CLUSTERING OF TRANSACTIONAL DATA

Capstone Project Report

Submitted by:
ASHIQUE NAWAZ CHOWDHURY
AYUSHI ARORA

Email Id/RUID: ashique.chowdhury@rutgers.edu **/ 197000391**
aa1992@scarletmail.rutgers.edu **/197005649**

Advisor:

PROF. SERGEI SCHREIDER

Table of Content

ACKNOWLEDGEMENT

The implementation and outcomes of the project are crucial and valuable for Dime Community Bank. However, without the kind help and assistance of many people, it would not have been possible. We extend our sincere gratitude to all of them. We are indebted to Prof. Sergei Schreider for his guidance and continuous supervision, we also extend our heartful thanks to our mentors and Supervisors from Dime community for giving us this opportunity with their constant support and invaluable feedback. We would like to express our gratitude for their kind cooperation and encouragement to our parents, friends & members of Rutgers, and Dime Community Bank which enabled us to complete this project.

INTRODUCTION

The paradigm shift in technology which has enabled corporations to handle copious amount of data and extract meaningful information has increased the competition among various financial corporations which has made creation of data driven business and marketing strategy a crucial part of Banking sector. Following the same path, the business case was to apply data analytics to make empower strategies based on targeted marketing and focus on customers which would have been possible through understanding the transactional behavior of customers and reckoning similarities among customers for grouping. By focusing on the business problem and unsupervised transactional data from DataMart the solution was to impose unsupervised learning and apply clustering on the dataset. The dataset is comprised of both categorical and numerical data with ample number of uncertain columns and thus different types of clustering methods and machine learning techniques along with feature engineering has been used.

Clustering is a type of unsupervised learning which involves grouping a set of objects in such a way that objects in the same group are more like each other than to those in other groups. Data mining, EDA and different types of clustering models has been used to perceive the best clusters substantiated by Data mining, EDA, and cluster visualizations to understand the clusters and infer busines outcome.

Methods Used

- **ACCOUNT LEVEL-**

K MEANS:

K MEANS is a type of clustering which is the most applied one in field of Unsupervised Learning and clustering. It is a type of Partitional clustering in which a no object can be a member of more than one cluster, and every cluster must have at least one object. It randomly selects select k centroids, where k is equal to the number of clusters we choose. Centroids are data points representing the center of a cluster.

Since K means requires data only to be numerical hence in our dataset, we convert the dataset from transactional level to account level by using group by function making every column as a count of categories of specific columns. Then we made pipelines of the methods and PCA for dimensionality reduction for obtaining results.

DBSCAN:

It is a type of density clustering and since our dataset is a large one, we used minimum number of samples to form a cluster as 4000 and thus obtained 2 clusters

- **TRANSACTIONAL LEVEL-**

K MODES:

It is a type of clustering which is implemented by installing kmodes package and is meant for categorical data. It defines clusters based on the number of matching categories between data points it also accepts nan values. The dataset where K modes is applied is required to be converted into a matrix first. In our dataset for applying k modes we kept the categorical columns as it is and converted the numerical columns into categorical. And as per requirement of consistent data type; all the columns are converted into string data type.

K medoids using Gower distance:

Gower's distance is a measure to find the similarity between two rows of a dataset consisting of mixed type attributes. It uses the concept of Manhattan distance for continuous variables and dice distance for measuring similarity between Binary variables

Challenge: Gower distance matrix works only for small dataset and hence no impacting and complete results were obtained.

K prototype:

It can work directly with the categorical data, without the need for encodings. A record is allocated to the cluster which has the most similar looking reference point i.e. prototype of the cluster. It is a combination of K means and K mode It uses Euclidean distance to find similarity between numerical columns and dice distance for measuring similarity between Binary variables

Import Libraries

- First, import the required libraries for the project

```
import pandas as pd
import numpy as np

## ML
import sklearn
import scipy
from sklearn.cluster import KMeans
from scipy import stats
from sklearn.preprocessing import MinMaxScaler
from pylab import rcParams
import sklearn.metrics as sm
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist
from pandas.plotting import parallel_coordinates
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from pyclustering.cluster import cluster_visualizer
from pyclustering.cluster.center_initializer import kmeans_plusplus_initializer
from pyclustering.cluster.kmedoids import kmedoids
from kmodes.kmodes import KModes

## Visualizations
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

## Dime Imports - Import Connections
from dimepy.custom_connections import connect_to_fiserv, connect_to_rocketdev
```

- Next step is to load the dataset, we have extracted the data from MS SQL server

```
#con_rocketprod = connect_to_rocketprod()
con_rocketdev = connect_to_rocketdev()

query = '''
    SELECT *
    FROM DM_DEMO.dbo.Transactions
    WHERE
        TranOrigPostDate >= '2020-09-01' and
        TranOrigPostDate < '2020-10-01'
'''

transactions = pd.read_sql(query, con_rocketdev)
```

- Understanding the data is the most important thing at the beginning. Constructing machine learning models does not make sense if we do not understand what we are looking for in our data and what our goals are. Let's see some of the simple data set stuff.

- Which are the variables included in the dataset?

```
transactions.columns
```

```
Index(['TransactionNumber', 'AccountNumber', 'TranAmount', 'TranTypeCode',
       'TranTypeDesc', 'TellerCIF', 'TranSourceCatCode', 'TranSourceCode',
       'TranSourceDesc', 'TranOrigPostDate', 'TranInteractionMeth',
       'TranInteractionChannel', 'TranInteractionDirection'],
      dtype='object')
```

- Checking the size of the data

```
transactions_df.shape
```

```
(56975, 14)
```

- Checking the datatype of the columns

```
transactions.dtypes
```

```
TransactionNumber                    int64
AccountNumber                        int64
TranAmount                         float64
TranTypeCode                        object
TranTypeDesc                        object
TellerCIF                           object
TranSourceCatCode                   object
TranSourceCode                      object
TranSourceDesc                      object
TranOrigPostDate            datetime64[ns]
TranInteractionMeth                 object
TranInteractionChannel              object
TranInteractionDirection            object
dtype: object
```

- Data Dictionary
    - TransactionNumber : Number of transactions made by a particular account number
    - AccountNumber: Account number
    - TranAmount: Amount of money withdrawal or deposited
    - TrantypeCode: Mode of Transaction in code
    - TrantypeDesc: Description of the mode of transaction
    - TransOrigPostDate: Date when the transaction has been made
    - TranSourseCode: Type of transaction code
    - TranSourseDesc: Description of transaction type code

Data Cleaning

- Checking for null values

```
transactions.isnull().sum()
```

```
TransactionNumber              0
AccountNumber                  0
TranAmount                     0
TranTypeCode                   0
TranTypeDesc                   0
TellerCIF                   6440
TranSourceCatCode              0
TranSourceCode                 0
TranSourceDesc                 0
TranOrigPostDate               0
TranInteractionMeth            0
TranInteractionChannel         0
TranInteractionDirection       0
dtype: int64
```

- Dropping the columns which are not required

```
transactions = transactions.drop(['TranTypeDesc', 'TellerCIF', 'TranSourceDesc',
                                  'TranTypeCode','TranSourceCatCode','TranSourceCode'], axis=1)
```

- Removed Outliers from all the numerical columns

```
#Removing outliers
#Change limits of outliers
def outlier_removal(data,column):
    q1 = data[column].quantile(.10)
    q3 = data[column].quantile(.90)
    mask = data[column].between(q1, q3, inclusive=True)
    iqr = data.loc[mask, column]
    return (iqr)
```

Feature Engineering

- Separating the TranAmount column into Deposit and Withdrawal columns

```
transactions['dep'] = np.where(transactions['TranAmount'] < 0, np.nan, transactions['TranAmount'])
transactions['withd'] = np.where(transactions['TranAmount'] > 0, np.nan, transactions['TranAmount'])
transactions['withd'] = transactions['withd'].abs()
```

- Creating a score column which contains bin of TransAmount

```
transactions['Score'] = pd.cut(transactions.TranAmount,
      bins=[-float("inf"), -10000,-9000,-8000,-7000,-6000,-5000,-4000,-3000,-2000, -1000, 0, 1000, 2000,
      include_lowest=True, labels=[-11,-10,-9,-8,-7,-6,-5,-4,-3,-2, -1, 1, 2, 3,4,5,6,7,8,9,10,11] )
```

- Deriving day_of_the_week and day_of_the_month columns from TransOrigPostDate

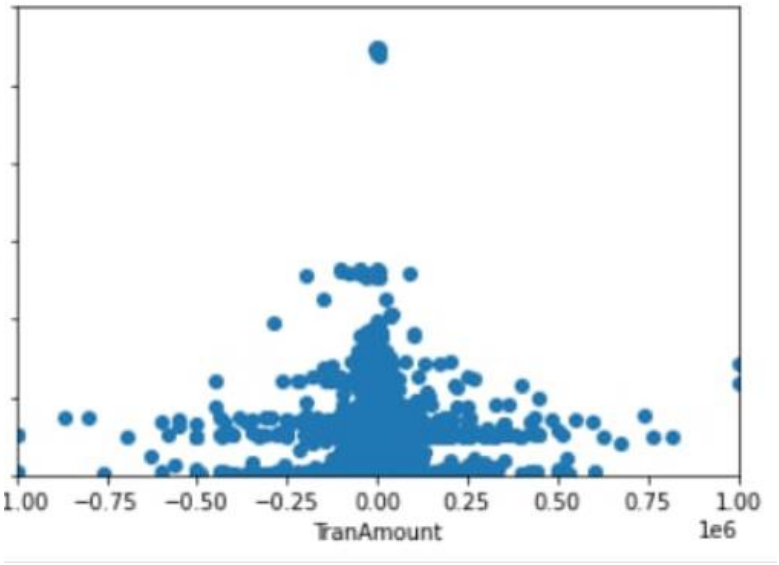
```
transactions["day_of_week"] = transactions.TranOrigPostDate.dt.dayofweek
transactions["day_of_month"] = transactions['TranOrigPostDate'].dt.day
```

Exploratory Data Analysis

One of the essential steps in the data analysis process is Exploratory Data Analysis. Here the emphasis is on making sense of the information at hand, such as formulating the best questions to ask the dataset, how to manipulate the sources of data to get the answers needed, and others. Using a visual approach, this is achieved by taking an elaborate look at trends, patterns, and outliers. Before we jump to machine learning or simulation of our data, exploratory data analysis is a crucial phase. It provides the context necessary to build an appropriate model and correctly interpret the outcomes.

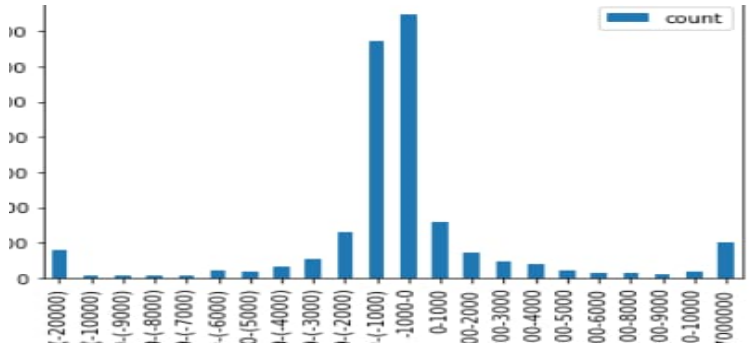
- Distribution of the data
  - Scatter Plot number of transactions and Transaction amount

```
plt.scatter(transactions['TranAmount'],transactions['TransactionNumber'])
plt.xlim([-1000000,1000000])
plt.ylim([0,30000])
plt.xlabel('TranAmount')
plt.ylabel('TransactionNumber')
```



- Bar Graph between scores and the number of transactions

```
df1 = pd.DataFrame(
    {'count' : transactions.Score.value_counts(sort=False),
     'names' : ["-2000000-(-20000)","-2000000-(-10000)","-10000-(-9000)","-9000-(-8000)",
                "-8000-(-7000)","-7000-(-6000)","-6000-(5000)","-5000-(-4000)","-4000-(-3000)",
                "-3000-(-2000)","-2000-(-1000)", "-1000-0", "0-1000", "1000-2000","2000-3000",
                "3000-4000","4000-5000","5000-6000","7000-8000","8000-9000","9000-10000","10000-7000000"]}
)
df1.plot.bar(x='names', y='count')
plot.show()
```

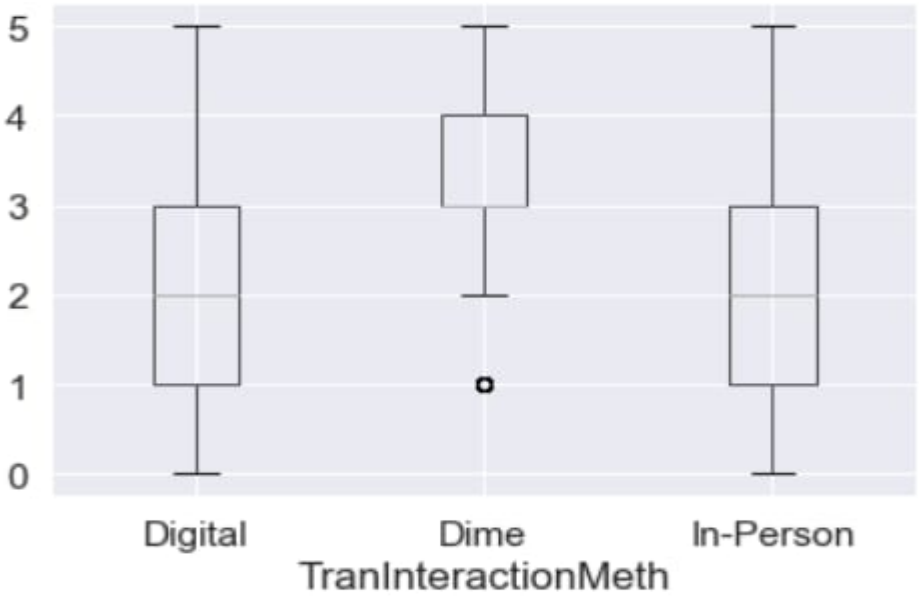


- Number of Transactions V/S weekdays

```
#Number of transactions vs Weekday
sns.set(font_scale=1.4)
transactions['day_of_week'].value_counts().plot(
    kind='bar', figsize=(7, 6), rot=0,
)
plt.xlabel("day_of_week", labelpad=14)
plt.ylabel("Number of Transactions", labelpad=14)
#plt.title("Count of People Who Received Tips by Gender", y=1.02);
```

```
Text(0, 0.5, 'Number of Transactions')
```

- Visualization of Modes of Transactions
  - Box plot for Transaction Interaction Method

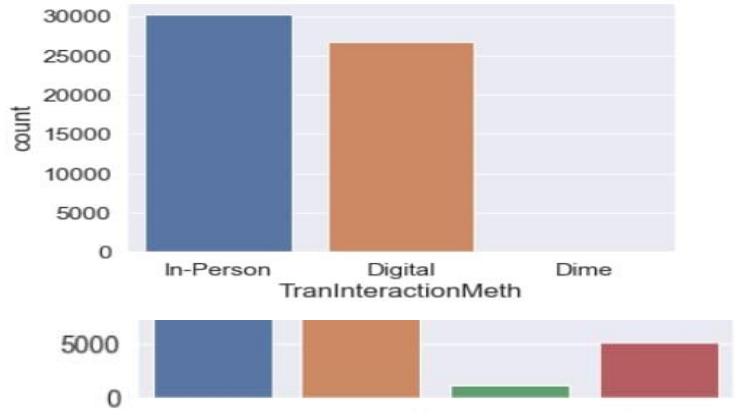


- Bar chat for modes of transactions

```
#Defining the function count plot
def count_plot(transactions, column_name, title =None, hue = None):
    base_color = sns.color_palette()[0]
    sns.countplot(data = transactions, x = column_name, hue=hue)
    plt.title(title)
```

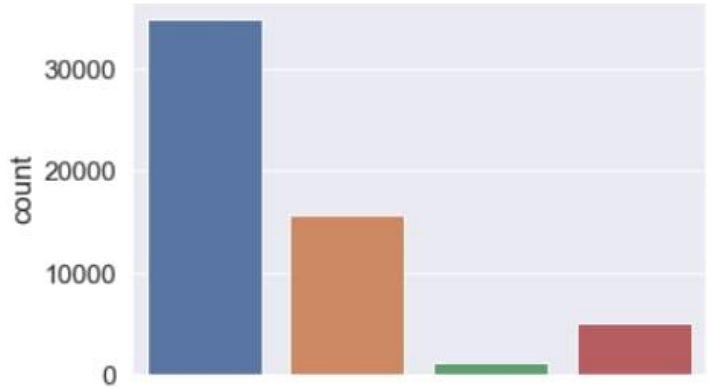
```
sns.countplot(data=transactions, x ='TranInteractionMeth')
```

```
<AxesSubplot:xlabel='TranInteractionMeth', ylabel='count'>
```




```
sns.countplot(data=transactions, x ='TranSourceCatCode')
```

```
<AxesSubplot:xlabel='TranSourceCatCode', ylabel='count'>
```



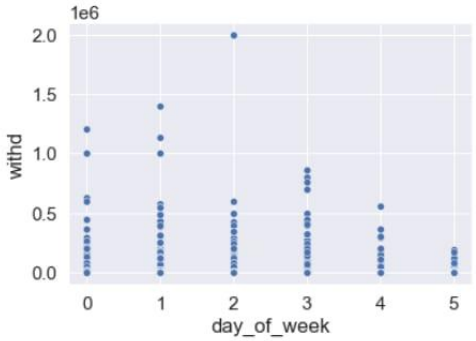
- Number of transactions V/S Deposit and Withdrawal

```
ountplot(data=transactions, x ='TranTypeCode')
```

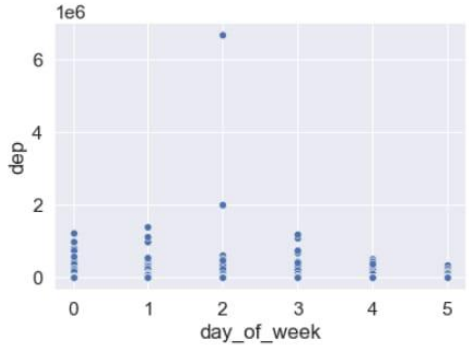
```
Subplot:xlabel='TranTypeCode', ylabel='count'>
```

- Visualization for Days of the week

```
#Scatter plot to see which day of the week customers withdraw the most
scatter_plot(transactions,"day_of_week","withd")
```




```
#Scatter plot to see which day of the week customers deposit the most
scatter_plot(transactions, "day_of_week", "dep")
```



Technical Approach

- Before the application of machine learning methods, we have separated the data into two levels Account Level and Transactional Level

Account Level

- Created a pivot table with count of each category in a particular column
- Grouping of all the columns w.r.t unique account number

```
account_df = transactions_pivot.groupby('AccountNumber')['dep','withd','Digital','Dime','In-Person',
             'Online Banking','Other','Teller','VRU_Interaction_channel', 'Deposit','Withdrawal',
             'Transfer'].count()
```

dep	withd	Digital	Dime	In-Person	Online Banking	Other	Teller	VRU_Interaction_channel	Deposit	Withdrawal	Transfer
2	0	0	0	2	0	0	2	0	2	0	0
0	1	0	0	1	0	0	1	0	0	1	0
0	1	0	0	1	0	0	1	0	0	1	0
2	0	0	0	2	0	0	2	0	2	0	0
0	5	3	0	2	3	0	2	0	0	2	3

- Feature Selection - Variance Threshold Method

```
#Remove this box and re run to get previous results
#For removing features of low variance using sklearn
from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold(.6)
selector.fit(account_df)
account_df = account_df[account_df.columns[selector.get_support(indices=True)]]
```

- Feature Selection - Retaining only the important features
- Pipeline for Principal Component Analysis and Kmeans
  - Pipeline is a process of automating machine learning workflows
  - Created pipelines for the steps involved in PCA and Kmeans to use it recursively whenever required

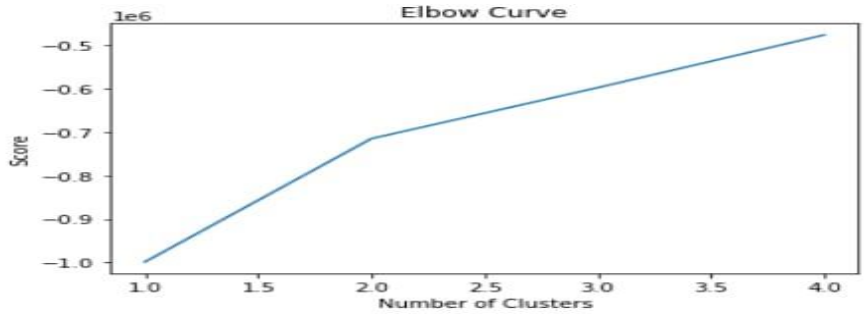
```
pca_pipeline = Pipeline(
        [
              ("pca", PCA(n_components=2))
        ]
    )
```

```
pipe = Pipeline(
    [
        ("clusterer", clusterer),
        ("pca_pipeline", pca_pipeline)
    ]
)
```

- Account Level Clustering – Kmeans

  - Elbow Curve to find the optimal number of clusters

```
# Run a number of tests, for 1, 2, ... num_clusters
num_clusters = 5
kmeans_tests = [KMeans(n_clusters=i, init='random', n_init=10) for i in range(1, num_clusters)]
score = [kmeans_tests[i].fit(account_df).score(account_df) for i in range(len(kmeans_tests))]

# Plot the curve
plt.plot(range(1, num_clusters),score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```



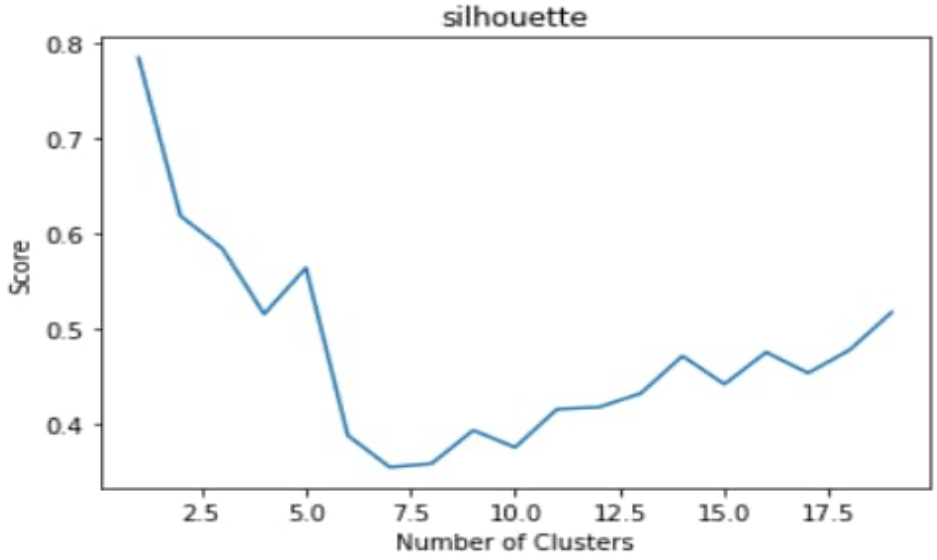
  - Silhouette_Score to find out optimal number of k

```
sil = []
kmax = 20

# dissimilarity would not be defined for a single cluster, thus, minimum number of clusters should be 2
for k in range(2, kmax+1):
  kmeans = KMeans(n_clusters = k).fit(account_df)
  labels = kmeans.labels_
  sil.append(silhouette_score(account_df, labels, metric = 'euclidean'))
```

```
sil
```

```
[0.7852847313695429,
 0.6271114225288045,
 0.5877554912117477,
 0.5155926697740951,
 0.564175225954167,
 0.45070084452973397,
```

```
# Plot the curve
kmax = 20
plt.plot(range(1, kmax),sil)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('silhouette')
plt.show()
```

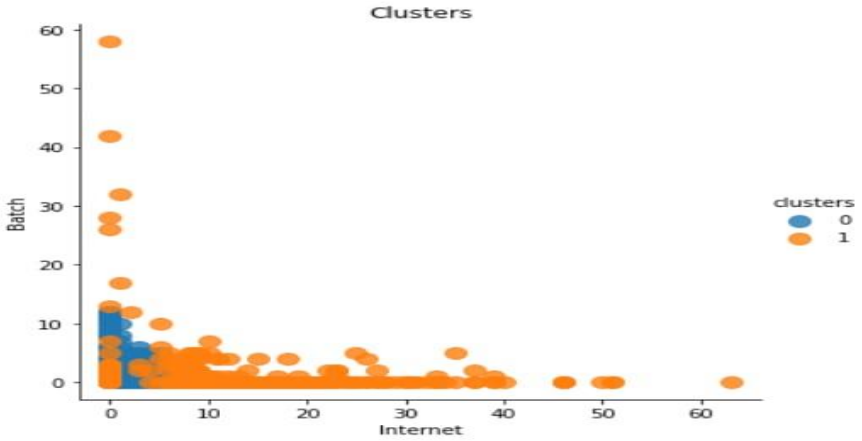


- Optimal number of clusters reckoned for clustering through elbow graph and silhouette score is 2 for Kmeans clustering
- Initiated the Kmeans model with parameter as int = kmeans++ and randon_state = 0

```
#Cluster the data
kmeans = KMeans(n_clusters = 2, random_state=0).fit(account_df)
labels = kmeans.labels_

#Glue back to originaal data
account_df['clusters'] = labels

#Add the column into our list
columns.extend(['clusters'])

#Creating clusters
clusters_col =  kmeans.predict(account_df)
```

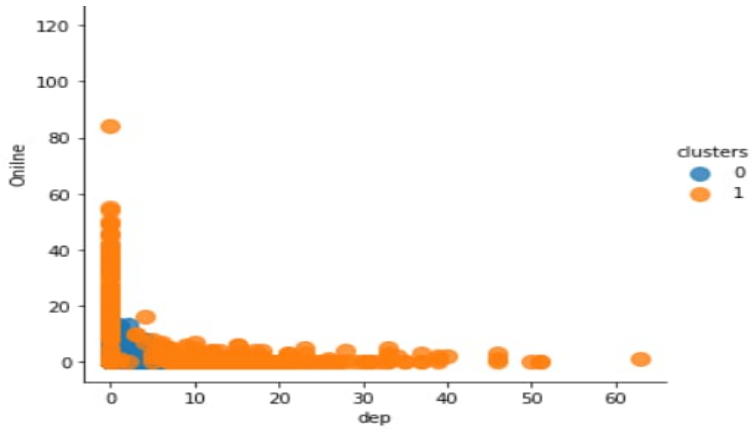
- Saved the clustering output in a variable and added back to the original dataset for visualization
- Clusters Visualization – Internet VS Batch

```
sns.lmplot('Internet','Batch',
           data=account_df,
           fit_reg=False,
           hue="clusters",
           scatter_kws={"marker": "D",
                        "s": 100})
plt.title('Clusters')
plt.xlabel('Internet')
plt.ylabel('Batch')
```



- Clusters Visualization – Deposit VS Onilne

```
sns.lmplot('Internet','Onilne',
           data=account_df,
           fit_reg=False,
           hue="clusters",
           scatter_kws={"marker": "D",
                        "s": 100})
plt.title('Clusters')
plt.xlabel('dep')
plt.ylabel('Onilne')
```



- Account Level Clustering – PCA

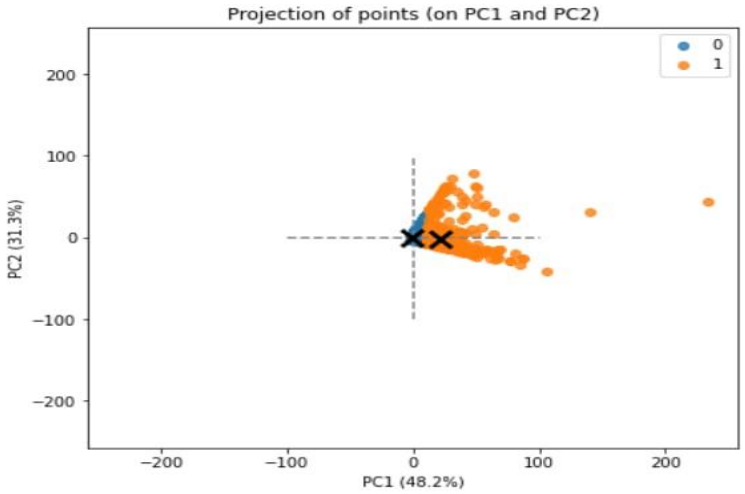
  - Used PCA to reduce our data to 2 dimensional for better understanding

```
# Create a PCA model to reduce our data to 2 dimensions for visualisation
pca = PCA(n_components=2)
pca.fit(account_df)

# Transfor the scaled data to the new PCA space
account_df_reduced = pca.transform(account_df)
```

- Parameter Used n_component = 2
- Biopolar graph shows the centroid of two clusters

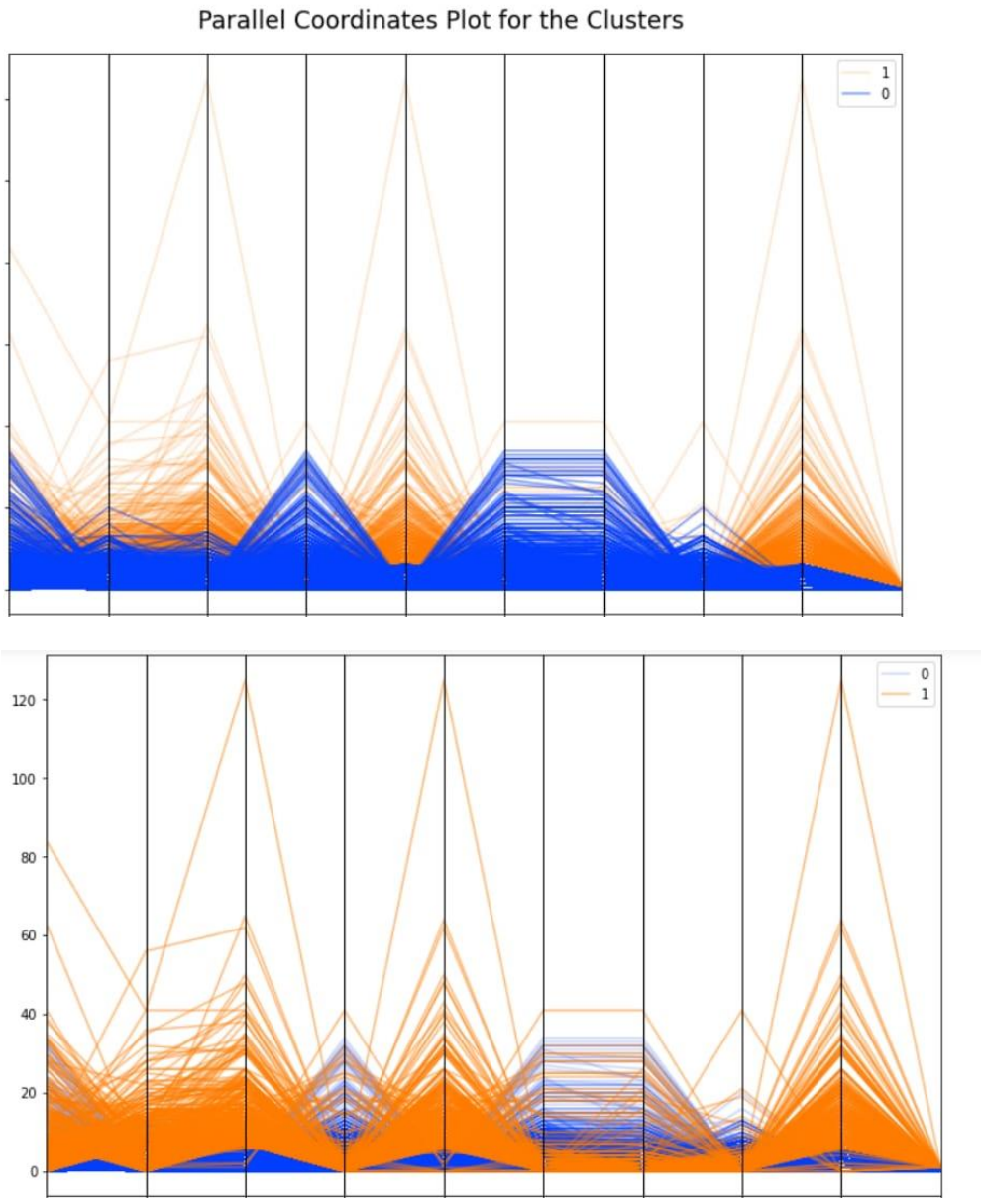
```
# Convert to a data frame
account_df_reduced_df = pd.DataFrame(account_df_reduced, index=account_df.index, columns=['PC1','PC2'])
account_df_reduced_df['cluster'] = clusters_col
account_df_reduced_df.head()
```

```
display_factorial_planes(account_df_reduced, 2, pca, [(0,1)], illustrative_var = clusters_col, alpha = 0.8)
plt.scatter(centres_reduced[:, 0], centres_reduced[:, 1],
            marker='x', s=169, linewidths=3,
            color='k', zorder=10)
```



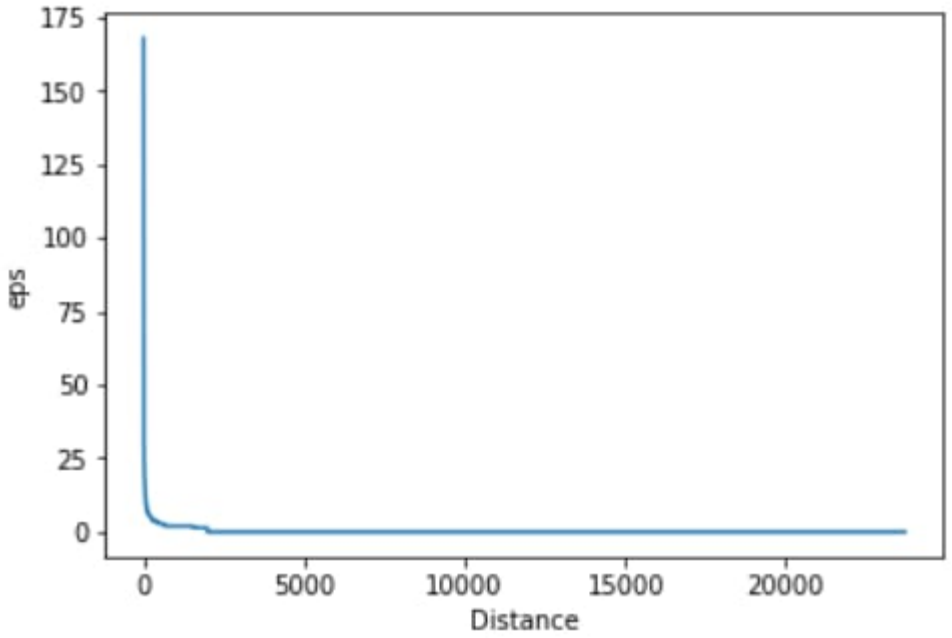
- Account Level Clustering – Parallel Coordinate Plot

```
# Add the cluster number to the original scaled data
X_clustered = pd.DataFrame(transactions_encoded, index=transactions_encoded.index,
                           columns=transactions_encoded.columns)
X_clustered["cluster"] = clusters_col

# Display parallel coordinates plots, one for each cluster
display_parallel_coordinates(X_clustered, 2)
```



- Account Level Clustering – DB Scan Cluster
  - Calculating the value for Epsilon

```
#Visualize DBSCAN clustering
df_DBSCAN=Data_DBSCAN
df_DBSCAN['Cluster_id_DBSCAN']=results
print (df_DBSCAN['Cluster_id_DBSCAN'].value_counts())
sns.pairplot(df_DBSCAN,hue='Cluster_id_DBSCAN',palette='Dark2',diag_kind='kde')
```

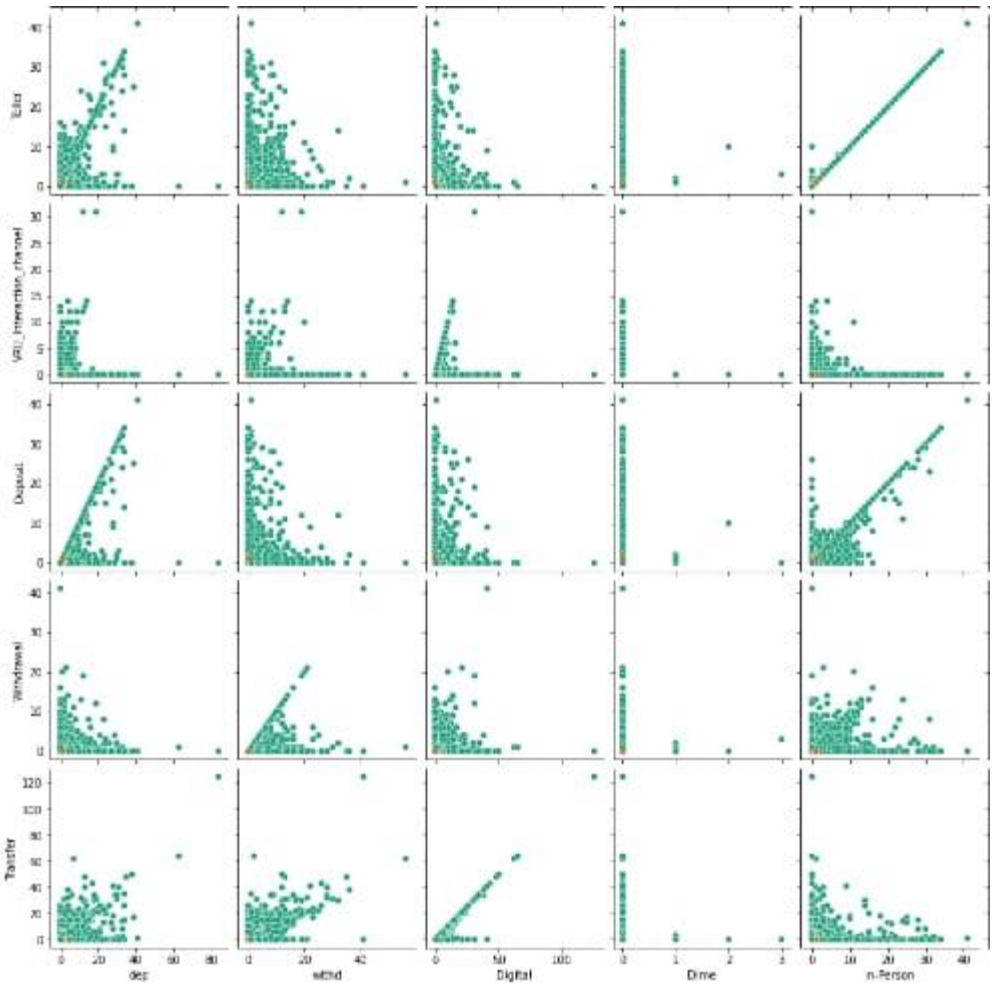


- Implementing DB Scan

```
#DBSCAN Algorithm
from sklearn.cluster import DBSCAN
dbs_1= DBSCAN(eps=0.065, min_samples = 4000)
results = dbs_1.fit(Data_DBSCAN).labels_
```

- Visualization of DB Scan

```
from sklearn.neighbors import NearestNeighbors
nbrs=NearestNeighbors().fit(Data_DBSCAN)
distances, indices = nbrs.kneighbors(Data_DBSCAN,20)
kDis = distances[:,10]
kDis.sort()
kDis = kDis[range(len(kDis)-1,0,-1)]
plt.plot(range(0,len(kDis)),kDis)
plt.xlabel('Distance')
plt.ylabel('eps')
plt.show()
```



Transactional Level

- Transactional Level data looks like

	TranInteractionMeth	TranInteractionChannel	TranInteractionDirection	day_of_week	day_of_month
0	In-Person	Teller	Withdrawal	1	22
1	In-Person	Teller	Deposit	1	1
2	Digital	Online Banking	Transfer	2	16
3	In-Person	Teller	Deposit	5	12
4	In-Person	Teller	Withdrawal	1	15

- Removed Outliers and Normalized the numerical columns

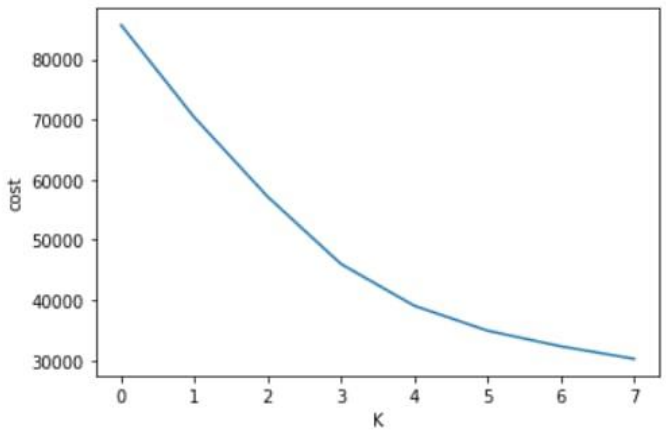
```
#Removing outliers
#Change limits of outliers
def outlier_removal(data,column):
    q1 = data[column].quantile(.10)
    q3 = data[column].quantile(.90)
    mask = data[column].between(q1, q3, inclusive=True)
    iqr = data.loc[mask, column]
    return (iqr)
```

```
transactions_normalized_3col[columns_to_normalize] = transactions_otlr_rmvd_2col[columns_to_normalize].
                                                     apply(lambda x: (x - x.mean()) / np.std(x))
```

- Transactional level Clustering – K-Prototype

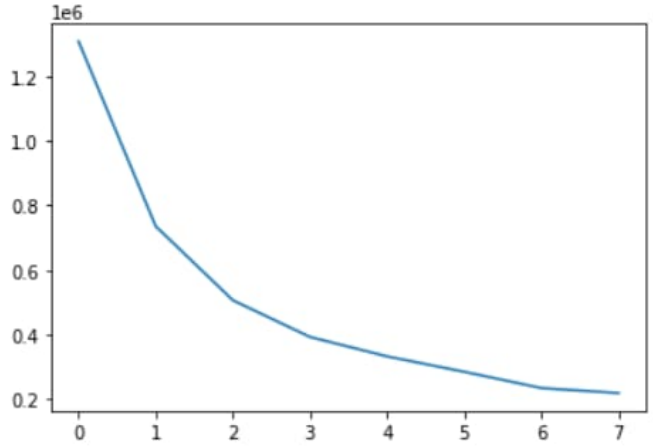
  - K-Prototype works directly with the categorical data without the need for encoding
  - Calculating the value for optimal K using Huang

```
#Choosing optimal K value using Huang
cost = []
X = transactions_normalized_array
for num_clusters in list(range(2,10)):
    kproto = KPrototypes(n_clusters=num_clusters, init='Huang',
                         random_state=42,n_jobs=-2,max_iter=15,n_init=50)
    kproto.fit_predict(X, categorical=[0, 1, 2])
    cost.append(kproto.cost_)

plt.plot(cost)
plt.xlabel('K')
plt.ylabel('cost')
plt.show
```



  - Calculating the value for optimal K using Cao

```
#Choosing optimal K
cost = []
for cluster in list(range(2,10)):
    kproto = KPrototypes(n_clusters=cluster, init='Cao')
    kproto.fit_predict(transactions_normalized_array, categorical=[0, 1, 2])
    cost.append(kproto.cost_)
plt.plot(cost);
```



- Converting data frame into array – prerequisite of K-Prototype

```
# Coverting dataframe into array
transactions_normalized_array=transactions_normalized_3col.values
```

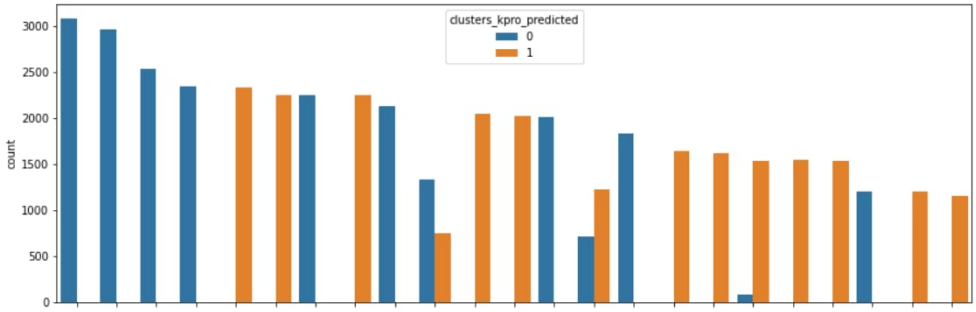
```
# Coverting neumerical columns of array into float
transactions_normalized_array[:, 3] = transactions_normalized_array[:, 3].astype(float)
transactions_normalized_array[:, 4] = transactions_normalized_array[:, 4].astype(float)
transactions_normalized_array[:, 5] = transactions_normalized_array[:, 5].astype(float)
```

```
transactions_normalized_array
```

```
array([['In-Person', 'Teller', 'Withdrawal', 0.7250136834636244, 0.0,
        -0.11524840213863105],
       ['In-Person', 'Teller', 'Deposit', -1.6582051548341366,
        -0.24529528770798806, 0.0],
       ['Digital', 'Online Banking', 'Transfer', 0.04409401537854982,
        0.0, 0.4582793947486272],
       ...,
       ['In-Person', 'Teller', 'Deposit', 0.2710672380735747,
        1.7628074676769212, 0.0],
       ['In-Person', 'Teller', 'Deposit', -0.4098524300114999,
        -0.6757350239905808, 0.0],
       ['In-Person', 'Teller', 'Deposit', 0.611527072116112,
        -0.7041001841116082, 0.0]], dtype=object)
```

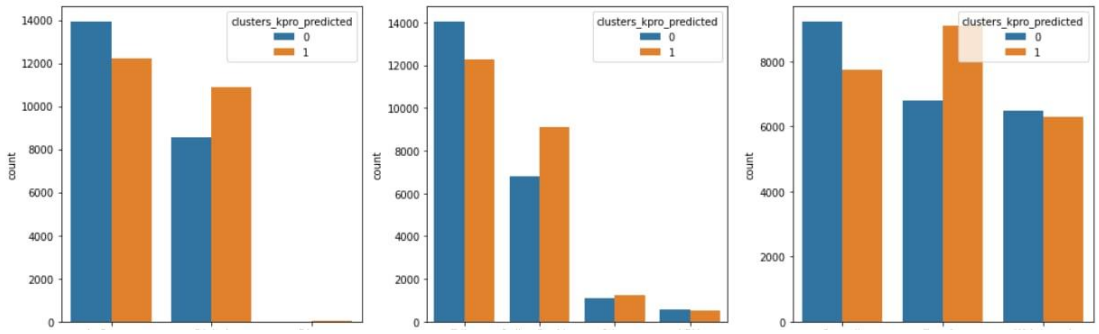
- K-Prototype Visualization – DayOfTheMonth VS Clusters

```
plt.subplots(figsize = (15,5))
sns.countplot(x=combined_Kpro['day_of_month'],order=combined_Kpro['day_of_month'].
              value_counts().index,hue=combined_Kpro['clusters_kpro_predicted'])
plt.show()
```



- K-Prototype Visualization – DayOfTheMonth VS Clusters

```
# Viewing all the categorical column together
f, axs = plt.subplots(1,3,figsize = (15,5))
sns.countplot(x=combined_Kpro['TranInteractionMeth'],order=combined_Kpro['TranInteractionMeth']
              .value_counts().index,hue=combined_Kpro['clusters_kpro_predicted'],ax=axs[0])
sns.countplot(x=combined_Kpro['TranInteractionChannel'],order=combined_Kpro['TranInteractionChannel']
              .value_counts().index,hue=combined_Kpro['clusters_kpro_predicted'],ax=axs[1])
sns.countplot(x=combined_Kpro['TranInteractionDirection'],order=combined_Kpro['TranInteractionDirection']
              .value_counts().index,hue=combined_Kpro['clusters_kpro_predicted'],ax=axs[2])

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



- Transactional level Clustering – K-Mode

  - For K-Mode first we do label encoding

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
transactions_Kmode = transactions_Kmode.apply(le.fit_transform)
transactions_Kmode.head()
```

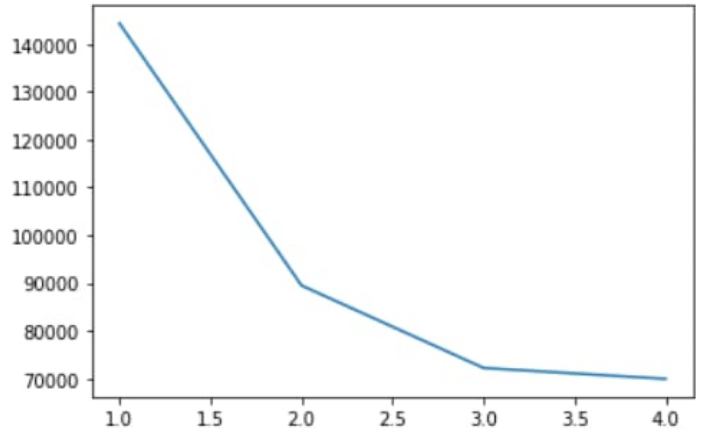
	TranInteractionMeth	TranInteractionChannel	TranInteractionDirection	day_of_week	day_of_month	Tran_bin
0	2	2	2	1	12	0
1	2	2	0	1	0	2
2	0	0	1	2	6	1
3	2	2	0	5	3	3
4	2	2	2	1	5	0

- Calculate Optimal number of K using Cao method

```
cost = []
for num_clusters in list(range(1,5)):
    kmode = KModes(n_clusters=num_clusters, init = "Cao", n_init = 1, verbose=1)
    kmode.fit_predict(transactions_Kmode)
    cost.append(kmode.cost_)
```

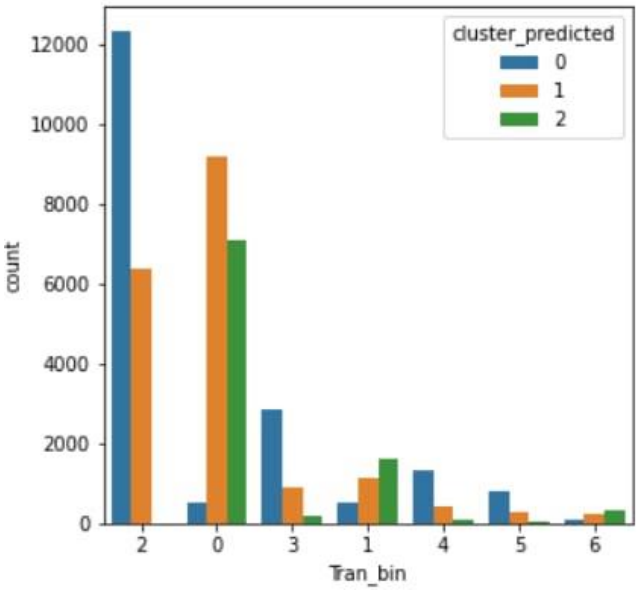
```
y = np.array([i for i in range(1,5,1)])
plt.plot(y,cost)
```

```
[<matplotlib.lines.Line2D at 0x1f5219e26a0>]
```



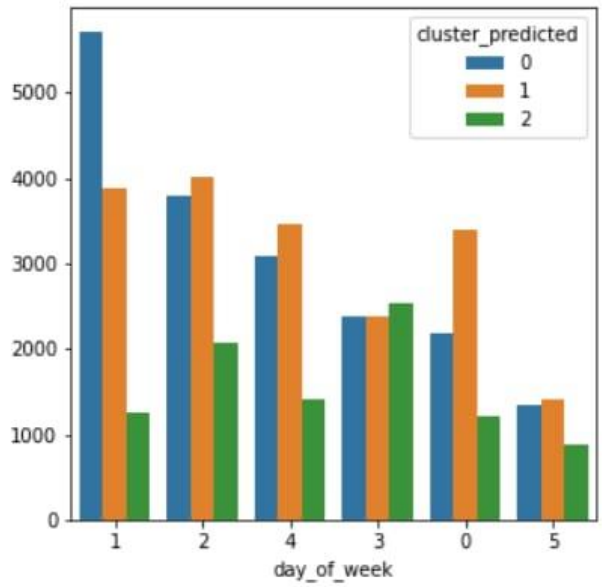
- Implemented K-Modes and optimal come out to be two
- K-Modes Visualization – Bar graph of Tran_Bin w.r.t clusters predicted

```
plt.subplots(figsize = (5,5))
sns.countplot(x=combined_Kmod['Tran_bin'],order=combined_Kmod['Tran_bin']
              .value_counts().index,hue=combined_Kmod['cluster_predicted'])
plt.show()
```



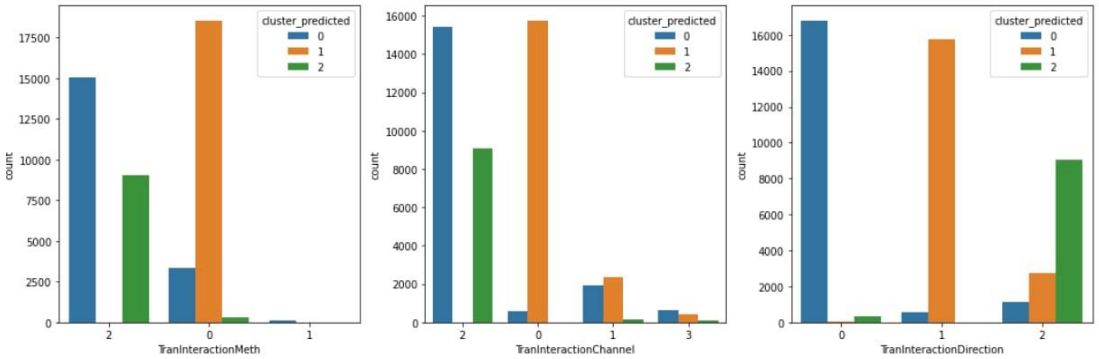
- K-Modes Visualization – Bar graph of days of the week w.r.t clusters predicted

```
plt.subplots(figsize = (5,5))
sns.countplot(x=combined_Kmod['day_of_week'],order=combined_Kmod['day_of_week']
              .value_counts().index,hue=combined_Kmod['cluster_predicted'])
plt.show()
```



- K-Modes Visualization – Bar graph of Modes of Transactions w.r.t clusters predicted

```
f, axs = plt.subplots(1,3,figsize = (15,5))
sns.countplot(x=combined_Kmod['TranInteractionMeth'],order=combined_Kmod['TranInteractionMeth']
              .value_counts().index,hue=combined_Kmod['cluster_predicted'],ax=axs[0])
sns.countplot(x=combined_Kmod['TranInteractionChannel'],order=combined_Kmod['TranInteractionChannel']
              .value_counts().index,hue=combined_Kmod['cluster_predicted'],ax=axs[1])
sns.countplot(x=combined_Kmod['TranInteractionDirection'],order=combined_Kmod['TranInteractionDirection']
              .value_counts().index,hue=combined_Kmod['cluster_predicted'],ax=axs[2])

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

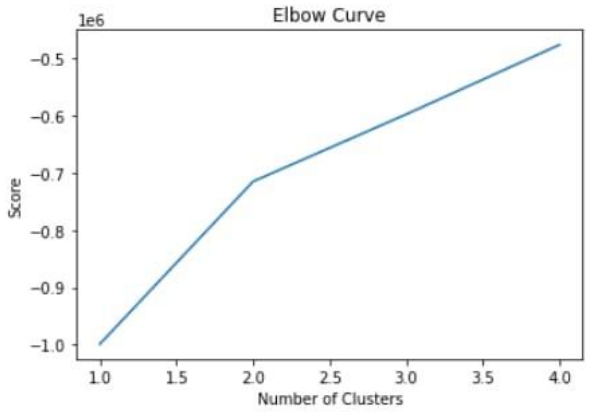


- Transactional level Clustering – Kmeans
  - Before starting the Kmeans clustering in transactional level we have done one hot encoding

	dep	withd	day_of_week	day_of_month	TranInteractionMeth_Digital	TranInteractionMeth_Dime	TranInteractionMeth_In-Person	TranInteractionChannel_Online Banking
0	0.00	800.0	1	22	0	0	1	0
1	954.38	0.0	1	1	0	0	1	0
2	0.00	1500.0	2	16	1	0	0	1
3	2000.00	0.0	5	12	0	0	1	0
4	0.00	5000.0	1	15	0	0	1	0

- Elbow Curve to find the optimal number of clusters

```
# Run a number of tests, for 1, 2, ... num_clusters
num_clusters = 5
kmeans_tests = [KMeans(n_clusters=i, init='random', n_init=10) for i in range(1, num_clusters)]
score = [kmeans_tests[i].fit(account_df).score(account_df) for i in range(len(kmeans_tests))]

# Plot the curve
plt.plot(range(1, num_clusters),score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```



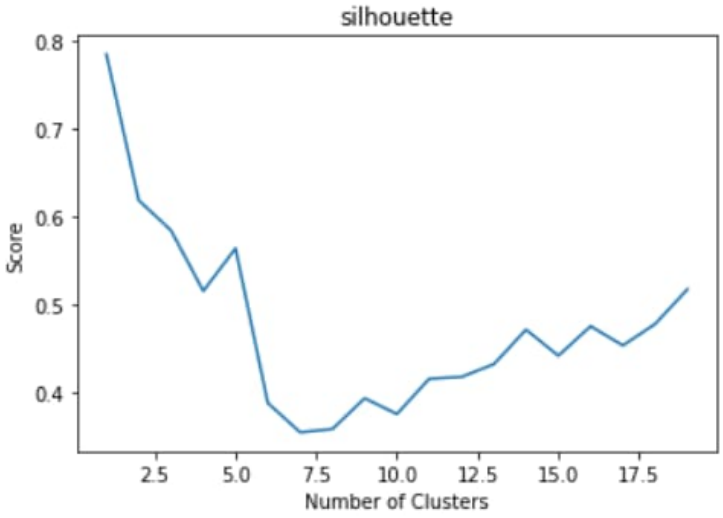
- Silhouette_Score to find out optimal number of k

```
sil = []
kmax = 20

# dissimilarity would not be defined for a single cluster, thus, minimum number of clusters should be 2
for k in range(2, kmax+1):
  kmeans = KMeans(n_clusters = k).fit(account_df)
  labels = kmeans.labels_
  sil.append(silhouette_score(account_df, labels, metric = 'euclidean'))
```

```
sil
```

```
[0.7852847313695429,
 0.6271114225288045,
 0.5877554912117477,
 0.5155926697740951,
 0.564175225954167,
 0.45070084452973397,
```

```
# Plot the curve
kmax = 20
plt.plot(range(1, kmax),sil)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('silhouette')
plt.show()
```

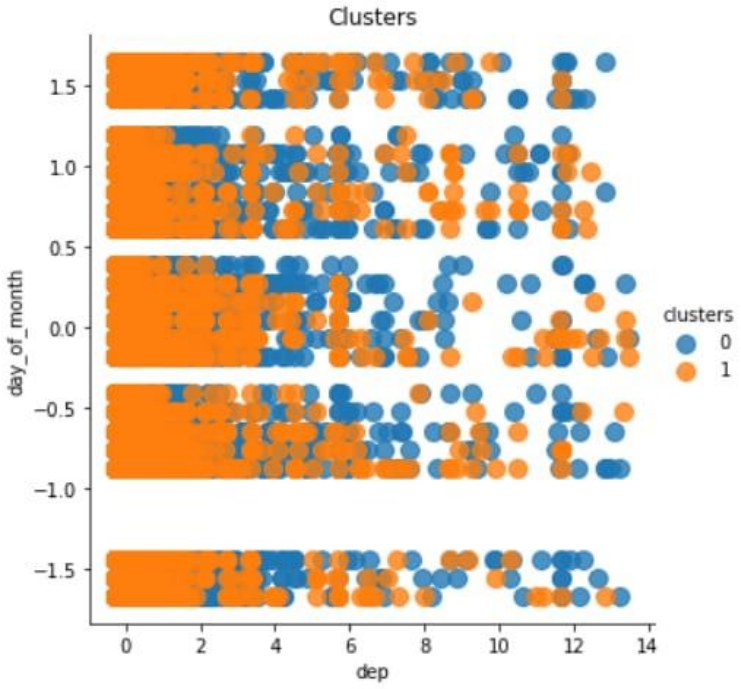


- Optimal number of clusters reckoned for clustering through elbow graph and silhouette score is 2 for Kmeans clustering
- Initiated the Kmeans model with parameter as int = kmeans++ and randon_state = 0

```
#Cluster the data
kmeans = KMeans(n_clusters = 2, random_state=0).fit(account_df)
labels = kmeans.labels_

#Glue back to originaal data
account_df['clusters'] = labels

#Add the column into our list
columns.extend(['clusters'])

#Creating clusters
clusters_col =  kmeans.predict(account_df)
```

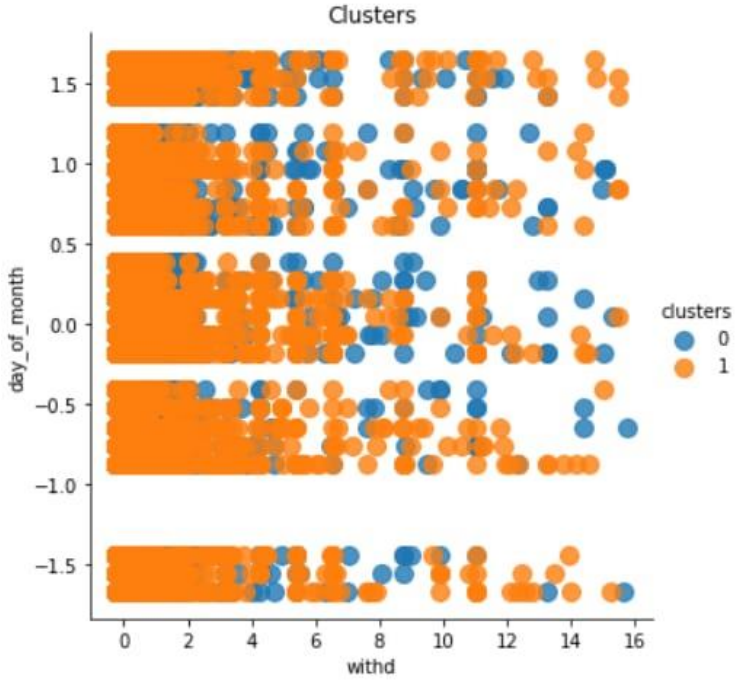
- Saved the clustering output in a variable and added back to the original dataset for visualization
- Clusters Visualization – Internet VS Batch

```
sns.lmplot('dep','day_of_month',
           data=filtered_transaction_df,
           fit_reg=False,
           hue="clusters",
           scatter_kws={"marker": "D",
                        "s": 100})
plt.title('Clusters')
plt.xlabel('dep')
plt.ylabel('day_of_month')
```



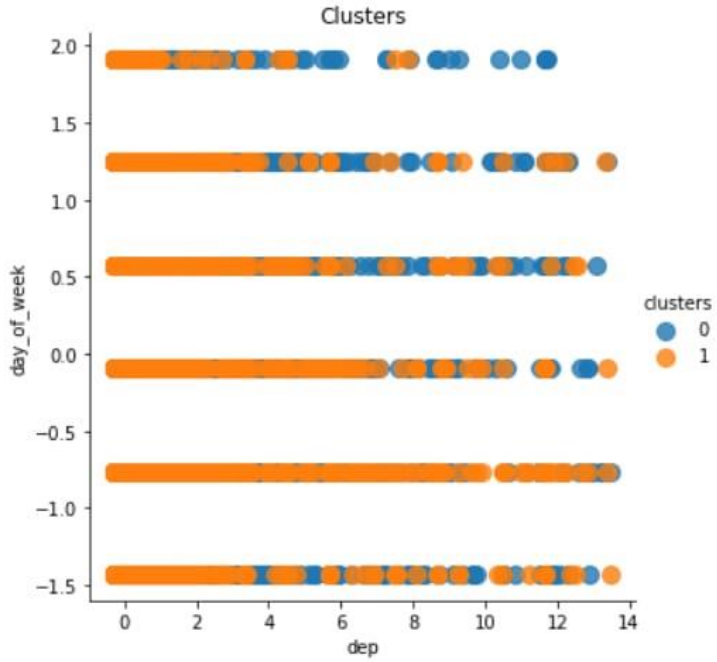
- Clusters Visualization – Internet VS Batch

```
sns.lmplot('withd','day_of_month',
           data=filtered_transaction_df,
           fit_reg=False,
           hue="clusters",
           scatter_kws={"marker": "D",
                        "s": 100})
plt.title('Clusters')
plt.xlabel('withd')
plt.ylabel('day_of_month')
```



- Clusters Visualization – Internet VS Batch

```
sns.lmplot('dep','day_of_week',
           data=filtered_transaction_df,
           fit_reg=False,
           hue="clusters",
           scatter_kws={"marker": "D",
                        "s": 100})
plt.title('Clusters')
plt.xlabel('dep')
plt.ylabel('day_of_week')
```



Conclusion:

Post application of different types of clustering and checking in accordance with various statistical attributes in account level the representation of 2 clusters through K means and in transactional level representation of 3 clusters trough K modes was found to be most meaningful in terms of business and various statistical attributes. To endorse the alignment between business and technical perspective the calinski harabasz score and cost is also found out in k mode and k prototype models for optimal numbers of k value while Silhouette score for k means is found out. In conclusion, from the whole project the strategies and optimization of existing modes of business's decisions could made as data driven ones.

```
from sklearn import metrics
labels_kmode3 = km_cao.labels_
metrics.calinski_harabasz_score(transactions_KmodeK3, labels_kmode3)
```

```
1119.8176869980014
```

```
km_cao.cost_
```

```
107935.0
```

Current/ Future Work:

Deep learning through application of neural network using Tensor Flow for clustering. This is made possible using auto-encoders, a model that is trained to reconstruct the original vector from a smaller representation (hidden layer activations) with reconstruction error (distance from the ID function) as cost function. This process does not give clusters, but it creates meaningful representations that is used for clustering.

Predictive analytics: Creation of a neural network classifier with TensorFlow Keras and training the model with output from the former clustering.

References

- https://towardsdatascience.com/dbscan-algorithm-complete-guide-and-application-with-python-scikit-learn-d690cbae4c5d
- https://towardsdatascience.com/clustering-algorithms-a-one-stop-shop-6cd0959f9b8f
- https://www.bmc.com/blogs/create-machine-learning-pipeline/#:~:text=What%20is%20a%20machine%20learning,works%20on%20their%20ML%20platform.