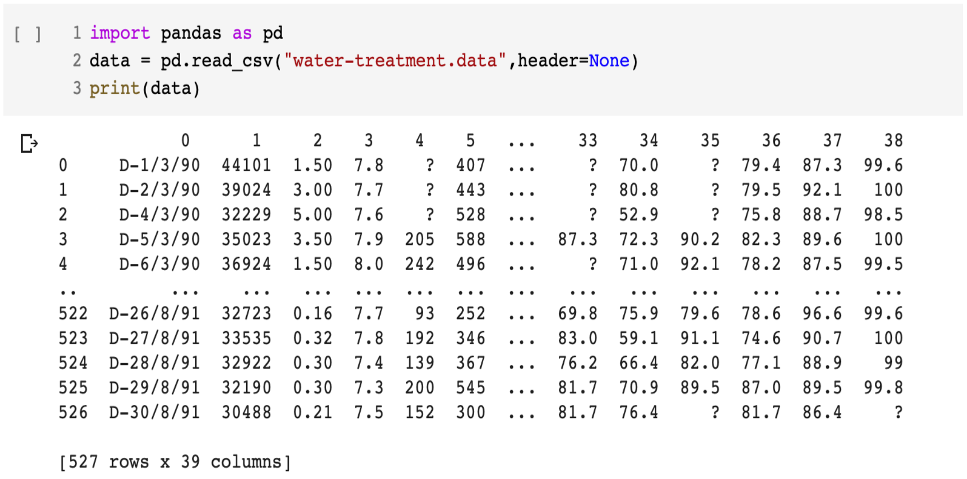
CS-535
INTRODUCTION TO DATA MINING

Aditya Bhagwat B00811694

1. Clean up the data set. This includes filling up the missing values and normalizing all the data items. Please state clearly the methods you use for filling up the missing values and normalizing the values in English to answer this question.

Solution :

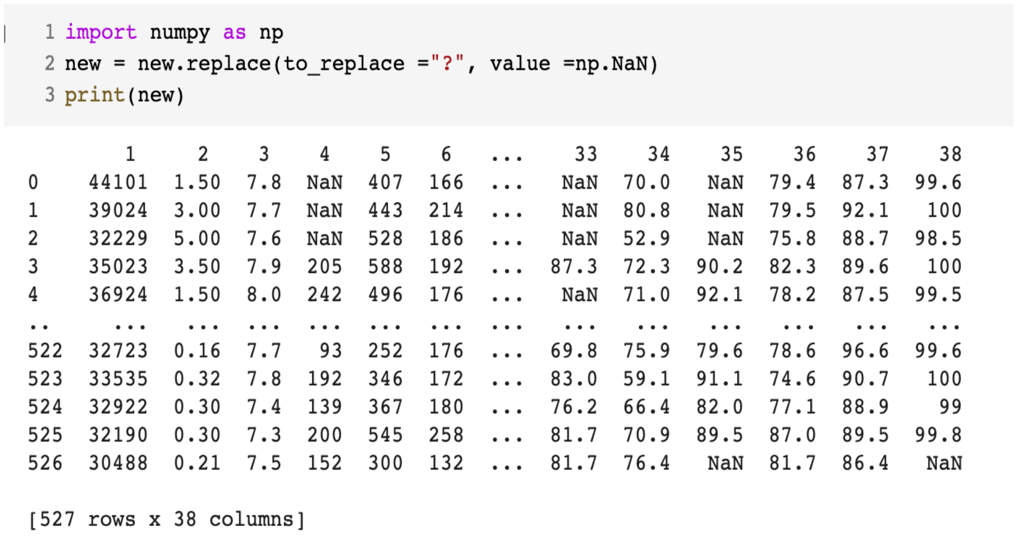
Initial Look of the dataset.



```
import pandas as pd
data = pd.read_csv("water-treatment.data",header=None)
print(data)
```

```
             0      1     2    3    4    5  ...    33    34    35    36    37    38
0      D-1/3/90  44101  1.50  7.8    ?  407  ...     ?  70.0     ?  79.4  87.3  99.6
1      D-2/3/90  39024  3.00  7.7    ?  443  ...     ?  80.8     ?  79.5  92.1   100
2      D-4/3/90  32229  5.00  7.6    ?  528  ...     ?  52.9     ?  75.8  88.7  98.5
3      D-5/3/90  35023  3.50  7.9  205  588  ...  87.3  72.3  90.2  82.3  89.6   100
4      D-6/3/90  36924  1.50  8.0  242  496  ...     ?  71.0  92.1  78.2  87.5  99.5
..          ...    ...   ...  ...  ...  ...  ...   ...   ...   ...   ...   ...   ...
522  D-26/8/91  32723  0.16  7.7   93  252  ...  69.8  75.9  79.6  78.6  96.6  99.6
523  D-27/8/91  33535  0.32  7.8  192  346  ...  83.0  59.1  91.1  74.6  90.7   100
524  D-28/8/91  32922  0.30  7.4  139  367  ...  76.2  66.4  82.0  77.1  88.9    99
525  D-29/8/91  32190  0.30  7.3  200  545  ...  81.7  70.9  89.5  87.0  89.5  99.8
526  D-30/8/91  30488  0.21  7.5  152  300  ...  81.7  76.4     ?  81.7  86.4     ?

[527 rows x 39 columns]
```

I. Firstly we need to replace all the missing and unhandled values with the NumPy value as np.NaN



```
import numpy as np
new = new.replace(to_replace ="?", value =np.NaN)
print(new)
```

```
         1     2    3    4    5    6  ...    33    34    35    36    37    38
0    44101  1.50  7.8  NaN  407  166  ...   NaN  70.0   NaN  79.4  87.3  99.6
1    39024  3.00  7.7  NaN  443  214  ...   NaN  80.8   NaN  79.5  92.1   100
2    32229  5.00  7.6  NaN  528  186  ...   NaN  52.9   NaN  75.8  88.7  98.5
3    35023  3.50  7.9  205  588  192  ...  87.3  72.3  90.2  82.3  89.6   100
4    36924  1.50  8.0  242  496  176  ...   NaN  71.0  92.1  78.2  87.5  99.5
..     ...   ...  ...  ...  ...  ...  ...   ...   ...   ...   ...   ...   ...
522  32723  0.16  7.7   93  252  176  ...  69.8  75.9  79.6  78.6  96.6  99.6
523  33535  0.32  7.8  192  346  172  ...  83.0  59.1  91.1  74.6  90.7   100
524  32922  0.30  7.4  139  367  180  ...  76.2  66.4  82.0  77.1  88.9    99
525  32190  0.30  7.3  200  545  258  ...  81.7  70.9  89.5  87.0  89.5  99.8
526  30488  0.21  7.5  152  300  132  ...  81.7  76.4   NaN  81.7  86.4   NaN

[527 rows x 38 columns]
```

II. Now we replace all the np.NaN values with respective median values column-wise.

```
for i in range(1,39):
  median = new[i].median()
  new[i].fillna(median, inplace=True)

print(new)
```

```
        1     2    3      4    5    6  ...    33    34    35    36    37    38
0    44101  1.50  7.8  182.5  407  166  ...  85.4  70.0  90.2  79.4  87.3  99.6
1    39024  3.00  7.7  182.5  443  214  ...  85.4  80.8  90.2  79.5  92.1   100
2    32229  5.00  7.6  182.5  528  186  ...  85.4  52.9  90.2  75.8  88.7  98.5
3    35023  3.50  7.9    205  588  192  ...  87.3  72.3  90.2  82.3  89.6   100
4    36924  1.50  8.0    242  496  176  ...  85.4  71.0  92.1  78.2  87.5  99.5
..     ...   ...  ...    ...  ...  ...  ...   ...   ...   ...   ...   ...   ...
522  32723  0.16  7.7     93  252  176  ...  69.8  75.9  79.6  78.6  96.6  99.6
523  33535  0.32  7.8    192  346  172  ...  83.0  59.1  91.1  74.6  90.7   100
524  32922  0.30  7.4    139  367  180  ...  76.2  66.4  82.0  77.1  88.9    99
525  32190  0.30  7.3    200  545  258  ...  81.7  70.9  89.5  87.0  89.5  99.8
526  30488  0.21  7.5    152  300  132  ...  81.7  76.4  90.2  81.7  86.4  99.7

[527 rows x 38 columns]
```

III. Now we normalize all the values using MinMax Scaler from Scikit-learn.

IV. We store the cleaned dataset in ‘MinMax.csv’

```
import pandas as pd
from sklearn import preprocessing
x = new.values.astype(float)

# Create a minimum and maximum processor object
min_max_scaler = preprocessing.MinMaxScaler()

# Create an object to transform the data to fit minmax processor
x_scaled = min_max_scaler.fit_transform(x)

# Run the normalizer on the dataframe
df_normalized = pd.DataFrame(x_scaled)

print(df_normalized)

s = pd.DataFrame(df_normalized)

s.to_csv('MinMax.csv')
```

```
            0         1         2  ...        35        36        37
0    0.680598  0.041916  0.500000  ...  0.762991  0.864198  0.993711
1    0.579121  0.086826  0.444444  ...  0.764259  0.918070  1.000000
2    0.443305  0.146707  0.388889  ...  0.717364  0.879910  0.976415
3    0.499151  0.101796  0.555556  ...  0.799747  0.890011  1.000000
4    0.537147  0.041916  0.611111  ...  0.747782  0.866442  0.992138
..        ...       ...       ...  ...       ...       ...       ...
522  0.453179  0.001796  0.444444  ...  0.752852  0.968575  0.993711
523  0.469409  0.006587  0.500000  ...  0.702155  0.902357  1.000000
524  0.457157  0.005988  0.277778  ...  0.733840  0.882155  0.984277
525  0.442526  0.005988  0.222222  ...  0.859316  0.888889  0.996855
526  0.408507  0.003293  0.333333  ...  0.792142  0.854097  0.995283

[527 rows x 38 columns]
```

CS-535

INTRODUCTION TO DATA MINING

Aditya Bhagwat B00811694

2. It is well-known that the k-means algorithm requires that the number of clusters, k, be given in advance. In this problem, we do not know the k value in advance. Propose a specific termination condition for the modified k-means when searching the true k value. State clearly your proposed condition or method in English.

Solution :

The silhouette score displays the separation distance between the resulting clusters. The range of this score is between [-1, 1]. Silhouette coefficients near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

We compare the silhouette score by iterating over the number of clusters expected and find the second best silhouette score so that the no of clusters required is optimized.

Algorithm :

```
SCORE = []
for n_cluster in range(2, 11):
kmeans = KMeans(n_clusters=n_cluster).fit(X)
label = kmeans.labels_
sil_coeff = silhouette_score(X, label, metric='euclidean')
SCORE.append(sil_coeff)

newScore = SCORE.copy()

max_value = max(newScore)
max_index = newScore.index(max_value)
newScore[max_index] = -1000
max_value = max(newScore)
max_index_2 = newScore.index(max_value)
print(SCORE)
print(max_value)
print(max_index_2)
```

CS-535
INTRODUCTION TO DATA MINING

Aditya Bhagwat B00811694

3. Implement the modified k-means algorithm with your proposed termination condition and run the algorithm using the water-treatment dataset. Please note that you must use the output format given in the description file. Report your output.

Solution :

Implementation :

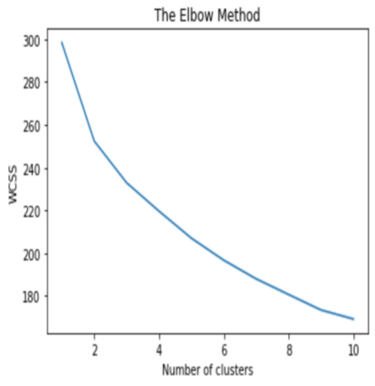
```
import numpy as numpy
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('MinMax.csv')
X = dataset.iloc[:, 1:]
from sklearn.metrics import silhouette_score
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
SCORE = []
for n_cluster in range(2, 11):
    kmeans = KMeans(n_clusters=n_cluster).fit(X)
    label = kmeans.labels_
    sil_coeff = silhouette_score(X, label, metric='euclidean')
    SCORE.append(sil_coeff)
newScore = SCORE.copy()
max_value = max(newScore)
max_index = newScore.index(max_value)
newScore[max_index] = -1000
max_value = max(newScore)
max_index_2 = newScore.index(max_value)
print(SCORE)
print(max_value)
print(max_index_2)
from sklearn.cluster import KMeans
WCSS = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-
means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    WCSS.append(kmeans.inertia_)
    print(WCSS)
plt.plot(range(1,11), WCSS)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
X = X.values
```

Aditya Bhagwat B00811694

```
print(max_index_2)
kmeans = KMeans(n_clusters = (max_index_2 + 2), init = 'k-
means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(X)
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'Pur
ple', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'mag
enta', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'blu
e', label = 'Cluster 3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1
], s = 100, c = 'yellow', label = 'Centroids')

plt.title('Clusters')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

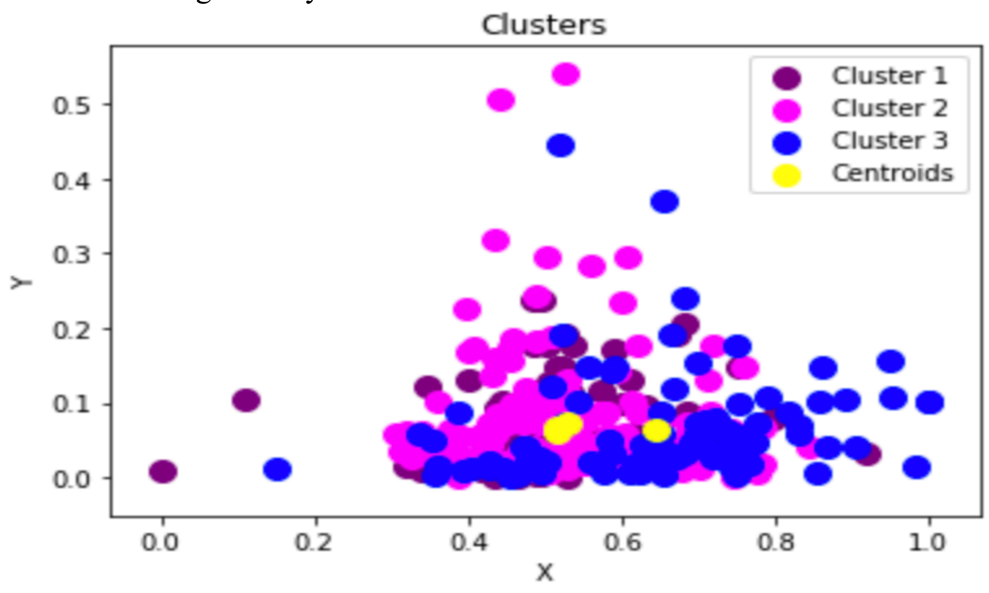
Output :

```
[0.14378752777199044, 0.11909341795676999, 0.12054306443602979, 0.10748325862081745, 0.10069971029211287, 0.09721365903768948, 0.09813582918414465, 0.10105
0.12054306443602979
2
[298.4784160885072]
[298.4784160885072, 252.46378394494963]
[298.4784160885072, 252.46378394494963, 232.94051111919913]
[298.4784160885072, 252.46378394494963, 232.94051111919913, 219.65279032247452]
[298.4784160885072, 252.46378394494963, 232.94051111919913, 219.65279032247452, 207.00515220121324]
[298.4784160885072, 252.46378394494963, 232.94051111919913, 219.65279032247452, 207.00515220121324, 196.69216097303172]
[298.4784160885072, 252.46378394494963, 232.94051111919913, 219.65279032247452, 207.00515220121324, 196.69216097303172, 187.97918655108862]
[298.4784160885072, 252.46378394494963, 232.94051111919913, 219.65279032247452, 207.00515220121324, 196.69216097303172, 187.97918655108862, 180.67068883994
[298.4784160885072, 252.46378394494963, 232.94051111919913, 219.65279032247452, 207.00515220121324, 196.69216097303172, 187.97918655108862, 180.67068883994
[298.4784160885072, 252.46378394494963, 232.94051111919913, 219.65279032247452, 207.00515220121324, 196.69216097303172, 187.97918655108862, 180.67068883994
```



```
2
```

Clustering Initially :



```
f = y_kmeans
file = open('Asciiii','w')
for n in range(527):
  file.writelines([str(n),"  ",str(f[n]),'\n'])
```

Generated Output file would be ‘Ascii_1’ and its content would look like this,

```
0  1
1  1
2  3
3  1
.   .
.   .
.   .
524  3
525  3
526  3
```

CS-535
INTRODUCTION TO DATA MINING

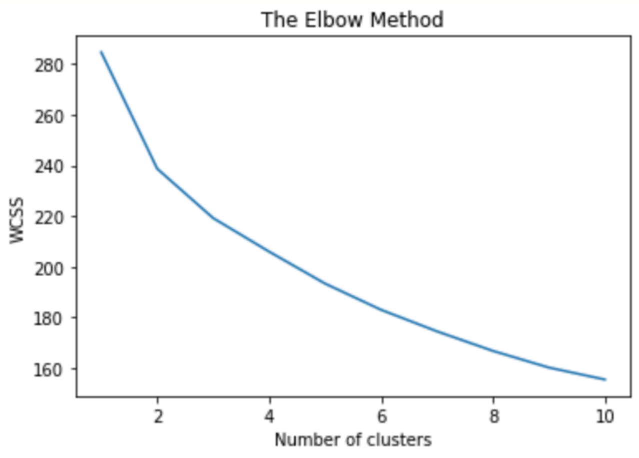
Aditya Bhagwat B00811694

4. Apply the PCA method you implemented in the first assignment to this dataset. Then apply the implemented modified k-means method above to this reduced data set to report the output. Please follow the same protocol of the output format specified in the description file.

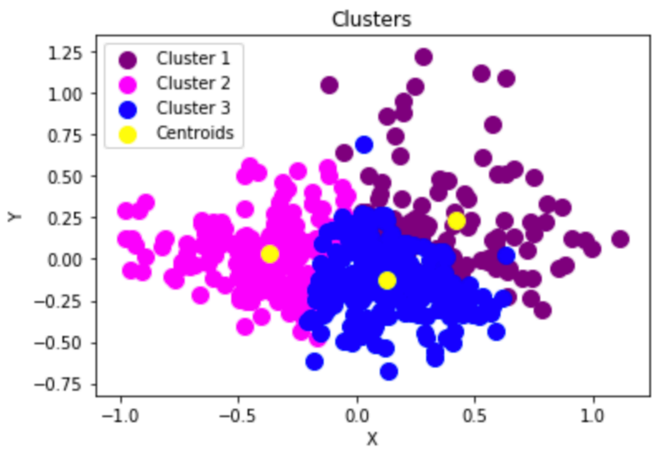
   Solution :

   Implementation :

```
#PCA
import numpy as numpy
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('MinMax.csv')
X = dataset.iloc[:, 1:]
from sklearn.decomposition import PCA
pca = PCA(n_components = 19)
Y = pca.fit_transform(X)
Y = pca.transform(X)
variance = pca.explained_variance_ratio_
sum = 0
j = 0
for i in variance:
    if(sum <= 0.95):
        sum = sum + i
        j = j + 1
print(j)
print(sum)
print(variance)
j = pd.DataFrame(Y)
j.to_csv('PCARed.csv')
print(j.shape)

import numpy as numpy
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('PCARed.csv')
X = dataset.iloc[:, 1:]
from sklearn.metrics import silhouette_score
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
SCORE = []
for n_cluster in range(2, 11):
    kmeans = KMeans(n_clusters=n_cluster).fit(X)
    label = kmeans.labels_
    sil_coeff = silhouette_score(X, label, metric='euclidean')
    SCORE.append(sil_coeff)
newScore = SCORE.copy()
max_value = max(newScore)
max_index = newScore.index(max_value)
newScore[max_index] = -1000
max_value = max(newScore)
max_index_2 = newScore.index(max_value)
print(SCORE)
print(max_value)
print(max_index_2)
from sklearn.cluster import KMeans
WCSS = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-
means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    WCSS.append(kmeans.inertia_)
    print(WCSS)

plt.plot(range(1,11), WCSS)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
X = X.values
print(max_index_2)
kmeans = KMeans(n_clusters = (max_index_2 + 2), init = 'k-
means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(X)
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = '
Purple', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = '
magenta', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = '
blue', label = 'Cluster 3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:
, 1], s = 100, c = 'yellow', label = 'Centroids')

plt.title('Clusters')

plt.xlabel('X')

plt.ylabel('Y')
plt.legend()
plt.show()
```



1



Storing Results as ‘Ascii_2’ :

```
f = y_kmeans
file = open('Ascii_2','w')
for n in range(527):
  file.writelines([str(n),"  ",str(f[n]),'\n'])
```

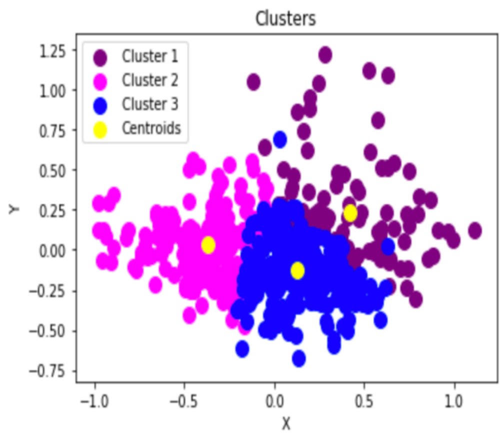
5. Compare the two clustering results and analyze any differences that you have observed and state why there is such difference if there is or why there is no difference if there is no.

Solution :

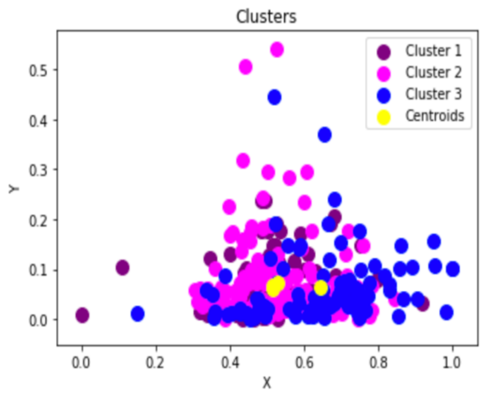
K-means has several drawbacks like the number of clusters has to be defined in advance and the algorithm is dependent upon the starting centroid locations. Another weakness, which is common to clustering in general, concerns with the visualization of determined clusters.

A possible solution is to preprocess the data first using PCA. After applying PCA to the dataset we use the principal components of the data and map them into a new feature space. Then, the modified k-means algorithm is applied to the data in the feature space. This is done so that we must be able to distinguish the different clusters clearly.

After PCA and applied K-means :



Before applying K-means :



CS-535
INTRODUCTION TO DATA MINING

Aditya Bhagwat B00811694

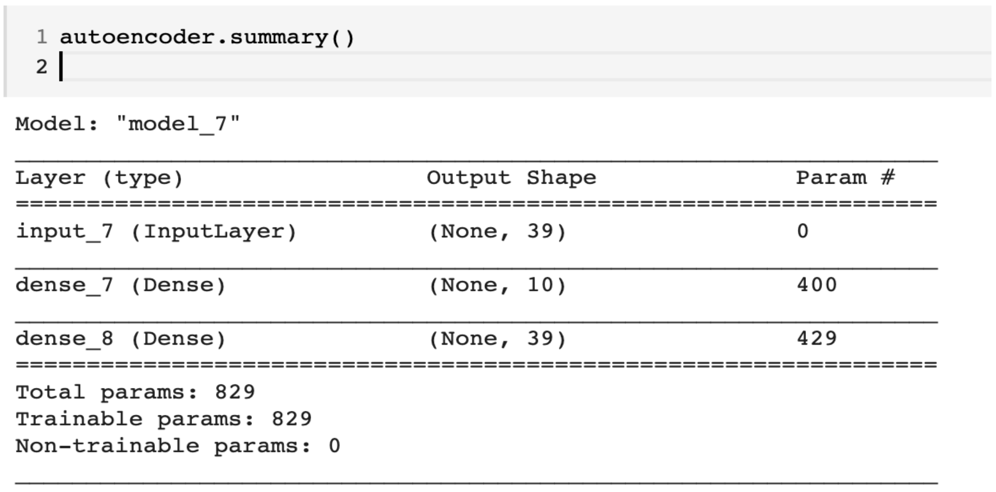
6. Implement an autoencoder (either shallow or deep) for dimensionality reduction and apply the implemented autoencoder to the given dataset. Report the dimensionality reduction result using the autoencoder and discuss the difference between PCA and autoencoder for dimensionality reduction with this dataset.

Solution :

Layers :

```
encoding_dim = 10
input_img = Input(shape=(39,))
encoded = Dense((encoding_dim), activation='relu')(input_img)
decoded = Dense(39, activation='sigmoid')(encoded)
autoencoder = Model(input_img, decoded)
encoder = Model(input_img, encoded)
encoded_input = Input(shape=(encoding_dim,))
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
autoencoder.fit(data ,data, epochs=50,batch_size=200,shuffle=True)
encoded_imgs = encoder.predict(data)
```

Summary :



```
autoencoder.summary()
```

```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 39)                0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                400
_________________________________________________________________
dense_8 (Dense)              (None, 39)                429
=================================================================
Total params: 829
Trainable params: 829
Non-trainable params: 0
_________________________________________________________________
```

Training :

```
Epoch 1/50
528/528 [==============================] - 0s 501us/step - loss: nan
Epoch 2/50
528/528 [==============================] - 0s 22us/step - loss: nan
Epoch 3/50
```

```
528/528 [==============================] - 0s 19us/step - loss: nan
Epoch 4/50
         .
         .
         .
         .
         .
         .
Epoch 47/50
528/528 [==============================] - 0s 18us/step - loss: nan
Epoch 48/50
528/528 [==============================] - 0s 18us/step - loss: nan
Epoch 49/50
528/528 [==============================] - 0s 20us/step - loss: nan
Epoch 50/50
528/528 [==============================] - 0s 21us/step - loss: nan
```

Differences :

1. PCA Algorithm is essentially a linear transformation whereas Auto-encoders are capable of modelling complex non-linear functions.

2. PCA features are totally linearly uncorrelated with each other since features are projections onto the orthogonal basis. But auto-encoded features might have correlations since they are just trained for accurate reconstruction.

3. PCA is faster and computationally cheaper than autoencoders.

4. A Single layered autoencoder with a linear activation function is very similar to PCA.

5. Autoencoder is prone to overfitting due to high number of parameters.

6. PCA reduces the dimensions to 19 whereas Autoencoders reduces the dimensions to 10.