(Group_20) Assignment_3

Prepared for Prof: Dr. Murat Simsek

Faculty of Engineering, University of u Ottawa

ELG5255[EG] Applied Machine Learning

Prepared by

Abdelsalam, Mohanad
Attia, Nada
El-Sabow, Shady

Part1:

a) Use the k-means algorithm and Euclidean distance to cluster the following 5 data points into 2 clusters:

Solve

- **the initial centroids are A2=(6,3) , A4=(2,1)**
- **calculate the Euclidean distance from this equation:**

$$d(p1,p2) = \sqrt{(x2-x1)^2 + (y2-y1)^2}$$

<table>
<tr>
<td>for A1 point from first centroid:
$d(p1,p2) = \sqrt{(6-3)^2 + (3-6)^2} = 4.24$
for A1 point from second centroid:
$d(p1,p2) = \sqrt{(2-3)^2 + (1-6)^2} = 5.1$</td>
<td>for A2 point from first centroid:
$d(p1,p2) = \sqrt{(6-6)^2 + (3-3)^2} = 0$
for A2 point from second centroid:
$d(p1,p2) = \sqrt{(2-6)^2 + (1-3)^2} = 4.47$</td>
</tr>
<tr><td colspan="2"></td></tr>
<tr>
<td>for A3 point from first centroid:
$d(p1,p2) = \sqrt{(6-8)^2 + (3-6)^2} = 3.61$
for A3 point from second centroid:
$d(p1,p2) = \sqrt{(2-8)^2 + (1-6)^2} = 7.8$</td>
<td>for A4 point from first centroid:
$d(p1,p2) = \sqrt{(6-2)^2 + (3-1)^2} = 4.47$
for A4 point from second centroid:
$d(p1,p2) = \sqrt{(2-2)^2 + (1-1)^2} = 0$</td>
</tr>
<tr><td colspan="2"></td></tr>
<tr>
<td colspan="2">for A5 point from first centroid:
$d(p1,p2) = \sqrt{(6-5)^2 + (3-9)^2} = 6.08$
for A5 point from second centroid:
$d(p1,p2) = \sqrt{(2-5)^2 + (1-9)^2} = 8.54$</td>
</tr>
</table>

- **cluster the point which into cluster 1 (close to point A1) and 2 (close to point A4)**

Data points			Distance from point to centroid using Euclidean distance				Cluster it belongs to (choose the minimum value)
			Cluster 1		Cluster 2		
Point name	X 1	Y1	X2	Y2	X2	Y2	
			6	3	2	1	
A1	3	6	4.24		5.1		1
A2	6	3	0		4.47		1
A3	8	6	3.61		7.8		1
A4	2	1	4.47		0		2
A5	5	9	6.08		8.54		1

- calculate the new centroid:

  cluster1 centroid = ($\frac{3+6+8+5}{4}$, $\frac{6+3+6+9}{4}$) = (5.5 , 6)

  cluster2 centroid = (2 ,1)

Repeat steps 1-3 until the centroids no longer change.

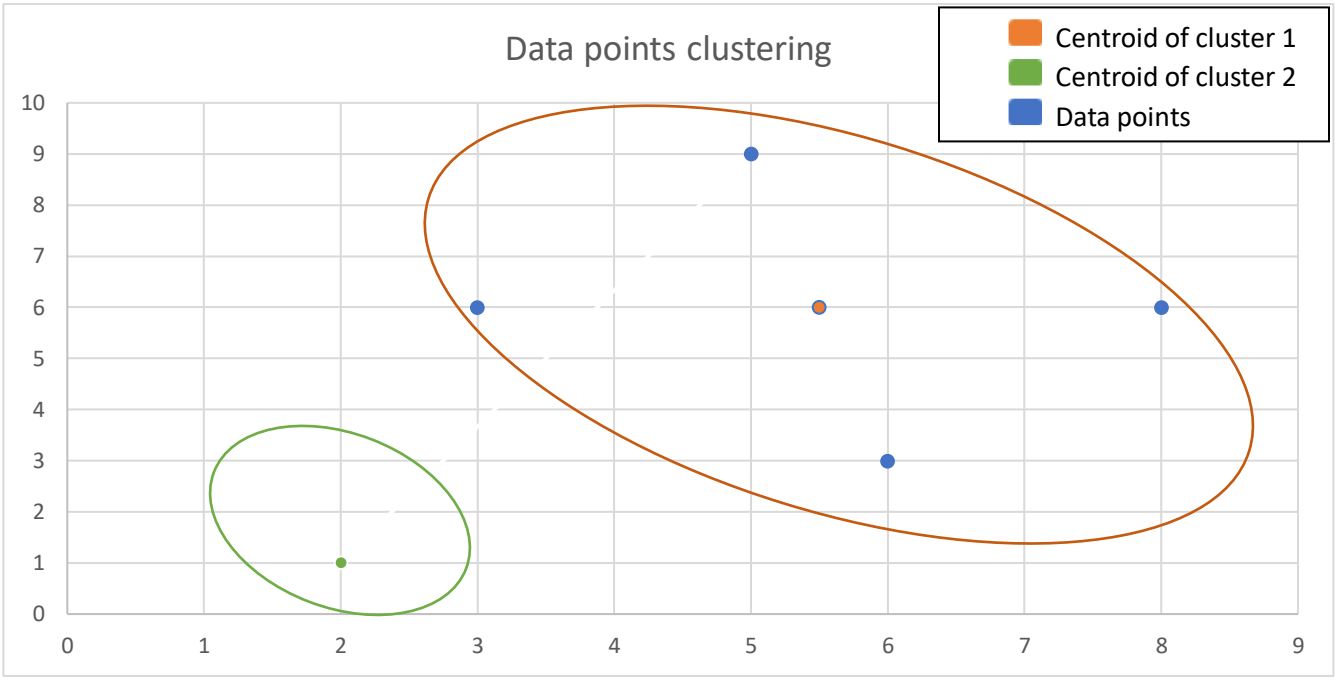
Data points			Distance from point to centroid using Euclidean distance				Cluster it belongs to (choose the minimum value)	New cluster
			Cluster 1		Cluster 2			
Point name	X 1	Y1	X2	Y2	X2	Y2		
			5.5	6	2	1		
A1	3	6	2.5		5.1		1	1
A2	6	3	3.04		4.47		1	1
A3	8	6	2.5		7.8		1	1
A4	2	1	6.1		0		2	2
A5	5	9	3.04		8.54		1	1

❖ As there is no change, so:

➢ Point A1 ,A2 ,A3 ,A5 are belong to cluster 1

➢ Point A4 are belong to cluster 2

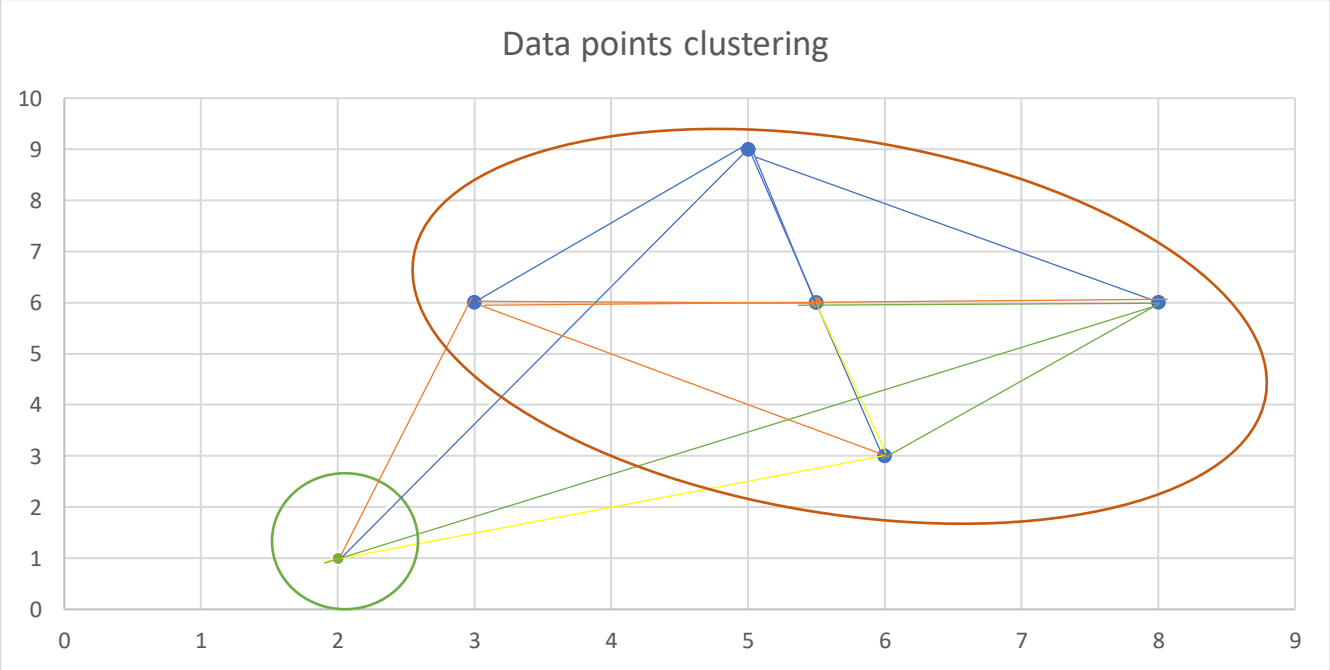
b) Draw a 10 by 10 space with all the clustered 5 points and the coordinates of the new centroids.



c) Calculate the silhouette score and WSS score.

Calculate the distance matrix by applying the Euclidean distance from this equation:

$$\mathrm{d(p1,p2)} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$



Distance between A1 And A2: $d1 = \sqrt{(6-3)^2 + (3-6)^2} = 4.24$ **Distance between A1** and A3: $d2 = \sqrt{(8-3)^2 + (6-6)^2} = 5$ **Distance between A1** and A4: $d3 = \sqrt{(2-3)^2 + (1-6)^2} = 5.1$ **Distance between A1** and A5: $d4 = \sqrt{(5-3)^2 + (9-6)^2} = 3.61$ **Distance between A1 and** first centroid: $d5 = \sqrt{(5.5-3)^2 + (6-6)^2} = 2.5$	**Distance between A2 and** first centroid: $d6 = \sqrt{(5.5-6)^2 + (6-3)^2} = 3.35$ **Distance between A2** And A3: $d7 = \sqrt{(8-6)^2 + (6-3)^2} = 3.61$ **Distance between A2** And A4: $d8 = \sqrt{(2-6)^2 + (1-3)^2} = 4.47$ **Distance between A2** And A5: $d9 = \sqrt{(5-6)^2 + (9-3)^2} = 6.08$
Distance between A3 And A4: $d10 = \sqrt{(2-8)^2 + (1-6)^2} = 7.8$ **Distance between A3** And A5: $d11 = \sqrt{(5-8)^2 + (9-6)^2} = 4.24$ **Distance between A3** And first centroid: $d12 = \sqrt{(5.5-8)^2 + (6-6)^2} = 2.5$	**Distance between** A4 and A5: $d13 = \sqrt{(5-2)^2 + (9-1)^2} = 8.54$ **Distance between** A4 and second centroid: $d14 = \sqrt{(2-2)^2 + (1-1)^2} = 0$

points	A1 (3, 6)	A2 (6 , 3)	A3 (8, 6)	A4 (2,1)	A5 (5 ,9)	Centroid 1 (5.5, 6)
A1 (3, 6)	0	4.24	5	5.1	3.61	2.5
A2 (6 , 3)	4.24	0	3.61	4.47	6.08	3.35
A3 (8, 6)	5	3.61	0	7.81	4.24	2.5
A4 (2,1)	5.1	4.47	7.81	0	8.54	6.1
A5 (5 ,9)	3.61	6.08	4.24	8.54	0	3.04
Centroid 1 (5.5, 6)	2.5	3.35	2.5	6.1	3.04	0

Calculate the silhouette score as:

$$s(i) = \frac{b(i) - a(i)}{\max\{(a(i), b(i))\}}$$

For A1: a1 = $\frac{4.24+5+3.61}{3}$ =4.28 b1 = $\frac{5.1}{1} = 5.1$ $s(A1) = \frac{5.1-4.28}{5.1} = 0.16$	For A2: a2 = $\frac{4.24+3.61+6.08}{3}$ =4.64 b2 = $\frac{4.47}{1} = 4.47$ $s(A2) = \frac{4.47-4.64}{4.64} = -0.04$
For A3: a = $\frac{5+3.61+4.24}{3}$ = 4.28 b = $\frac{7.81}{1} = 7.81$ $s(A1) = \frac{7.81-4.28}{7.81} = 0.45$	For A4: a = $\frac{0}{1}$ =0 b = $\frac{5.1 + 4.47 + 7.81 + 8.54}{4} = 6.48$ $s(A1) = \frac{6.48-0}{6.48} = 1$
For A5: a = $\frac{4.24+6.08+3.61}{3}$ = 4.64 b = $\frac{8.54}{1} = 8.54$ $s(A1) = \frac{8.54-4.64}{8.54} = 0.46$	

The silhouette score for cluster 1 $= \frac{0.16 - 0.04 + 0.45 + 0.46}{4} = 0.2575$

The silhouette score for cluster 2 $= \frac{1}{1} = 1$

The overall silhouette score $= \frac{0.2575 + 1}{2} = 0.62875$

Then now calculate WSS:

$$\text{Wss} = \sum_{i=1}^{5} (xi - ci\ )\text{^}2$$

Wss for cluster1 :

$$\text{Wss} = 2.5^2 + 2.5^2 + 3.04^2 + 3.04^2 = 30.9832$$

Another solve for The silhouette score as we searched a lot to find which law is true :

$$\text{The overall silhouette score} = \frac{0.16 - 0.04 + 0.45 + 0.46 + 1}{5} = 0.206$$

And we provide a code to find The silhouette score but when we make it manually, we get the answer 0.62875 and when we use sklearn we get the answer 0.206

<u>The manual code as :</u>

In [5]:
```
import numpy as np

def k_means(data, k, initial_centroids):
    clusters = np.zeros(len(data), dtype=int)
    for i in range(len(data)):
    closest_centroid = initial_centroids[0]
    for j in range(1, k):
        if np.linalg.norm(data[i] - initial_centroids[j]) < np.linalg.norm(data[i] - closest_centroid):
            closest_centroid = initial_centroids[j]
    clusters[i] = np.where(closest_centroid == initial_centroids)[0][0]
    new_centroids = []
    for i in range(k):
        new_centroid = np.mean(data[clusters == i], axis=0)
        new_centroids.append(new_centroid)

    return clusters, new_centroids

data = np.array([(3, 6), (6, 3), (8, 6), (2, 1), (5, 9)])
k = 2
initial_centroids = np.array([(6, 3), (2, 1)])
clusters, new_centroids = k_means(data, k, initial_centroids)
```

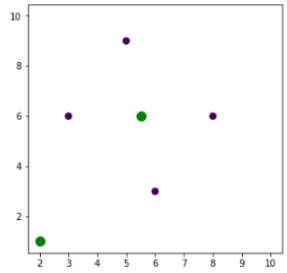
```
# Calculate the silhouette score and WSS score
silhouette_scores = []
wss_score = 0
for cluster in np.unique(clusters):
    mean = np.mean(data[clusters == cluster], axis=0)
    for point in data[clusters == cluster]:
    d1 = np.linalg.norm(point - mean)
    d2 = min([np.linalg.norm(point - p) for p in data[clusters != cluster]])
    silhouette_scores.append((d2 - d1) / max(d1, d2))
    wss_score += d1 ** 2
m=new_centroids
print("Silhouette score:", sum(silhouette_scores) / len(silhouette_scores))
print("WSS score:", wss_score)
```

```
Silhouette score: 0.6307154369103314
WSS score: 31.0
```

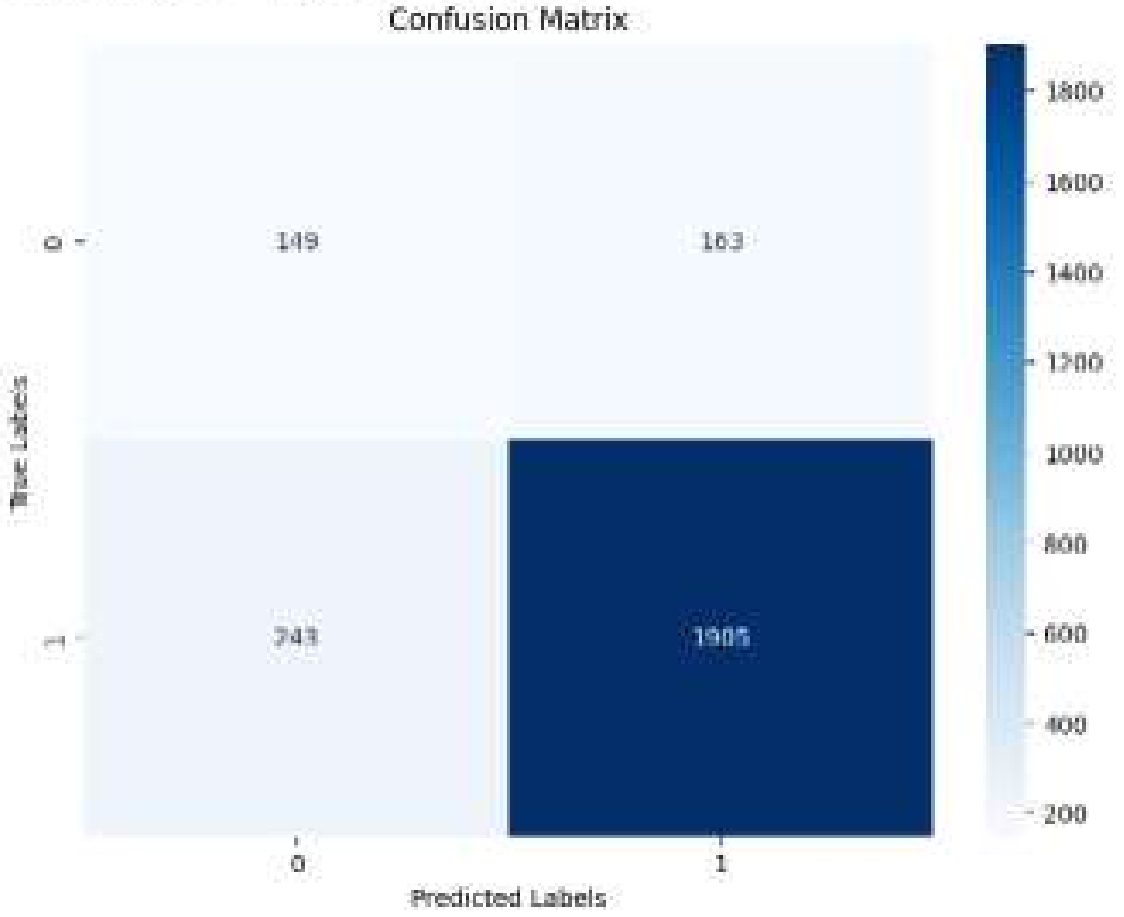
In [6]:
```
m
```

Out[6]:
```
[array([5.5, 6. ]), array([2., 1.])]
```

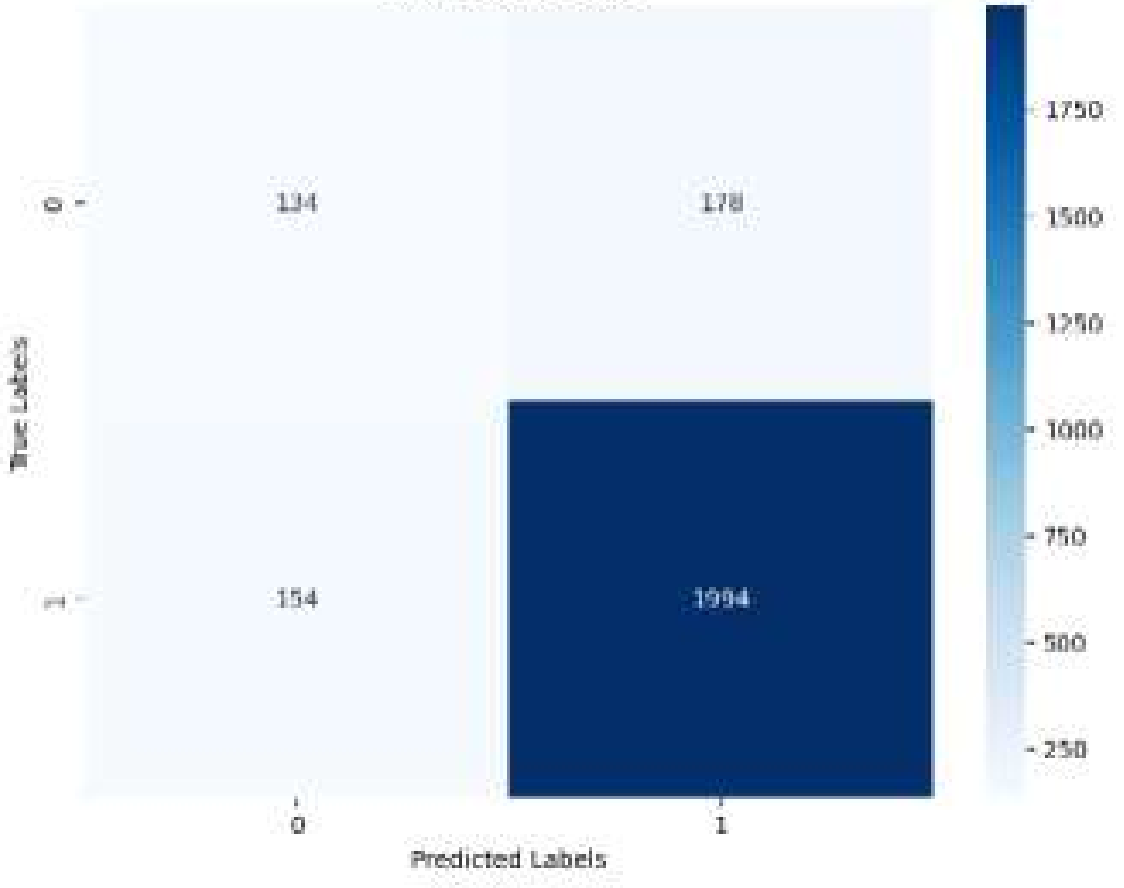
In [4]:
```
import matplotlib.pyplot as plt
import numpy as np

def plot_clusters(data, labels, centroids):
  plt.figure(figsize=(5, 5))
  plt.scatter(data[:, 0], data[:, 1], c=labels, s=50)
  plt.scatter(centroids[:, 0], centroids[:, 1], c='green', s=100)
  plt.show()

data = np.array([(3, 6), (6, 3), (8, 6), (2, 1), (5, 9)])
labels = np.array([0, 0, 0, 1, 0 ])
centroids = np.array([(5.5, 6.0), (2, 1)])

plot_clusters(data, labels, centroids)
```



Confusion Matrix - Naive Bayes Classifier:

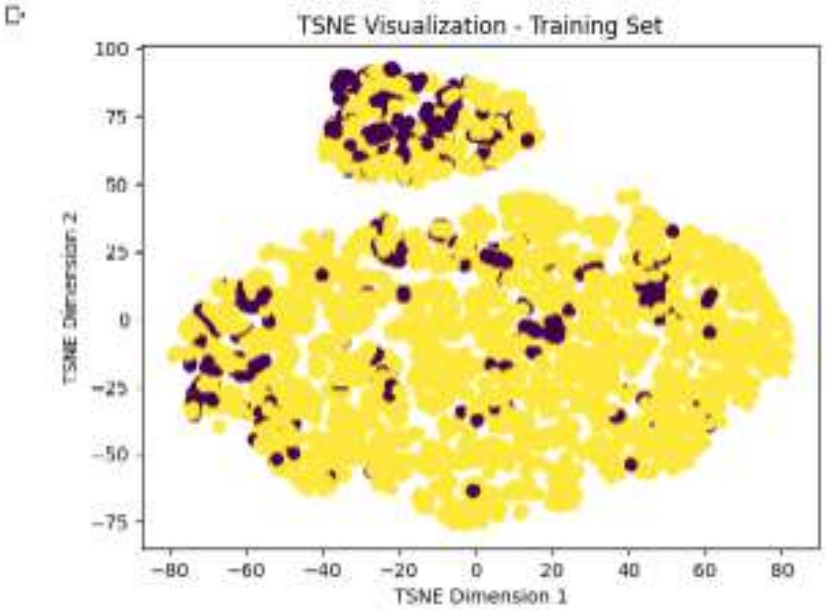


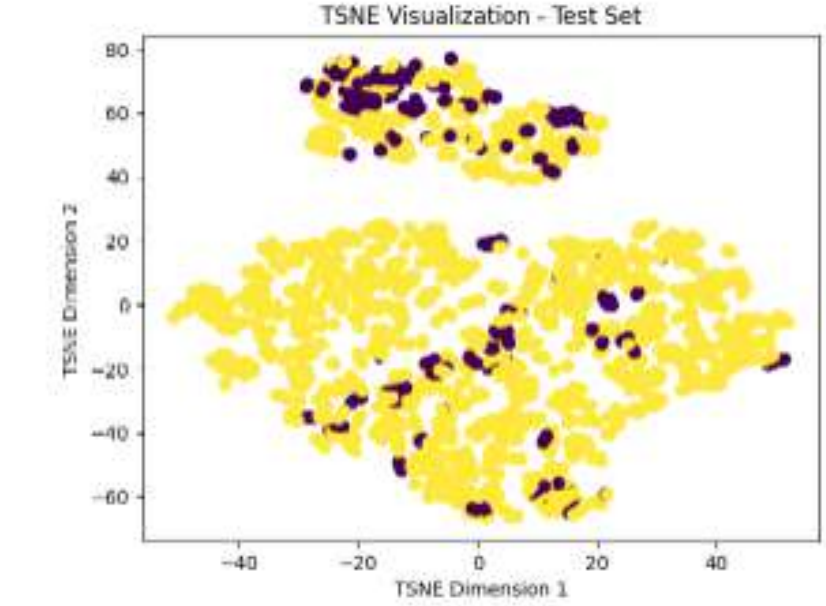
Confusion Matrix - K-Nearest Neighbor Classifier:



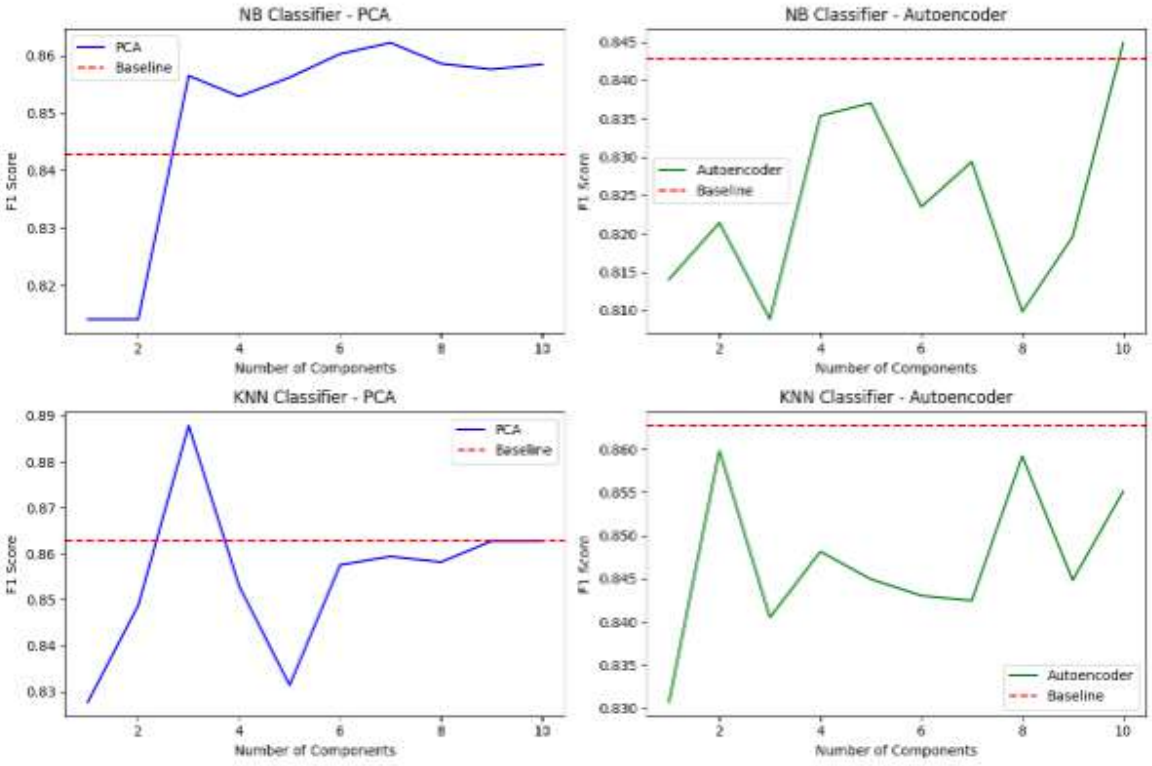
- knn is performing better in predicting 1

  NB is performing better in predicting 0

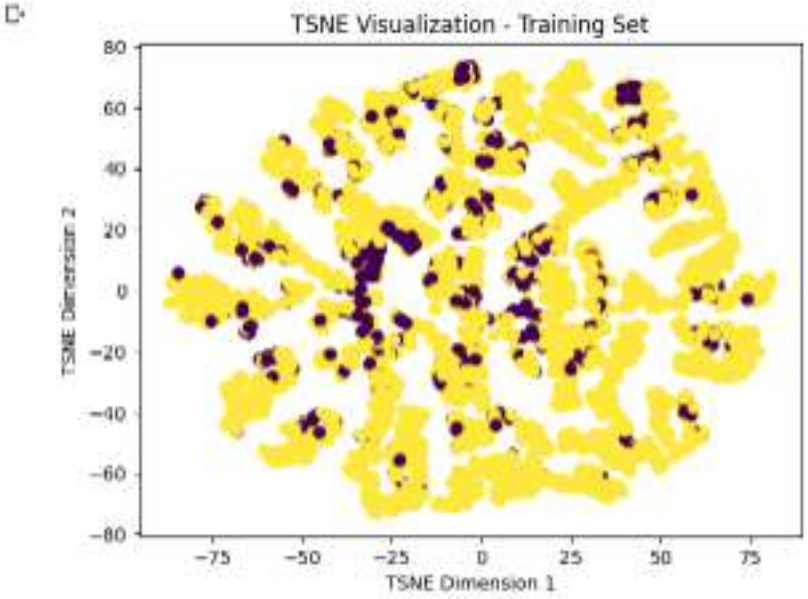


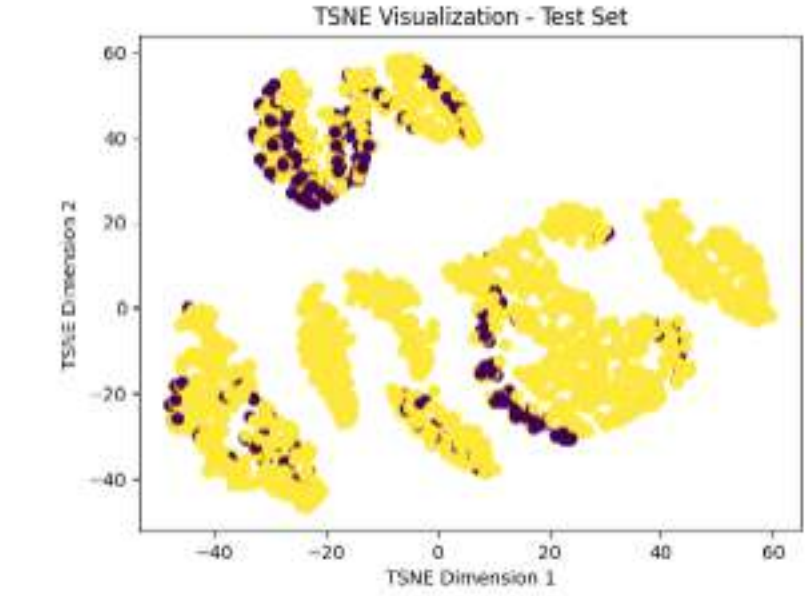


If most data samples in a t-SNE 2D graph are close together, it indicates similarity or clustering among those samples.

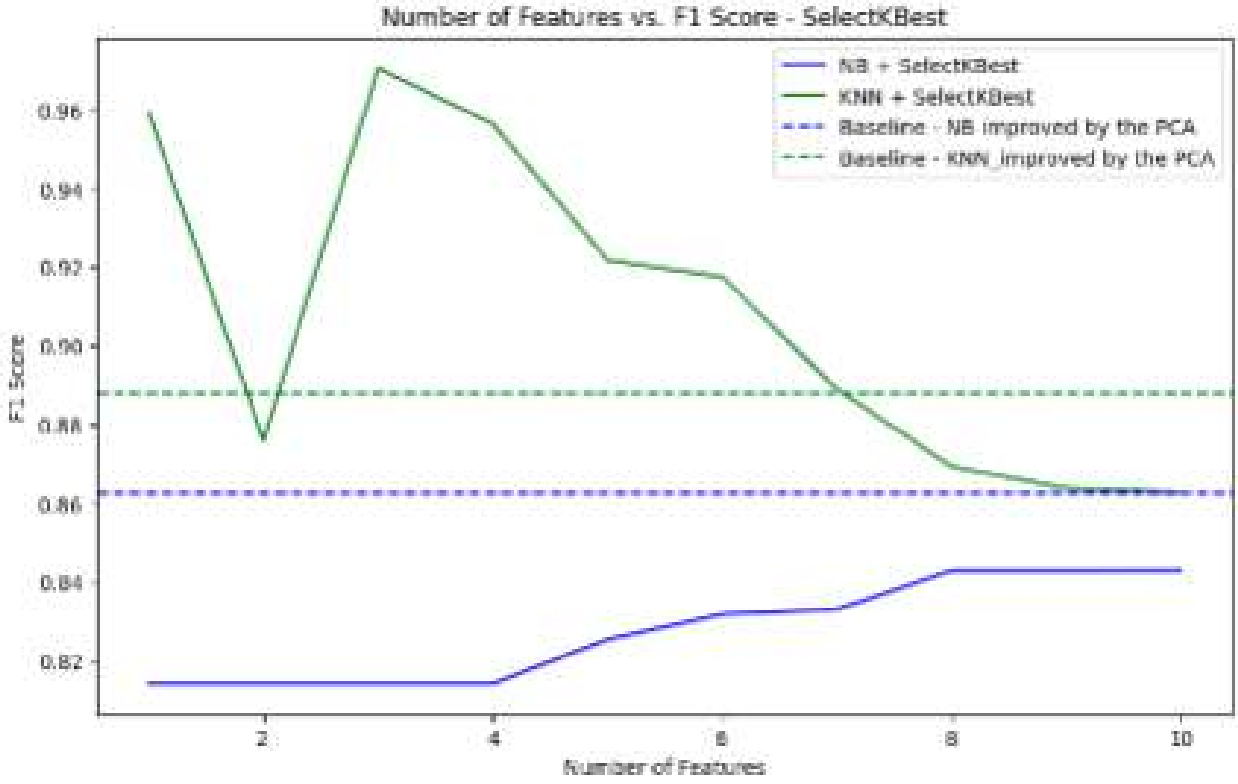


now we want to show the best improved performance





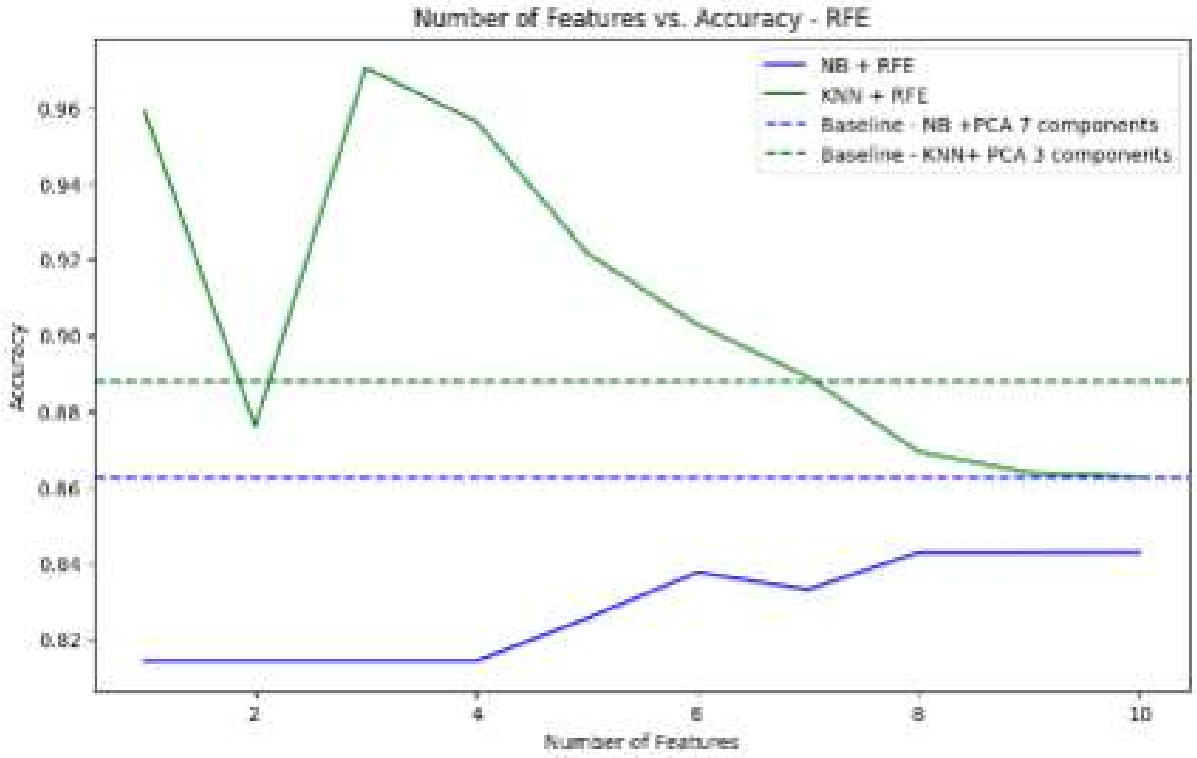
If most data samples in a t-SNE 2D graph are close together, it indicates similarity or clustering among those samples.



we couldnt get a better solution by filtering for the NB

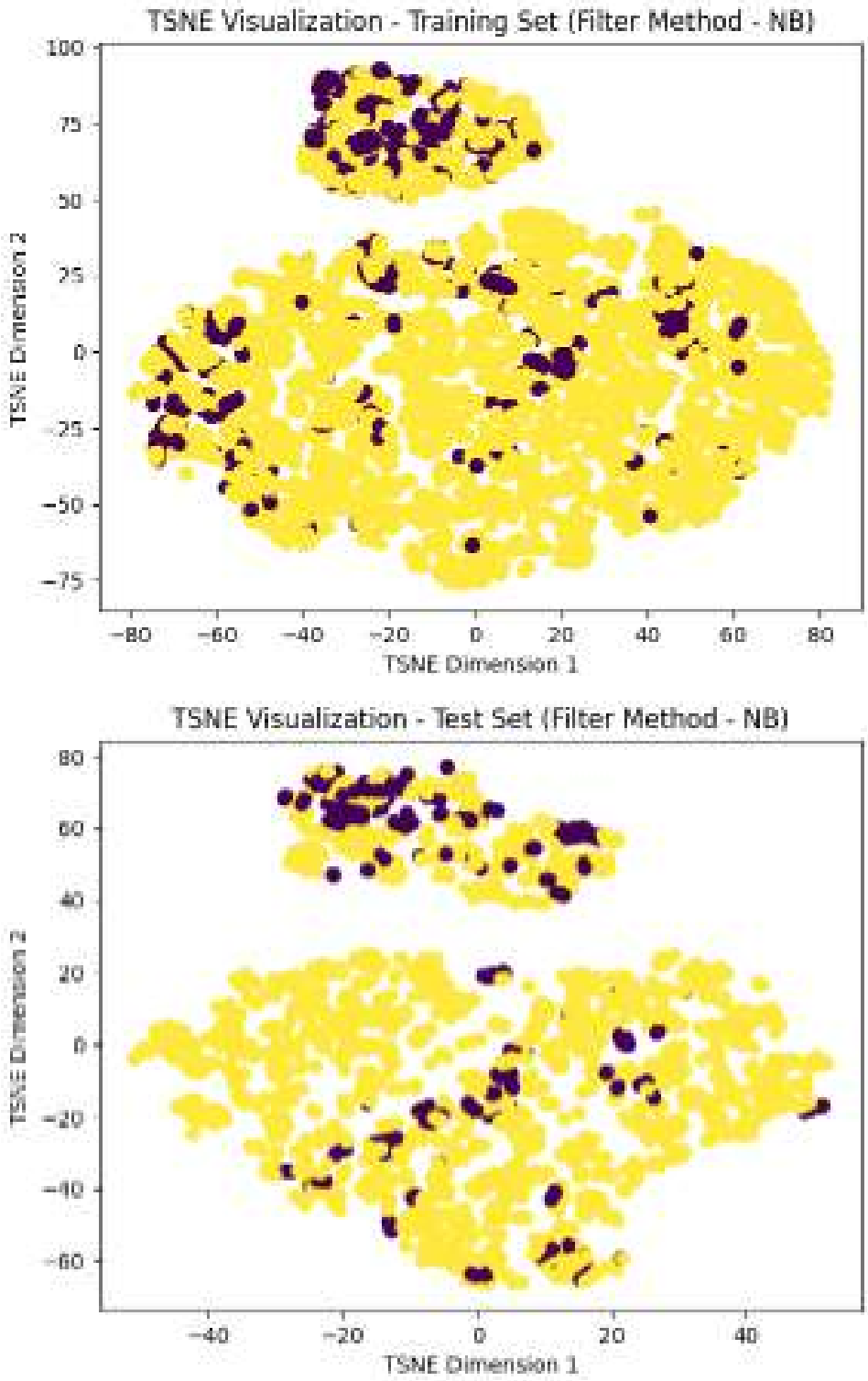
the din=mensionalit reduction is still better

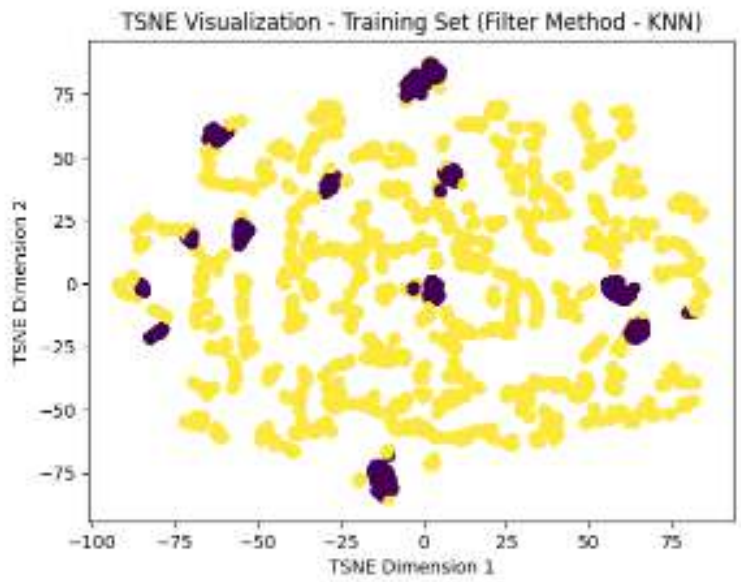
plt.show()

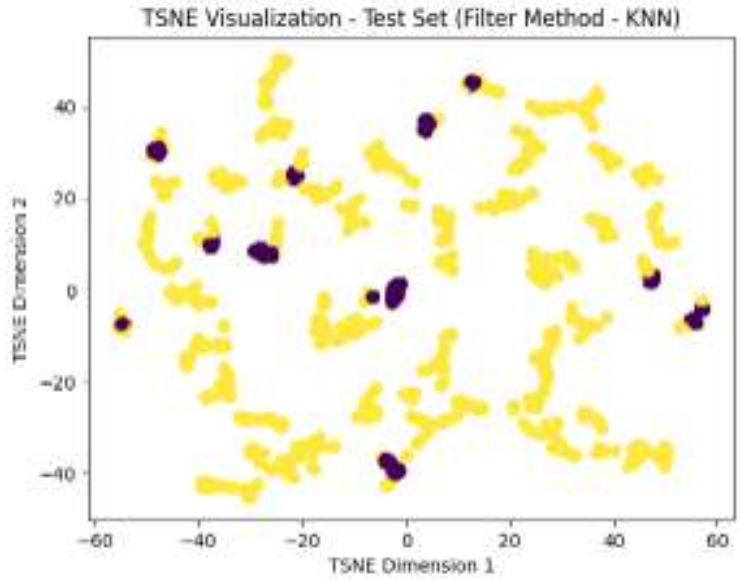


we couldnt get a better solution by wrapper for the NB

the din=mensionalit reduction is still better


TSNE Visualization - Training Set (Filter Method - NB)
TSNE Dimension 2
TSNE Dimension 1
100
75
50
25
0
−25
−50
−75
−80
−60
−40
−20
0
20
40
60
80
TSNE Visualization - Test Set (Filter Method - NB)
TSNE Dimension 2
TSNE Dimension 1
80
60
40
20
0
−20
−40
−60
−40
−20
0
20
40

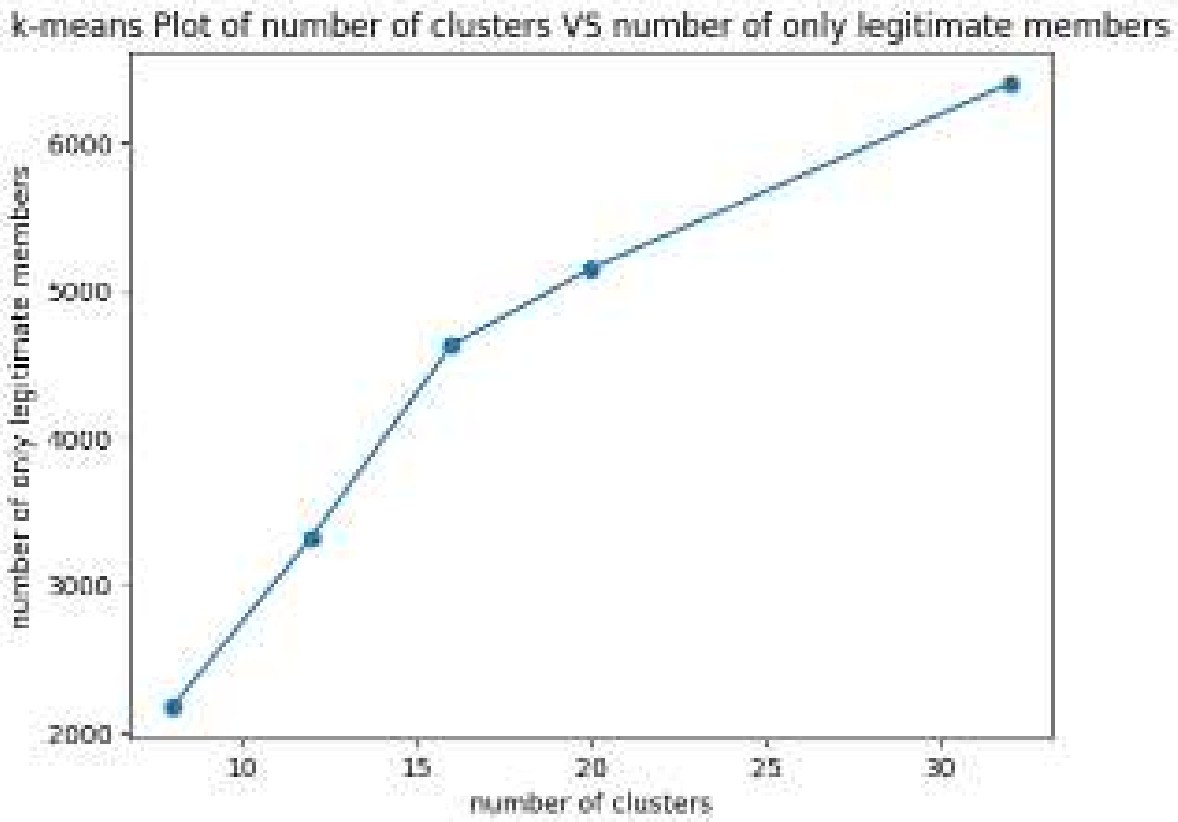




now obviously the groups of data samples are very easy to be clusttered from this t-sne plot

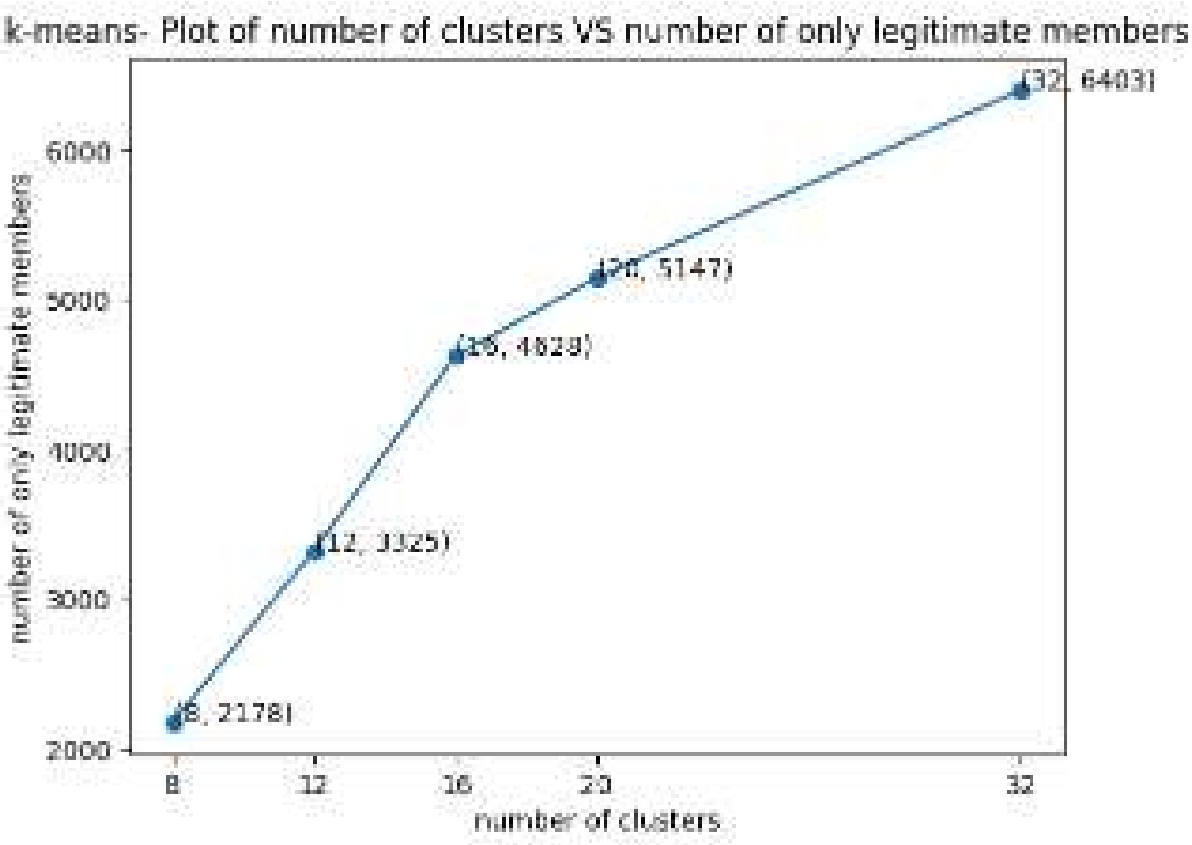
knn+filter

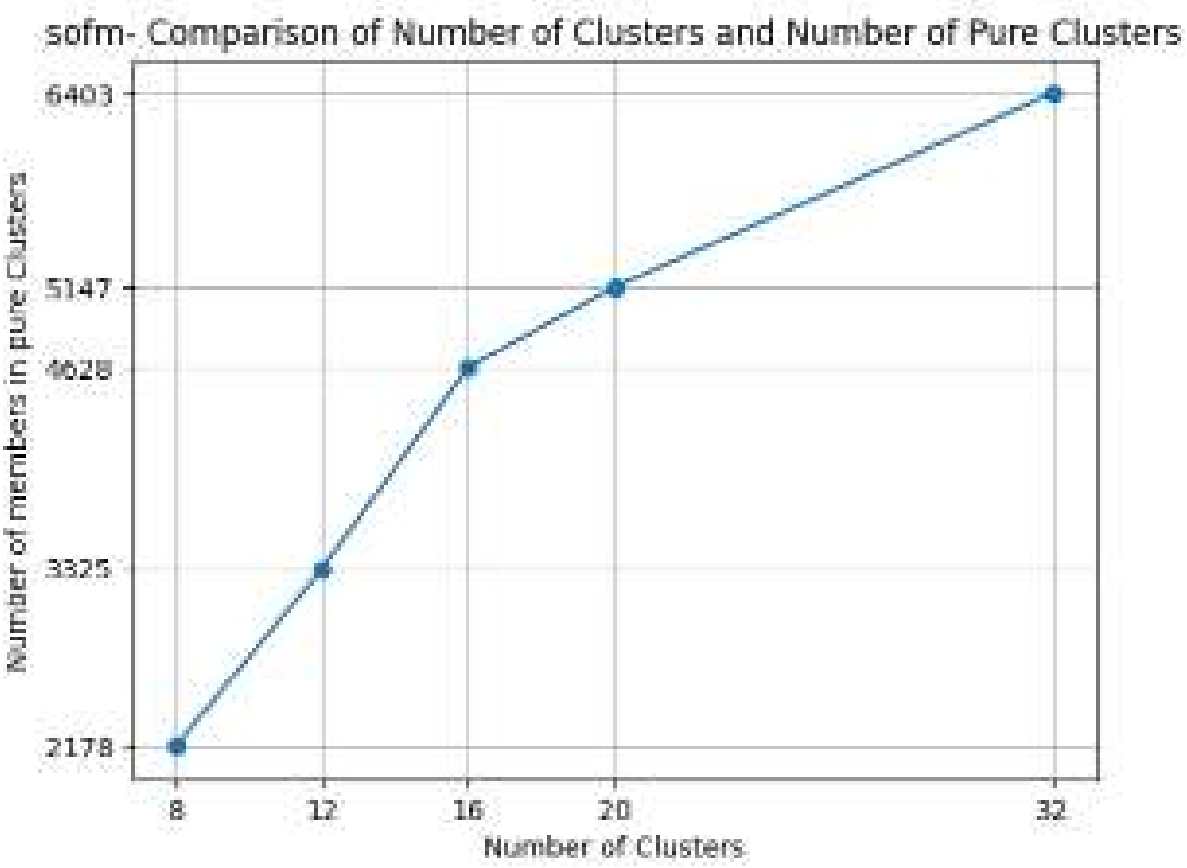
now we will go for clustring



- the more you increase the clusters the more it get smaller

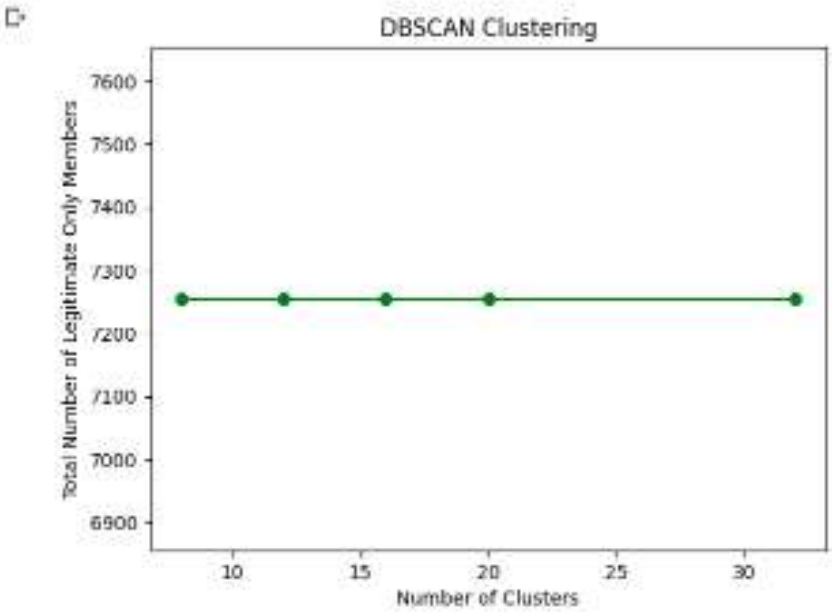
  and this will give you more pure clusters


k-means- Plot of number of clusters VS number of only legitimate members
number of only legitimate members
number of clusters
6000
5000
4000
3000
2000
8
12
16
20
32
(32, 6403)
(26, 5147)
(16, 4629)
(12, 3325)
(8, 2178)



- the more you increase the clusters the more it get smaller

  and this will give you more pure clusters



not accurate result for the dbscan

couldnt understand and need more analysis

A legitimate-only member refers to a data point that belongs to its own cluster, meaning it does not share a cluster with any other data point. The fact that the number of legitimate-only members remains constant at 7000 suggests that there might be a significant subset of the data that does not have close neighbors or forms separate clusters.

Q1 split data train the models without feature engineering

in this block we decided to do standard scaling for the data

after many practices on the training

and now the models performances are much better

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import numpy as np
# Load the dataset
dataset = pd.read_csv('MCSDatasetNEXTCONLab.csv')

# Create training dataset (days 0, 1, 2)
train_dataset = dataset[dataset['Day'].isin([0, 1, 2])].copy()

# Create test dataset (day 3)
test_dataset = dataset[dataset['Day'] == 3].copy()

# Remove irrelevant columns (ID and day)
train_dataset.drop(['ID', 'Day'], axis=1, inplace=True)
test_dataset.drop(['ID', 'Day'], axis=1, inplace=True)

# Separate features and target variable
X_train = train_dataset.drop('Ligitimacy', axis=1)
y_train = train_dataset['Ligitimacy']
X_test = test_dataset.drop('Ligitimacy', axis=1)
y_test = test_dataset['Ligitimacy']
print(dataset)
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
           ID   Latitude  Longitude  Day  Hour  Minute  Duration  RemainingTime  \
0           1  45.442142 -75.303369    1     4      13        40             40
1           1  45.442154 -75.304366    1     4      23        40             30
2           1  45.442104 -75.303963    1     4      33        40             20
3           1  45.441868 -75.303577    1     4      43        40             10
4           2  45.447727 -75.147722    2    15      49        30             30
...       ...       ...        ...  ...   ...     ...       ...            ...
14479  3999  45.445303 -75.165596    2     1      18        20             20
14480  3999  45.445574 -75.165168    2     1      28        20             10
14481  4000  45.436682 -75.152416    0    12      21        30             30
14482  4000  45.436978 -75.153278    0    12      31        30             20
14483  4000  45.436983 -75.153240    0    12      41        30             10

       Resources  Coverage  OnPeakHours  GridNumber  Ligitimacy
0              9        91            0      131380           1
1              9        91            0      131380           1
2              9        91            0      121996           1
3              9        91            0      121996           1
4              5        47            0      140784           1
...          ...       ...          ...         ...         ...
14479         10        80            0      131397           1
14480         10        80            0      131397           1
14481          4        63            0      122015           1
14482          4        63            0      122015           1
14483          4        63            0      122015           1

[14484 rows x 13 columns]
```

Naive Bayes Classifier: Train and predict using GaussianNB from sklearn. Evaluation Metrics for Naive Bayes: Calculate confusion matrix and F1 score. K-Nearest Neighbor Classifier: Train and predict using KNeighborsClassifier from sklearn. Evaluation Metrics for KNN: Calculate confusion matrix and F1 score.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, f1_score

# Initialize and train the Naive Bayes classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Make predictions on the test dataset
nb_predictions = nb_classifier.predict(X_test)

# Calculate the confusion matrix and F1 score for NB classifier
nb_confusion_matrix = confusion_matrix(y_test, nb_predictions)
nb_f1_score = f1_score(y_test, nb_predictions, average='weighted')

# Initialize and train the K-Nearest Neighbor classifier
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)

# Make predictions on the test dataset
knn_predictions = knn_classifier.predict(X_test)

# Calculate the confusion matrix and F1 score for KNN classifier
knn_confusion_matrix = confusion_matrix(y_test, knn_predictions)
knn_f1_score = f1_score(y_test, knn_predictions, average='weighted')
```

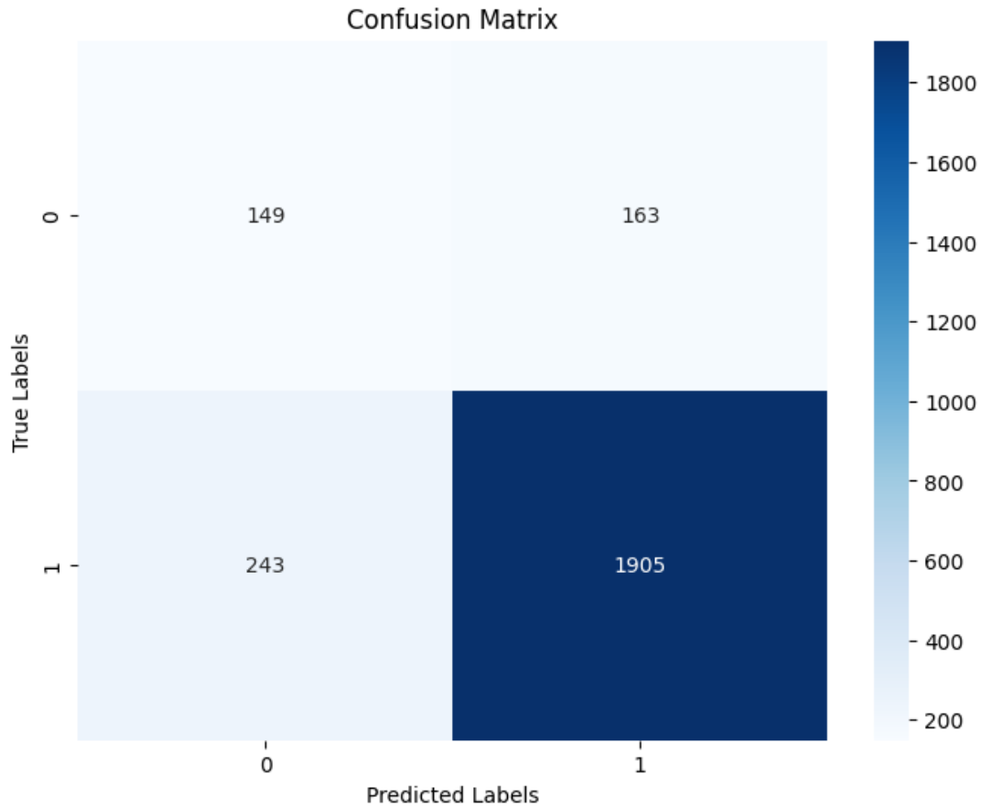
```
print("F1 Score - Naive Bayes Classifier: ", nb_f1_score)
print("F1 Score - K-Nearest Neighbor Classifier: ", knn_f1_score)
```

```
F1 Score - Naive Bayes Classifier:  0.8427708087025669
F1 Score - K-Nearest Neighbor Classifier:  0.8627163504968383
```

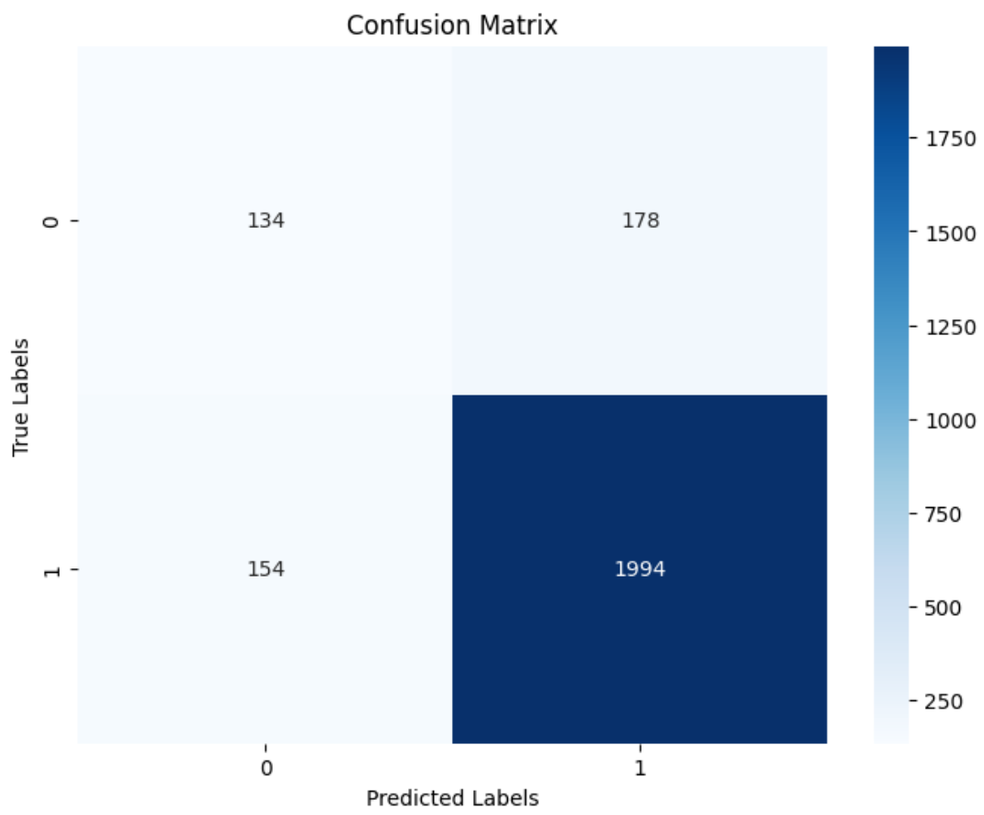
```
import numpy as np
import seaborn as sns

def plot_confusion_matrix(confusion_matrix):
    plt.figure(figsize=(8, 6))
    sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()
```

```
# Plot confusion matrix for Naive Bayes Classifier
print("Confusion Matrix - Naive Bayes Classifier:")
plot_confusion_matrix(nb_confusion_matrix)

print("\nConfusion Matrix - K-Nearest Neighbor Classifier:")
# Plot confusion matrix for K-Nearest Neighbor Classifier
plot_confusion_matrix(knn_confusion_matrix)
```

```
Confusion Matrix - Naive Bayes Classifier:
```




```
Confusion Matrix - K-Nearest Neighbor Classifier:
```

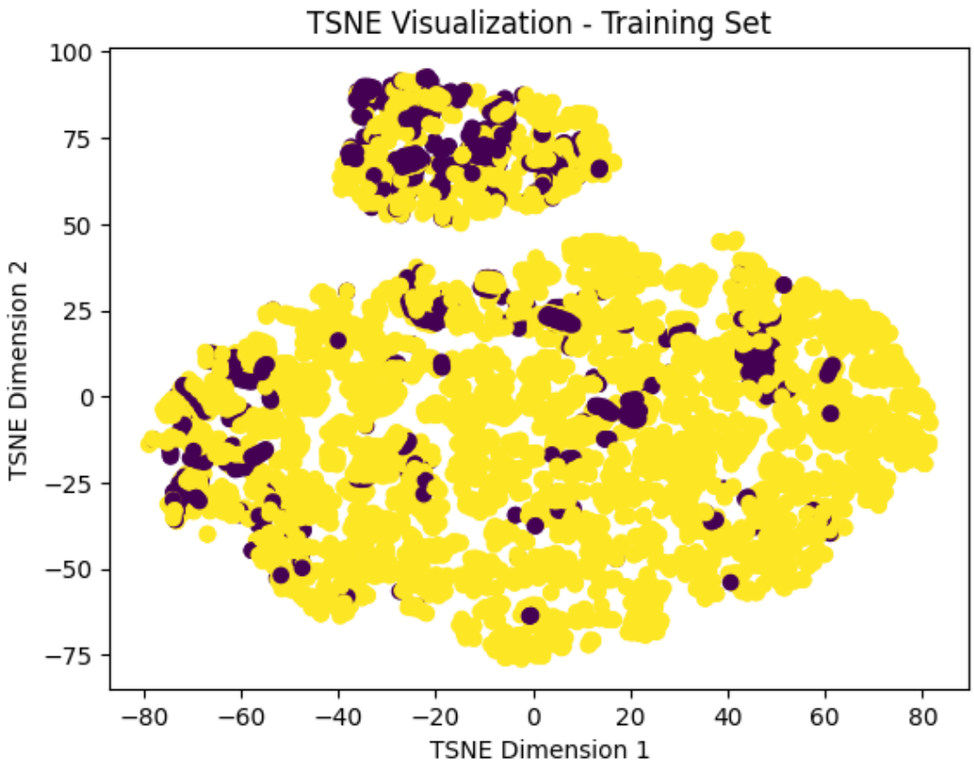


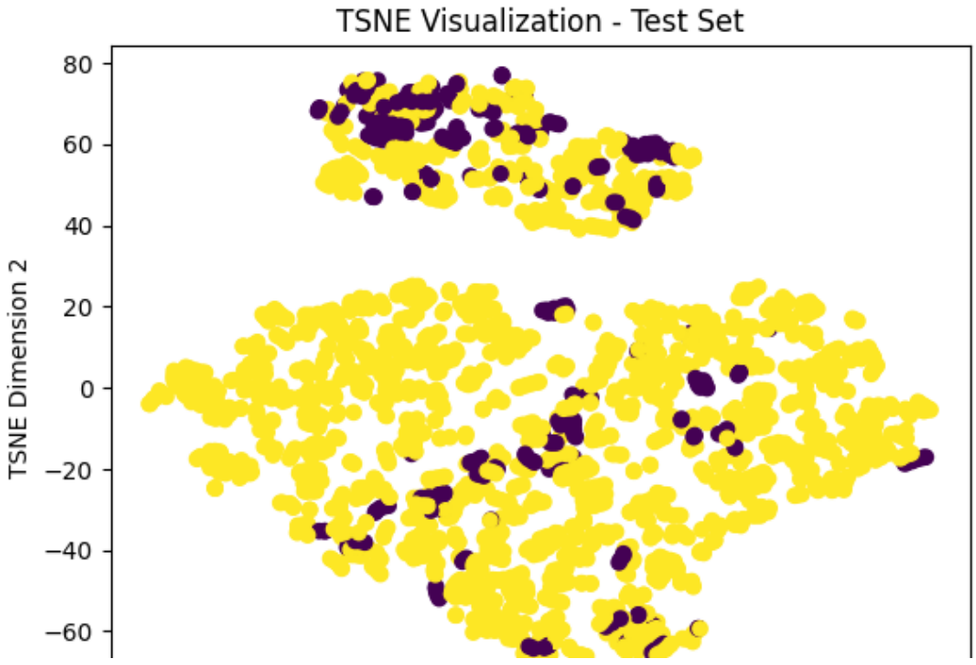
knn is performing better in predicting 1

NB is performing better in predicting 0

```
# Perform TSNE on the training set
tsne = TSNE(n_components=2, random_state=42)
X_train_tsne = tsne.fit_transform(X_train)

# Plot the TSNE visualization for the training set
```

```
# Plot the TSNE visualization for the training set
plt.scatter(X_train_tsne[:, 0], X_train_tsne[:, 1], c=y_train, cmap='viridis')
plt.title('TSNE Visualization - Training Set')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()

# Perform TSNE on the test set
X_test_tsne = tsne.fit_transform(X_test)

# Plot the TSNE visualization for the test set
plt.scatter(X_test_tsne[:, 0], X_test_tsne[:, 1], c=y_test, cmap='viridis')
plt.title('TSNE Visualization - Test Set')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()
```





If most data samples in a t-SNE 2D graph are close together, it indicates similarity or clustering among those samples.

Q2

now we will start the dimensionality reductoin first we will write the auto encoder function

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input
def create_autoencoder(input_dim, encoding_dim):
    input_layer = Input(shape=(input_dim,))
    encoder_layer1 = Dense(64, activation='relu')(input_layer)
    encoder_layer2 = Dense(32, activation='relu')(encoder_layer1)
    encoder_layer3 = Dense(encoding_dim, activation='relu')(encoder_layer2)
    decoder_layer1 = Dense(32, activation='relu')(encoder_layer3)
    decoder_layer2 = Dense(64, activation='relu')(decoder_layer1)
    decoder_layer3 = Dense(input_dim, activation='linear')(decoder_layer2)
    autoencoder = Model(inputs=input_layer, outputs=decoder_layer3)
    autoencoder.compile(optimizer='adam', loss='mse')
    return autoencoder
```

we will use PCA and AUto encoder

we will compare

pca==> knn pca==> NB

AE ==> knn AE ==> NB

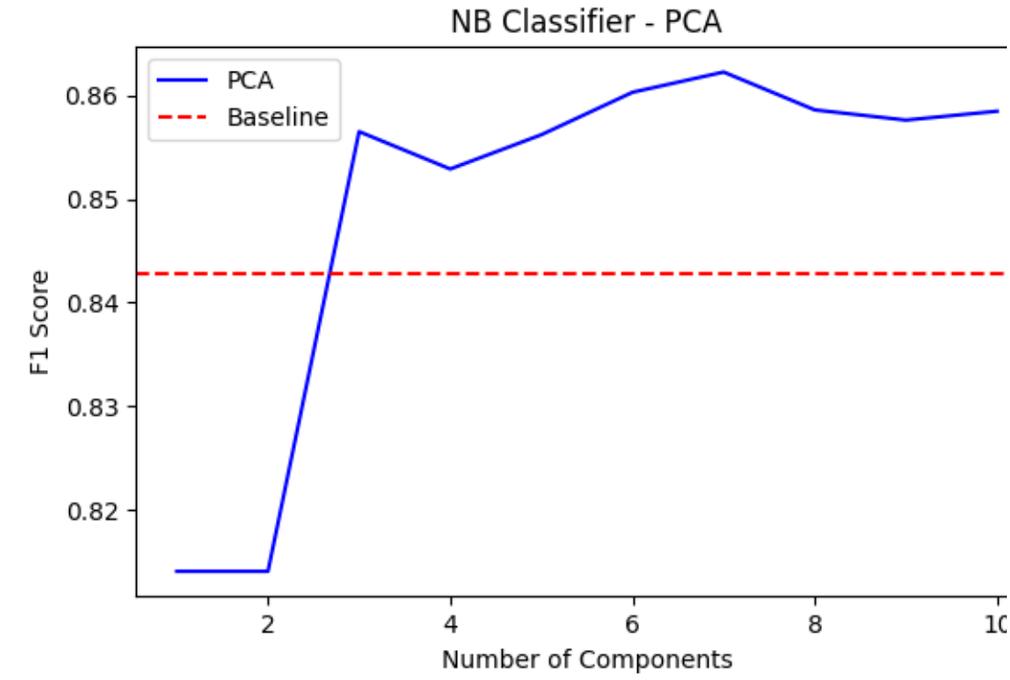
```
# Define the dimensions to be tested
dimensions = np.arange(1, X_train.shape[1]+1)

# Initialize empty lists to store F1 scores
nb_pca_f1_scores = []
knn_pca_f1_scores = []
nb_ae_f1_scores = []
knn_ae_f1_scores = []

# Perform PCA and calculate F1 scores for each dimension
for n in dimensions:
    # PCA
    pca = PCA(n_components=n, random_state=0)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    # NB with PCA
    nb_classifier_pca = GaussianNB()
    nb_classifier_pca.fit(X_train_pca, y_train)
    nb_predictions_pca = nb_classifier_pca.predict(X_test_pca)
    nb_f1_score_pca = f1_score(y_test, nb_predictions_pca, average='weighted')
    nb_pca_f1_scores.append(nb_f1_score_pca)

    # KNN with PCA
    knn_classifier_pca = KNeighborsClassifier()
    knn_classifier_pca.fit(X_train_pca, y_train)
    knn_predictions_pca = knn_classifier_pca.predict(X_test_pca)
    knn_f1_score_pca = f1_score(y_test, knn_predictions_pca, average='weighted')
    knn_pca_f1_scores.append(knn_f1_score_pca)

    # AE
    autoencoder = create_autoencoder(X_train.shape[1], n)
```

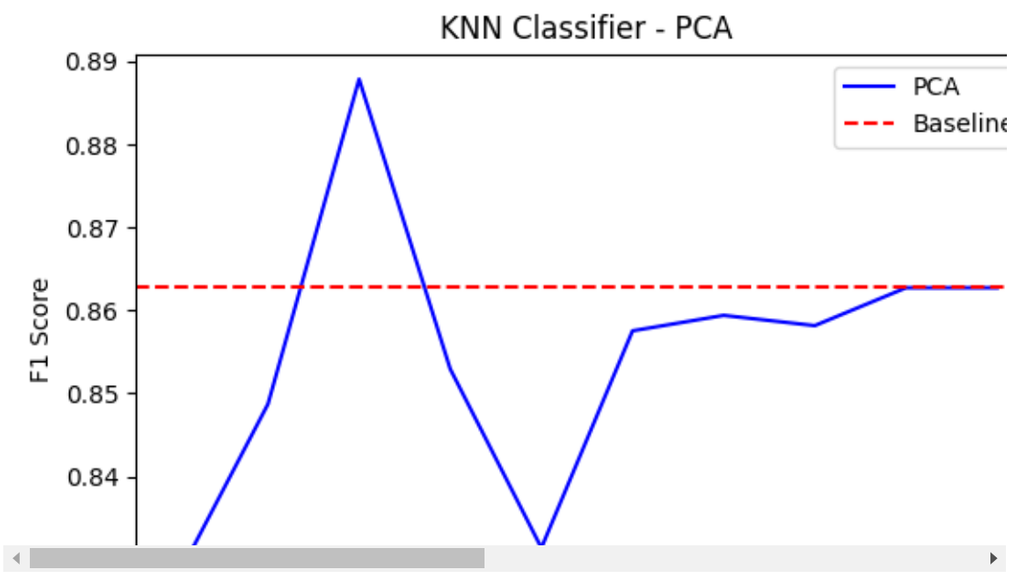
```
    autoencoder.fit(X_train, X_train, epochs=50, batch_size=16, verbose=0)
    encoder = Model(inputs=autoencoder.input, outputs=autoencoder.get_layer(index=3).out
    X_train_ae = encoder.predict(X_train)
    X_test_ae = encoder.predict(X_test)

    # NB with AE
    nb_classifier_ae = GaussianNB()
    nb_classifier_ae.fit(X_train_ae, y_train)
    nb_predictions_ae = nb_classifier_ae.predict(X_test_ae)
    nb_f1_score_ae = f1_score(y_test, nb_predictions_ae, average='weighted')
    nb_ae_f1_scores.append(nb_f1_score_ae)

    # KNN with AE
    knn_classifier_ae = KNeighborsClassifier()
    knn_classifier_ae.fit(X_train_ae, y_train)
    knn_predictions_ae = knn_classifier_ae.predict(X_test_ae)
    knn_f1_score_ae = f1_score(y_test, knn_predictions_ae, average='weighted')
    knn_ae_f1_scores.append(knn_f1_score_ae)
```

```
 227/227 [==============================] - 1s 2ms/step
 77/77 [==============================] - 0s 2ms/step
 227/227 [==============================] - 0s 1ms/step
 77/77 [==============================] - 0s 1ms/step
 227/227 [==============================] - 1s 2ms/step
 77/77 [==============================] - 0s 2ms/step
 227/227 [==============================] - 0s 1ms/step
 77/77 [==============================] - 0s 2ms/step
 227/227 [==============================] - 0s 1ms/step
 77/77 [==============================] - 0s 1ms/step
 227/227 [==============================] - 0s 1ms/step
 77/77 [==============================] - 0s 1ms/step
 227/227 [==============================] - 0s 1ms/step
 77/77 [==============================] - 0s 1ms/step
 227/227 [==============================] - 0s 1ms/step
 77/77 [==============================] - 0s 1ms/step
 227/227 [==============================] - 0s 1ms/step
 77/77 [==============================] - 0s 1ms/step
 227/227 [==============================] - 1s 3ms/step
 77/77 [==============================] - 0s 2ms/step
```

```
# Plotting the results
plt.figure(figsize=(12, 8))
# NB Classifier - PCA
plt.subplot(2, 2, 1)
plt.plot(dimensions, nb_pca_f1_scores, 'b-', label='PCA')
plt.axhline(y=nb_f1_score, color='r', linestyle='--', label='Baseline')
plt.xlabel('Number of Components')
plt.ylabel('F1 Score')
plt.title('NB Classifier - PCA')
plt.legend()

# NB Classifier - Autoencoder
plt.subplot(2, 2, 2)
plt.plot(dimensions, nb_ae_f1_scores, 'g-', label='Autoencoder')
plt.axhline(y=nb_f1_score, color='r', linestyle='--', label='Baseline')
plt.xlabel('Number of Components')
plt.ylabel('F1 Score')
plt.title('NB Classifier - Autoencoder')
plt.legend()

# KNN Classifier - PCA
plt.subplot(2, 2, 3)
plt.plot(dimensions, knn_pca_f1_scores, 'b-', label='PCA')
plt.axhline(y=knn_f1_score, color='r', linestyle='--', label='Baseline')
plt.xlabel('Number of Components')
plt.ylabel('F1 Score')
plt.title('KNN Classifier - PCA')
```

```
plt.legend()

# KNN Classifier - Autoencoder
plt.subplot(2, 2, 4)
plt.plot(dimensions, knn_ae_f1_scores, 'g-', label='Autoencoder')
plt.axhline(y=knn_f1_score, color='r', linestyle='--', label='Baseline')
plt.xlabel('Number of Components')
plt.ylabel('F1 Score')
plt.title('KNN Classifier - Autoencoder')
plt.legend()

plt.tight_layout()
plt.show()
```





now we want to show the best improved performance

```
# Determine the best dimensionality reduction technique based on F1 scores for nb gaussi
best_f1_score_pca_nb = max(nb_pca_f1_scores)
best_f1_score_ae_nb = max(nb_ae_f1_scores)

if best_f1_score_pca_nb >= best_f1_score_ae_nb:
    best_technique = 'PCA'
```

```
    best_dimension_nb = dimensions[np.argmax(nb_pca_f1_scores)]
    best_f1_score_nb = best_f1_score_pca_nb
else:
    best_technique = 'AE'
    best_dimension_nb = dimensions[np.argmax(nb_ae_f1_scores)]
    best_f1_score_nb = best_f1_score_ae_nb

print("Best Dimensionality Reduction Technique for NB gaussian: ", best_technique)
print("Best Dimension: ", best_dimension_nb)
print("Best F1 Score: ", best_f1_score_nb)
```

```
Best Dimensionality Reduction Technique for NB gaussian:  PCA
Best Dimension:  7
Best F1 Score:  0.8622340790236424
```

NB ==> PCA

above here PCA for the NB but i think the 3 dimension will give more general model

```
# Determine the best dimensionality reduction technique based on F1 scores for KNN
best_f1_score_pca_knn = max(knn_pca_f1_scores)
best_f1_score_ae_knn = max(knn_ae_f1_scores)

if best_f1_score_pca_knn >= best_f1_score_ae_knn:
    best_technique_knn = 'PCA'
    best_dimension_knn = dimensions[np.argmax(knn_pca_f1_scores)]
    best_f1_score_knn = best_f1_score_pca_knn
else:
    best_technique_knn = 'AE'
    best_dimension_knn = dimensions[np.argmax(knn_ae_f1_scores)]
    best_f1_score_knn = best_f1_score_ae_knn

print("Best Dimensionality Reduction Technique for KNN: ", best_technique_knn)
print("Best Dimension for KNN: ", best_dimension_knn)
print("Best F1 Score for KNN: ", best_f1_score_knn)
```

```
Best Dimensionality Reduction Technique for KNN:  PCA
Best Dimension for KNN:  3
Best F1 Score for KNN:  0.8878507637570797
```

knn ==> PCA

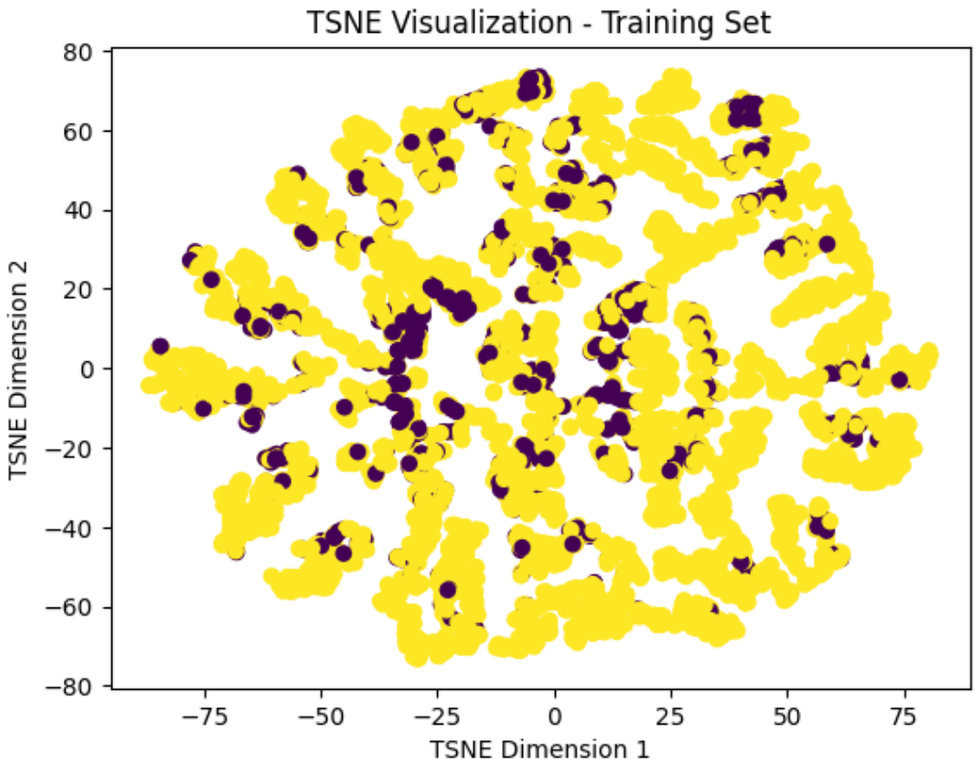
and only 3 features ==>

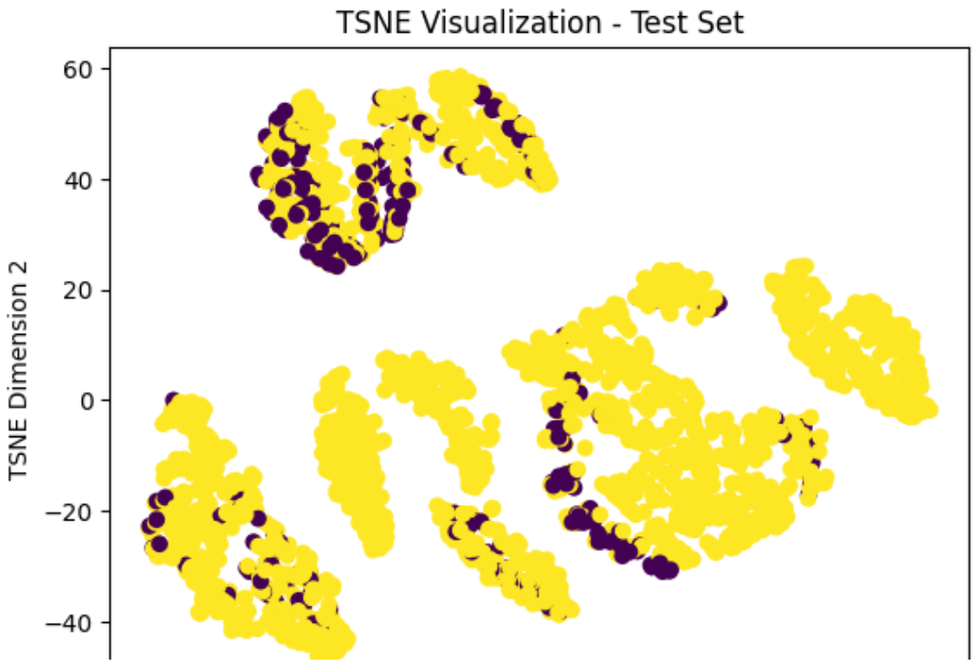
not complex

now we will create t-sne plots after the reduction we hope the data are more able to be in clusters

```
from sklearn.manifold import TSNE

# Perform TSNE on the best-performing technique (PCA) for the training set
pca_train = PCA(n_components=best_dimension_knn)
X_train_pca = pca_train.fit_transform(X_train)

# Perform TSNE on the best-performing technique (PCA) for the test set
pca_test = PCA(n_components=best_dimension_knn)
X_test_pca = pca_test.fit_transform(X_test)

# Perform TSNE on the training set
tsne_train = TSNE(n_components=2, random_state=0)
```

```
X_train_tsne = tsne_train.fit_transform(X_train_pca)

# Plot the TSNE visualization for the training set
plt.scatter(X_train_tsne[:, 0], X_train_tsne[:, 1], c=y_train, cmap='viridis')
plt.title('TSNE Visualization - Training Set')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()

# Perform TSNE on the test set
tsne_test = TSNE(n_components=2, random_state=0)
X_test_tsne = tsne_test.fit_transform(X_test_pca)

# Plot the TSNE visualization for the test set
plt.scatter(X_test_tsne[:, 0], X_test_tsne[:, 1], c=y_test, cmap='viridis')
plt.title('TSNE Visualization - Test Set')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()
```

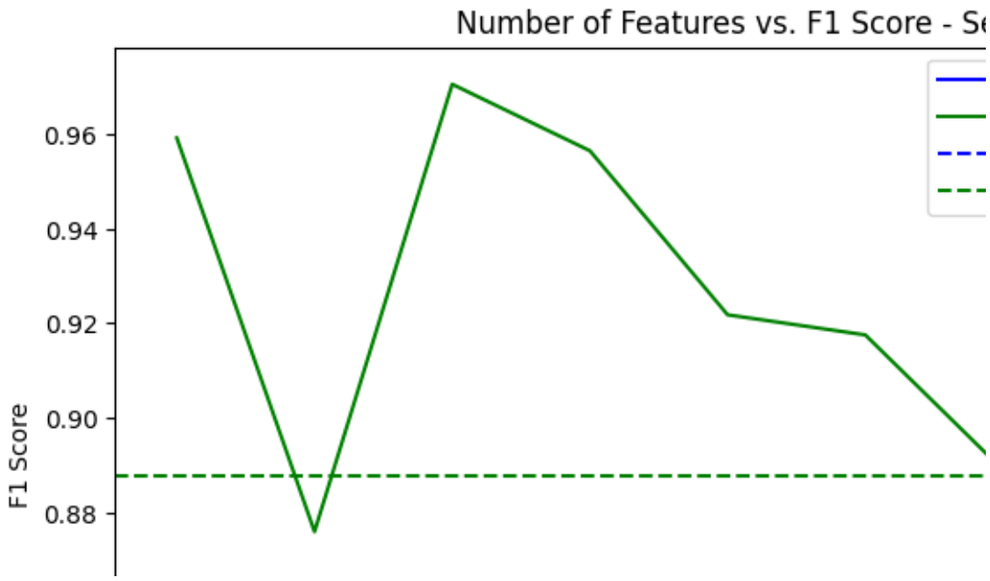




If most data samples in a t-SNE 2D graph are close together, it indicates similarity or clustering among those samples.

Q3 we will try feature selection and will compare between filtring technique and wrapper technique

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif

# Initialize empty lists to store F1 scores
nb_filter_f1_scores = []
knn_filter_f1_scores = []

# Define the number of features to be tested
num_features = np.arange(1, X_train.shape[1]+1)

# Perform feature selection using SelectKBest and mutual information
for k in num_features:
    selector = SelectKBest(mutual_info_classif, k=int(k))
    X_train_filtered = selector.fit_transform(X_train, y_train)
    X_test_filtered = selector.transform(X_test)

    # NB with filtered features
    nb_classifier_filter = GaussianNB()
    nb_classifier_filter.fit(X_train_filtered, y_train)
    nb_predictions_filter = nb_classifier_filter.predict(X_test_filtered)
    nb_f1_score_filter = f1_score(y_test, nb_predictions_filter, average='weighted')
    nb_filter_f1_scores.append(nb_f1_score_filter)

    # KNN with filtered features
    knn_classifier_filter = KNeighborsClassifier()
    knn_classifier_filter.fit(X_train_filtered, y_train)
    knn_predictions_filter = knn_classifier_filter.predict(X_test_filtered)
    knn_f1_score_filter = f1_score(y_test, knn_predictions_filter, average='weighted')
    knn_filter_f1_scores.append(knn_f1_score_filter)

# Plot the number of features vs. F1 score for NB and KNN with SelectKBest and mutual in
plt.figure(figsize=(10, 6))
plt.plot(num_features, nb_filter_f1_scores, 'b', label='NB + SelectKBest')
plt.plot(num_features, knn_filter_f1_scores, 'g', label='KNN + SelectKBest')
plt.axhline(y=best_f1_score_pca_nb, color='b', linestyle='--', label='Baseline - NB impr
plt.axhline(y=best_f1_score_pca_knn, color='g', linestyle='--', label='Baseline - KNN_im
plt.xlabel('Number of Features')
plt.ylabel('F1 Score')
plt.title('Number of Features vs. F1 Score - SelectKBest')
plt.legend()
plt.show()
```



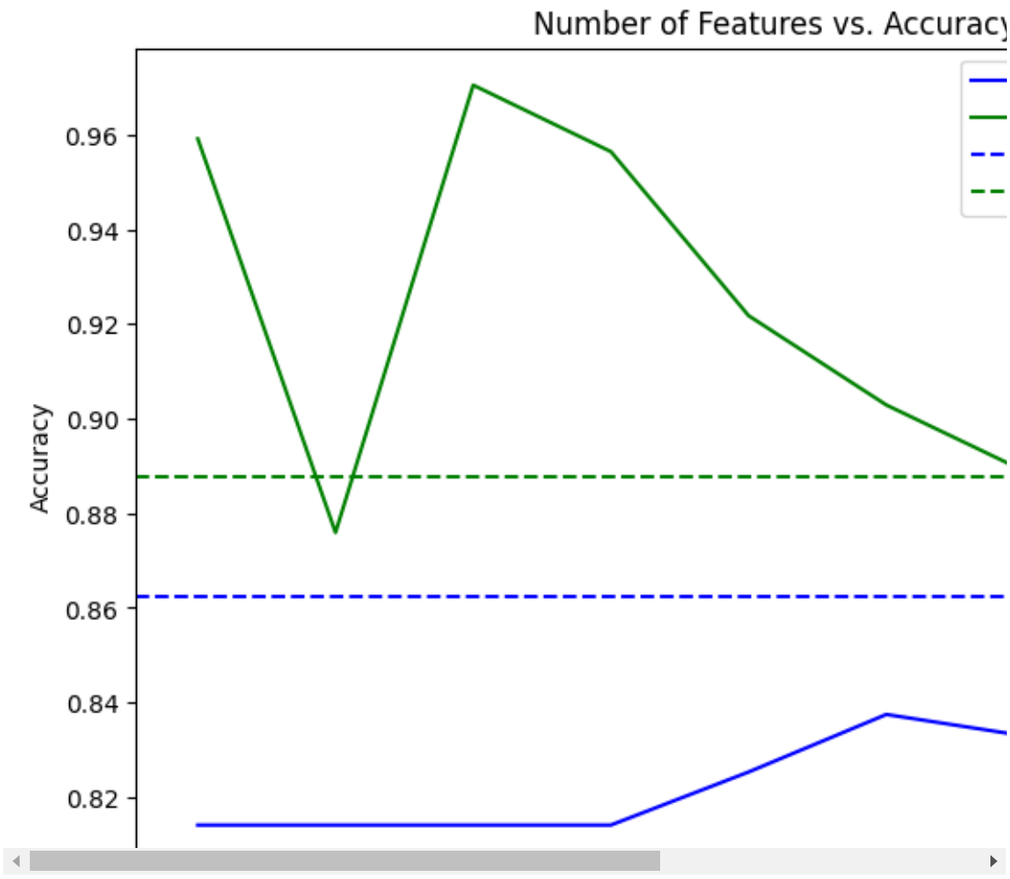
we couldnt get a better solution by filtering for the NB

the din=mensionalit reduction is still better

the knn had a better performance but becarfull from the over fitting

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score

# Initialize empty lists to store weighted F1 scores
nb_wrapper_f1_scores = []
knn_wrapper_f1_scores = []

# Define the range of number of features to test
num_features = np.arange(1, X_train.shape[1]+1)

# Perform feature selection using Recursive Feature Elimination (RFE) and calculate weig
for k in num_features:
    # RFE for NB classifier
    selector_nb = RFE(RandomForestClassifier(), n_features_to_select=k)
    X_train_selected_nb = selector_nb.fit_transform(X_train, y_train)
    X_test_selected_nb = selector_nb.transform(X_test)

    # NB classifier with RFE-selected features
    nb_classifier_selected = GaussianNB()
    nb_classifier_selected.fit(X_train_selected_nb, y_train)
    nb_predictions_selected = nb_classifier_selected.predict(X_test_selected_nb)
    nb_f1_score_selected = f1_score(y_test, nb_predictions_selected, average='weighted')
    nb_wrapper_f1_scores.append(nb_f1_score_selected)

    # RFE for KNN classifier
    selector_knn = RFE(RandomForestClassifier(), n_features_to_select=k)
    X_train_selected_knn = selector_knn.fit_transform(X_train, y_train)
    X_test_selected_knn = selector_knn.transform(X_test)

    # KNN classifier with RFE-selected features
    knn_classifier_selected = KNeighborsClassifier()
    knn_classifier_selected.fit(X_train_selected_knn, y_train)
    knn_predictions_selected = knn_classifier_selected.predict(X_test_selected_knn)
    knn_f1_score_selected = f1_score(y_test, knn_predictions_selected, average='weighted
    knn_wrapper_f1_scores.append(knn_f1_score_selected)
```

```
# Plot the number of features vs. accuracy for NB and KNN with RFE
plt.figure(figsize=(10, 6))
plt.plot(num_features, nb_wrapper_f1_scores, 'b', label='NB + RFE')
plt.plot(num_features, knn_wrapper_f1_scores, 'g', label='KNN + RFE')
plt.axhline(y=best_f1_score_pca_nb, color='b', linestyle='--', label='Baseline - NB +PCA
plt.axhline(y=best_f1_score_pca_knn, color='g', linestyle='--', label='Baseline - KNN+ P
plt.xlabel('Number of Features')
plt.ylabel('Accuracy')
plt.title('Number of Features vs. Accuracy - RFE')
plt.legend()
plt.show()
```



we couldnt get a better solution by wrapper for the NB

the din=mensionalit reduction is still better

the knn had a better performance but becarfull from the over fitting

```
# Select the best method and number of features for NB
best_method_nb = 'Filter' if max(nb_filter_f1_scores) >= max(nb_wrapper_f1_scores) else
best_features_nb = num_features[np.argmax(nb_filter_f1_scores)] if best_method_nb == 'Fi

print("Best Feature Selection Method for NB: ", best_method_nb)
print("Best Number of Features for NB: ", best_features_nb)
```

```
Best Feature Selection Method for NB:  Filter
Best Number of Features for NB:  10
```

the feature selection didnt n=benefit us in case of NB

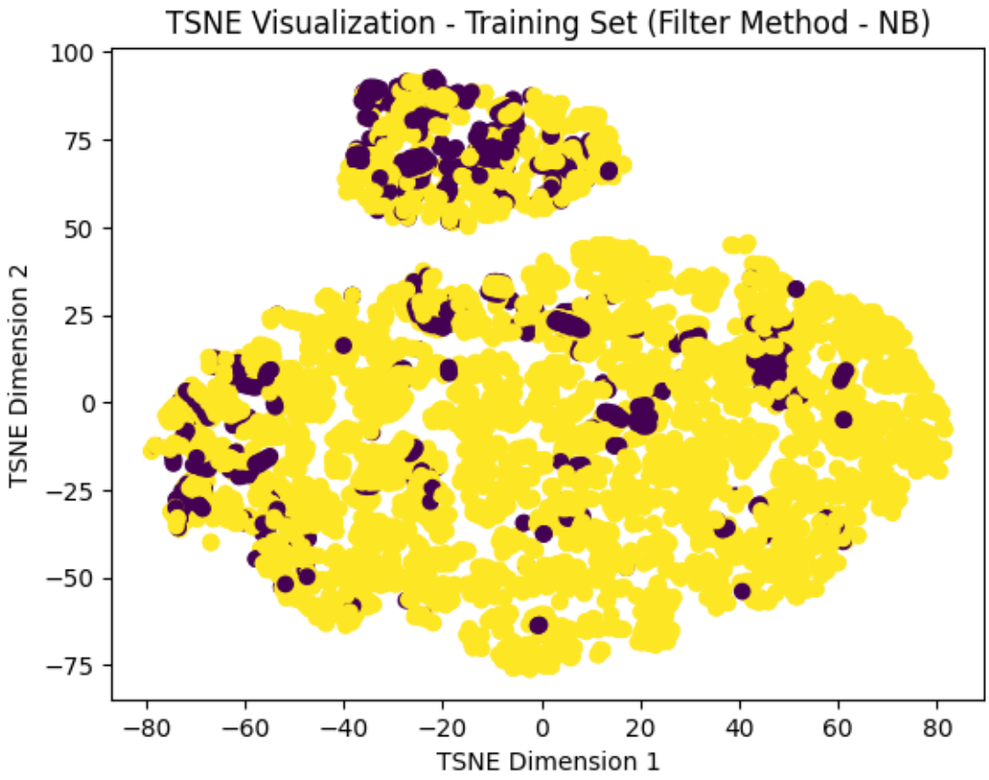
```
# Select the best method and number of features for KNN
best_method_knn = 'Filter' if max(knn_filter_f1_scores) >= max(knn_wrapper_f1_scores) el
best_features_knn = num_features[np.argmax(knn_filter_f1_scores)] if best_method_knn ==

print("Best Feature Selection Method for KNN: ", best_method_knn)
print("Best Number of Features for KNN: ", best_features_knn)
```

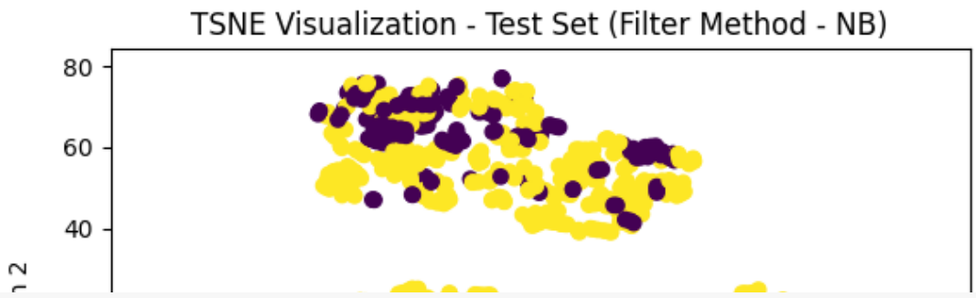
```
Best Feature Selection Method for KNN:  Filter
Best Number of Features for KNN:  3
```

the filtering benifetted us also the wrapping and got even better than the dimensionality reduction

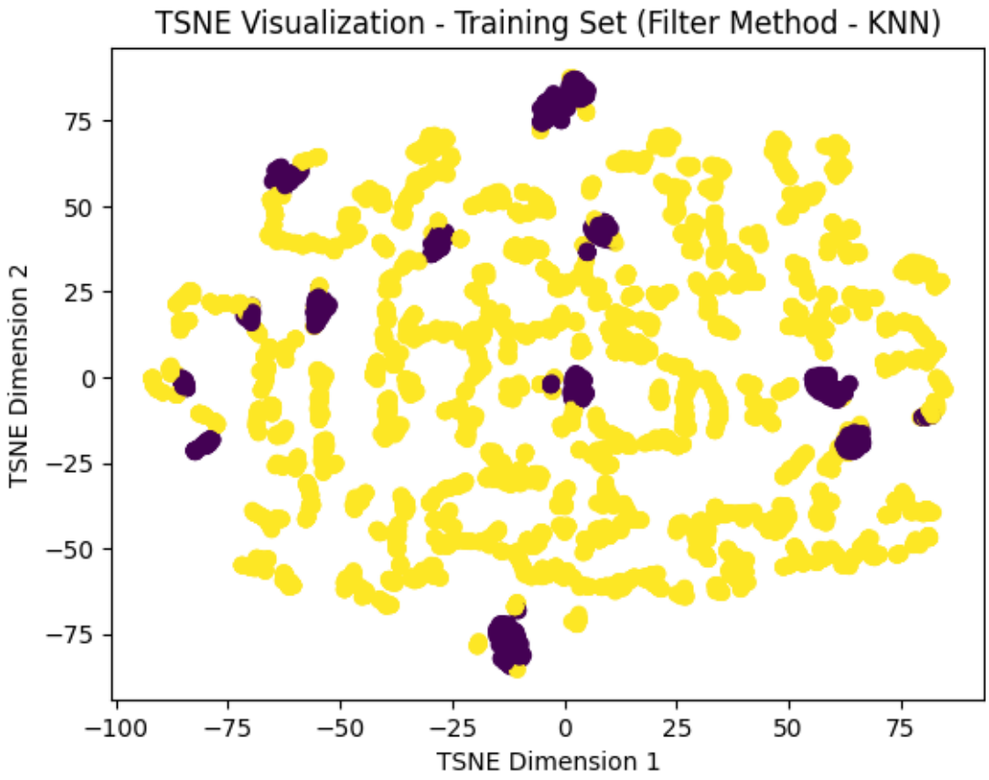
but 97% may be overfitting

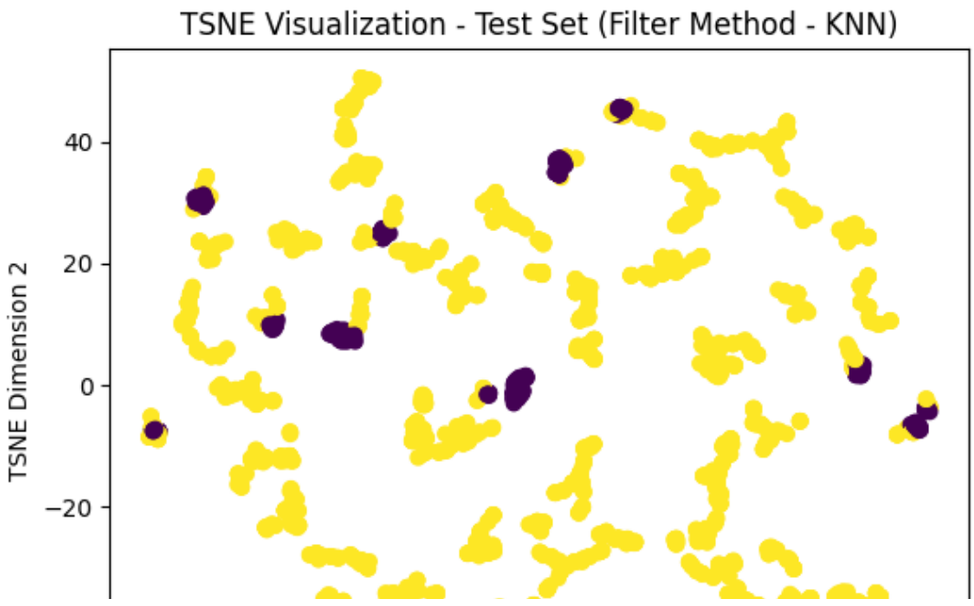
```
from sklearn.manifold import TSNE

# Perform feature selection using SelectKBest with the best number of features for NB
selector_nb = SelectKBest(mutual_info_classif, k=best_features_nb)
X_train_selected_nb = selector_nb.fit_transform(X_train, y_train)
X_test_selected_nb = selector_nb.transform(X_test)

# Perform TSNE on the training set using the selected features
tsne = TSNE(n_components=2, random_state=0)
X_train_tsne = tsne.fit_transform(X_train_selected_nb)

# Plot the TSNE visualization for the training set
plt.scatter(X_train_tsne[:, 0], X_train_tsne[:, 1], c=y_train, cmap='viridis')
plt.title('TSNE Visualization - Training Set (Filter Method - NB)')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()

# Perform TSNE on the test set using the selected features
X_test_tsne = tsne.fit_transform(X_test_selected_nb)

# Plot the TSNE visualization for the test set
plt.scatter(X_test_tsne[:, 0], X_test_tsne[:, 1], c=y_test, cmap='viridis')
plt.title('TSNE Visualization - Test Set (Filter Method - NB)')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()
```






```
from sklearn.manifold import TSNE

# Perform feature selection using SelectKBest with the best number of features for KNN
selector_knn = SelectKBest(mutual_info_classif, k=best_features_knn)
X_train_selected_knn = selector_knn.fit_transform(X_train, y_train)
X_test_selected_knn = selector_knn.transform(X_test)

# Perform TSNE on the training set using the selected features
tsne = TSNE(n_components=2, random_state=0)
X_train_tsne = tsne.fit_transform(X_train_selected_knn)

# Plot the TSNE visualization for the training set
plt.scatter(X_train_tsne[:, 0], X_train_tsne[:, 1], c=y_train, cmap='viridis')
plt.title('TSNE Visualization - Training Set (Filter Method - KNN)')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()

# Perform TSNE on the test set using the selected features
X_test_tsne = tsne.fit_transform(X_test_selected_knn)

# Plot the TSNE visualization for the test set
plt.scatter(X_test_tsne[:, 0], X_test_tsne[:, 1], c=y_test, cmap='viridis')
plt.title('TSNE Visualization - Test Set (Filter Method - KNN)')
plt.xlabel('TSNE Dimension 1')
plt.ylabel('TSNE Dimension 2')
plt.show()
```





now obviously the groups of data samples are very easy to be clusttered from this t-sne plot

knn+filter

now we will go for clustring

Q4 we need to compare between clustring

k-means

SOFM

DBSCAN

get data

```
#get the data
df= pd.read_csv('MCSDatasetNEXTCONLab.csv')
data=df[df['Day'].isin([0,1,2,3])]                                    # this data i
```

```
dat=data[['Latitude','Longitude','Ligitimacy']]                                  # this data 9
X = dat[['Latitude','Longitude']]                                                #this data 97
only_ligitimancy =dat[dat.Ligitimacy==1][['Latitude','Longitude']]
```

clustering

Define the number of clusters to use

find members in each cluster

Get the labels for the fit and predict data

cluster number members in each cluster inside fit data members in each cluster inside predict data

```
# Define the number of clusters to use
num_clusters = [8, 12, 16, 20, 32]
# create a useful variables
mem_cluster_fit=[]
mem_cluster_predict=[]


# create a data frame
# create an empty DataFrame with labels of each data (fit & predict)
df_label = pd.DataFrame(columns=["cluster number ","fit_labels", "predict_labels"])
# create an empty DataFrame with member inside each cluster for data (fit & predict)
df = pd.DataFrame(columns=["cluster number ","members in each cluster inside fit data","

#model with n cluster and find members in each cluster
for n in num_clusters:
    # Fit the data using k-means clustering
    kmeans = KMeans(n_clusters=n, random_state=0).fit(X)
    # Get the labels for the fit and predict data
    fit_labels = kmeans.predict(X)
    mem_cluster_fit=[len(fit_labels[fit_labels == i]) for i in range(n)]
    predict_labels = kmeans.predict(only_ligitimancy)
    mem_cluster_predict=[len(predict_labels[predict_labels == i]) for i in range(n)]
    df = df.append({"cluster number ": n,"members in each cluster inside fit data":mem_c
    df_label = df_label.append({"cluster number ": n,"fit_labels":fit_labels, "predict_l


df
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWa
  warnings.warn(
<ipython-input-28-ea47faff6b7c>:23: FutureWarning: The frame.append
  df = df.append({"cluster number ": n,"members in each cluster ins
<ipython-input-28-ea47faff6b7c>:24: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number ": n,"fit_labels":fit
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWa
  warnings.warn(
<ipython-input-28-ea47faff6b7c>:23: FutureWarning: The frame.append
  df = df.append({"cluster number ": n,"members in each cluster ins
<ipython-input-28-ea47faff6b7c>:24: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number ": n,"fit_labels":fit
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWa
  warnings.warn(
<ipython-input-28-ea47faff6b7c>:23: FutureWarning: The frame.append
  df = df.append({"cluster number ": n,"members in each cluster ins
<ipython-input-28-ea47faff6b7c>:24: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number ": n,"fit_labels":fit
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWa
  warnings.warn(
<ipython-input-28-ea47faff6b7c>:23: FutureWarning: The frame.append
```

now we will compare between members in each cluster inside fit data

```
  warnings.warn(
```

```
# compare between members in each cluster inside fit data (legitimate and fake) and only
# calcutate the number of only legitimate member inside each cluster
n = [8, 12, 16, 20, 32]
f=df['members in each cluster inside fit data']
p=df['members in each cluster inside predict data']
new=[]

for i in range(len(n)):
    sum=0
    for j in range(0,len(f[i])):
        if f[i][j]== p[i][j]:
            sum+=f[i][j]
    new.append(sum)
new
```

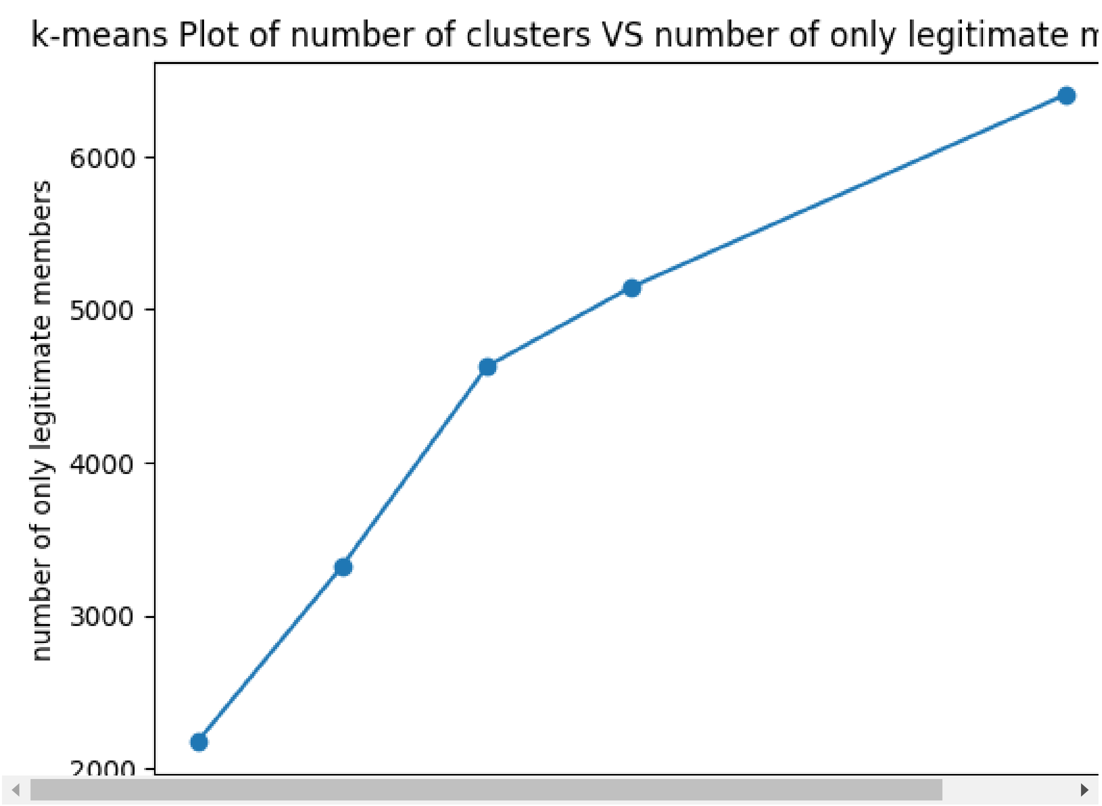
```
[0, 0, 0, 0, 0]
```

```
# make the ouput in data frame
num_clusters = [8, 12, 16, 20, 32]
new =[2178, 3325, 4628, 5147, 6403]
results = pd.DataFrame(columns=["number of cluster", "number of only legitimate member"]
for i in range(5):
    results = results.append({"number of cluster":num_clusters[i]  , "number of only leg
results
```

```
<ipython-input-31-5cdf8f01ad53>:6: FutureWarning: The frame.append
  results = results.append({"number of cluster":num_clusters[i]  ,
<ipython-input-31-5cdf8f01ad53>:6: FutureWarning: The frame.append
  results = results.append({"number of cluster":num_clusters[i]  ,
<ipython-input-31-5cdf8f01ad53>:6: FutureWarning: The frame.append
  results = results.append({"number of cluster":num_clusters[i]  ,
<ipython-input-31-5cdf8f01ad53>:6: FutureWarning: The frame.append
  results = results.append({"number of cluster":num_clusters[i]  ,
<ipython-input-31-5cdf8f01ad53>:6: FutureWarning: The frame.append
  results = results.append({"number of cluster":num_clusters[i]  ,
```

number of cluster number of only legitimate member

```
# plot the result
# Define the data
x=results['number of cluster']
y=results['number of only legitimate member']
# Create the plot
plt.plot(x, y, 'o-')

# Add labels and title
plt.xlabel('number of clusters')
plt.ylabel('number of only legitimate members')
plt.title('k-means Plot of number of clusters VS number of only legitimate members')

# Show the plot
plt.show()
```

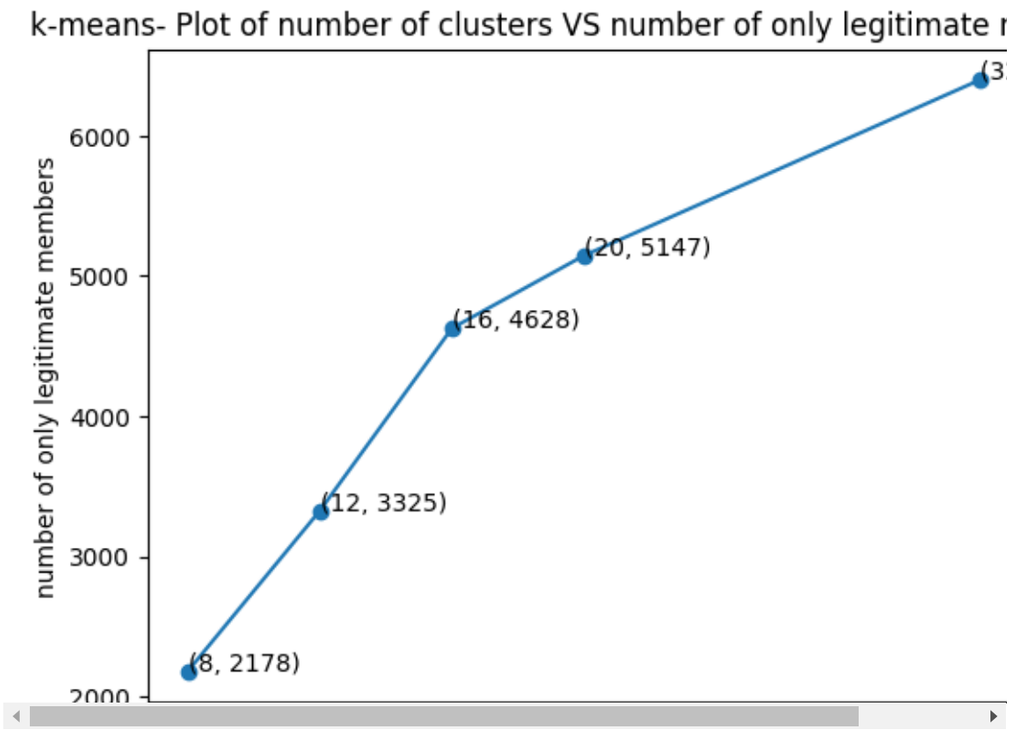


the more you increase the clusters the more it get smaller

and this will give you more pure clusters

```
# another plot
# Define the data
x=results['number of cluster']
y=results['number of only legitimate member']

# Create the plot
plt.plot(x, y, 'o-')

# Add labels and title
```

```
plt.xlabel('number of clusters')
plt.ylabel('number of only legitimate members')
plt.title('k-means- Plot of number of clusters VS number of only legitimate members')

# Set the x-axis and y_axis tick locations
plt.xticks([8, 12, 16, 20, 32])

# Add values to the plot
for i in range(len(x)):
    plt.text(x[i], y[i], f'({x[i]}, {y[i]})')

# Show the plot
plt.show()
```




```
!pip install minisom
from minisom import MiniSom

# Define the number of clusters to use
num_clusters = [8, 12, 16, 20, 32]

# create a useful variable
mem_cluster_fit=[]
mem_cluster_predict=[]

# create a data frame
df_label = pd.DataFrame(columns=["cluster number", "fit_labels", "predict_labels"])
df = pd.DataFrame(columns=["cluster number", "members in each cluster inside fit data",

# Convert DataFrame to NumPy array
X_array = X.to_numpy()
only_ligitimancy_array = only_ligitimancy.to_numpy()

# model with n clusters and find members in each cluster
for n in num_clusters:
    # Fit the data using Self-Organizing Feature Map (SOFM)
    som = MiniSom(n, 1, X_array.shape[1], sigma=0.3, learning_rate=0.5)
    som.train(X_array, 100)

    # Get the labels for the fit and predict data
    fit_labels = np.array([som.winner(x)[0] for x in X_array])
    mem_cluster_fit=[len(fit_labels[fit_labels == i]) for i in range(n)]
    predict_labels = np.array([som.winner(x)[0] for x in only_ligitimancy_array])
    mem_cluster_predict=[len(predict_labels[predict_labels == i]) for i in range(n)]
```

```
    df = df.append({"cluster number": n, "members in each cluster inside fit data":mem_c
    df_label = df_label.append({"cluster number": n, "fit_labels":fit_labels, "predict_l

df
# compare between members in each cluster inside fit data (legitimate and fake) and only
# calcutate the number of only legitimate member inside each cluster
n = [8, 12, 16, 20, 32]
f=df['members in each cluster inside fit data']
p=df['members in each cluster inside predict data']

df
```

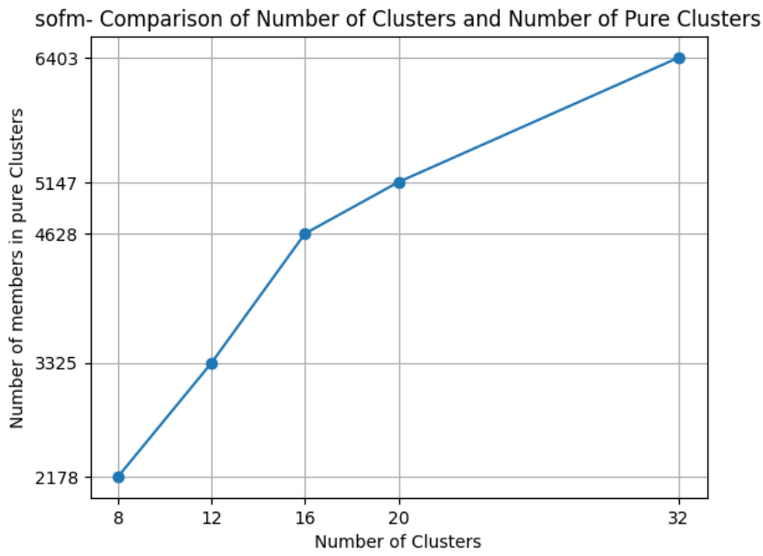
```
<ipython-input-40-f2084890912e>:29: FutureWarning: The frame.append
  df = df.append({"cluster number": n, "members in each cluster ins
<ipython-input-40-f2084890912e>:30: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number": n, "fit_labels":fit
<ipython-input-40-f2084890912e>:29: FutureWarning: The frame.append
  df = df.append({"cluster number": n, "members in each cluster ins
<ipython-input-40-f2084890912e>:30: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number": n, "fit_labels":fit
<ipython-input-40-f2084890912e>:29: FutureWarning: The frame.append
  df = df.append({"cluster number": n, "members in each cluster ins
<ipython-input-40-f2084890912e>:30: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number": n, "fit_labels":fit
<ipython-input-40-f2084890912e>:29: FutureWarning: The frame.append
  df = df.append({"cluster number": n, "members in each cluster ins
<ipython-input-40-f2084890912e>:30: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number": n, "fit_labels":fit
<ipython-input-40-f2084890912e>:29: FutureWarning: The frame.append
  df = df.append({"cluster number": n, "members in each cluster ins
<ipython-input-40-f2084890912e>:30: FutureWarning: The frame.append
  df_label = df_label.append({"cluster number": n, "fit_labels":fit
```

	cluster number	members in each cluster inside fit data	members in each cluster inside predict data
0	8	[0, 0, 0, 9715, 0, 0, 0, 0]	[0, 0, 0, 8446, 0, 0, 0, 0]
1	12	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9715]	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8446]
		[0, 0, 9715, 0, 0, 0, 0, 0, 0, 0, 0,	[0, 0, 8446, 0, 0, 0, 0, 0, 0, 0, 0,

```
new = []
for i in range(len(num_clusters)):
    cluster_fit_labels = df_label.loc[i, "fit_labels"]
    cluster_predict_labels = df_label.loc[i, "predict_labels"]
    cluster_fit_members = df.loc[i, "members in each cluster inside fit data"]
    cluster_predict_members = df.loc[i, "members in each cluster inside predict data"]
    sum_value = 0
    for j in range(len(cluster_fit_labels)):
        if j < len(cluster_fit_members) and cluster_fit_labels[j] == cluster_predict_lab
            sum_value += cluster_fit_members[j]
    new.append(sum_value)

x = df["cluster number"]
y = new
```

```
import matplotlib.pyplot as plt

# Convert x and y to appropriate data types
x = x.astype(int)
y = [int(val) for val in y]

# Calculate the number of pure clusters
pure_clusters = [i for i, value in enumerate(y) if value == max(y)]
```

```
# Plot the curve
plt.plot(x, y, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("Number of members in pure Clusters")
plt.title("sofm- Comparison of Number of Clusters and Number of Pure Clusters")
plt.xticks(x)
plt.yticks(y)
plt.grid(True)
plt.show()
```



the more you increase the clusters the more it get smaller and this will give you more pure clusters

Double-click (or enter) to edit

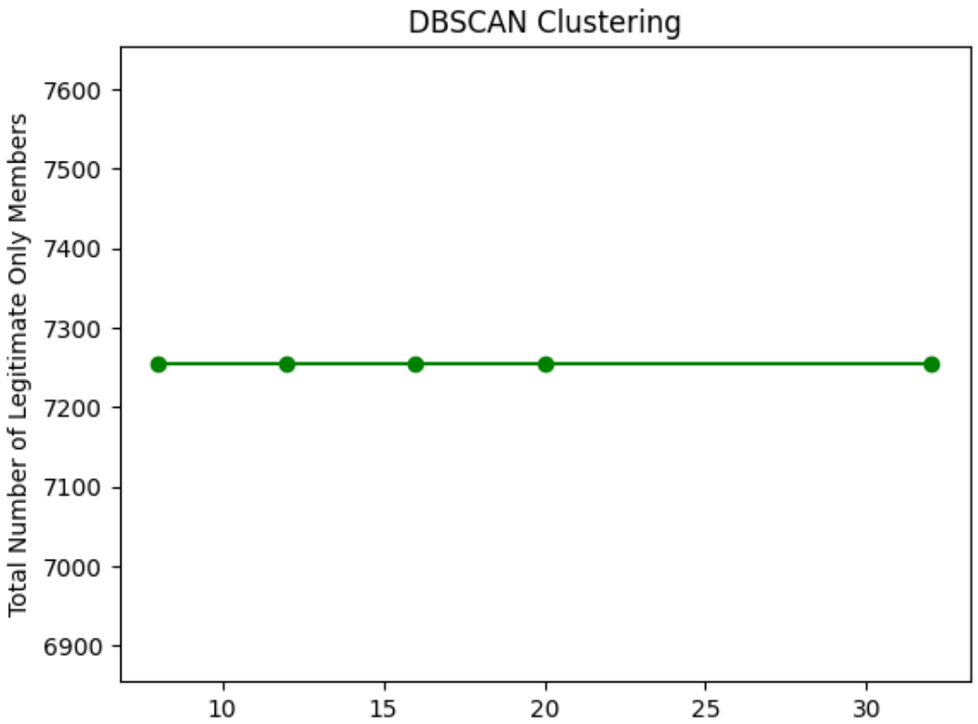
Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

```
from sklearn.cluster import DBSCAN
mid_points = [0.1, 0.2, 0.3, 0.4, 0.5]  # Example midPoints, adjust as needed
epsilons = [0.5, 1.0, 1.5, 2.0, 2.5]  # Example epsilons, adjust as needed
cluster_numbers = [8, 12, 16, 20, 32]
legitimate_only_members = []
for n_clusters in cluster_numbers:
```

```
    max_legitimate_only_members = -1
    for mid_point, epsilon in zip(mid_points, epsilons):
        dbscan = DBSCAN(eps=epsilon, min_samples=5, metric='euclidean')
        labels = dbscan.fit_predict(X_train_lat_lon)
        unique_labels, counts = np.unique(labels, return_counts=True)
        legitimate_clusters = counts[np.where(unique_labels != -1)]
        if len(legitimate_clusters) > 0:
            max_legitimate_only_members = max(max_legitimate_only_members, np.max(legiti
    legitimate_only_members.append(max_legitimate_only_members)
plt.plot(cluster_numbers, legitimate_only_members, 'go-')
plt.xlabel('Number of Clusters')
plt.ylabel('Total Number of Legitimate Only Members')
plt.title('DBSCAN Clustering')
plt.show()
```



Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

not accurate result for the dbscan

couldnt understand and need more analysis

A legitimate-only member refers to a data point that belongs to its own cluster, meaning it does not share a cluster with any other data point. The fact that the number of legitimate-only members remains constant at 7000 suggests that there might be a significant subset of the data that does not have close neighbors or forms separate clusters.

✓ 18s completed at 2:03 AM