ICP6

Answer 1:

Program Description: A program**,** to apply k means clustering in data set provided below:

Data Set: https://umkc.box.com/s/a9lzu9qoqfkbhjwk5nz9m6dyybhl1wqy

i) Remove all null values by the mean.
ii) Use of the elbow method to find a good number of clusters with KMeans Algorithm.

This program also calculates the silhouette score for the above clustering and trying feature scaling to see if it will improve the Silhouette score. Applying PCA on the same dataset.

Explanation:

Imported the necessary libraries and created our environment.
Importing **pandas** library as we will be doing dataframe manipulation.
The **matplotlib** is a plotting library for the python programming language and matplotlib.pyplot is a collection of command style function which makes it work like MATLAB.
Principal Component Analysis (PCA) which is a linear dimensionality reduction technique.
Seaborn uses matplotlib to draw its plots.

```
# Imported the necessary libraries and created our environment.
# Importing pandas library as we will be doing dataframe manipulation.
import pandas as pd
#Importing pandas library as we will be doing dataframe manipulation.
import matplotlib.pyplot as plt
# Importing Principal Component Analaysis(PCA) which is a linear dimensionality reduction technique.
from sklearn.decomposition import PCA
# seaborn uses matplotlib to draw its plots.
import seaborn as sns
```

Step 1) Data collection and Engineering:
Below part is to import 'CC.csv' file into dataframe. We are using pd.read_csv() method of Pandas for that. It creates a Dataframe from a csv file.

```
# Next is importing 'CC.csv' file into dataframe.
# We are using pd.read_csv() method of Pandas for that. It creates a Dataframe from a csv file.
dataset = pd.read_csv('CC.csv')
# Printing the number of rows and column in Data set
print('The number of rows and column in Dataset:', dataset.shape,'\n')
```

Output: We have 8950 rows and 18 column

```
The number of rows and column in Dataset: (8950, 18)
```

In our data, the CUST_ID column is just the index of the row and does not look that much necessary. So removed the CUST_ID.

```
# to drop the CUST_ID column
dataset = dataset.drop('CUST_ID', axis=1)
```

After that we will check the total numbers of null values in our dataset and will replace the null values with the mean. For this we used **fillna()** function to fill out the missing values in the given series object.

```
# Check total number of null values
print("Total number of null values:  ", + dataset.isnull().sum().sum(),'\n')

# Replace all the null with mean
dataset.fillna(dataset.mean(), inplace=True)
print("Total number of null after mean replace:  ", + dataset.isnull().sum().sum(),'\n')
```

OUTPUT:

```
Total number of null values:   314

Total number of null after mean replace:   0
```

In our dataset the total number of null values is 314, which are replaced by the mean and the total number of null after the mean is 0.

Step 2) Use of Elbow method with K means Algorithm:

The main use of Elbow method is to estimate optimal number of Clusters to be used for K means**.**

I have defined a method **elb_met** for this purpose. The required parameter is Dataframe on which Elbow methods needs to be applied.

This method runs k-means clustering on the dataset for a range of values for k (say from 1-10) and then for each value of k computes Within Cluster Sum of Squared Errors (WCSS). WCSS is the sum of squares of the distances of each data point in all clusters to their respective centroids. For this we have used for loop in range (1,10).

We also Visualized the plot between WCSS and number of clusters.

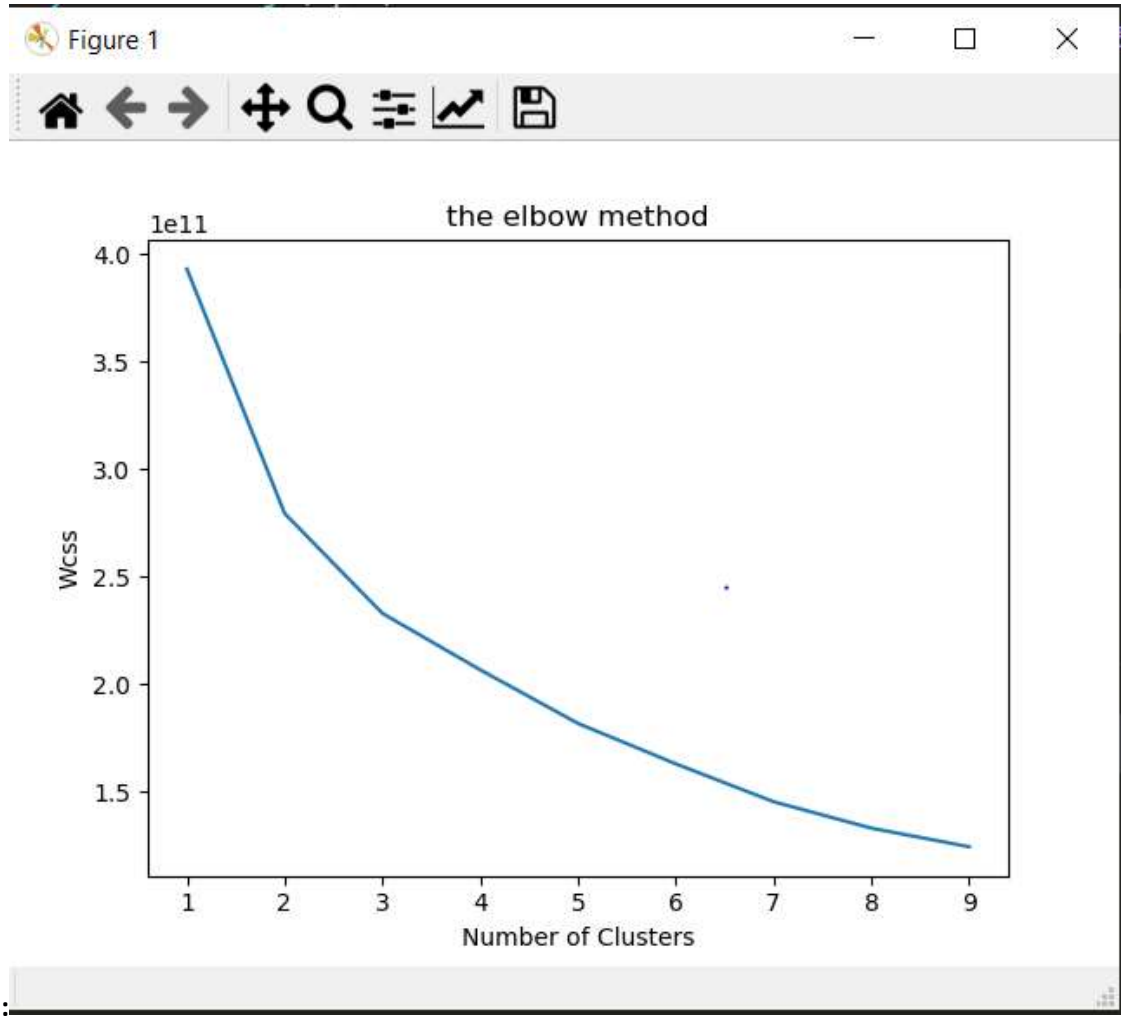
```
from sklearn.cluster import KMeans
# Creating elbow method to get optimal number of Clusters
def elb_met(eldf):
# Calculating wcss for a range of values for k
        wcss = []
# using for loop in range (1,10)
        for i in range(1,10):
            kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, random_state=42)
            kmeans.fit(eldf)
            wcss.append(kmeans.inertia_)
```

```
# provided range (1,10)
        plt.plot(range(1,10),wcss)
# Set tittle as The elbow method
        plt.title('the elbow method')
# label for x- axis
        plt.xlabel('Number of Clusters')
# label for y - axis
        plt.ylabel('Wcss')
# to show the graph
        plt.show()

#calling the elbow method to show elbow plot based on original dataset
elb_met(dataset)
```

In the end we called the elbow method to show elbow plot based on original dataset.

OUTPUT:



:

In this Output we see an elbow graph showing the WCSS after running k-means clustering for k going from 1 to 10. We see a pretty clear **elbow at k = 3**, indicating that 3 is the best number of clusters. After 3 the error value is diminishing consistently.

Step 3) Calculate the Silhouette score:

Here we first defined a method (**sil_met**) to get kmean from passed dataframe and also get the Silhouette Score based on that. This method will return the Silhouette Score and predicted K mean value.

As from elbow graph we derived that optimal number of clusters for this dataset = 3, we will pass same to Kmeans.

```
from sklearn import metrics
# Define a method to get Kmean from passed dataframe and also get Silhouette score based on that.
# This method returns Silhouette score and predicted K mean values
def sil_met(sldf):
 # from elbow graph we derived that optimal number of clusters for this dataset = 3
        nclusters =3
        seed = 0
 # calculate Kmeans for given dataset
        km = KMeans(n_clusters=nclusters, random_state=seed)
        km.fit(sldf)
 # predict the cluster for each data point
        y_cluster_km = km.predict(sldf)
#
```

```
#
# Calculate the Silhouette Score.
# +1: Means clusters are well apart from each other and clearly distinguished.
#  0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.
# -1: Means clusters are assigned in the wrong way.
#
       score = metrics.silhouette_score(sldf,y_cluster_km)
       return(score, y_cluster_km)

# Call method to get Silhouette score for keamns on original dataset
sil_scr, y_cluster_km = sil_met(dataset)
print("Silhouette Score on Original data: ", sil_scr, '\n')
```

We called the method to get **Silhouette Score for Kmean on original dataset.**

OUTPUT:

```
Silhouette Score on Original data:  0.466123875928961
```

Silhouette scores values has significance as below.

+1: Means clusters are well apart from each other and clearly distinguished.

0: Means clusters are indifferent, or we can say that the distance between clusters is not significant.

-1: Means clusters are assigned in the wrong way.

Step 4) Feature Scaling on input dataframe:

Feature scaling is used to bring all the features to similar level of magnitude and it ensure that the larger features does not dominates the others in clustering.

Once Scaler transformation is done on original data, we created new Dataframe with this Scaled data (X_Scaled).

```
# Doing the Feature Scaling so that the larger features should not dominate the others in clustering, etc.
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(dataset)
# Scaler.transform will transform our data
X_scaled_array = scaler.transform(dataset)
X_scaled = pd.DataFrame(X_scaled_array, columns= dataset.columns)
print('Dataframe after Scaling: ')
# printing the 2 sample from X_scaled
print(X_scaled.sample(2),'\n')
```

OUTPUT:

```
Dataframe after Scaling:
       BALANCE  BALANCE_FREQUENCY  ...  PRC_FULL_PAYMENT   TENURE
1644 -0.661113           0.518084  ...          0.329200  0.36068
5976 -0.730167          -0.249434  ...          1.960998  0.36068

[2 rows x 17 columns]
```

Now we will call the **sil_met** to get the kmean values and Silhouette score based on scaled data.

```
# Call the sil_met to get Kmean and Silhouette score based on Scaled data
sil_scr, y_scl_km = sil_met(X_scaled)
print("Silhouette Score after Scaling: ", sil_scr, '\n')
```

OUTPUT:

```
Silhouette Score after Scaling:  0.2500952686352568
```

We got the Silhouette Score after Scaling as 0.2500...
Before scaling the Silhouette score was 0.46612...
Which means the Feature Scaling is not improving the Silhouette score because more the value close to 1 it will be considered as a good value.

Step 5) PCA on Original data set:

Principal Component Analysis (PCA), is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

```
# Applying PCA and Kmean on orignal dataset
# Principal Component Analysis, or PCA, is a dimensionality-reduction method
pca = PCA(2)
X_pca = pca.fit_transform(dataset)
pcadf = pd.DataFrame(data = X_pca, columns= ['PC1', 'PC2'])
print('PCA data sample: ')
print(pcadf.sample(),'\n')
```

OUTPUT:

```
PCA data sample:
              PC1          PC2
3573 -2587.664996  1126.267065
```

We got two sample values from **pcadf** of PC1 and PC2.

Bonus Question answer: As per Bonus question requirement below part of code is to apply K means on PCA result. Different combination are used as PCA + KMEANS and SCALING + PCA + KMEANS. In the end data Visualization will be done on PCA result.

Part I) PCA+KMEANS:

We will be applying kmeans algorithm on the PCA result and report and check if the score improved or not.

Calling **sil_met** method by passing PCA data (**pcadf**) we got above. sil_met method will do Kmean on it and also get the Silhouette Score.

```
# Calling sil_met method by passing PCA dataframe and doing Kmean on it.
# method will return Silhouette Score
# *** (PCA+KMEANS) ***

pca_score, y_pca_km = sil_met(pcadf)
print("PCA Silhouette Score for original data: ", pca_score, '\n')
#print('kmean: ', y_cluster_km)
```

OUTPUT:

```
PCA Silhouette Score for original data:  0.5720003152910856
```

We got a Silhouette score of 0.57200

As we can see score is improved compared to the score calculated on Original and Scaled data.

Part 2) (SCALING+PCA+KMEANS):

We have done the Feature scaling on original data (X_Scaled dataframe). This time we calculated the PCA on X_scaled dataframe.

Once done we will call sil_met method again to calculate Kmeans and Silhouette Score.

```
# Applying PCA and Kmean on Scaled dataset
# *** (SCALING+PCA+KMEANS) ***
X_pca = pca.fit_transform(X_scaled)
pcadf = pd.DataFrame(data = X_pca, columns= ['PC1', 'PC2'])
print(pcadf.sample(),'\n')
# Calling sil_met method by passing PCA dataframe and doing Kmean on it.
pca_score, y_scl_km = sil_met(pcadf)
print("PCA Silhouette Score after Scaling: ", pca_score,'\n')
```

OUTPUT:

```
PCA Silhouette Score after Scaling:  0.45239015688421386

         PC1       PC2  Kmeans
52 -1.726873  0.471398       0
```

Silhouette Score after Scaling as 0.4523....

As we can see doing PCA on Scaled data reduced the Score.

Part III) VISIUALIZATION:

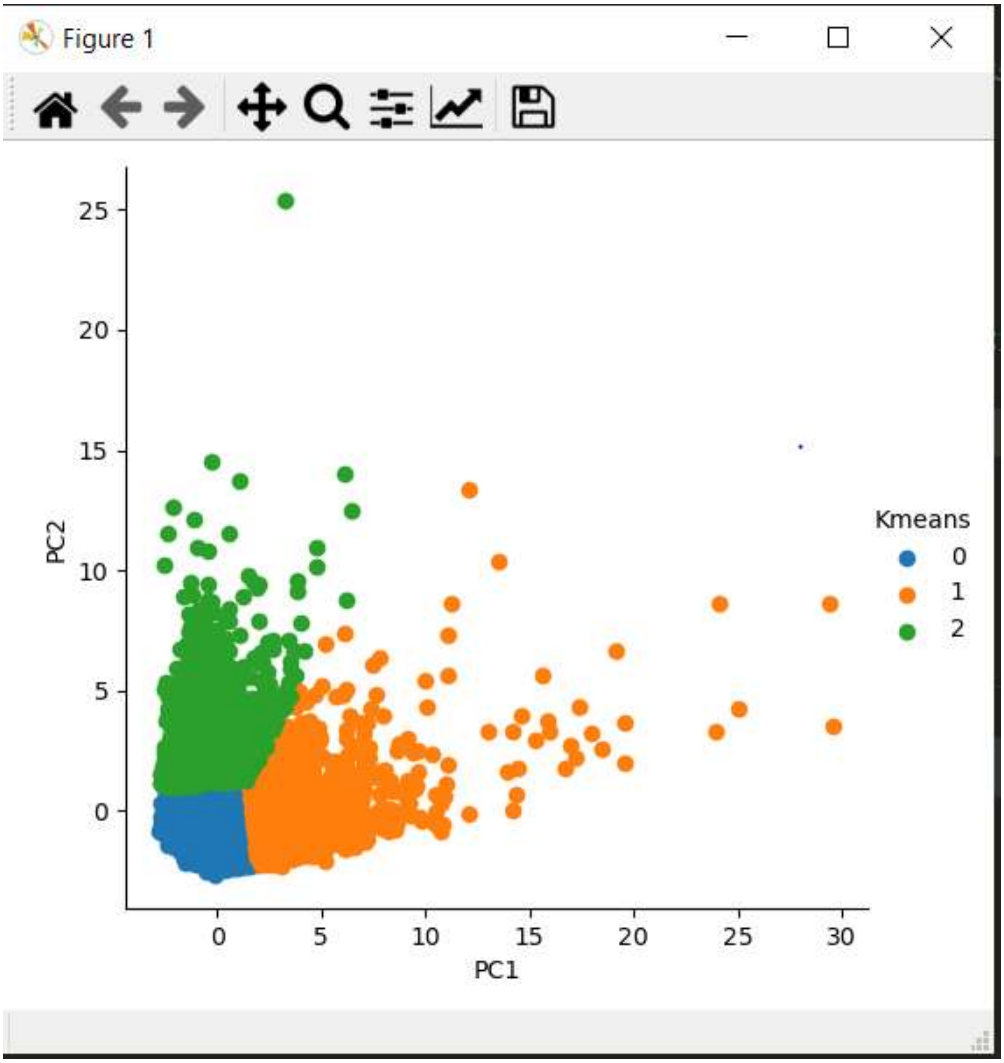
For doing visualisation on PCA result, first created dataframe which has PCA data and calculated Kmeans. The resulting Dataframe – **pcadf2.**

We will calculate the Scatter plot between PC1, PC2 columns (PCA dataframe) and Kmean values.

```
# Visualization
# Create Dataframe with Scaled PCA data columns plus PCA based Kmeans values columns
km_df = pd.DataFrame(data = y_scl_km, columns=['Kmeans'])
pcadf2 = pd.concat([pcadf, km_df],axis = 1)
print(pcadf2.sample())

# Scatter plot for Scaled PCA data and Kmean values
sns.FacetGrid(pcadf2, hue="Kmeans", height=5).map(plt.scatter, "PC1", "PC2").add_legend()
plt.show()
```

OUTPUT:



In this output as we can see clusters are more distinguishable.