.1 – 4.3)/ 5.1 = 0.16

A2: a = (4.2+3.6+6.1)/ 3 = 4.63, b = 4.5

SC = b - a / max(b,a) = (4.5 – 4.63)/ 4.63 = -0.028

A3: a = (5+3.6+4.2) / 3 = 4.3, b = 7.8

SC = b - a / max(b,a) = (7.8 – 4.3)/ 7.8 = 0.45

A4: a = 0, b = (4.1 + 4.5 + 7.8 + 8.5)/ 4 = 6.5

SC = b - a / max(b,a) = (6.5 – 0)/ 6.5 = 0

A5: a = (3.6 + 6.1 + 4.2)/3 = 4.6, b = 8.5

SC = b - a / max(b,a) = (8.5 – 4.6)/ 8.5 = 0.46

Average silhouette score for C1:

silhouette score = (0.16 − 0.028 + 0.45 + 0.46) / 4 = 0.2605

Average silhouette score for C2:

silhouette score = 1

Average silhouette score = (0.16 - 0.028 +0.45 +1 + 0.46) / 5 = 0.4084

WSS Score:

$$\textbf{WSS} = \sum_{1}^{n}(X_i - C_i)^2$$

For: C1 : (5.5 , 6) C2 : (2 , 1)

(A1 , C1) : (3 - 5.5)^2 + (6 - 6)^2 = 6.25

(A2 , C1) : (6 - 5.5)^2 + (3 - 6)^2 =9.25

(A3 , C1) : (8 - 5.5)^2 + (6 - 6)^2 =6.25

(A4 , C1) : (2 - 2)^2 + (1 - 1)^2 = 0

(A5, C1) : (5 - 5.5)^2 + (9 - 6)^2 = 9.25

WSS Score = 6.25 + 9.25 + 6.25 + 0 + 9.25 = 31

Part2

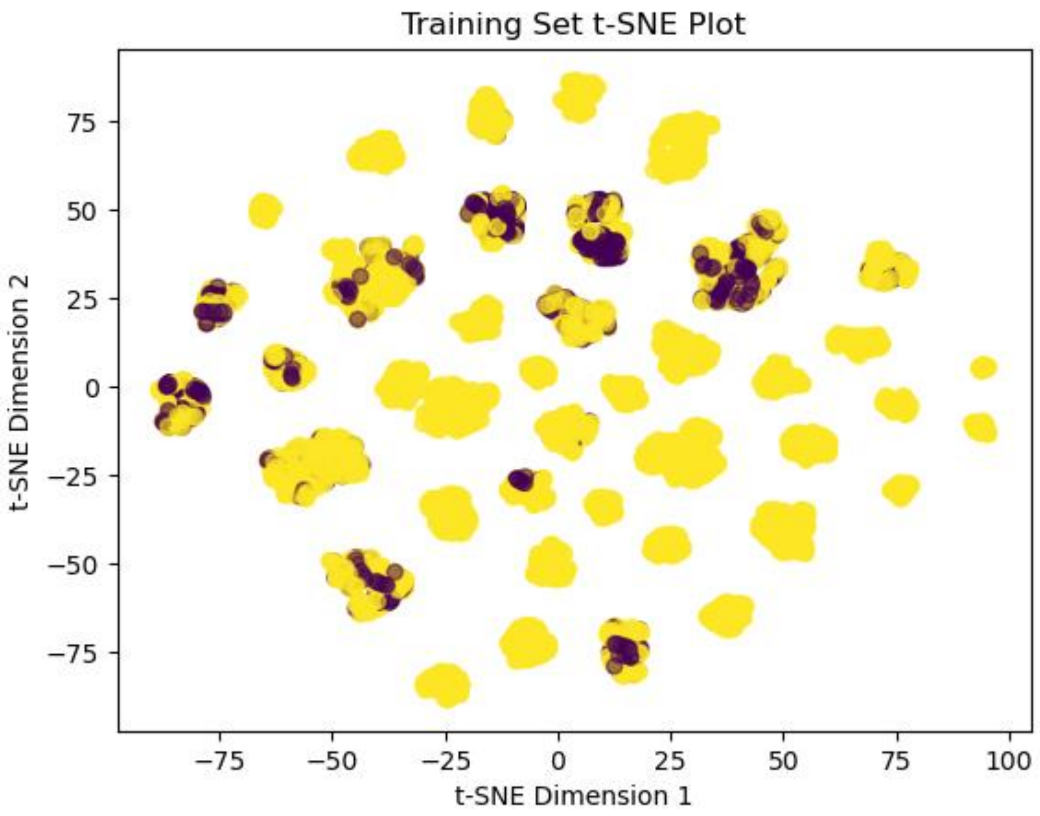
a) we load a dataset from a CSV file using the pd.read_csv() function and assigns the resulting DataFrame object to the variable df. then selects the rows of the DataFrame df where the Day column has values 0, 1, or 2 and assigns them to the DataFrame train_df. Similarly, it selects the rows where the Day column has value 3 and assigns them to the DataFrame test_df. and drops the columns ID, Day, and Ligitimacy from the DataFrames train_df and test_df using the drop() method. The resulting DataFrames without these columns are assigned to the variables x_train and x_test, respectively.

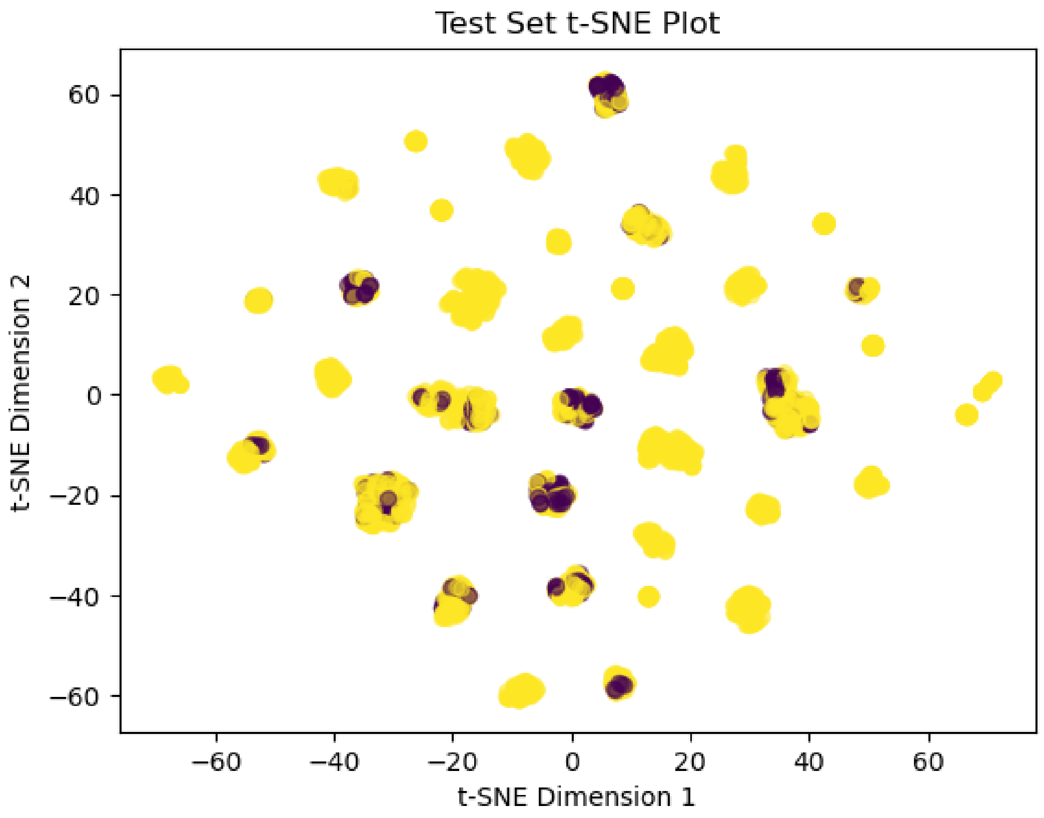
b) Gaussian Naive Bayes (GaussianNB) classifier on the training data x_train and y_train using the fit() method of the GaussianNB object nb. then uses the trained classifier to make predictions on the test data x_test using the predict() method of the GaussianNB object nb. The predicted labels are assigned to the variable nb_pred. calculates the F1 score of the classifier's predictions using the f1_score() function from scikit-learn. The F1 score is a metric that combines precision and recall to measure the accuracy of a binary classification model.OUTPUT:F1 Score: 0.9322 ,

K-Nearest Neighbors (KNN) classifier on the training data x_train and y_train using the fit() method of the KNeighborsClassifier object knn. OUTPUT:F1 Score: 0. 0.922

c) we reduces the dimensionality of the data using t-distributed Stochastic Neighbor Embedding (t-SNE) and then plots the resulting two-dimensional embeddings for the training and test sets.

we create a TSNE object with n_components=2 to specify that the dimensionality of the data should be reduced to two dimensions. The random_state parameter is set to 42 to ensure reproducibility of the results. we use the fit_transform() method of the TSNE object to transform the training and test sets (X_train and X_test, respectively) into two-dimensional embeddings. The resulting embeddings are assigned to the variables X_train_tsne and X_test_tsne, respectively. then creates a scatter plot of the training set embeddings using the plt.scatter() function. The c parameter specifies that the color of each point should correspond to its label (y_train), and the cmap parameter specifies the color map to use (viridis). The alpha parameter sets the transparency level of the points to 0.5.

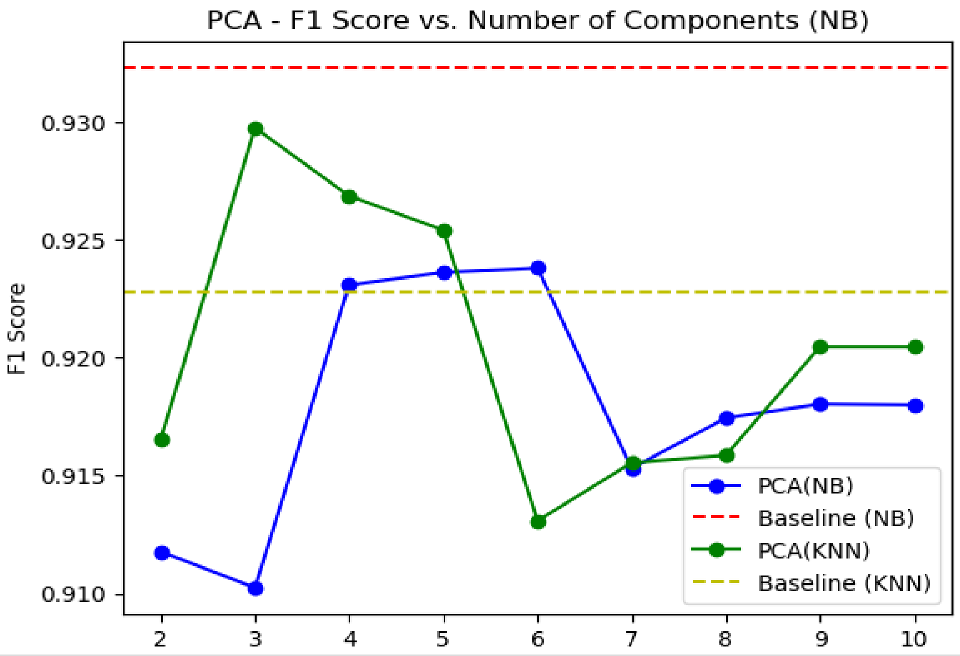

Training Set t-SNE Plot
t-SNE Dimension 2
t-SNE Dimension 1
75
50
25
0
−25
−50
−75
−75
−50
−25
0
25
50
75
100

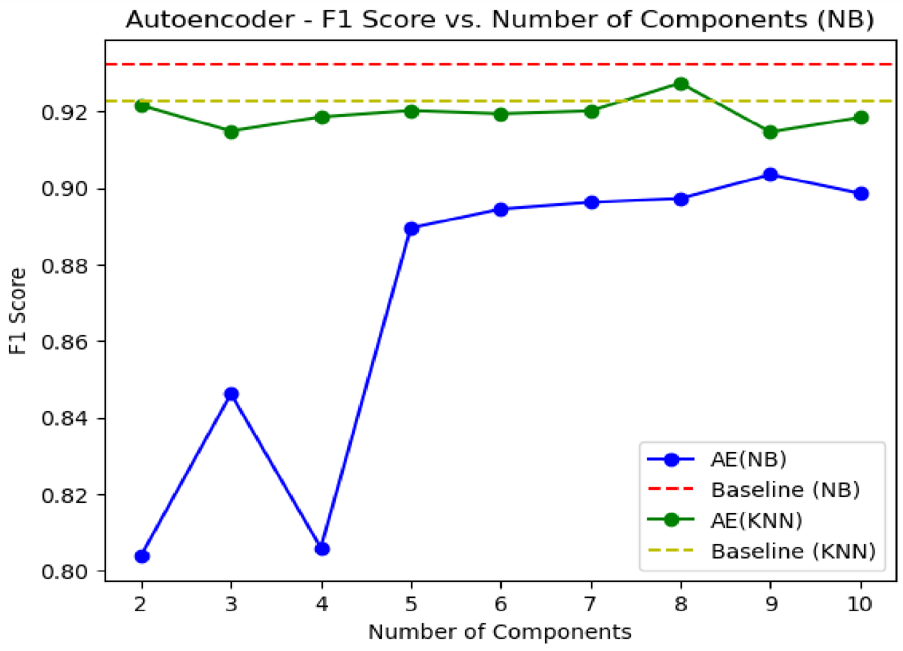

Test Set t-SNE Plot
t-SNE Dimension 2
t-SNE Dimension 1
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60

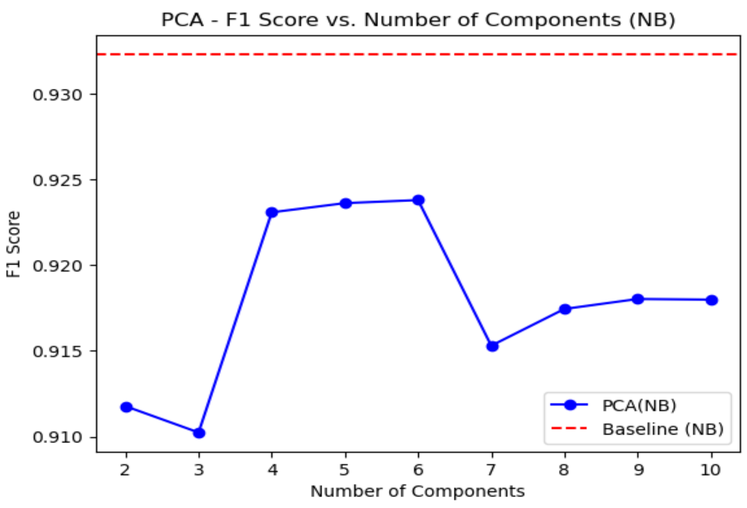
2)

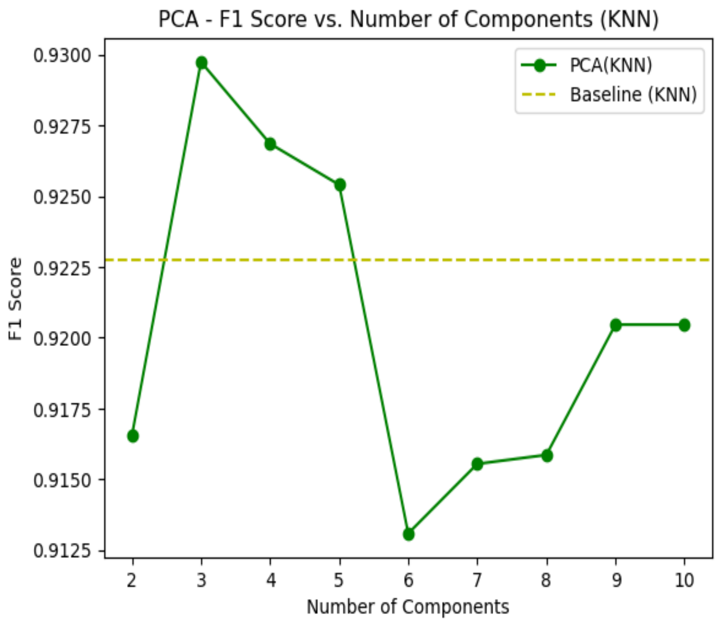
a) performs dimensionality reduction using Principal Component Analysis (PCA) and Autoencoder (AE) and then applies two classifiers, Gaussian Naive Bayes and K-Nearest Neighbors, to the reduced data. The F1 scores of the classifiers' predictions on the test set are stored in separate lists for PCA and AE. defines a range of dimensions (from 2 to 10) to use for PCA and AE using the range() and list() functions. initializes two empty lists (pca_f1_scores_nb and pca_f1_scores_knn) to store the F1 scores of the Gaussian Naive Bayes and K-Nearest Neighbors classifiers, respectively, for each dimensionality reduction method (PCA and AE). then enters a loop that performs dimensionality reduction using PCA and applies the classifiers to the reduced data for each dimension in the range defined earlier.Inside the loop, the code creates a PCA object with n_components set to the current dimension (n) and a Pipeline object that scales the data using MinMaxScaler() and applies PCA. The fit_transform() and transform() methods of the pipeline are then used to transform the training and test sets into lower dimensions, respectively. The resulting transformed data is assigned to the variables X_train_pca and X_test_pca. then fits the Gaussian Naive Bayes and K-Nearest Neighbors classifiers to the reduced data (X_train_pca and y_train) using the fit() method of each classifier. The classifiers are then used to predict the labels of the test data (X_test_pca) using the predict() method. The F1 scores of the classifiers' predictions on the test data are then computed using the f1_score() function and appended to the pca_f1_scores_nb and pca_f1_scores_knn lists, respectively. then enters a similar loop that performs dimensionality reduction using Autoencoder and applies the classifiers to the reduced data for each dimension in the range defined earlier.Inside the loop, the code creates an MLPRegressor object with a single hidden layer of size n, and fits it to the training data using the fit() method. The trained autoencoder is then used to reduce the dimensionality of the training and test sets using the predict() method, and the resulting reduced data is assigned to the variables X_train_ae and X_test_ae.

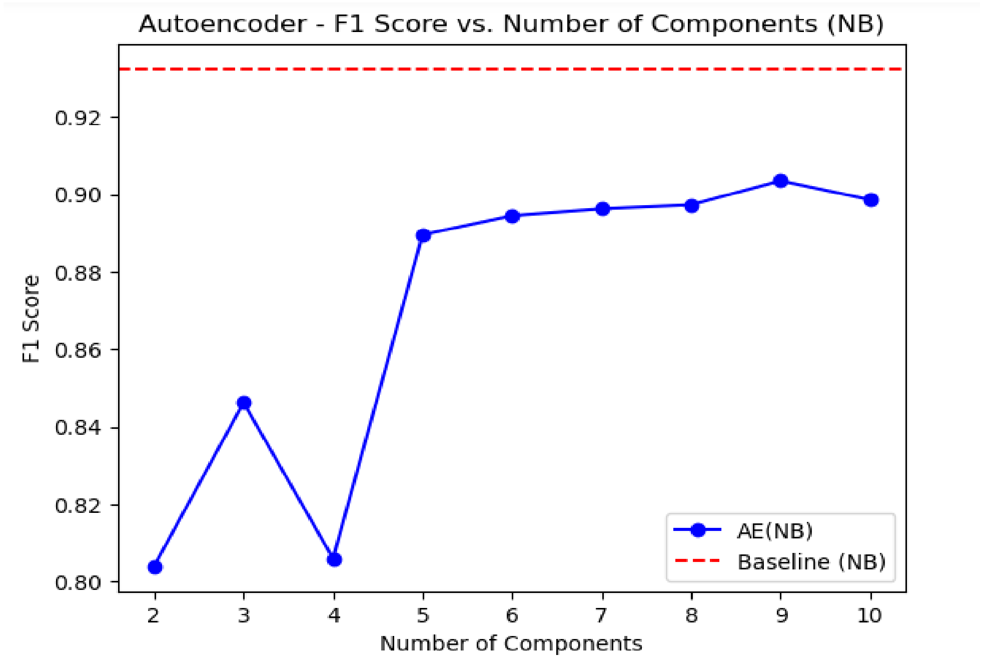
then fits the Gaussian Naive Bayes and K-Nearest Neighbors classifiers to the reduced data (X_train_ae and y_train) using the fit() method of each classifier. The classifiers are then used to predict the labels of the test data (X_test_ae) using the predict() method. The F1 scores of the classifiers' predictions on the test data are then computed using the f1_score() function and appended to the ae_f1_scores_nb and ae_f1_scores_knn lists, respectively.

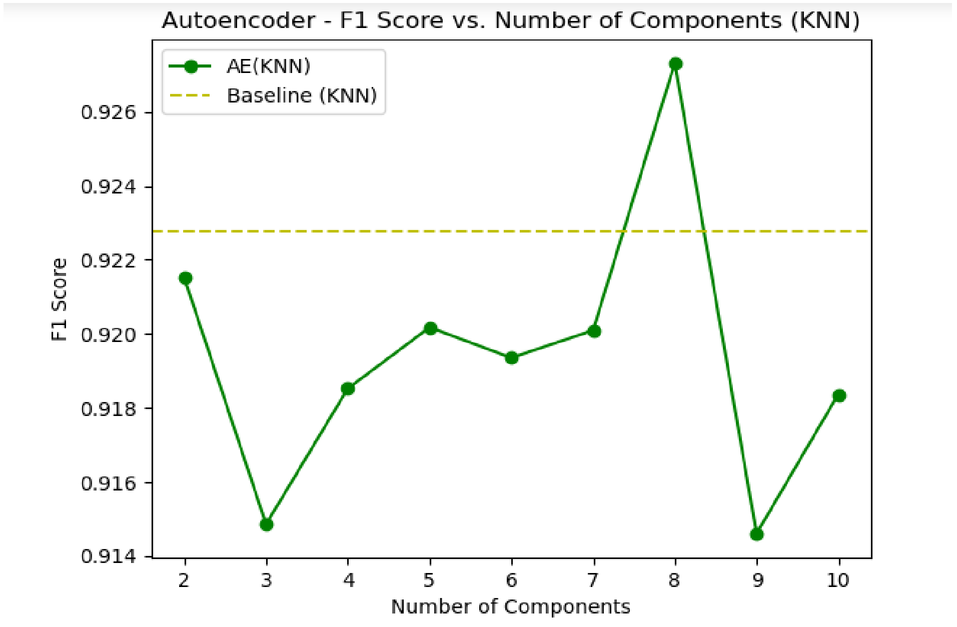

PCA - F1 Score vs. Number of Components (NB)
F1 Score
0.930
0.925
0.920
0.915
0.910
2
3
4
5
6
7
8
9
10
PCA(NB)
Baseline (NB)
PCA(KNN)
Baseline (KNN)


Autoencoder - F1 Score vs. Number of Components (NB)
F1 Score
0.92
0.90
0.88
0.86
0.84
0.82
0.80
2
3
4
5
6
7
8
9
10
Number of Components
AE(NB)
Baseline (NB)
AE(KNN)
Baseline (KNN)


PCA - F1 Score vs. Number of Components (NB)
F1 Score
0.910
0.915
0.920
0.925
0.930
Number of Components
2
3
4
5
6
7
8
9
10
PCA(NB)
Baseline (NB)

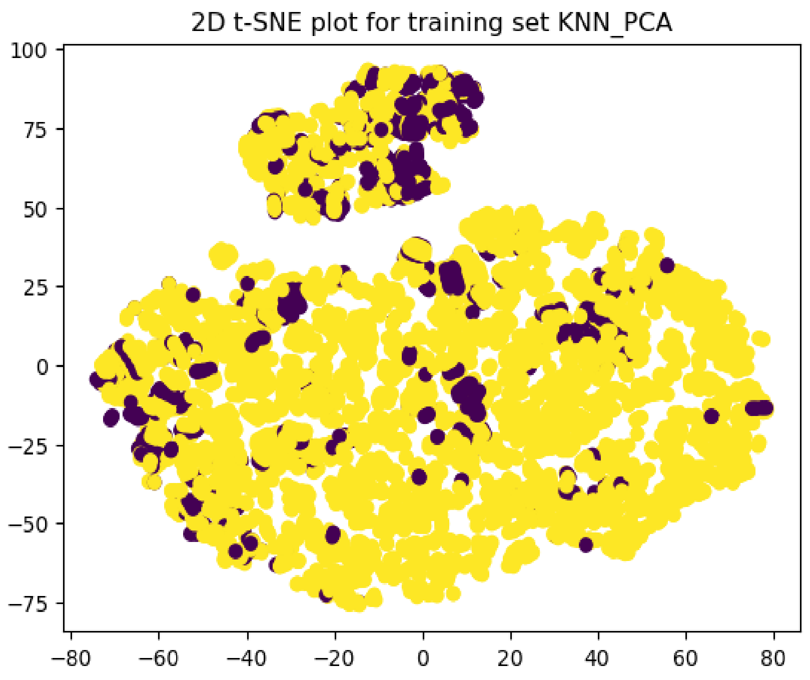

PCA - F1 Score vs. Number of Components (KNN)
F1 Score
0.9125
0.9150
0.9175
0.9200
0.9225
0.9250
0.9275
0.9300
Number of Components
2
3
4
5
6
7
8
9
10
PCA(KNN)
Baseline (KNN)

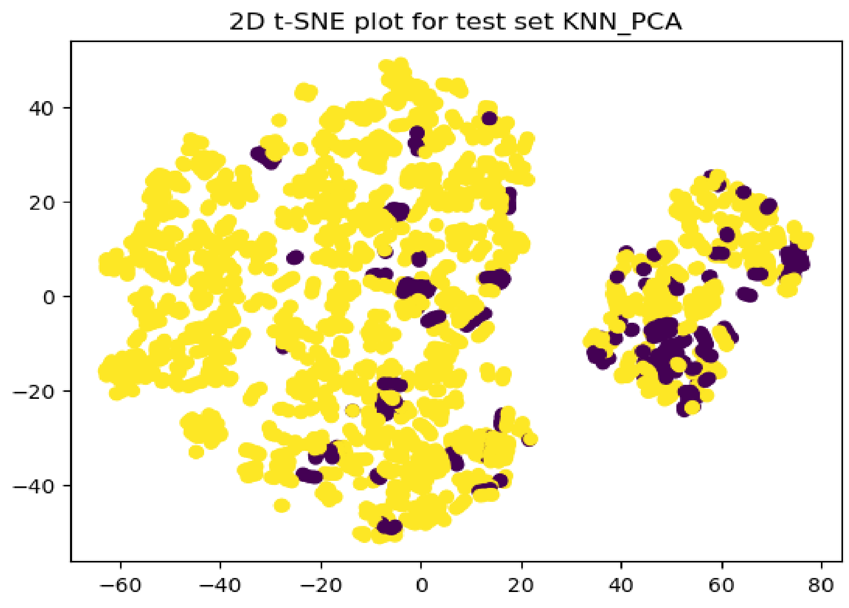




b)

Best Performance (PCA KNN) allows to visualize the reduced data in two dimensions using t-SNE and to see if the KNN classifier can separate the different classes in the reduced space. The PCA dimensionality reduction step is performed to speed up the training of the KNN classifier and reduce overfitting, while the t-SNE plots provide a visual representation of the reduced data that can help us understand the clustering and separability of the different classes.



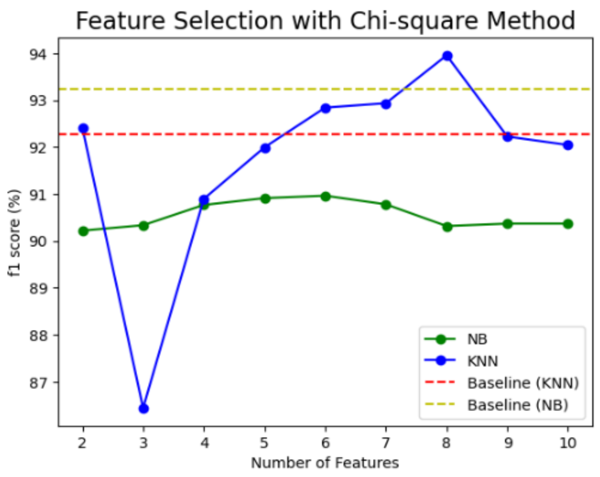


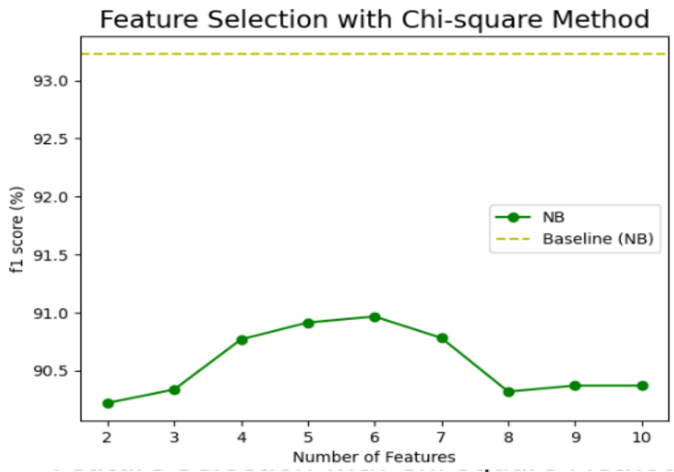
3)

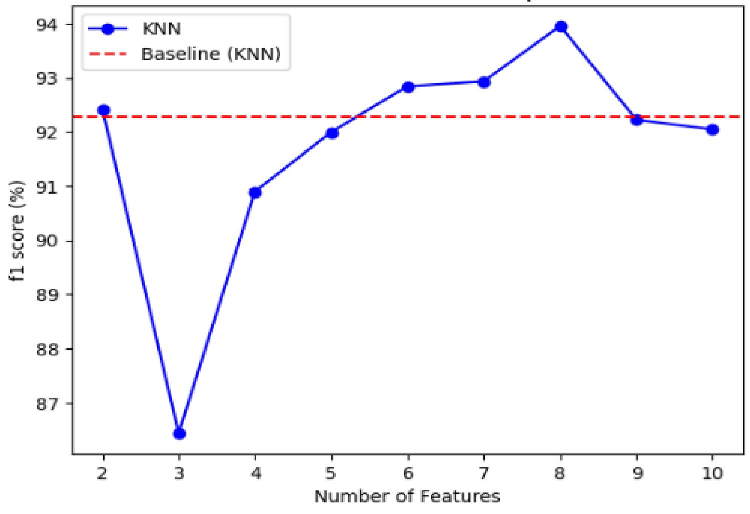
a) selects the best number of features using the chi-square method with the SelectKBest feature selection object, and then plots the f1 score versus the number of features for the Naive Bayes and KNN classifiers. initializes a GaussianNB() and KNeighborsClassifier() object as the classifiers, and initializes a dictionary to store the f1 scores for each number of features for both classifiers.

then iterates over a range of number of features, from 2 to 10, and for each number of features, it initializes a SelectKBest object with the chi2 scoring metric, fits and transforms the training and test data to the selected number of features using the fit_transform() method, selects the best features, trains the classifiers using the transformed training data, and evaluates their performance on the test set using the f1_score() function. The f1 scores for each classifier and each number of features are stored in their corresponding dictionaries.

```
Maximum f1 score nb: 90.96470588235293
Best number of features nb: 6
Maximum f1 score KNN: 93.95565927654609
Best number of features KNN: 8
```

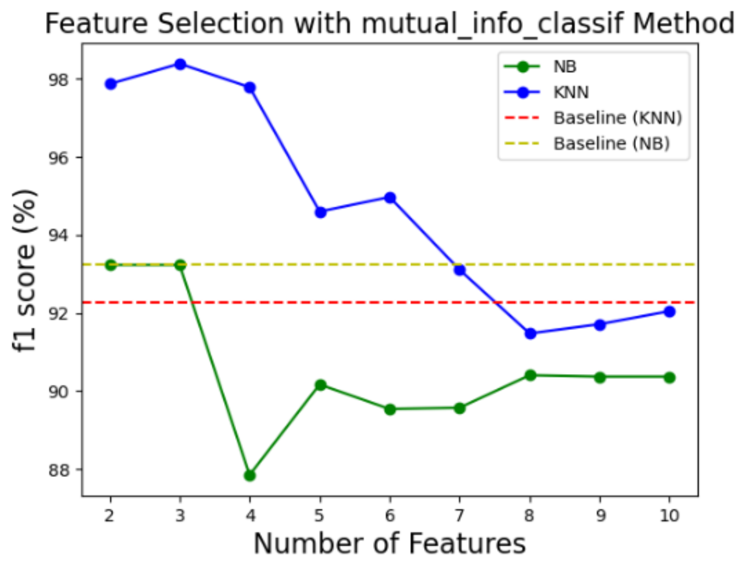


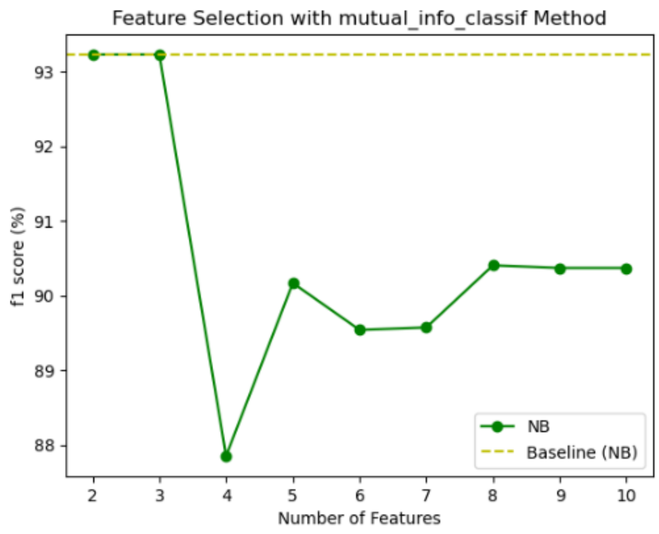


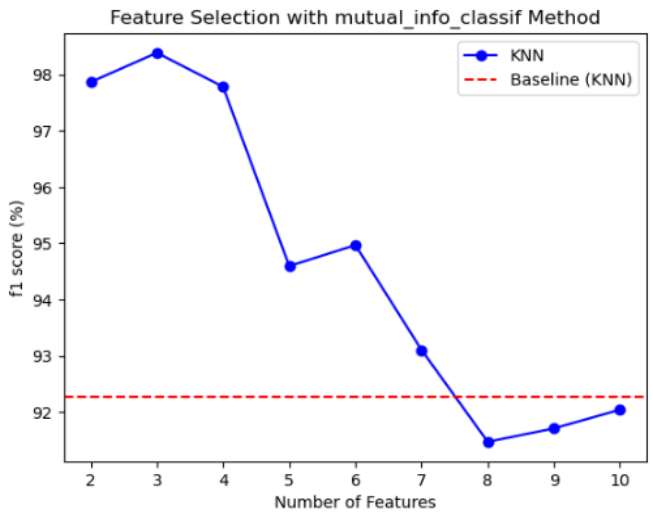


b) it uses the Recursive Feature Elimination (RFE) method instead of the chi-square method for feature selection. The RFE method selects the best features by recursively training a model and eliminating the least important feature until the desired number of features is reached.

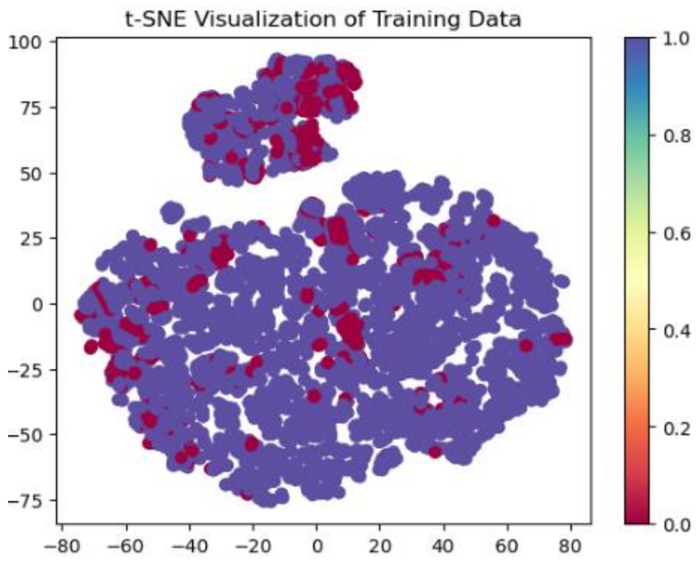
```
Maximum f1 score nb: 93.22916666666667
Best number of features nb: 2
Maximum f1 score knn: 98.38369641602249
Best number of features knn: 3
```

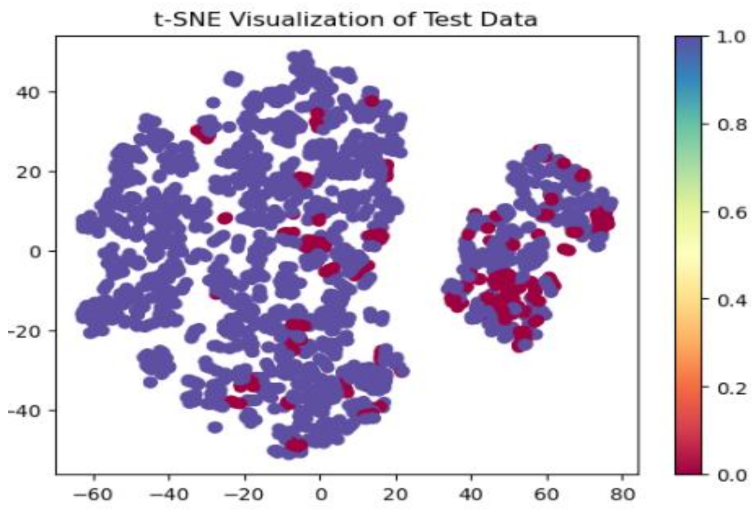






c) performs t-SNE visualization on the reduced feature set obtained from the Recursive Feature Elimination (RFE) method using a logistic regression model as the estimator.





4)

a)

- Latitude and longitude features only used as features as name “`processed_data`”

- then we start by filter data by 'Legitimacy' into two sets: filtered_data_1 for instances where the 'Legitimacy' column is equal to 1, and filtered_data_0 for instances where 'Legitimacy' is equal to 0.

-A dictionary cc to store the number of filtered_data_1 instances for each desired cluster number.

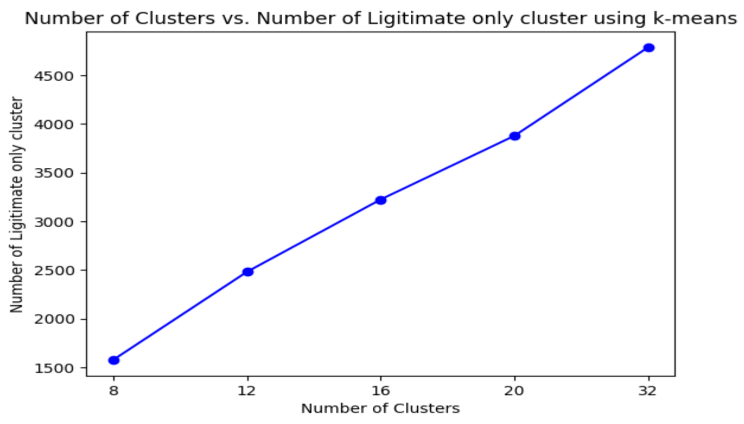
- iterates over the desired cluster numbers.
- Inside the loop, it creates an instance of KMeans with the current number of clusters.
- It fits the K-means model to the processed_data (assuming processed_data is “ processed_data = data[['Latitude', 'Longitude']].values”
-)
- Then predict the cluster labels for the filtered data where 'Ligitimacy' is 1 using the predict method of the KMeans model, AND also where 'Ligitimacy' is 0.
- then count the occurrences of each cluster label for both the 'Ligitimacy' equal to 1 and 0 cases using collections.Counter.
- then calculate the ratio of 'Ligitimacy' equal to 1 instances to the total instances (equal to 1 and 0) for each cluster label.

Then I put threshold at `0.95 where we can allow for small error by 0.5`

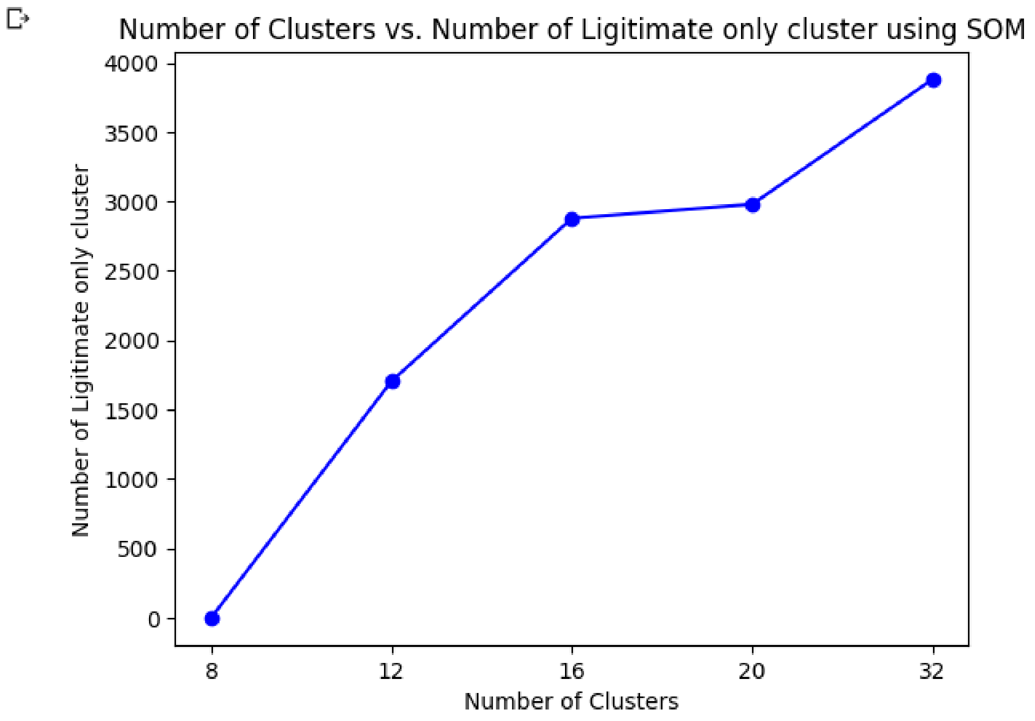
```
    for i in Ligitimacy_1:

        if (Ligitimacy_1[i] / (Ligitimacy_1[i] + Ligitimacy_0[i])) >=
0.95:

            kmeans_dict_desired_clusters[str(n_clusters)] +=
Ligitimacy_1[i]
```

this line graph appear

and insight from it (the total number of legitimate only members inside the legitimate only cluster in 32 clusters so this make scene as when the number of cluster is high the model can separate class (0) and (1) **well)**

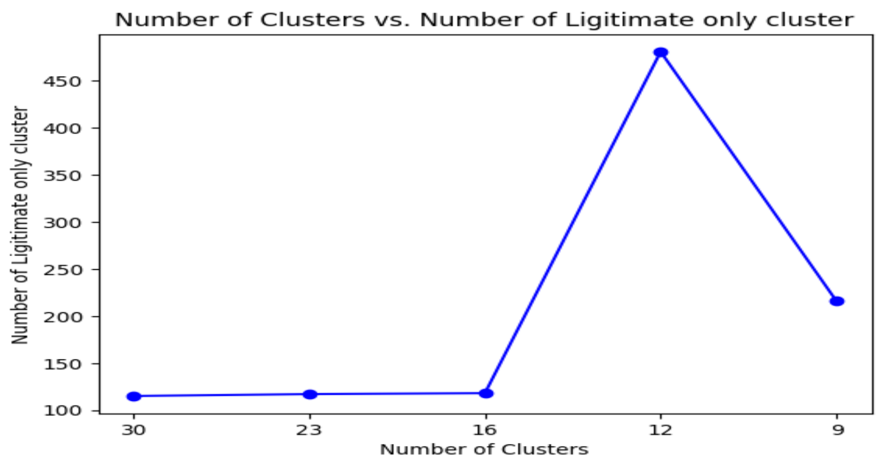


b)
we import the necessary libraries, including MiniSom from minisom and Counter from collections
we iterate over the desired numbers of clusters.
Inside the loop, we calculate the grid size based on the desired number of clusters using square root calculations.we create an instance of MiniSom with the current grid size, number of features, sigma, and learning rate.we initialize the weights of the SOM using the random_weights_init method and the processed_data.we train the SOM with the train_batch method and the processed_data.
It calculates the best matching unit (BMU) indices for the filtered data where 'Ligitimacy' is 1 using the winner method of the SOM.we calculate the BMU indices for the filtered data where 'Ligitimacy' is 0.we count the occurrences of each BMU index for both the 'Ligitimacy' equal to 1 and 0 cases using Counter.we calculate the ratio of 'Ligitimacy' equal to 1 instances to the total instances (equal to 1 and 0) for each BMU index.Then I put threshold at 0.95 where we can allow for small error by 0.5, we increment the count for the corresponding number of clusters in the dict_desired_clusters dictionary.After iterating over all BMU indices.
this line graph appear
and insight from it (the total number of legitimate only members inside the legitimate only cluster in 32 clusters so this make scene as when the number of cluster is high the model can separate class (0) and (1) **well)**



c)
we import the necessary libraries, including DBSCAN from sklearn.cluster, Counter from collections, matplotlib.pyplot for plotting, numpy as np for numerical operations, and tqdm for progress tracking.
we initialize an empty dictionary comb to store the combinations of epsilon, minpoints, and resulting clusters that match the desired cluster numbers.we perform nested loops over different epsilon and minpoints values using np.arange and range.Inside the loops, we create an instance of DBSCAN with the current epsilon and minpoints values.we fit the DBSCAN model to the processed_data.we extract the core sample indices using db.core_sample_indices_ and create a boolean mask indicating the core samples.we assign the cluster labels to the labels variable.we calculate the number of clusters by counting the unique labels (excluding -1, which represents outliers) using set(labels).If the number of clusters matches any of the desired cluster numbers, it appends the epsilon, minpoints, and clusters to the comb dictionary.
After iterating over all combinations, the code stores the epsilon, minpoints, and clusters information in the comb dictionary.

we iterate over the desired numbers of clusters and corresponding minimum points (min_samples) from the result_dict.Inside the loop, we create an instance of DBSCAN with the desired epsilon (eps) and the current minimum points value.we fit the DBSCAN model to the processed_data.we get the predicted cluster labels for the filtered data where 'Ligitimacy' is 1 using the labels_ attribute of the DBSCAN model.we get the predicted cluster labels for the filtered data where 'Ligitimacy' is 0.we count the occurrences of each cluster label for both the 'Ligitimacy' equal to 1 and 0 cases using Counter.We calculate the ratio of 'Ligitimacy' equal to 1 instances to the total instances (equal to 1 and 0) for each cluster label.Then I put threshold at `0.95 where we can allow for small error by 0.5`, it increments the count for the corresponding number of clusters in the DBSCAN_desired_clusters dictionary.After iterating over all cluster labels, the code updates the counts of pure instances in the DBSCAN_desired_clusters dictionary.this line graph appear and insight from it (the total number of legitimate only members inside the legitimate only cluster in 12 clusters so this make scene as when the number of cluster is high the model can separate class (0) and (1) **well)**



5) Conclusion Part:

(a) conclusion in:

part 1:

After prepare the data .We use Gaussian Naive Bayes and K-Nearest Neighbors (KNN) classifier to the training data and uses it to make predictions on the test data. It then calculates the F1 score, which is a performance metric that measures the balance between precision and recall of the classifier . allow for the comparison of the performance classifier with other classifiers that are trained and tested on the same dataset. This can help in identifying the best classifier for the given task and dataset . reduces the dimensionality of the data using t-Distributed Stochastic Neighbor Embedding (t-SNE) and plots the transformed data in a 2D scatter plot.

part 2:

provided performs dimensionality reduction using Principal Component Analysis (PCA) and Autoencoder (AE) for different dimensions and evaluates the performance of Naive Bayes (NB) and K-Nearest Neighbors (KNN) classifiers on the reduced data. constructs a pipeline for PCA that includes scaling the data and applying PCA. This allows for easy modification of the pipeline to include additional preprocessing steps or different dimensionality reduction techniques. easily extensible to include additional dimensionality reduction techniques or classifiers, allowing for a wide range of experiments and evaluations to be performed. provided applies Principal Component Analysis (PCA) to reduce the dimensionality of the input data to 10 components, trains a K-Nearest Neighbors (KNN) classifier on the reduced data, and generates 2D t-SNE plots for the training and test sets

part 3:

provided performs feature selection using the chi-square method for different numbers of features, evaluates the performance of Naive Bayes (NB) and K-Nearest Neighbors (KNN) classifiers on the selected features, and plots the F1 scores for each classifier as a function of the number of features.
using the Recursive Feature Elimination (RFE) method with a decision tree estimator for different numbers of features . provided applies Recursive Feature Elimination (RFE) with logistic regression estimator to select the top 100 features, and then applies t-SNE to visualize the reduced training and test data in two dimensions.

part 4:

For (K-means, MiniSom, and DBSCAN) to cluster data based on latitude and longitude features while filtering based on the 'Legitimacy' column.

Method 4a) uses K-means clustering to separate data into two sets: 'Legitimacy' equal to 1 and 0. The number of instances for each desired cluster is stored, and ratios of 'Legitimacy' equal to 1 instances are calculated for each cluster label. A threshold of 0.95 is applied, incrementing counts if the ratio meets the threshold.

Method 4b) applies MiniSom clustering. The best matching unit (BMU) indices are obtained for 'Legitimacy' equal to 1 and 0 cases. Ratios of 'Legitimacy' equal to 1 instances are calculated for each BMU index, and counts are incremented based on a threshold of 0.95.

Method 4c) utilizes DBSCAN clustering. Combinations of epsilon, minpoints, and resulting clusters that match desired numbers are stored. The DBSCAN model is fitted, and cluster labels are obtained. Ratios of 'Legitimacy' equal to 1 instances are calculated for each cluster label, and counts are incremented if the threshold of 0.95 is met.

Overall, the code generates line graphs showing the increase in legitimate-only members within the legitimate-only cluster as the number of clusters increases. This indicates better class separation.