Clustering Unsupervised Assignment

Author : @mohameddhameem

Date : 2022-02-02

Problem Statement

Comany XYZ is looking to find potential customers for their new product. They have a large data set of customers who have purchased products using Credit Card. XYZ now wants to find out which customers are most likely to purchase XYZ's product.

The goal is to segment the customers into groups based on their credit card usage and assign each customer to a group. With this assignment we will help to create clusters and present findings. The business team will assign appropriate name for each of the clusters.

Data Set

The dataset is downloaded from Kaggle. - https//www.kaggle.com/arjunbhasin2013/ccdata#

The sample Dataset summarizes the usage behavior of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioral variables.

Data Set description

The dataset contains the following features:

Variable Name	Description
CUSTID	Identification of Credit Card holder (Categorical)
BALANCE	Balance amount left in their account to make purchases
BALANCEFREQUENCY	How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
PURCHASES	Amount of purchases made from account
ONEOFFPURCHASES	Maximum purchase amount done in one-go

Variable Name	Description
INSTALLMENTSPURCHASES	Amount of purchase done in installment
CASHADVANCE	Cash in advance given by the user
PURCHASESFREQUENCY	How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
ONEOFFPURCHASESFREQUENCY	How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
PURCHASESINSTALLMENTSFREQUENCY	How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
CASHADVANCEFREQUENCY	How frequently the cash in advance being paid
CASHADVANCETRX	Number of Transactions made with "Cash in Advanced"
PURCHASESTRX	Numbe of purchase transactions made
CREDITLIMIT	Limit of Credit Card for user
PAYMENTS	Amount of Payment done by user
MINIMUM_PAYMENTS	Minimum amount of payments made by user
PRCFULLPAYMENT	Percent of full payment paid by user
TENURE	Tenure of credit card service for user

Data Exploration

In [ ]:

```
# lets import all the libraries we need
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt, seaborn as sns

# for feature engineering
from scipy.special import boxcox1p
from sklearn.cluster import KMeans
```

In [ ]:

```
plt.rcParams['figure.figsize'] = [6,6]
sns.set_style("whitegrid")
sns.set_context("talk")
```

In [ ]:
```python
# lets read the dataset
dataset = pd.read_csv('Data_CC_GENERAL.csv')
```

In [ ]:
```python
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64  
 12  PURCHASES_TRX                     8950 non-null   int64  
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [ ]:
```python
# lets check missing values
dataset.isnull().sum().sort_values(ascending=False).head()
```

Out[ ]:
```
MINIMUM_PAYMENTS    313
CREDIT_LIMIT          1
CUST_ID               0
BALANCE               0
PRC_FULL_PAYMENT      0
dtype: int64
```

In [ ]:

```python
# there are 2 columns with missing values
# lets check the distribution of the data
dataset.describe()
```

Out[ ]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_F
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8!
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	

In [ ]:

```python
dataset.head()
```

Out[ ]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASE
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	

In [ ]:

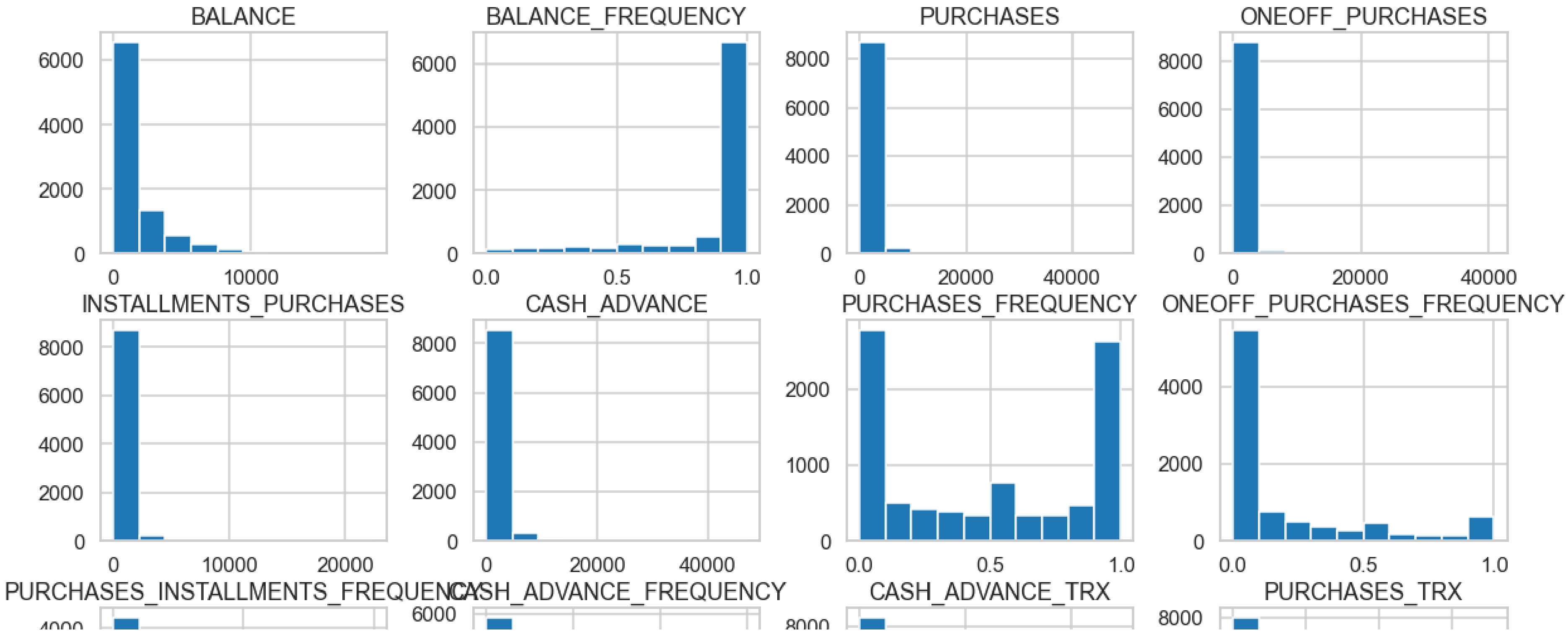
```python
# lets impute the missing values
# for the column 'MINIMUM_PAYMENTS' lets impute with 0
dataset['MINIMUM_PAYMENTS'].fillna(0, inplace=True)
# for the colmun 'CREDIT_LIMIT' lets impute with 50. This is the minimum value for the column. The reason for this is t
dataset['CREDIT_LIMIT'].fillna(50, inplace=True)
```

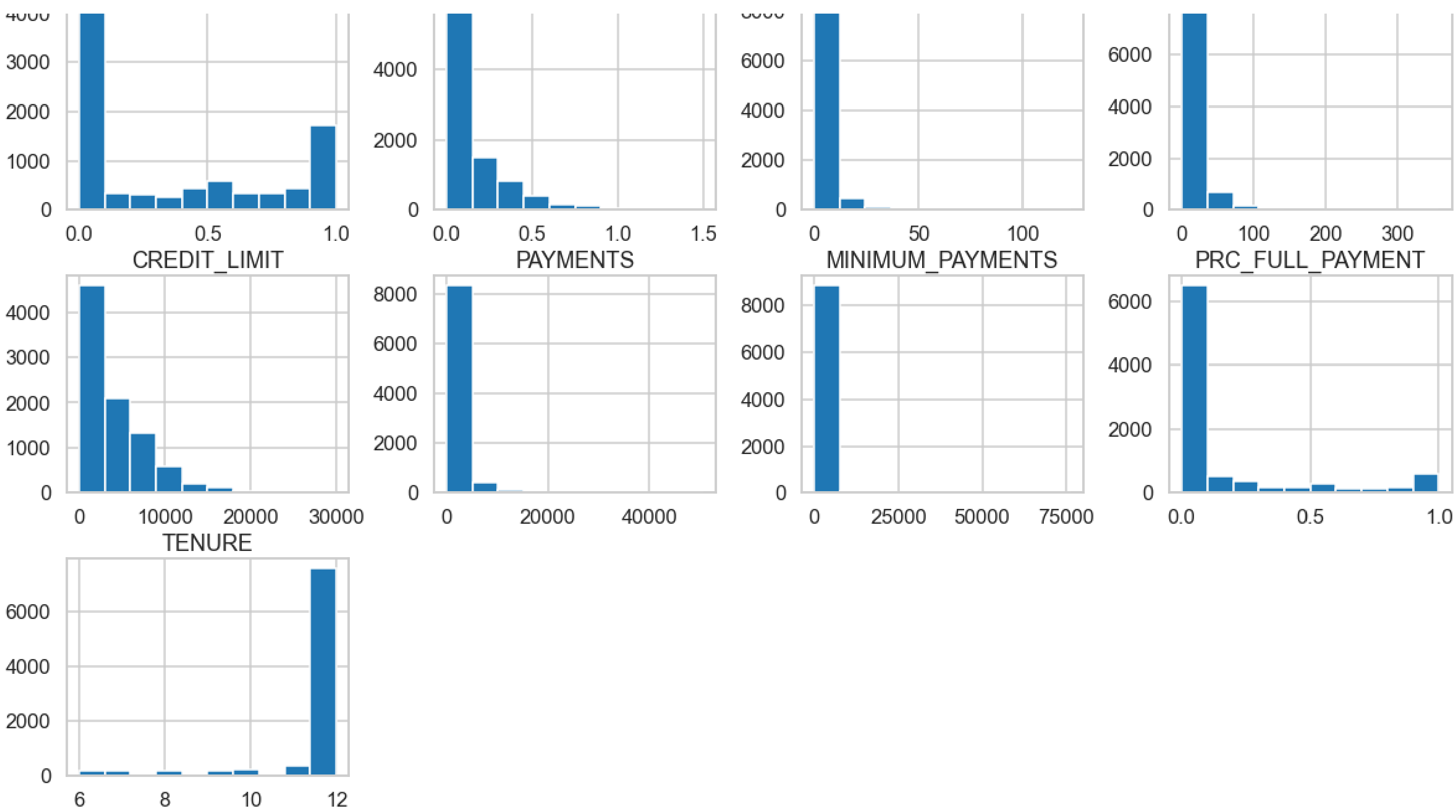
In [ ]:

```python
# Lets try to plot the data distribution for all the columns
dataset.hist(figsize=(20, 20))
```

Out[ ]:

```
array([[<AxesSubplot:title={'center':'BALANCE'}>,
        <AxesSubplot:title={'center':'BALANCE_FREQUENCY'}>,
        <AxesSubplot:title={'center':'PURCHASES'}>,
        <AxesSubplot:title={'center':'ONEOFF_PURCHASES'}>],
       [<AxesSubplot:title={'center':'INSTALLMENTS_PURCHASES'}>,
        <AxesSubplot:title={'center':'CASH_ADVANCE'}>,
        <AxesSubplot:title={'center':'PURCHASES_FREQUENCY'}>,
        <AxesSubplot:title={'center':'ONEOFF_PURCHASES_FREQUENCY'}>],
       [<AxesSubplot:title={'center':'PURCHASES_INSTALLMENTS_FREQUENCY'}>,
        <AxesSubplot:title={'center':'CASH_ADVANCE_FREQUENCY'}>,
        <AxesSubplot:title={'center':'CASH_ADVANCE_TRX'}>,
        <AxesSubplot:title={'center':'PURCHASES_TRX'}>],
       [<AxesSubplot:title={'center':'CREDIT_LIMIT'}>,
        <AxesSubplot:title={'center':'PAYMENTS'}>,
        <AxesSubplot:title={'center':'MINIMUM_PAYMENTS'}>,
        <AxesSubplot:title={'center':'PRC_FULL_PAYMENT'}>],
       [<AxesSubplot:title={'center':'TENURE'}>, <AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```





From the above plot we could see some skew in the data. During our feature engineering we will try to remove the skew.

Identify skweness in the data

In [ ]:

```
# utils function to identify skewness
def skewness(data):
    # create a mask for the data with datatype float and integer
    # mask = data.dtypes == np.float64 or data.dtypes == np.int64 or data.dtypes == np.int32 or data.dtypes == np.int16
    mask = (data.dtypes == np.float64) | (data.dtypes == np.int64)
    float_cols = data.columns[mask]

    skew_limit = 0.75 # define a limit above which we will log transform
```

```python
    skew_vals = data[float_cols].skew()
    # Showing the skewed columns
    skew_cols = (skew_vals
                 .sort_values(ascending=False)
                 .to_frame()
                 .rename(columns={0:'Skew'})
                 .query('abs(Skew) > {}'.format(skew_limit)))

    print('Number of skewed columns :', skew_cols.shape[0])
    return skew_cols
```

In [ ]:

```python
skew_cols = skewness(dataset)
skew_cols
```

```
Number of skewed columns : 15
```

Out[ ]:

	Skew
MINIMUM_PAYMENTS	13.808430
ONEOFF_PURCHASES	10.045083
PURCHASES	8.144269
INSTALLMENTS_PURCHASES	7.299120
PAYMENTS	5.907620
CASH_ADVANCE_TRX	5.721298
CASH_ADVANCE	5.166609
PURCHASES_TRX	4.630655
BALANCE	2.393386
PRC_FULL_PAYMENT	1.942820
CASH_ADVANCE_FREQUENCY	1.828686
ONEOFF_PURCHASES_FREQUENCY	1.535613
CREDIT_LIMIT	1.522374
BALANCE_FREQUENCY	-2.023266
TENURE	-2.943017

Feature Engineering

In [ ]:

```
# remove unwanted columns from the dataset. Customer ID is not needed
dataset.drop(['CUST_ID'], axis=1, inplace=True)
```

Remove Skew with boxcox1p transformation

In [ ]:

```
# to remove the skewness we will use boxcox transformation
for col in skew_cols.index:
    # apply boxcox1p transformation
    dataset[col] = boxcox1p(dataset[col], 0.15)
```

Scaling the data with RobustScaler

In [ ]:

```
# Robust scaling
from sklearn.preprocessing import RobustScaler
scalar = RobustScaler()
# get the mask for float and integer columns
mask = (dataset.dtypes == np.float64) | (dataset.dtypes == np.int64)
float_cols = dataset.columns[mask]
dataset[float_cols] = scalar.fit_transform(dataset[float_cols])
```

Model Building

K-Means Clustering

Lets apply PCA to reduce the dimensionality of the data

In [ ]:

```
num_clusters = 5
# we will consider there are only 5 clusters based on our understanding of the data
```

In [ ]:

```
# PCA - we will use this in ploting the clusters
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
dataset_pca = pca.fit_transform(dataset)
```

In [ ]:

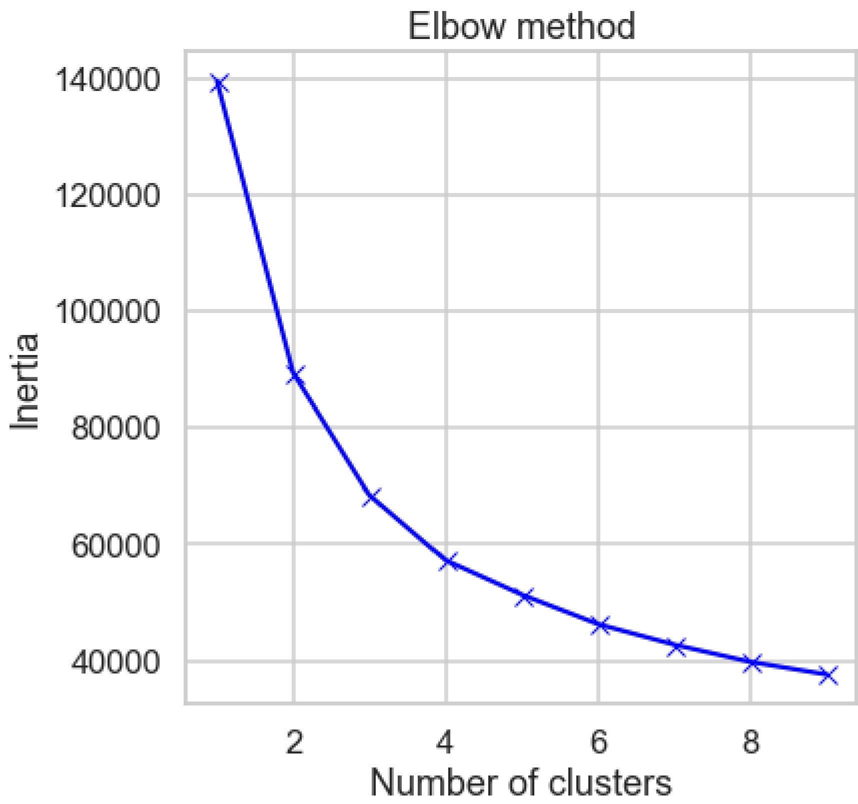
```
# helper function that allows us to display data in 2 dimensions an highlights the clusters
def display_cluster(X,km=[],num_clusters=0):
    color = 'brgcmyk'
    alpha = 0.5
    s = 20
    if num_clusters == 0:
        plt.scatter(X[:,0],X[:,1],c = color[0],alpha = alpha,s = s)
    else:
        for i in range(num_clusters):
            plt.scatter(X[km.labels_==i,0],X[km.labels_==i,1],c = color[i],alpha = alpha,s=s)
            plt.scatter(km.cluster_centers_[i][0],km.cluster_centers_[i][1],c = color[i], marker = 'x', s = 100)
```

In [ ]:

```
# Lets try to find optimal number of clusters using elbow method
# lets define a function that will help us to find the optimal number of clusters
def elbow_method(X, max_clusters=10):
    # define a list to store the values of the inertia
    inertias = []
    # loop through the number of clusters
    for i in range(1, max_clusters):
        # create a KMeans object
        km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=300, random_state=0)
        # fit the data to the KMeans object
        km.fit(X)
        # append the value of the inertia to the list
        inertias.append(km.inertia_)
    # plot the values of the inertia
    plt.plot(range(1, max_clusters), inertias, 'bx-')
    plt.xlabel('Number of clusters')
    plt.ylabel('Inertia')
    plt.title('Elbow method')
    plt.show()
```

In [ ]:

```
elbow_method(dataset)
```



Explore the clusters

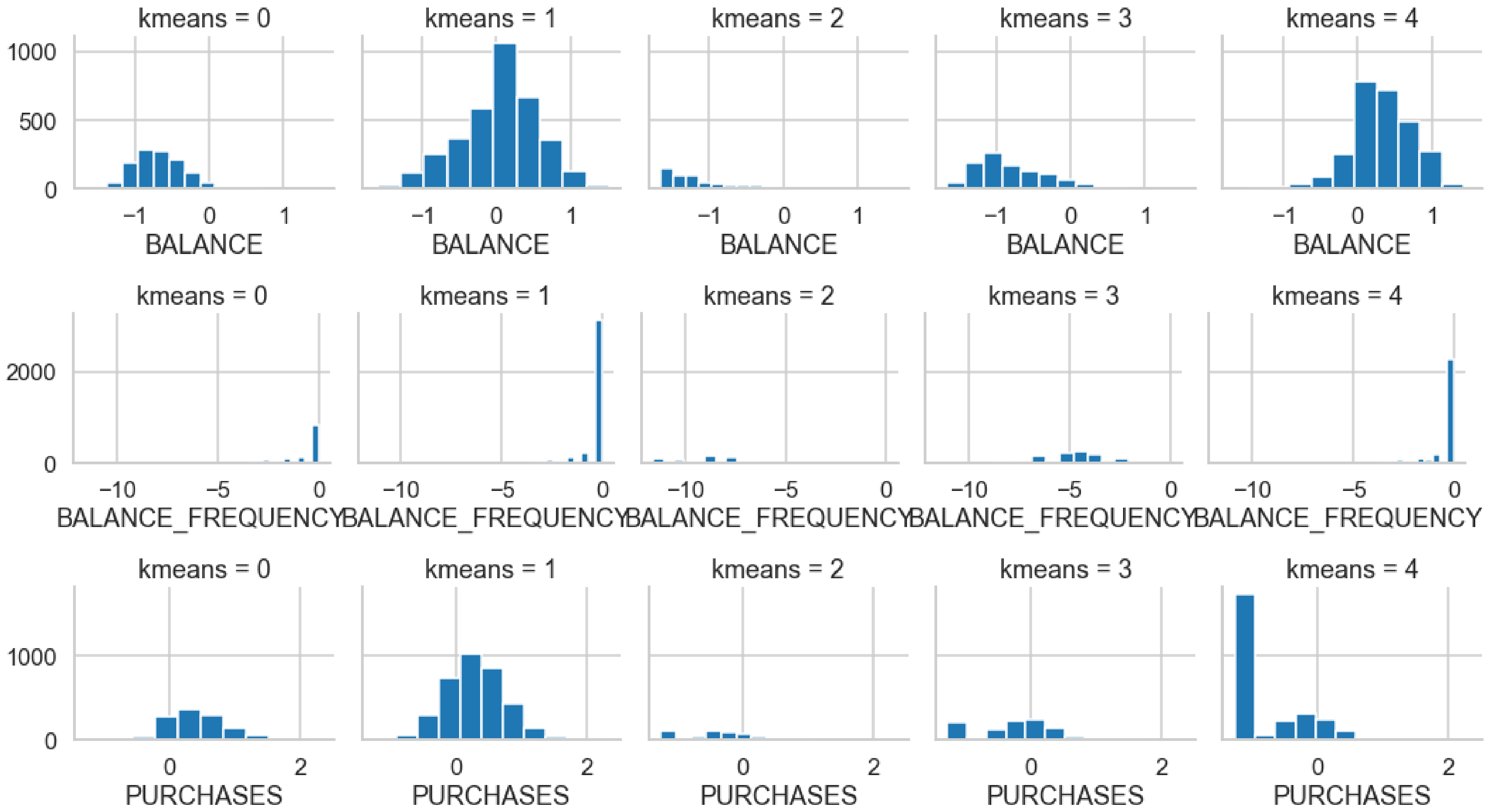
In [ ]:

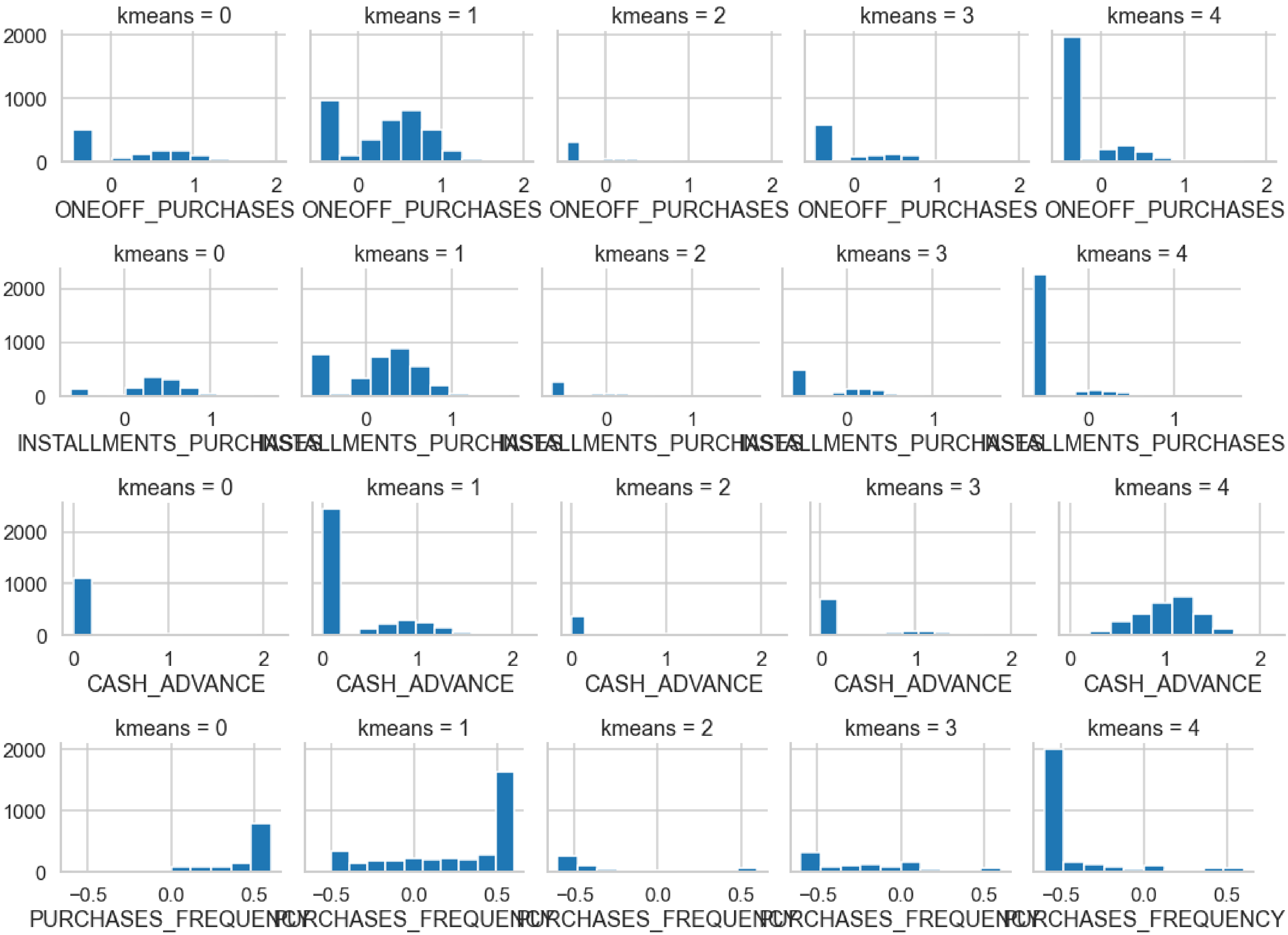
```
# Lets take the optimal number of clusters as 5
km = KMeans(n_clusters=num_clusters, init='k-means++', n_init=10, max_iter=300, random_state=0)
km.fit(dataset)
labels = km.labels_
```

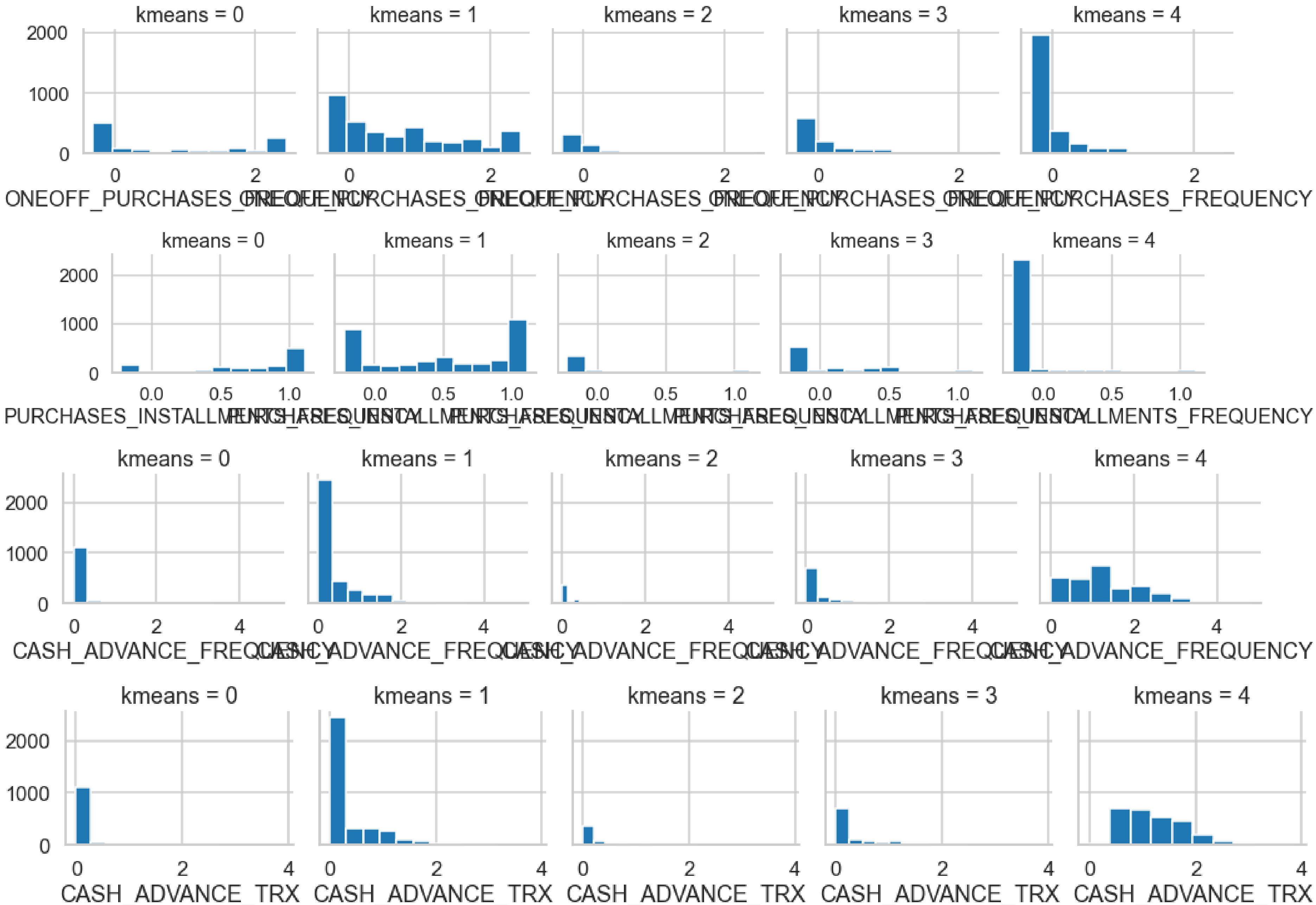
In [ ]:

```
# Append the cluster labels to the result dataframe
# empty dataframe
df_results = pd.DataFrame()
df_results['kmeans'] = km.labels_
# for next vizualization alone
dataset['kmeans'] = km.labels_
```

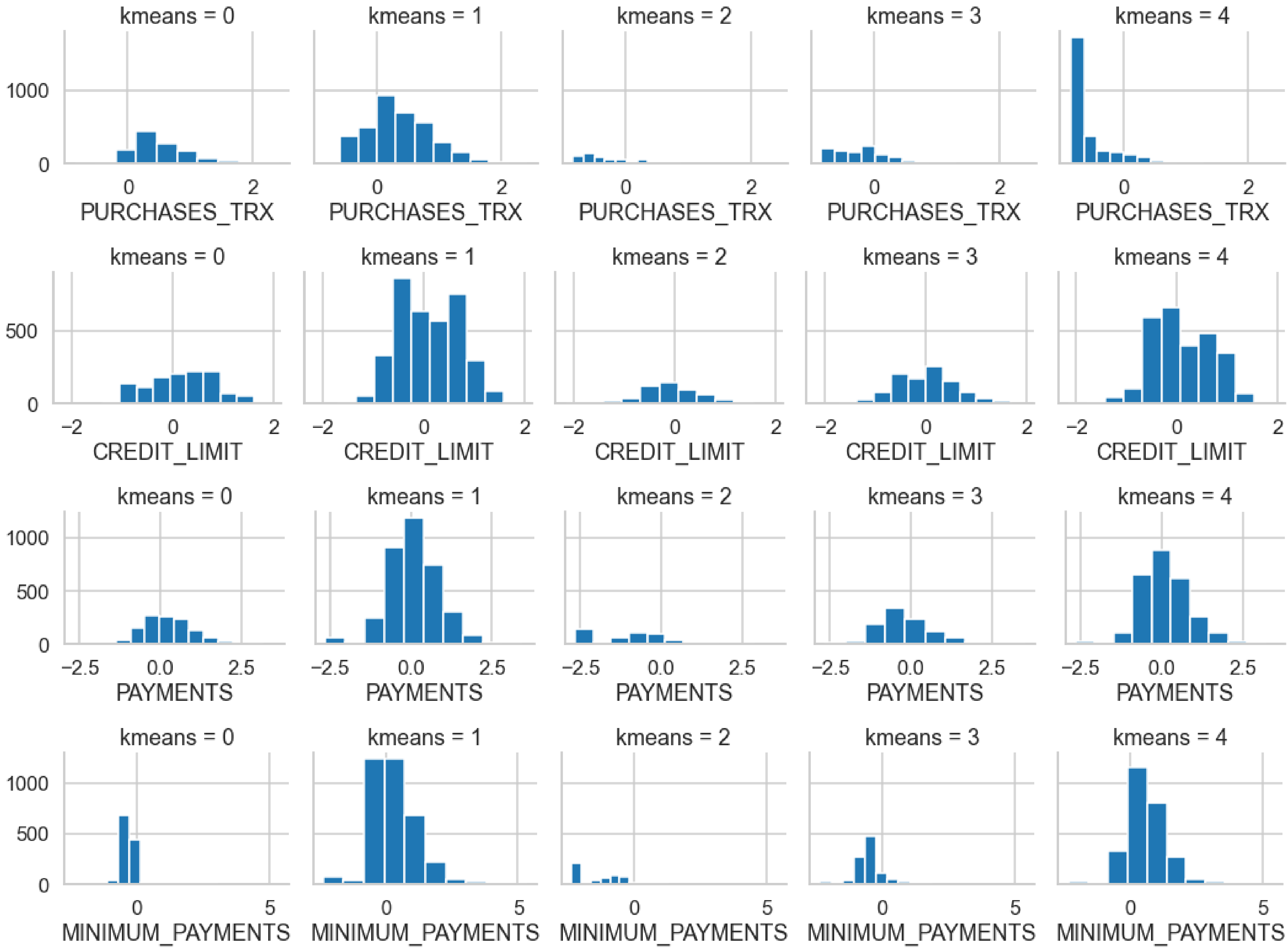
```
In [ ]: # Lets interpret the clusters
for c in dataset:
    if c != 'kmeans':
        grid= sns.FacetGrid(dataset, col='kmeans')
        grid.map(plt.hist, c)
```

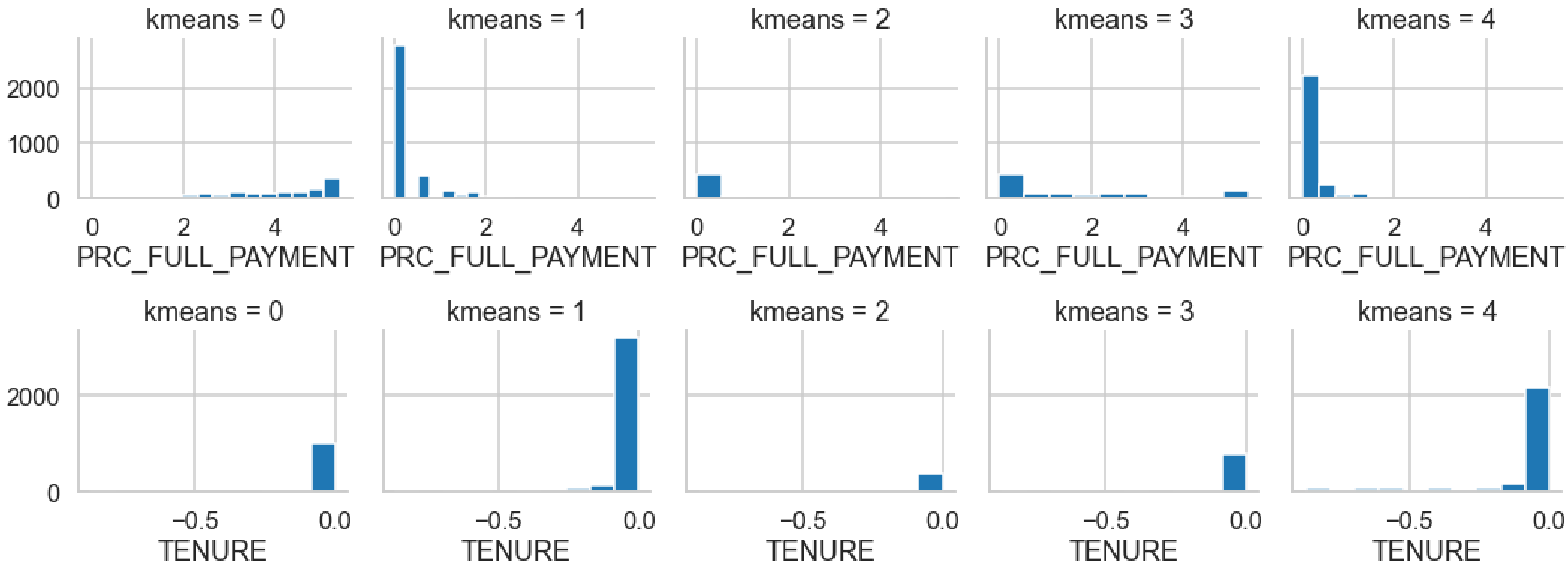



kmeans = 0
kmeans = 1
kmeans = 2
kmeans = 3
kmeans = 4
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY



kmeans = 0
kmeans = 1
kmeans = 2
kmeans = 3
kmeans = 4
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
CASH_ADVANCE_TRX
CASH_ADVANCE_TRX
CASH_ADVANCE_TRX
CASH_ADVANCE_TRX


kmeans = 0
kmeans = 1
kmeans = 2
kmeans = 3
kmeans = 4
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
1000
500
0
−2
2
−2.5
0.0
2.5
5

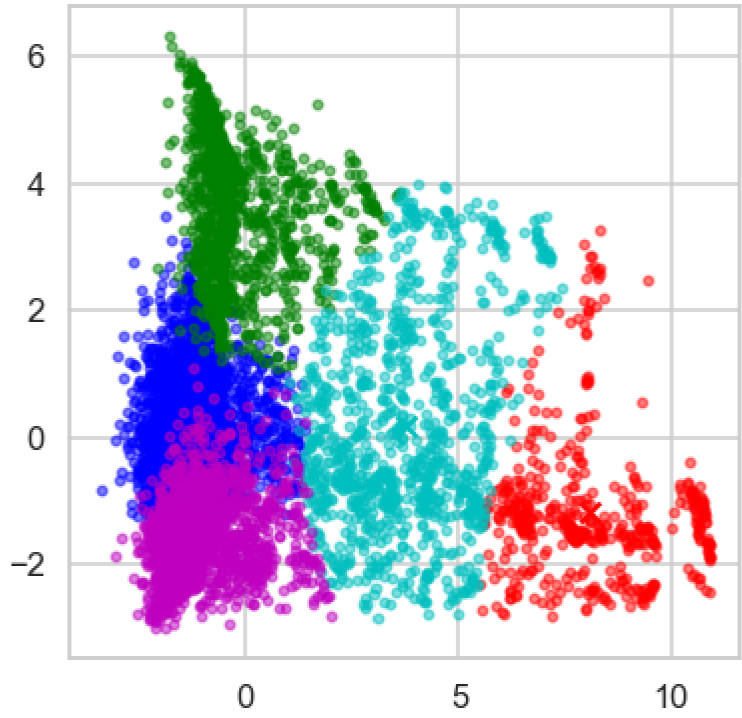


We will leave it to the business team to decide the name of the clusters.

Visualize the clusters

In [ ]:

```
# fit the data to the KMeans object - We will use the PCA data
km.fit(dataset_pca)
# Lets plot the data
display_cluster(dataset_pca,km,num_clusters)
```



From the above plot we can see the K-Means clustering is able to cluster the data into 2 clusters.

In [ ]:

```
# vizualize the clusters
# lets create a dataframe with the cluster labels
cluster_labels = pd.DataFrame(km.labels_)
# lets rename the column
cluster_labels.columns = ['CLUSTER_LABELS']
# lets merge the cluster labels with the dataset
dataset_clustered = pd.concat([dataset,cluster_labels],axis=1)
# lets check the cluster labels
dataset_clustered.head()
```

Out[ ]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENC
0	-0.948107	-1.662719	-0.389159	-0.458874	0.013416	0.000000	-0.3999
1	0.557005	-0.814527	-1.267909	-0.458874	-0.635738	1.462487	-0.6000
2	0.441399	0.000000	0.260677	0.613826	-0.635738	0.000000	0.6000

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENC
3	0.263423	-3.473316	0.513229	0.791058	-0.635738	0.656935	-0.5000(
4	-0.025435	0.000000	-0.795142	-0.127105	-0.635738	0.000000	-0.5000(

Hierarchical Agglomerative Clustering

In [ ]:

```
# hierarchical clustering
from sklearn.cluster import AgglomerativeClustering
ag_model = AgglomerativeClustering(n_clusters=num_clusters, affinity='euclidean', linkage='ward')

ag_model = ag_model.fit(dataset)
# Lets try to predict the clusters with Hierarchical Clustering
df_results['agglom'] = ag_model.labels_
```

In [ ]:

```
dataset.head()
```

Out[ ]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENC
0	-0.948107	-1.662719	-0.389159	-0.458874	0.013416	0.000000	-0.3999(
1	0.557005	-0.814527	-1.267909	-0.458874	-0.635738	1.462487	-0.6000(
2	0.441399	0.000000	0.260677	0.613826	-0.635738	0.000000	0.6000(
3	0.263423	-3.473316	0.513229	0.791058	-0.635738	0.656935	-0.5000(
4	-0.025435	0.000000	-0.795142	-0.127105	-0.635738	0.000000	-0.5000(

In [ ]:

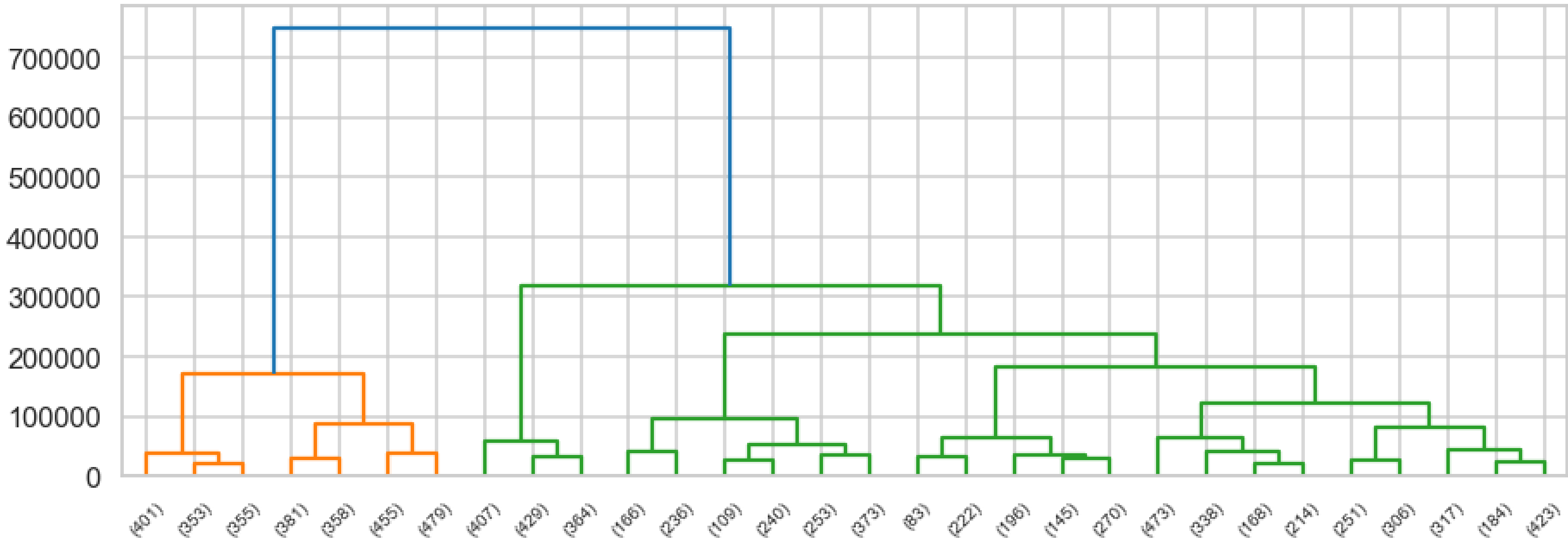
```
# Plot the dentogram
from scipy.cluster import hierarchy

Z = hierarchy.linkage(ag_model.children_, method='ward')

fig, ax = plt.subplots(figsize=(15,5))

den = hierarchy.dendrogram(Z, orientation='top',
```

```
                    p=30, truncate_mode='lastp',
                    show_leaf_counts=True, ax=ax,
                    # above_threshold_color=blue
                    )
```



DBScan Clustering

In [ ]:

```
# use DBSCAN to cluster the data
from sklearn.cluster import DBSCAN
db_model = DBSCAN(eps=0.3, min_samples=10).fit(dataset)
df_results['dbscan'] = db_model.labels_
```

In [ ]:

```
df_results.head()
```

Out[ ]:

	kmeans	agglom	dbscan
0	1	0	-1
1	4	2	-1

	kmeans	agglom	dbscan
2	1	0	-1
3	3	1	-1
4	1	0	-1

Key Findings and Recommendations

We have tried 3 different clustering algorithms. We will leave it to the business team to decide the name of the clusters.

The orignal scope of the problem was to find out the customer segments. Via our analysis we managed to segment the customers into 5 different segments.

KMeans Clustering seems the best choice for this problem.

Our Recommendations:

Work with business analyst and assign name to each cluster and if there are any additonal data set available , then we can convert this problem to supervised learning.

In [ ]:

```

```