Report Assignment 3

(Calculations & Programming)

Applied Machine Learning ELG5255[EG]

Group Number: G_26

Prepared by:

Sawsan Awad (300327224)

Sondos Ali (300327219)

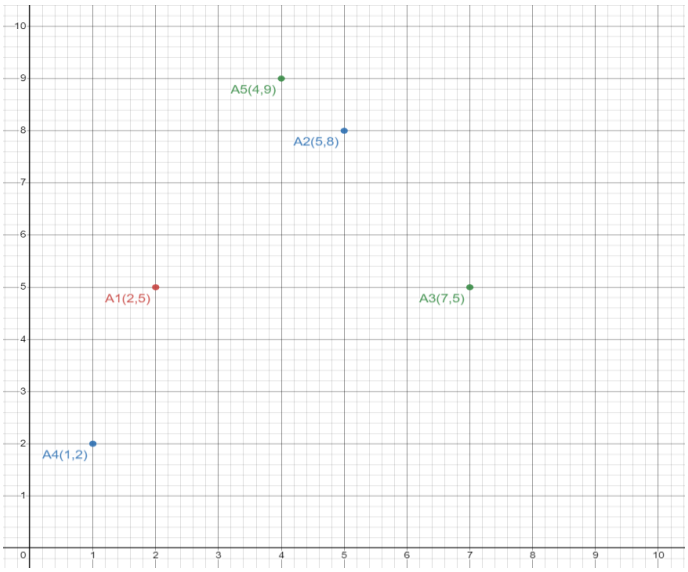
Toka Mostafa (300327284)

Part 1: Calculations

1. Use the k-means algorithm and Euclidean distance to cluster the following 5 data points into 2 clusters: A1= (2,5), A2= (5,8), A3= (7,5), A4= (1,2), A5= (4,9). 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.
    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.

Solution

- ✓ **K = 2**
- ✓ **C1= (5,8), C2= (1,2)**
- ✓ **A1= (2,5), A2= (5,8), A3= (7,5), A4= (1,2), A5= (4,9)**



1. **Step (1): Calculate the distance between points and centers by using Euclidean Distance:**

$$D = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}$$

1.1. Calculate distance between A1 and C1, C2:

$$D(A1, C1) = \sqrt{(5-2)^2 - (8-5)^2} = 4.242640687$$

$$D(A1, C2) = \sqrt{(1-2)^2 - (2-5)^2} = 3.16227766$$

1.2. Calculate distance between A2 and C1, C2:

$$D(A2, C1) = \sqrt{(5-5)^2 - (8-8)^2} = 0$$

$$D(A2, C2) = \sqrt{(1-5)^2 - (2-8)^2} = 7.211102551$$

1.3. Calculate distance between A3 and C1, C2:

$$D(A3, C1) = \sqrt{(5-7)^2 - (8-5)^2} = 3.605551275$$

$$D(A3, C2) = \sqrt{(1-7)^2 - (2-5)^2} = 6.708203932$$

1.4. Calculate distance between A4 and C1, C2:

$$D(A4, C1) = \sqrt{(5-1)^2 - (8-2)^2} = 7.211102551$$

$$D(A4, C2) = \sqrt{(1-1)^2 - (2-2)^2} = 0$$

1.5. Calculate distance between A5 and C1, C2:

$$D(A5, C1) = \sqrt{(5-4)^2 - (8-9)^2} = 1.414213562$$

$$D(A5, C2) = \sqrt{(1-4)^2 - (2-9)^2} = 7.615773106$$

2. Step (2): Classify the points to clusters by the nearest distance:

Given Points	Distance from center of cluster 1	Distance from center of cluster 2	Points belong to cluster
A1= (2,5)	4.242640687	3.16227766	C2
A2= (5,8)	0	7. 211102551	C1
A3= (7,5)	3.605551275	6.708203932	C1
A4= (1,2)	7.211102551	0	C2
A5= (4,9)	1.414213562	7.615773106	C1

- ✓ **Cluster (1):** A2= (5,8), A3= (7,5), A5= (4,9)
- ✓ **Cluster (2):** A1= (2,5), A4= (1,2)

3. **Step (3): Compute the new center to clusters:**

- **Mean of all points which are in the same cluster.**
  (New cluster center)

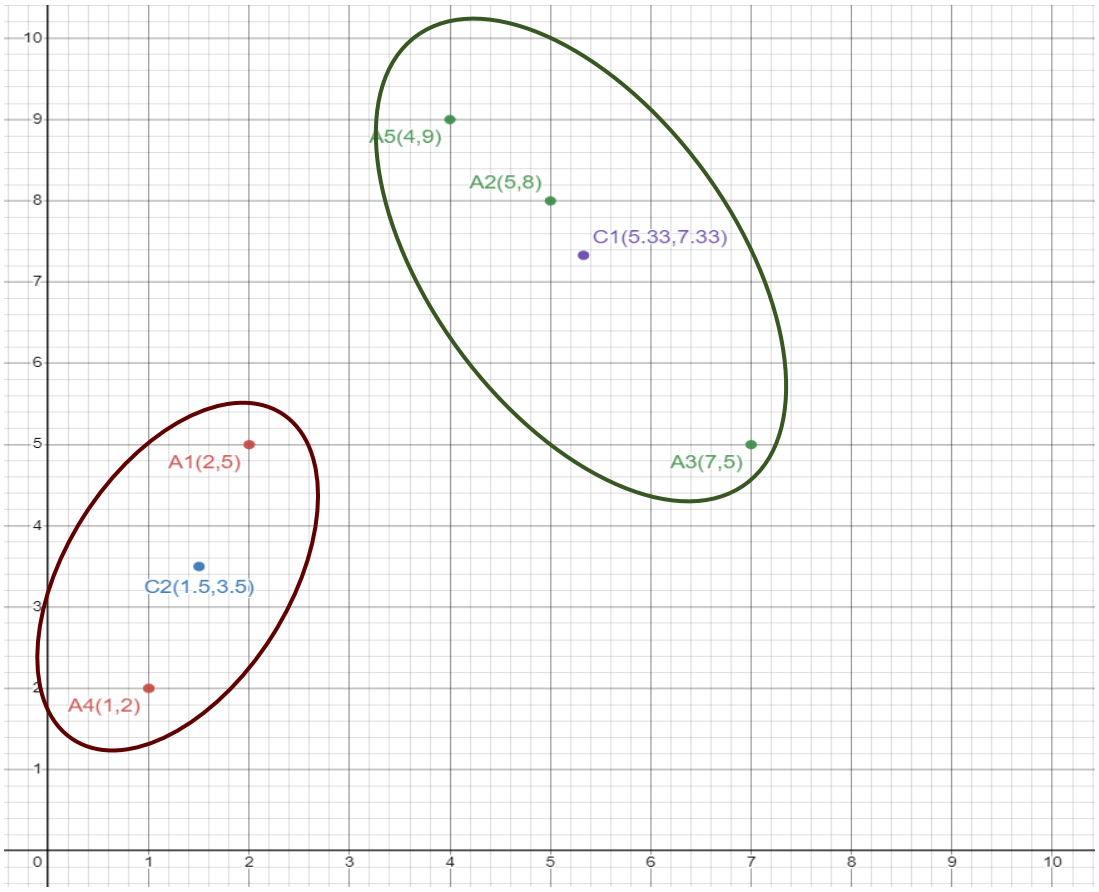
$$C_i = \left(\frac{x_1 + x_2 + x_n}{n}\right), \left(\frac{y_1 + y_2 + y_n}{n}\right)$$

1) **New center to cluster (1):**

$$C_1 = \left(\frac{5+7+4}{3}\right), \left(\frac{8+5+9}{3}\right) = (5.33, 7.33)$$

2) **New center to cluster (2):**

$$C_2 = \left(\frac{2+1}{2}\right), \left(\frac{5+2}{2}\right) = (1.5, 3.5)$$



4. Step (5): Compute the Silhouette:

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

- **The Silhouette consists of 2 parts:**
  - ✓ **First Part: Cohesion which is the calculation of the distance between points in the same cluster a(i).**
    - **a(i): is the average of all distances which we calculate to each point.**

$$a(i) = \left(\frac{All\ the\ distance\ of\ point\ i}{n}\right)$$

  - ✓ **Second Part: Separate which is the calculation of the distance between each point with the rest in the different cluster b(i).**
    - **b(i): is the average of all distances which we calculate to each point.**

$$b(i) = \left(\frac{All\ the\ distance\ of\ point\ i}{n}\right)$$

4.1. Calculate Cohesion:

4.1.1. Calculate a(A1):

$$D(A1, A4) = \sqrt{(1-2)^2 - (2-5)^2} = 3.16227766$$

$$a(A1) = \left(\frac{3.16227766}{1}\right) = 3.16227766$$

4. 1.2. Calculate a(A2):

$$D(A2, A3) = \sqrt{(7-5)^2 - (5-8)^2} = 3.605551275$$

$$D(A2, A5) = \sqrt{(4-5)^2 - (9-8)^2} = 1.414213562$$

$$a(A2) = \left(\frac{3.605551275 + 1.414213562}{2}\right) = 2.509882419$$

4. 1.3. Calculate a(A3):

$$D(A3, A2) = \sqrt{(5-7)^2 - (8-5)^2} = 3.605551275$$

$$D(A3, A5) = \sqrt{(4-7)^2 - (9-5)^2} = 5$$

$$a(A3) = \left(\frac{3.605551275 + 5}{2}\right) = 4.302775638$$

4. 1.4. Calculate a(A4):

$$D(A1, A4) = \sqrt{(1-2)^2 - (2-5)^2} = 3.16227766$$

$$a(A4) = \left(\frac{3.16227766}{1}\right) = 3.16227766$$

4. 1.5. Calculate a(A5):

$$D(A5, A2) = \sqrt{(5-4)^2 - (8-9)^2} = 1.414213562$$

$$D(A5, A3) = \sqrt{(7-4)^2 - (5-9)^2} = 5$$

$$a(A5) = \left(\frac{1.414213562 + 5}{2}\right) = 3.207106781$$

4.2. Calculate Separate:

4.2.1. Calculate b(A1):

$$D(A1, A2) = \sqrt{(5-2)^2 - (8-5)^2} = 4.242640687$$

$$D(A1, A3) = \sqrt{(7-2)^2 - (5-5)^2} = 5$$

$$D(A1, A5) = \sqrt{(4-2)^2 - (9-5)^2} = 4.472135955$$

$$b(A1) = \left(\frac{4.242640687 + 4.472135955 + 5}{3}\right) = 4.571592214$$

4. 2.2. Calculate b(A2):

$$D(A2, A1) = \sqrt{(2-5)^2 - (5-8)^2} = 4.242640687$$

$$D(A2, A4) = \sqrt{(1-5)^2 - (2-8)^2} = 7.211102551$$

$$b(A2) = \left(\frac{4.242640687 + 7.211102551}{2}\right) = 5.726871619$$

4. 2.3. Calculate b(A3):

$$D(A3, A1) = \sqrt{(2-7)^2 - (5-5)^2} = 5$$

$$D(A3, A4) = \sqrt{(1-7)^2 - (2-5)^2} = 6.708203932$$

$$b(A3) = \left(\frac{6.708203932 + 5}{2}\right) = 5.854101966$$

4. 2.4. Calculate b(A4):

$$D(A4, A2) = \sqrt{(5-1)^2 - (8-2)^2} = 7.211102551$$

$$D(A4, A3) = \sqrt{(7-1)^2 - (5-2)^2} = 6.708203932$$

$$D(A4, A5) = \sqrt{(4-1)^2 - (9-2)^2} = 7.615773106$$

$$b(A4) = \left(\frac{7.615773106 + 6.708203932 + 7.211102551}{3}\right) = 7.17835986$$

4. 2.5. Calculate b(A5):

$$D(A5, A1) = \sqrt{(2-4)^2 - (5-9)^2} = 4.472135955$$

$$D(A5, A4) = \sqrt{(1-4)^2 - (2-9)^2} = 7.615773106$$

$$b(A5) = \left(\frac{4.472135955 + 7.615773106}{2}\right) = 6.04395453$$

4.3. Calculate Silhouette:

4.3.1. Calculate S(A1):

$$S(A1) = \left(\frac{4.571592214 - 3.16227766}{4.571592214}\right) = 0.3082765234$$

4. 3.2. Calculate S(A2):

$$S(A2) = \left(\frac{5.726871619 - 2.509882419}{5.726871619}\right) = 0.561735868$$

4. 3.3. Calculate S(A3):

$$S(A3) = \left(\frac{5.854101966 - 4.302775638}{5.854101966}\right) = 0.2649981734$$

4. 3.4. Calculate S(A4):

$$S(A4) = \left(\frac{7.17835986 - 3.16227766}{7.17835986}\right) = 0.5594707257$$

4. 3.5. Calculate S(A5):

$$S(A5) = \left(\frac{6.04395453 - 3.207106781}{6.04395453}\right) = 0.4693694729$$

5. Step (6): Compute Within the sum of squares (WSS):

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

x_i : data point.

C_i : the closest point (the center of each cluster) to points in the cluster.

$$WSS1 = (5 - 5.33)^2 + (8 - 7.33)^2 + (7 - 5.33)^2 + (5 - 7.33)^2 + (4 - 5.33)^2 + (9 - 7.33)^2 = 13.1667$$

$$WSS2 = (2 - 1.5)^2 + (5 - 3.5)^2 + (1 - 1.5)^2 + (2 - 3.5)^2 = 5.1667$$

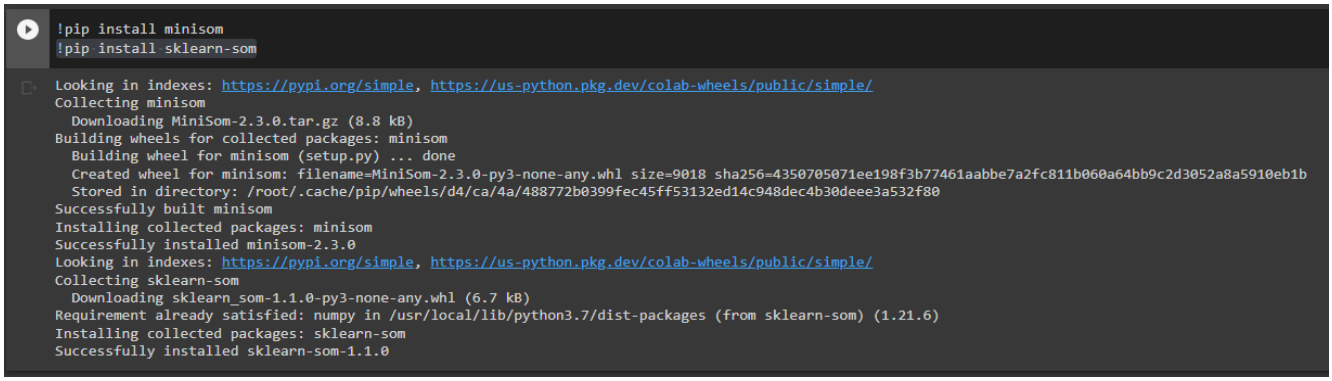
$$WSS = WSS1 + WSS2 = 13.1667 + 5.1667 = 18.3334$$

Part 2: Programming

We followed some defined steps to obtain the aimed results:

2.1. Installing the important package:

- !pip install minisom**:** is a minimalistic and NumPy based implementation of the Self Organizing Maps (SOM).
- !pip install sklearn-som: is a minimalist, simple implementation of a Kohonen self-organizing map with a planar (rectangular) topology.



2.2. Importing important libraries:

- NumPy library: it provides a lot of supporting functions that make working with ndarray very easy.
- Pandas library: it helps us to analyze and understand data better.
- from sklearn.model_selection import train_test_split: is a function in Sklearn model selection for splitting data arrays into two subsets for training data and for testing data. With this function, we don't need to divide the dataset manually. By default, Sklearn train_test_split will make random partitions for the two subsets.
- from sklearn.pipeline import Pipeline: is to assemble several steps that can be cross-validated together while setting different parameters.
- from sklearn.neighbors import KNeighborsClassifier: a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
- from sklearn.linear_model import LogisticRegression: is a classification algorithm rather than regression algorithm. Based on a given set of independent variables, it is used to estimate discrete value (0 or 1, yes/no, true/false).

- from sklearn.metrics import classification_report, accuracy_score:
    - Classification_report: is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of your trained classification model, and it will return accuracy.
    - The accuracy_score: is function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions.
- from sklearn.model_selection import GridSearchCV: is a function that comes in Scikit-learn's model_selection package to find the best values for hyperparameters of a model.
- from sklearn.cluster import KMeans: selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
- from sklearn.metrics import silhouette_score: is used to evaluate the quality of clusters created using clustering algorithms such as K-Means in terms of how well samples are clustered with other samples that are similar to each other. The Silhouette score is calculated for each sample of different clusters.
- Matplotlib.pyplot library: used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc.
- Other libraries will be shown their importance in the code.

```
import numpy as np
import pandas as pd

#sklearn
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif,chi2,f_classif
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SequentialFeatureSelector
from minisom import MiniSom
from sklearn_som.som import SOM
#Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.core.pylabtools import figsize
%matplotlib inline
```

2.3. Importing dataset:

- First, we use read the dataset.
- Second, we use .head() function to display the first five rows of the data frame by default.

```
[4] data = pd.read_csv('Assignment3_dataset.csv')
    data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.411765	0.623116	0.573770	0.333333	0.254137	0.380030	0.035440	0.266667	0
1	0.294118	0.542714	0.590164	0.434343	0.088652	0.538003	0.078992	0.200000	0
2	0.058824	0.437186	0.491803	0.373737	0.088652	0.554396	0.184031	0.016667	0
3	0.058824	0.723618	0.672131	0.464646	0.212766	0.687034	0.109735	0.416667	1
4	0.058824	0.557789	0.508197	0.131313	0.215130	0.357675	0.025619	0.033333	0

2.4. Functions:

We make many functions to many iterations to generalize.

2.4.1. Splitting Function:

- Here we will split the dataset to 2 sets: (training set & test set), test set size will be 25% from the dataset.

```
1. Splitting

[5] def split(x,y):
      x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25, random_state =0)
      return x_train,x_test,y_train,y_test
```

2.4.2. Models Function:

- ✓ KNeighborsClassifier: is used to implement classification based on voting by nearest k-neighbors of target point, t, while RadiusNeighborsClassifier implements classification based on all neighborhood points within a fixed radius, r, of target point, t.
- ✓ Logistics Regression: is a machine learning model that uses a hyperplane in an dimensional space to separate data points with number of features into their classes.

- First, we make def knn and give 3 parameters to it (x_train, y_train, x_test) to be able to all models to make fit and predict.
- Second, we make def lr and give 3 parameters to it (x_train, y_train, x_test) to be able to all models to make fit and predict.

2. Models

```
[6]  def knn(x_train,y_train,x_test):
       knn = KNeighborsClassifier(n_neighbors=27)
       knn.fit(x_train,y_train)
       y_knn = knn.predict(x_test)
       return y_knn

     def lr (x_train,y_train,x_test):
       lr = LogisticRegression()
       lr.fit(x_train,y_train)
       y_lr = lr.predict(x_test)
       return y_lr
```

2.4.3. Accuracy Function:

- First, we make def accuracy (for generalization) and give 2 parameters to it (y_test, y_pred) to be able to all models to generate the accuracy.
- Second, we use classification_report() function to generate all details for all models like: (Accuracy, Recall, F1-score, precision,..).

3. Accuracy

```
[7]  def accuracy(y_test,y_pred):
       acc = accuracy_score(y_test,y_pred)
       return acc
```

2.4.4. K-means Function:

- ✓ The silhouette coefficient is a measure of how similar a data point is within-cluster (cohesion) compared to other clusters (separation).
- ✓ K-means clustering: is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K.
- First, we make def Silhouette and give 1 parameter to it (x).
- Second, we make def kmeans and give 2 parameters to it (n_clusters, x).

4. KMeans

```
def Silhouette(x):
  Silhouette_Score =[]
  for i in range(2, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=1, random_state=100)
    cluster_labels = kmeans.fit_predict(x)
    silhouette_avg = silhouette_score(x, cluster_labels,metric="euclidean",sample_size=1000,random_state=0)
    Silhouette_Score.append(silhouette_avg )

  plt.plot(range(2,11),Silhouette_Score)
  plt.title('Silhouette score vs Numbers of clusters')
  plt.xlabel('Numbers of clusters')
  plt.ylabel('Silhouette score')
  return silhouette_avg

def kmeans(n_clusters,x):
  kmeans = KMeans(n_clusters=n_clusters, init='k-means++', max_iter=300, n_init=1, random_state=100)
  y_kmeans = kmeans.fit_predict(x)
  return y_kmeans
```

2.4.5. SOM:

- ✓ SOM A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction.

5. SOM

```
from sklearn.utils.multiclass import unique_labels
def silhoutte_n_neuron(feature):
  accList=[]
  silhouette_lst=[]
  n_Neuron =range(2,31)

  _ , dim = feature.shape
  for i in n_Neuron:

    som=SOM(m=i, n=1,dim=dim,random_state=0)
    predClusters=som.fit_predict(feature)

    score = silhouette_score(feature, predClusters, random_state=0)
    silhouette_lst.append(score)
    accList.append(accuracy)

  plt.plot(range(2,31),silhouette_lst)
  plt.title('Silhouette score vs Numbers of neurons')
  plt.xlabel('Numbers of neurons')
  plt.ylabel('Silhouette score')
  return silhouette_lst
```

2.4.6. PCA:

- ✓ PCA Principal Component Analysis (PCA): is a technique that comes from the field of linear algebra and can be used as a data preparation technique to create a projection of a dataset prior to fitting a model.

- We use PCA for dimensionality reduction.

6. PCA

```
[10] def pca(n_components,x_train,x_test):
       Pca = PCA(n_components = n_components,random_state =0)
       x_train_pca = Pca.fit_transform(x_train)
       x_test_pca = Pca.transform(x_test)
       return x_train_pca,x_test_pca
```

2.4.7. TSNE:

- ✓ TSNE :T-distributed Stochastic Neighbor Embedding is a tool for visualizing high-dimensional data. T-SNE, based on stochastic neighbor embedding, is a nonlinear dimensionality reduction technique to visualize data in a two- or three-dimensional space.

7. TSNE

```
def tsne(x,y):
    Tsne = TSNE(n_components = 2)
    tsne_results = Tsne.fit_transform(x)
    df = pd.DataFrame()
    df['feature 1'] = tsne_results[:,0]
    df['feature 2'] = tsne_results[:,1]
    df['y'] = y
    sns.scatterplot(x = 'feature 1', y = 'feature 2', hue = df.y.tolist(), data = df)
```

2.4.8. Filter Method:

8. Filter method

```
[12] def selectkbest(model,k,x_train,y_train,x_test):
       select = SelectKBest(score_func= model,k = k)
       x_train_fs = select.fit_transform(x_train,y_train)
       x_test_fs = select.transform(x_test)
       features_names = select.get_feature_names_out(np.array(['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']))
       return x_train_fs,x_test_fs,features_names

     def variance_threshold(threshold,x_train,x_test):
       v_threshold = VarianceThreshold(threshold= threshold)
       x_train_vt = v_threshold.fit_transform(x_train)
       x_test_vt = v_threshold.transform(x_test)
       features_names = v_threshold.get_feature_names_out(np.array(['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']))
       return x_train_vt,x_test_vt,features_names
```

2.4.9. Wrapper Method:

9. Wrapper method

```
[13] def wrapper (model,direction,i,x_train,y_train,x_test):
       model.fit(x,y)
       selector = SequentialFeatureSelector(model,n_features_to_select=i,direction=direction)
       x_train_new = selector.fit_transform(x_train,y_train)
       x_test_new = selector.transform(x_test)
       features_names = selector.get_feature_names_out(np.array(['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']))
       return x_train_new,x_test_new,features_names
```

2.4.10. Bar Plot Function:

```
10. Bar plot

[14] def bar(y_knn,y_lr,baseline_knn,baseline_lr):
       figsize(10,7)
       plt.subplot(1,2,1)
       sns.barplot(x = [1,2,3,4,5,6,7],y = y_knn)
       sns.lineplot(x=[1,2,3,4,5,6,7],y = baseline_knn)
       plt.title('Accuracies of KNN')
       plt.ylim(0.5,0.9,0.01)
       plt.subplot(1,2,2)
       sns.barplot(x=[1,2,3,4,5,6,7],y = y_lr)
       sns.lineplot(x=[1,2,3,4,5,6,7],y = baseline_lr)
       plt.title('Accuracies of LR')
       plt.ylim(0.5,0.9,0.01)
```

2.5. KNN & LR:

- First: we split the data into training (75%) and testing (25%).
- Modeling:

  a) KNN Model:
  - ✓ We train the model on training set. We use KNN Model for solving classification problems.
  - ✓ We make predictions on the testing set.
  - ✓ We calculate the accuracy for the model : 0.7643979057591623

  b) Logistic Regression model :
  - ✓ We train the model on training set. We use LR Model for solving classification problems.
  - ✓ We make predictions on the testing set.
  - ✓ We calculate the accuracy for the model : 0.774869109947644
- We Provide 2D TSNE plots, one for the training set and one for the test set

```
Question 1

[ ]  #Splitting
     x = data.iloc[:,:-1].values
     y = data.iloc[:,-1].values
     x_train,x_test,y_train,y_test = split(x,y)

     #knn
     y_knn = knn(x_train,y_train,x_test)
     acc_knn = accuracy(y_test,y_knn)

     #Logistic Regression
     y_lr = lr(x_train,y_train,x_test)
     acc_lr = accuracy(y_test,y_lr)

     #TSNE
     figsize(10,5)

     plt.subplot(1,2,1)
     tsne(x_train,y_train)
     plt.subplot(1,2,2)
     tsne(x_test,y_test)

     print(acc_knn)
     print(acc_lr)
```

```
0.7643979057591623
0.774869109947644
```

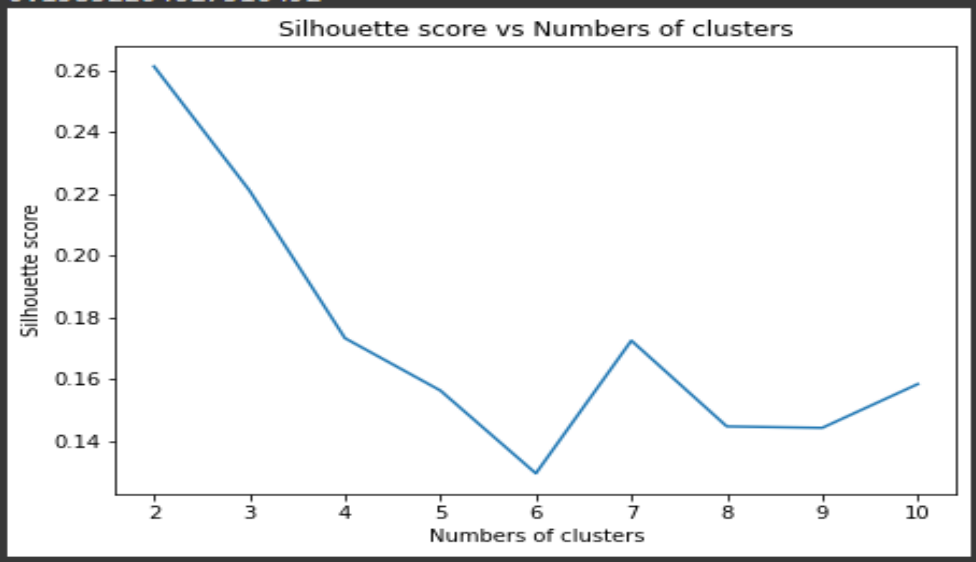
2.6. Determine the optimum K:

- We calculate silhouette score to get the optimum number of K and plot the data.

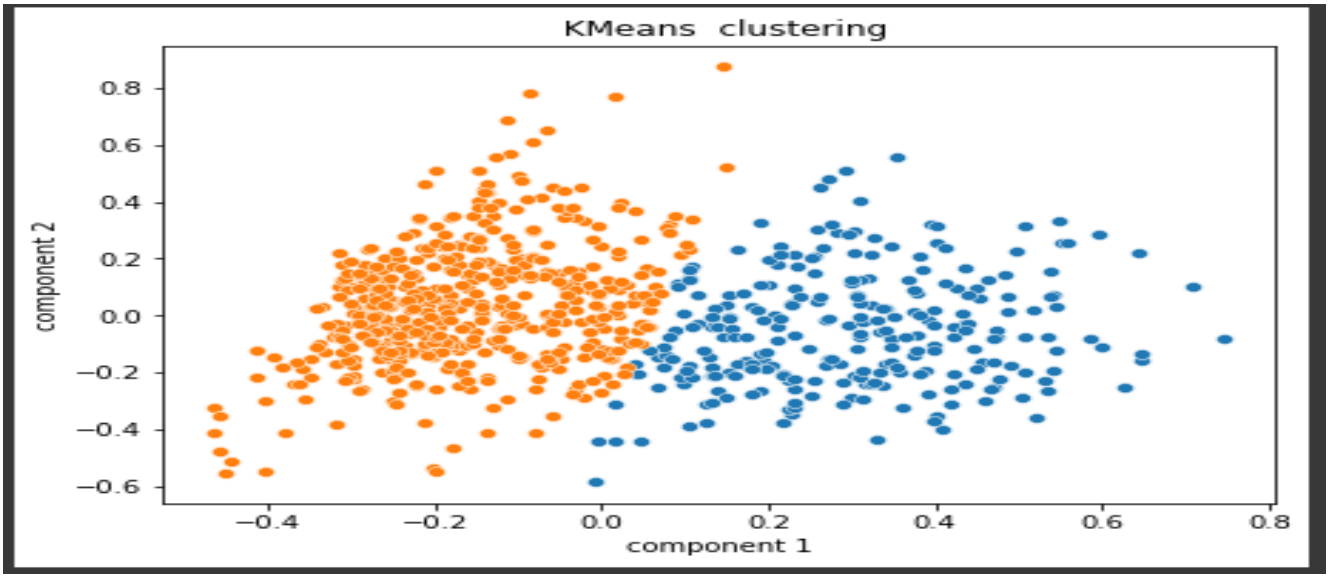
Question 2

```
figsize(7,5)
Silhouette(x)
```

```
0.15832264627316492
```



```
[ ]  #KMeans clustering
     figsize(7,5)
     x_train_pca,x_test_pca = pca(2,x_train,x_test)
     x_new_pca = np.concatenate((x_train_pca,x_test_pca))
     y_kmeans = kmeans(2,x_new_pca)
     sns.scatterplot(x_new_pca[y_kmeans == 0,0],x_new_pca[y_kmeans == 0,1])
     sns.scatterplot(x_new_pca[y_kmeans == 1,0],x_new_pca[y_kmeans == 1,1])
     plt.title(' KMeans  clustering')
     plt.xlabel("component 1")
     plt.ylabel("component 2")
     plt.show
```



2.7. PCA:

- We apply dimensionality reduction by PCA algorithm, then plot the Number of Components-Accuracy graph, and plot the TSNE after PCA.

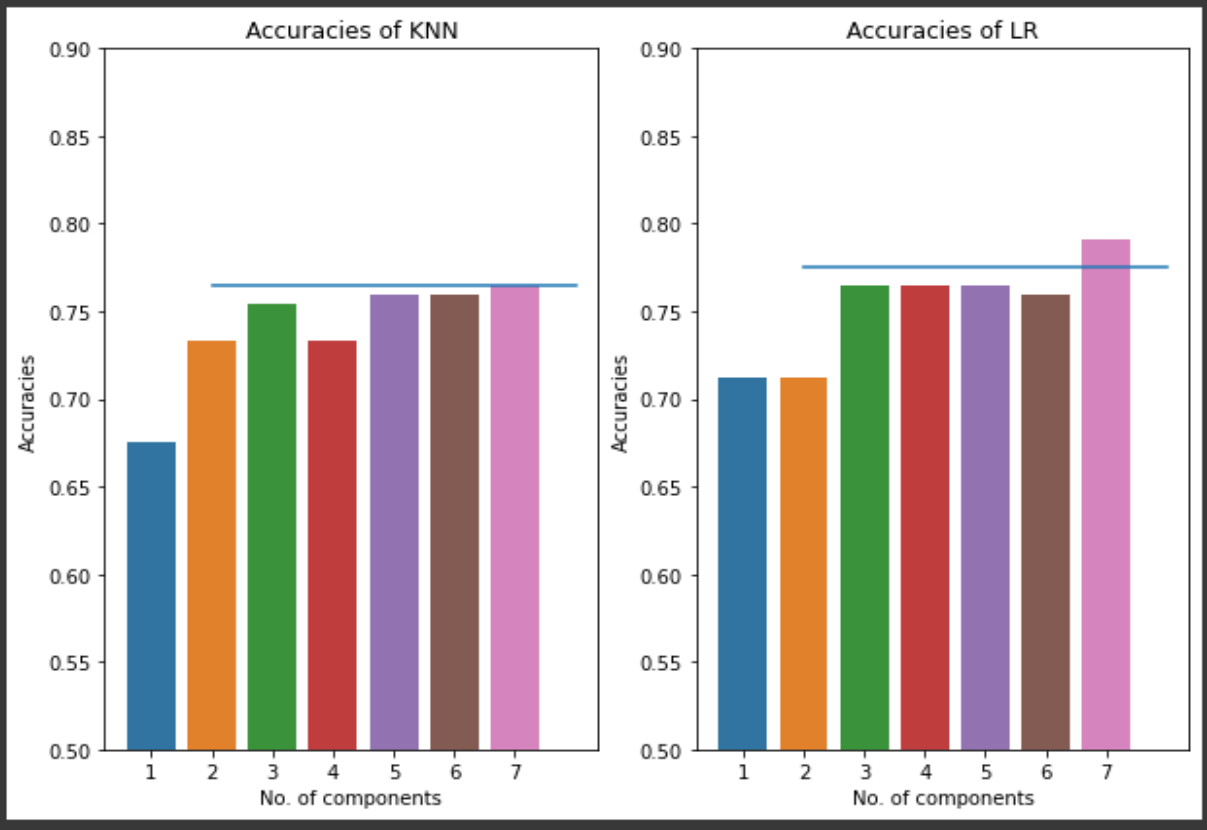
Question 3

```
[ ]  lr_pca = []
     knn_pca = []

     for i in range(1,8):
       #knn
       x_train_pca,x_test_pca = pca(i,x_train,x_test)
       y_knn_pca = knn(x_train_pca,y_train,x_test_pca)
       acc_knn_pca = accuracy(y_test,y_knn_pca)
       knn_pca.append(acc_knn_pca)

       #Logistic Regression
       y_lr_pca = lr(x_train_pca,y_train,x_test_pca)
       acc_lr_pca = accuracy(y_test,y_lr_pca)
       lr_pca.append(acc_lr_pca)

     print("Accuracies of KNN: ",knn_pca)
     print("Accuracies of LR: ",lr_pca)
     print(np.argmax(knn_pca),': ', knn_pca[np.argmax(knn_pca)])
     print(np.argmax(lr_pca), ': ', lr_pca[np.argmax(lr_pca)])

     Accuracies of KNN:  [0.675392670157068, 0.7329842931937173, 0.7539267015706806, 0.7329842931937173, 0.7591623036649214, 0.7591623036649214, 0.7643979057591623]
     Accuracies of LR:  [0.7120418848167539, 0.7120418848167539, 0.7643979057591623, 0.7643979057591623, 0.7643979057591623, 0.7591623036649214, 0.7905759162303665]
     6 :  0.7643979057591623
     6 :  0.7905759162303665
```

```
[ ]  x_train_pca,x_test_pca = pca(7,x_train,x_test)

     #knn
     y_knn_pca = knn(x_train_pca,y_train,x_test_pca)
     acc_knn_pca = accuracy(y_test,y_knn_pca)

     #Logistic Regression
     y_lr_pca = lr(x_train_pca,y_train,x_test_pca)
     acc_lr_pca = accuracy(y_test,y_lr_pca)


     #Concatenate all data
     x_pca = np.concatenate((x_train_pca,x_test_pca))

     print("Accuracy of KNN: ",acc_knn_pca)
     print("Accuracy of LR: ",acc_lr_pca)
```

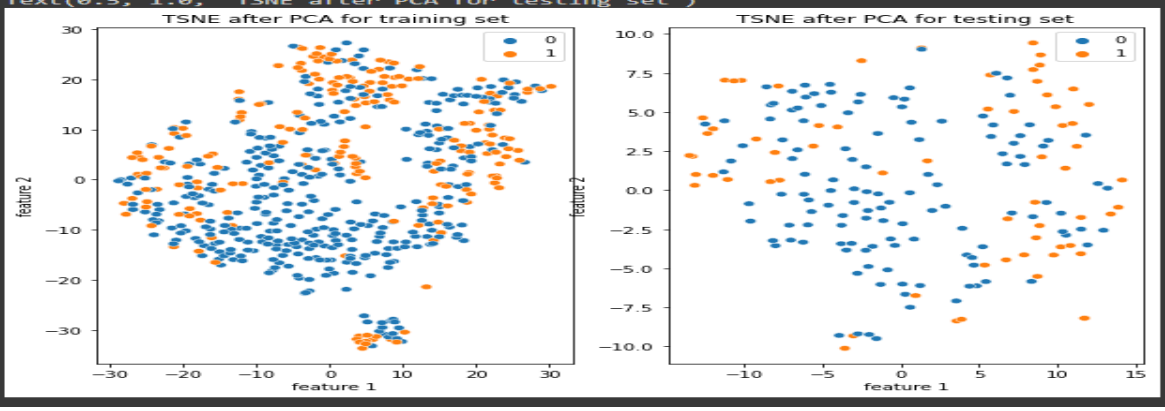
```
Accuracy of KNN:  0.7643979057591623
Accuracy of LR:  0.7905759162303665
```

```
[ ]  bar(knn_pca,lr_pca,acc_knn,acc_lr)
```



```
[ ]  plt.subplot(1,2,1)
     tsne(x_train_pca,y_train)
     plt.title('TSNE after PCA for training set')
     plt.subplot(1,2,2)
     tsne(x_test_pca,y_test)
     plt.title('TSNE after PCA for testing set')
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The
  FutureWarning,
Text(0.5, 1.0, 'TSNE after PCA for testing set')
```



2.8. Filter & Wrapper Methods:

- It's a method to decrease the number of feature data by removing noise and redundant data and using only the relevant data. It has four types: filter method, wrapper method, embedded method, and hybrid method.
- In this project, we used only the filter method and wrapper method. In the filter method, we used in filter method chi-squared which compares two variables in a contingency table to see if they are related and f-classif function which is a univariate feature selection works by selecting the best features based on univariate statistical tests.
- In the wrapper method, which is work based on a machine learning model, we used forward feature selection, backward feature selection, and recursive feature selection. The forward feature selection works by adding a feature each time in the model and the feature with a minimum p-value is added to the subset and repeats this process until reaching the optimal subset. While Backward feature selection method it's opposite to forward

feature selection because it selects all features at the beginning and then eliminates features one by one to reach the optimal subset of features. Then, we used recursive feature selection the method is similar to backward feature selection.

- Implementation of filter method:

  1.Using for loop to define the best number of features.

  2.Apply chi2 and f-classif functions.

  3.Apply the features to K-nearest neighbors and logistic regression models.

  4.Compute the accuracy

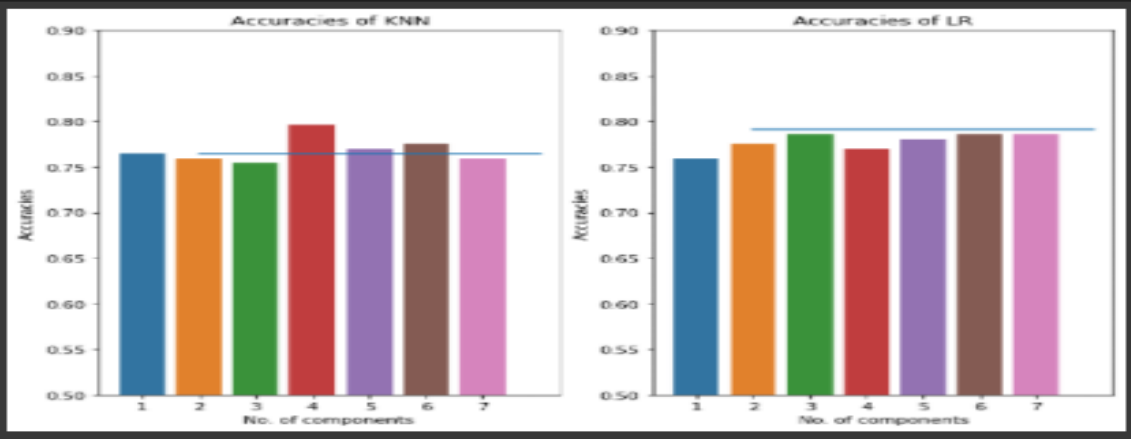
Question 4

Filter method

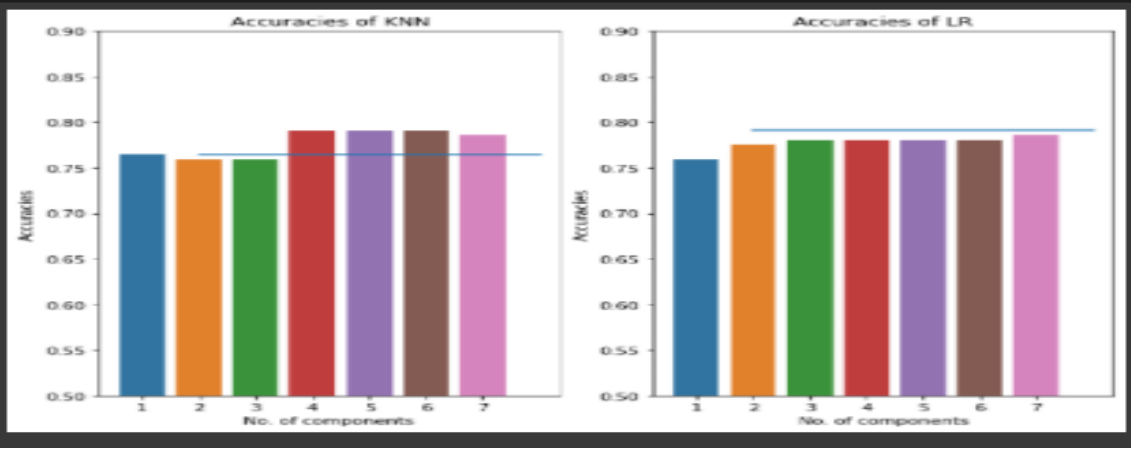
```
[ ]  knn_chi2 = []
     lr_chi2 = []
     knn_fclassif = []
     lr_fclassif = []

     for i in range(1,8):
       #chi2
       x_train_chi2,x_test_chi2,features_names_chi2 = selectkbest(chi2,i,x_train,y_train,x_test)
       #knn
       y_knn_chi2 = knn(x_train_chi2,y_train,x_test_chi2)
       acc_knn_chi2 = accuracy(y_test,y_knn_chi2)
       knn_chi2.append(acc_knn_chi2)
       #Logistic Regression
       y_lr_chi2 = lr(x_train_chi2,y_train,x_test_chi2)
       acc_lr_chi2 = accuracy(y_test,y_lr_chi2)
       lr_chi2.append(acc_lr_chi2)

       #f_classif
       x_train_fclassif,x_test_fclassif,features_names_fclassif = selectkbest(f_classif,i,x_train,y_train,x_test)
       #knn
       y_knn_fclassif = knn(x_train_fclassif,y_train,x_test_fclassif)
       acc_knn_fclassif = accuracy(y_test,y_knn_fclassif)
       knn_fclassif.append(acc_knn_fclassif)
       #Logistic Regression
       y_lr_fclassif = lr(x_train_fclassif,y_train,x_test_fclassif)
       acc_lr_fclassif = accuracy(y_test,y_lr_fclassif)
       lr_fclassif.append(acc_lr_fclassif)

     print('Accuracies of KNN with chi2: ',knn_chi2)
     print('Accuracies of LR with chi2: ',lr_chi2)
     print('Accuracies of KNN with f_classif: ',knn_fclassif)
     print('Accuracies of LR with f_classif: ',lr_fclassif)
     print('Max of KNN with chi2 index: ',np.argmax(knn_chi2),"Which are: ",knn_chi2[np.argmax(knn_chi2)])
     print('Max of LR with chi2 index: ',np.argmax(lr_chi2),"Which are: ",lr_chi2[np.argmax(lr_chi2)])
     print('Max of KNN with f_classif index: ',np.argmax(knn_fclassif),"Which are: ",knn_fclassif[np.argmax(knn_fclassif)])
     print('Max of LR with f_classif index: ',np.argmax(lr_fclassif),"Which are: ",lr_fclassif[np.argmax(lr_fclassif)])
```

```
Accuracies of KNN with chi2:  [0.6963350785340314, 0.7382198952879581, 0.7643979057591623, 0.774869109947644, 0.8010471204188482, 0.7853403141361257, 0.7643979057591623]
Accuracies of LR with chi2:  [0.6544502617801047, 0.7801047120418848, 0.7853403141361257, 0.7801047120418848, 0.7853403141361257, 0.7801047120418848, 0.7853403141361257]
Accuracies of KNN with f_classif:  [0.7643979057591623, 0.7539267015706806, 0.7696335078534031, 0.774869109947644, 0.7591623036649214, 0.7853403141361257, 0.7643979057591623]
Accuracies of LR with f_classif:  [0.7591623036649214, 0.774869109947644, 0.7853403141361257, 0.7905759162303665, 0.7801047120418848, 0.7801047120418848, 0.7853403141361257]
Max of KNN with chi2 index:  4 Which are:  0.8010471204188482
Max of LR with chi2 index:  2 Which are:  0.7853403141361257
Max of KNN with f_classif index:  5 Which are:  0.7853403141361257
Max of LR with f_classif index:  3 Which are:  0.7905759162303665
```

```
[ ]  #Forward feature selection
     bar(knn_forward,lr_forward,acc_knn_pca,acc_lr_pca)
```




```
[ ]  #Backward feature selection
     bar(knn_backward,lr_backward,acc_knn_pca,acc_lr_pca)
```

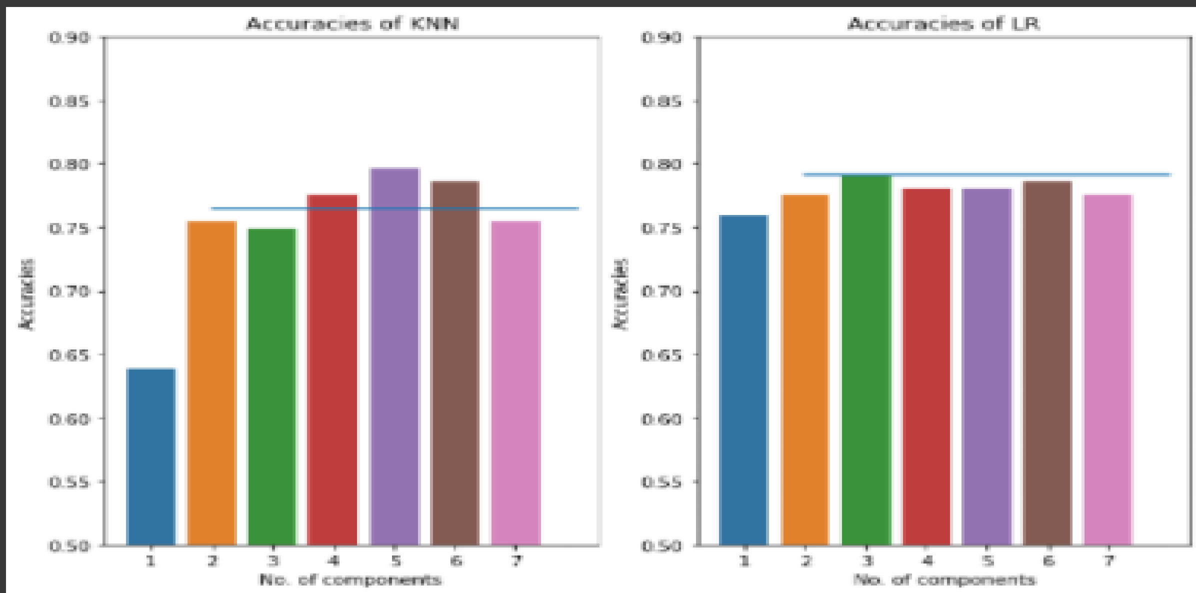


Recursive feature selection is the best

```
[ ]  #KNN
     x_train_knn_rfe,x_test_knn_rfe,features_names_knn_rfe = recursive(DecisionTreeClassifier(),5,x_train,y_train,x_test)
     y_knn_rfe = knn(x_train_knn_rfe,y_train,x_test_knn_rfe)
     acc_knn_rfe = accuracy(y_test,y_knn_rfe)

     #Logistic Regression
     x_train_lr_rfe,x_test_lr_rfe,features_names_lr_rfe = recursive(LogisticRegression(),3,x_train,y_train,x_test)
     y_lr_rfe = lr(x_train_lr_rfe,y_train,x_test_lr_rfe)
     acc_lr_rfe = accuracy(y_test,y_lr_rfe)

     print('Best accuracy of KNN with recursive feature selections',acc_knn_rfe)
     print('Features name used in KNN: ', features_names_knn_rfe)
     print('Best accuracy of LR with recursive feature selections',acc_lr_rfe)
     print('Features name used in LR: ', features_names_lr_rfe)
```

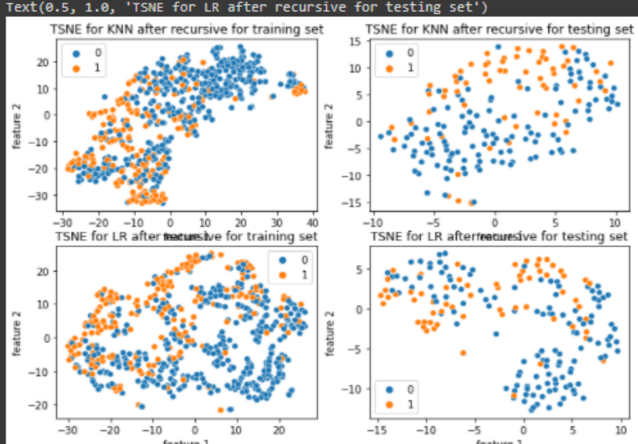
```
Best accuracy of KNN with recursive feature selections 0.7958115183246073
Features name used in KNN:  ['Glucose' 'BloodPressure' 'BMI' 'DiabetesPedigreeFunction' 'Age']
Best accuracy of LR with recursive feature selections 0.7905759162303665
Features name used in LR:  ['Pregnancies' 'Glucose' 'BMI']
```

```
#Recursive feature selection
bar(knn_rfe,lr_rfe,acc_knn_pca,acc_lr_pca)
```




```
#TSNE
plt.subplot(2,2,1)
tsne(x_train_knn_rfe,y_train)
plt.title('TSNE for KNN after recursive for training set')
plt.subplot(2,2,2)
tsne(x_test_knn_rfe,y_test)
plt.title('TSNE for KNN after recursive for testing set')
plt.subplot(2,2,3)
tsne(x_train_lr_rfe,y_train)
plt.title('TSNE for LR after recursive for training set')
plt.subplot(2,2,4)
tsne(x_test_lr_rfe,y_test)
plt.title('TSNE for LR after recursive for testing set')
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
Text(0.5, 1.0, 'TSNE for LR after recursive for testing set')
```



2.9. K-Means with PCA:

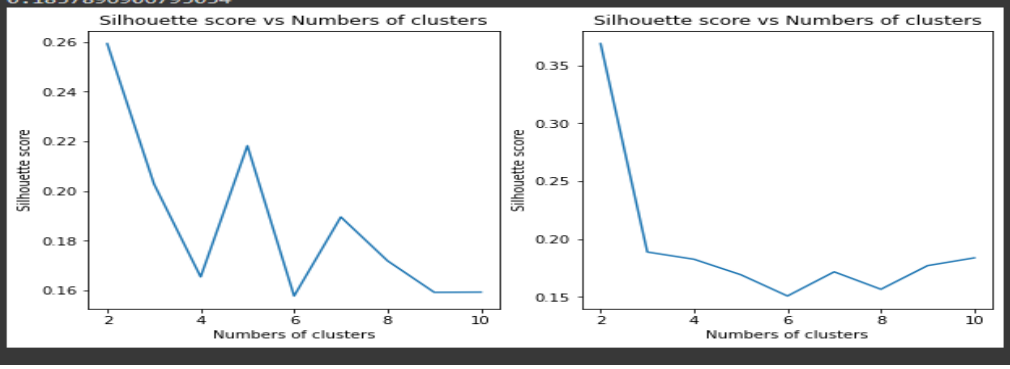
- We apply PCA on data and fit with K-Means model to get the optimum silhouette score.

Question 5

KMeans for PCA

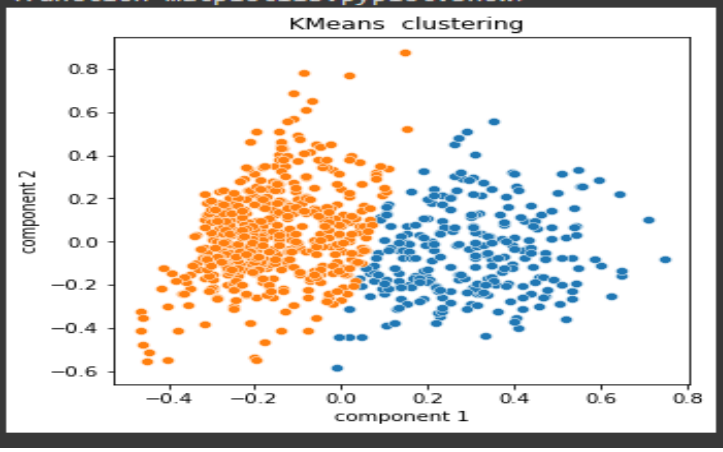
```
[ ]  figsize(10,5)
     plt.subplot(1,2,1)
     Silhouette(x_train_pca)
     plt.subplot(1,2,2)
     Silhouette(x_test_pca)
```

```
0.1837896906795054
```




```
[ ]  #KMeans  clustering
     x_train_pca_new,x_test_pca_new = pca(2,x_train_pca,x_test_pca)
     x_pca_fs = np.concatenate((x_train_pca_new,x_test_pca_new))
     y_kmeans = kmeans(2,x_pca_fs)
     figsize(5,5)
     sns.scatterplot(x_pca_fs[y_kmeans == 0,0],x_pca_fs[y_kmeans == 0,1])
     sns.scatterplot(x_pca_fs[y_kmeans == 1,0],x_pca_fs[y_kmeans == 1,1])
     plt.title(' KMeans  clustering')
     plt.xlabel("component 1")
     plt.ylabel("component 2")
     plt.show
```

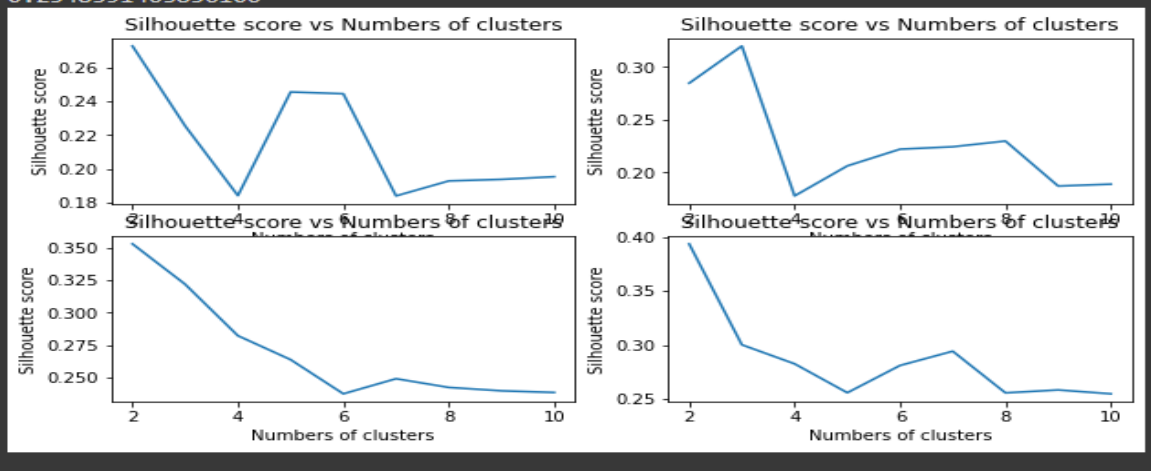
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarn
  FutureWarning
<function matplotlib.pyplot.show>
```



KMeans for forward feature selection

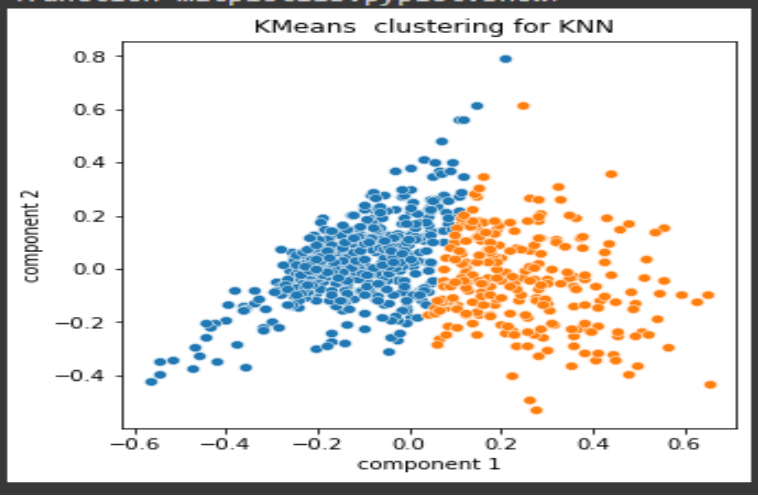
```
[ ]  figsize(10,5)
     plt.subplot(2,2,1)
     Silhouette(x_train_knn_rfe)
     plt.subplot(2,2,2)
     Silhouette(x_test_knn_rfe)
     plt.subplot(2,2,3)
     Silhouette(x_train_lr_rfe)
     plt.subplot(2,2,4)
     Silhouette(x_test_lr_rfe)
```

```
0.2548591463856106
```




```
x_train_pca_new,x_test_pca_new = pca(2,x_train_knn_rfe,x_test_knn_rfe)
x_rfe_pca = np.concatenate((x_train_pca_new,x_test_pca_new))
y_kmeans = kmeans(2,x_rfe_pca)
figsize(5,5)
sns.scatterplot(x_rfe_pca[y_kmeans == 0,0],x_rfe_pca[y_kmeans == 0,1])
sns.scatterplot(x_rfe_pca[y_kmeans == 1,0],x_rfe_pca[y_kmeans == 1,1])
plt.title(' KMeans  clustering for KNN')
plt.xlabel("component 1")
plt.ylabel("component 2")
plt.show
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureW
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureW
  FutureWarning
<function matplotlib.pyplot.show>
```



```
x_train_pca_new,x_test_pca_new = pca(2,x_train_lr_rfe,x_test_lr_rfe)
x_rfe_pca_lr = np.concatenate((x_train_pca_new,x_test_pca_new))
y_kmeans = kmeans(2,x_rfe_pca_lr)
figsize(5,5)
sns.scatterplot(x_rfe_pca_lr[y_kmeans == 0,0],x_rfe_pca_lr[y_kmeans == 0,1])
sns.scatterplot(x_rfe_pca_lr[y_kmeans == 1,0],x_rfe_pca_lr[y_kmeans == 1,1])
plt.title(' KMeans  clustering for the training set')
plt.xlabel("component 1")
plt.ylabel("component 2")
plt.show
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
<function matplotlib.pyplot.show>
```

2.10. SOM:

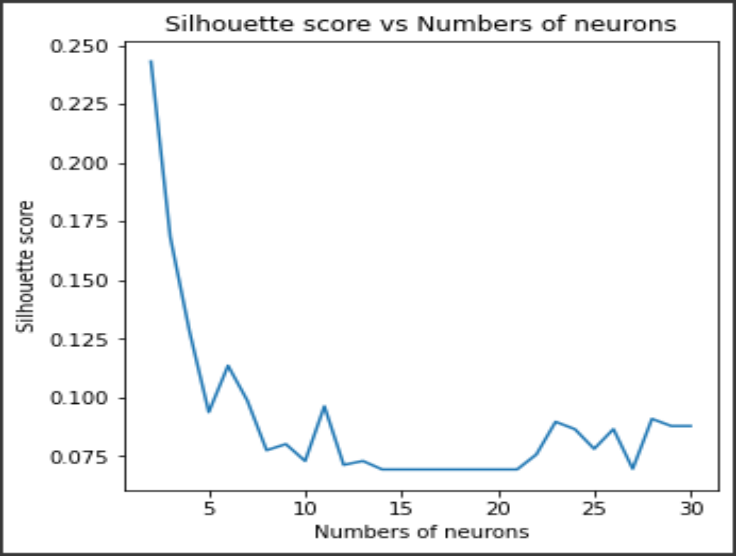
- ✓ SOM is an unsupervised learning technique used for clustering and mapping techniques to map multidimensional data onto lower-dimensional.
- We used the best 7 features From the data by using pca from question 3 : which named : (x_pca)
- We use SOM library from sklearn to plot the silhouette and get the best number of neurons.
- We made 2 nested for loops : which iterate on each neuron and determine the silhouette score of each neuron.
- Plot the silhouette score vs the number of neurons.
- From the previous graph , The best number of neurons = 2
  - We used (argmax) function to find the value of the optimal number of neurons

- The initial and final Neuron positions at 2 clusters
- We used MiniSom library to plot the initial and final positions of the neurons.
- The initial position:
  - We plot the initial position before starting the iterations of training the data and each neuron represents a cluster.
    - The final positions:
  - We plot the final position after 2000 iteration of training the data.

Question 6

a) Plot the silhouette score vs the number of neurons

```
[ ]  # we used the best features from q3 which called : (x_pca)
     som_x_pca=silhoutte_n_neuron(x_pca)
```




```
[ ]  print(x_pca.shape)
     print(x_pca.shape)
```

```
(767, 7)
(767, 7)
```

b) the value of the optimal number of neurons

```
[ ]  best_x_pca=np.argmax(som_x_pca)
     som_x_pca[best_x_pca]
```

```
0.24317409249397146
```

c) Plot the initial and final Neuron positions

- Plot the initial Neuron positions

```
[ ] x_pca.shape
```

```
(767, 7)
```

```
[ ] # Initialization and training
    som_shape = (2, 1)
    som = MiniSom(som_shape[0],som_shape[1], x_pca.shape[1], random_seed=0)
    intial = np.array(som.get_weights())
    som.train_batch(x_pca,2000, verbose=True)
    finial = np.array(som.get_weights())
```

```
 [  816 / 2000 ]  41% - 0:00:00 left /usr/local/lib/python3.7/dist-packages/minisom.py:160: UserWarning: Warning: sigma is too high for the dimension of the map.
  warn('Warning: sigma is too high for the dimension of the map.')
 [ 2000 / 2000 ] 100% - 0:00:00 left
 quantization error: 0.37416593953292715
```

```
[ ] # each neuron represents a cluster
    winner_coordinates = np.array([som.winner(x) for x in x_pca]).T
    # with np.ravel_multi_index we convert the bidimensional
    # coordinates to a monodimensional index
    cluster_index = np.ravel_multi_index(winner_coordinates, som_shape)
```

```
[ ] cluster_index
```

```
array([0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1])
```

```
[ ] intial.shape
```

```
(2, 1, 7)
```

```
[ ]
    intial = intial.reshape(intial.shape[0],x_pca.shape[1])
    finial = finial.reshape(finial.shape[0],x_pca.shape[1])
```

```
[ ] intial
```

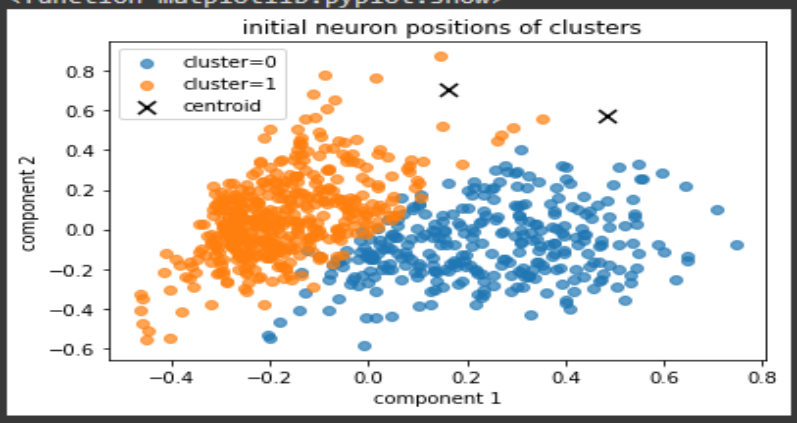
```
array([[ 0.16069505,  0.70840777,  0.33829913,  0.14775635, -0.25132995,
         0.48028639, -0.20546417],
       [ 0.48453665,  0.57344839, -0.14415709,  0.36079942,  0.03573663,
         0.08415609,  0.52636901]])
```

```
[ ] finial
```

```
array([[ 0.19433634, -0.0733264 ,  0.02816359, -0.02230897,  0.0042302 ,
        -0.04862309,  0.03654185],
       [-0.14324222,  0.12814719,  0.03732493,  0.00949006, -0.02485965,
        -0.04365312, -0.00722865]])
```

```
[ ] import matplotlib.pyplot as plt
    %matplotlib inline

    # plotting the clusters using the first 2 dimentions of the data
    for c in np.unique(cluster_index):
        plt.scatter(x_pca[cluster_index == c, 0],
                    x_pca[cluster_index == c, 1], label='cluster='+str(c), alpha=.7)

    # plotting centroids
    plt.scatter(intial[:, 0], intial[:, 1], marker='x',
                    s=80, linewidths=35, color='k', label='centroid')
    plt.legend();
    plt.title(' initial neuron positions of clusters')
    plt.xlabel("component 1")
    plt.ylabel("component 2")
    plt.show
```

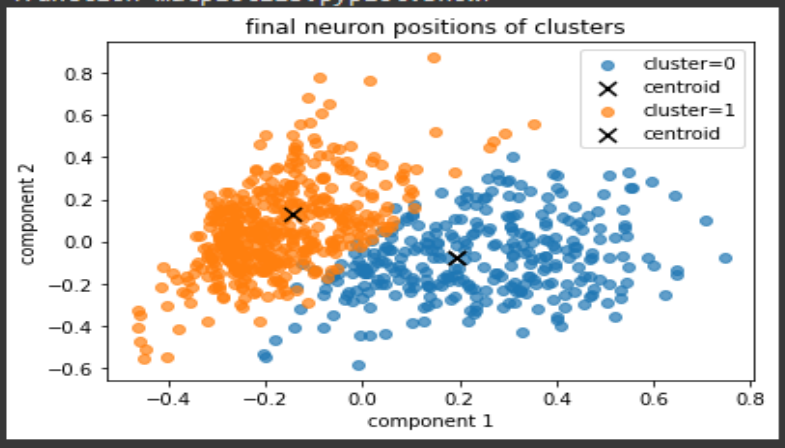
```
<function matplotlib.pyplot.show>
```



Plot the final Neuron positions

```
[ ]  import matplotlib.pyplot as plt
     %matplotlib inline

     # plotting the clusters using the first 2 dimentions of the data
     for c in np.unique(cluster_index):
         plt.scatter(x_pca[cluster_index == c, 0],
                     x_pca[cluster_index == c, 1], label='cluster='+str(c), alpha=.7)

     # plotting centroids
         plt.scatter(finial[:, 0], finial[:, 1], marker='x',
                     s=80, linewidths=35, color='k', label='centroid')

     plt.legend();
     plt.title(' final neuron positions of clusters')
     plt.xlabel("component 1")
     plt.ylabel("component 2")
     plt.show
```

```
<function matplotlib.pyplot.show>
```



2.11. DBSCAN Algorithm:

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an unsupervised machine learning technique used to identify clusters of varying shape in a data set
- We use DBSCAN library from sklearn which works on the 2 clusters from Q6.
- We initialize DBSCAN with our values for epsilon and minpoints:
  - ε (epsilon or “eps”): the maximum distance two points can be from one another while still belonging to the same cluster.
  - Minimum samples (“MinPoints”): indicates the minimum number of samples that should be within the epsilon range.

- We created 2 nested for loops one to iterate on the epsilon values in range from 0.3 to 0.7 and one to iterate on the minpoints values in range from 2 to 15 and fit the data to DBSCAN and determined the silhouette score.
- We print a data frame of which contains lists of epsilon, minpoints , sillouhtte_score and the number of clusters .

- We noticed from the iteration:
  - The best 10 combinations of epsilon and minpoints that brings us closer to the highest silhouette score When the number of clusters =2.
  - epsilon vs number of clusters for the best 10 combinations
  - minpoint vs number of clusters for The best 10 combinations

Question 7

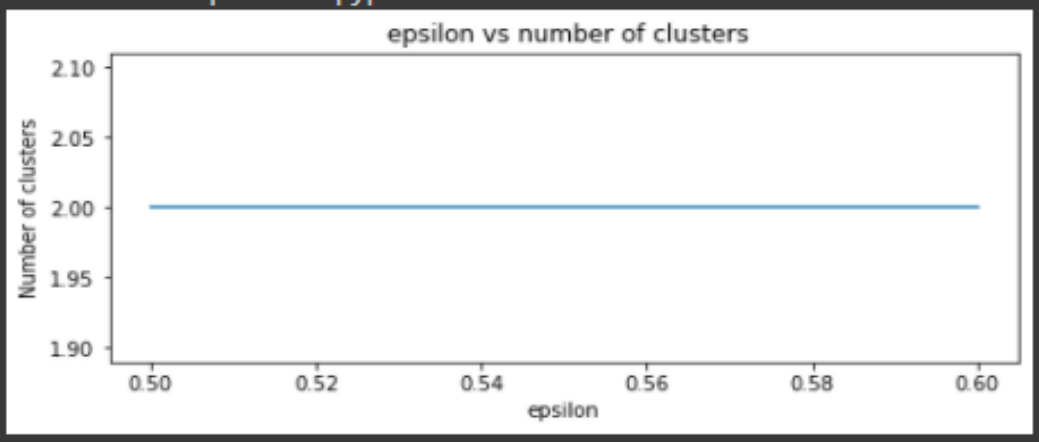
```
[ ] #find DBSCAN optimal eps and min-samples
    from tqdm import tqdm
    from sklearn.cluster import KMeans, DBSCAN
    from sklearn.utils.multiclass import unique_labels
    epsList, msList, accList ,pred_N_Cluster= list(), list(), list(),list()
    eps_minPoints=[]

    for eps in tqdm(np.arange(0.3,0.7,0.1)):
      for ms in range(2, 15,1):
        model = DBSCAN(eps=eps, min_samples=ms)
        predLabels = model.fit_predict(x)
        # print(predLabels)
        score = silhouette_score(x, predLabels, random_state=0)
        epsList.append(eps)
        msList.append(ms)
        accList.append(score)
        eps_minPoints.append((eps,ms))
        pred_N_Cluster.append(len(unique_labels(predLabels)))

    epsList, msList, accList = np.array(epsList), np.array(msList), np.array(accList)
    print(eps_minPoints)
    print(accList)
    print(pred_N_Cluster)
```

```
100%|██████████| 4/4 [00:02<00:00,  1.75it/s]
[(0.3, 2), (0.3, 3), (0.3, 4), (0.3, 5), (0.3, 6), (0.3, 7), (0.3, 8), (0.3, 9), (0.3, 10), (0.3, 11), (0.3, 12), (0.3, 13), (0.3, 14), (0.4, 2), (0.4, 3), (0
[0.12427501 0.18144829 0.25570374 0.25550342 0.25071393 0.25569042
 0.25546474 0.24137344 0.23483044 0.23167176 0.22396659 0.28383905
 0.28248988 0.28693763 0.41150418 0.40738576 0.39902473 0.39902473
 0.39902473 0.39475775 0.39208884 0.39133404 0.39133404 0.39133404
 0.39133404 0.38693072 0.44374417 0.44374417 0.48500793 0.48500793
 0.48500793 0.48500793 0.48228732 0.48228732 0.48407501 0.48407501
 0.48407501 0.48407501 0.48407501 0.44374417 0.44374417 0.44374417
 0.44374417 0.44374417 0.44374417 0.44374417 0.48500793 0.48500793
 0.48500793 0.48500793 0.48500793 0.48500793]
[6, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

epsilon vs number of clusters for the 10 combinations

```
[ ]  plt.figure(figsize=(8,3))
     plt.plot(data['epsilon'],data['N_Cluster'])
     plt.title(' epsilon vs number of clusters')
     plt.xlabel("epsilon")
     plt.ylabel("Number of clusters")
     plt.show
```

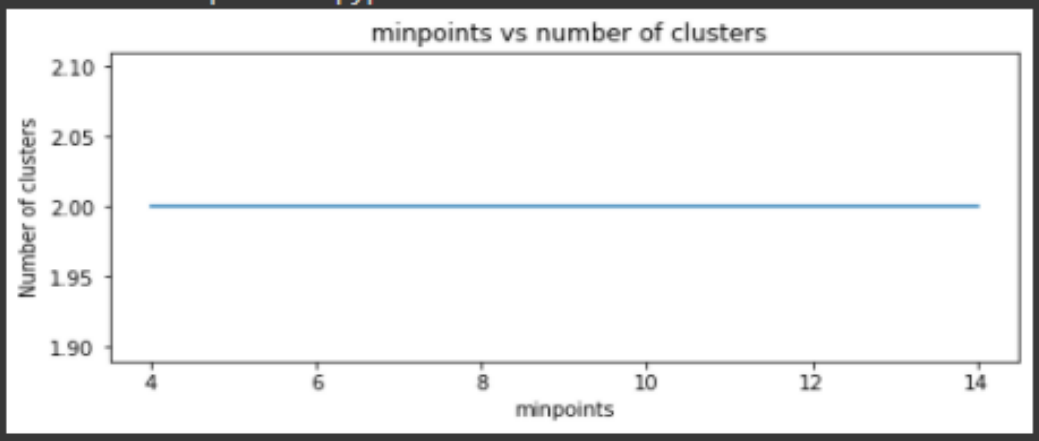
```
<function matplotlib.pyplot.show>
```



minpoints vs number of clusters for the 10 combinations

```
[ ]  plt.figure(figsize=(8,3))
     plt.plot(data['minpoints'],data['N_Cluster'])
     plt.title(' minpoints vs number of clusters')
     plt.xlabel("minpoints")
     plt.ylabel("Number of clusters")
     plt.show
```

```
<function matplotlib.pyplot.show>
```



Part 3: Conclusion

- Transforming data from high dimensions to low dimensions for its dense where the similar points collected together.
- gaining information by using PCA and extract new features from the original data.
- dropping features that can explained by another features will introduce the date more clearly.