Analysis Report

Domain: Banking, Finance

Prepared By:

Tohfa Siddika Barbhuiya,

EXL Analytics

31 Jan’2021

EXECUTIVE SUMMARY:

BACKGROUND:

Business Context:

This case requires to develop a customer segmentation to define marketing strategy. 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 Dictionary:

CUSTID: Identification of Credit Card holder (Categorical)

BALANCE: Balance amount left in their account to make purchases

BALANCE FREQUENCY: 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

ONEOFF PURCHASES: Maximum purchase amount done in one-go

INSTALLMENTS PURCHASES: Amount of purchase done in installment

CASHADVANCE: Cash in advance given by the user

PURCHASES FREQUENCY: How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)

ONEOFF PURCHASES FREQUENCY: How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)

PURCHASES INSTALLMENTS FREQUENCY: How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)

CASH ADVANCE FREQUENCY: How frequently the cash in advance being paid

CASH ADVANCE TRX: Number of Transactions made with "Cash in Advance"

PURCHASES TRX: Numbe of purchase transactions made

CREDIT LIMIT: Limit of Credit Card for user

PAYMENTS: Amount of Payment done by user

MINIMUM_PAYMENTS: Minimum amount of payments made by user

PRC FULL PAYMENT: Percent of full payment paid by user

TENURE: Tenure of credit card service for user

KEY FINDINGS:

```
import pandas as pd
import numpy as np

#Date stuff
from datetime import datetime
from datetime import timedelta

#Library for Nice graphing
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sn
%matplotlib inline

#Library for statistics operation
import scipy.stats as stats

# Date Time library
from datetime import datetime

#Machine learning Library
import statsmodels.api as sm
from sklearn import metrics

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

1. Preprocessing the data (15 points)

a. Check a few observations and get familiar with the data. (1 points)

```
df = pd.read_csv("C:/Users/user/Downloads/data_credit_card.csv")
df.head(3)
```

b. Check the size and info of the data set. (2 points)

```
df.info()

df.shape
```

c. Check for missing values. Impute the missing values if there is any. (2 points)

```
df.isnull().sum().values.sum()
```

Since there are missing values in the data so we are imputing them with median

```
df.isnull().any()
# CREDIT_LIMIT  and MINIMUM_PAYMENTS has missing values so we need to remove
with median.

df['CREDIT_LIMIT'].fillna(df['CREDIT_LIMIT'].median(),inplace=True)
```

```
df['CREDIT_LIMIT'].count()
```

```
df['MINIMUM_PAYMENTS'].median()
df['MINIMUM_PAYMENTS'].fillna(df['MINIMUM_PAYMENTS'].median(),inplace=True)
```

Now again check the missing values.

```
df.isnull().any()
```

```
df.isnull().sum().values.sum()
```

d. Drop unnecessary columns. (2 points)

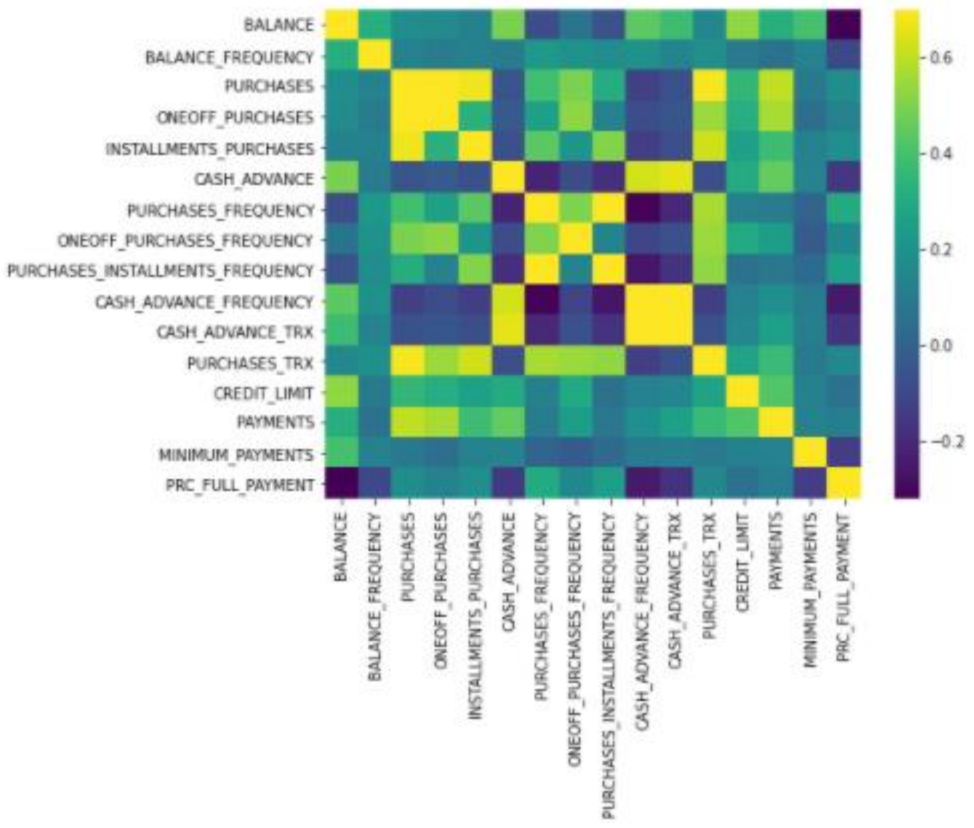
```
df = df.drop(columns=['CUST_ID','TENURE'])
df.sample(7)
```

```
df.shape
```

```
df_copy = df.copy()
```

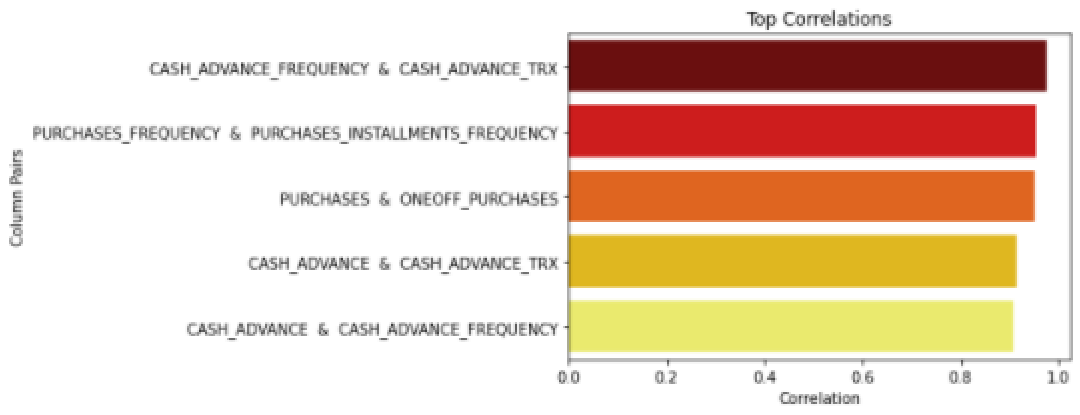
e. Check correlation among features and comment your findings. (3 points)

```
# Heatmap of the Pearson correlation between each pair of features
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), cmap='viridis', vmax=0.70) # I set vmax=0.7 to easily spot correlations greater than or close to 0.7
plt.show()
```



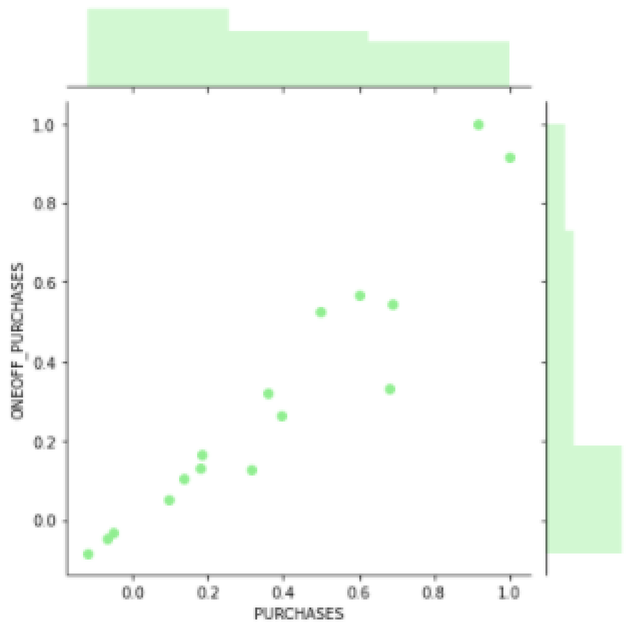
```
df=df.corr()
df


#Function to get unique correlations from the correlation matrix
def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop


# Function to get top n absolute correlation values and corresponding columns
```

```
def get_top_abs_correlations(df, n=5):
    au_corr = df.corr().abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    au_corr = au_corr.drop(labels=labels_to_drop).sort_values(ascending=False)
    return au_corr[0:n]

print("Top Absolute Correlations \n")
print(get_top_abs_correlations(df, 5))

# Visualization of the top correlated variables
x = get_top_abs_correlations(df, 5)
x = x.reset_index()
x.columns = ['Column_1', 'Column_2', 'Correlation' ]
x['Column Pairs'] = x['Column_1'] + '  &  ' + x['Column_2']
sns.barplot(x='Correlation', y='Column Pairs', data=x, palette='hot')
plt.title('Top Correlations')
plt.show()
```

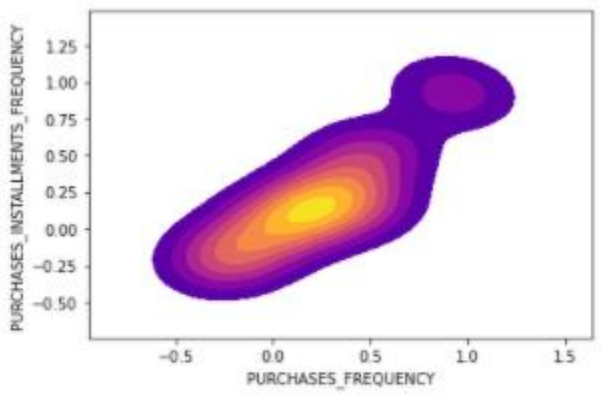


#Very high correlation between PURCHASES and ONEOFF_PURCHASES

```
# Jointplot to visualize the high correlation between PURCHASES and ONEOFF_PURCHASES
sns.jointplot(x='PURCHASES', y='ONEOFF_PURCHASES', data=df, color='lightgreen')
plt.show()
```



```
df[['PURCHASES', 'ONEOFF_PURCHASES']].head(5)


df[['PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES']].head(5)


df.drop('PURCHASES', axis=1, inplace=True)


# KDE plot to visualize high correlation between PURCHASES_FREQUENCY and
PURCHASES_INSTALLMENTS_FREQUENCY
sns.kdeplot( df['PURCHASES_FREQUENCY'],
df['PURCHASES_INSTALLMENTS_FREQUENCY'],
                cmap="plasma", shade=True, shade_lowest=False)
plt.show()
```

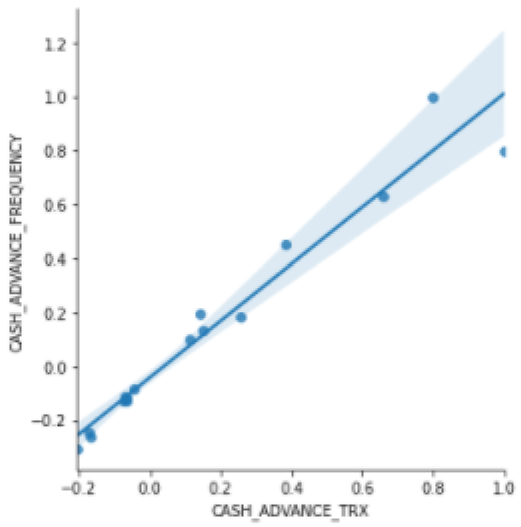


# Dropping PURCHASES_INSTALLMENTS_FREQUENCY

df.drop('PURCHASES_INSTALLMENTS_FREQUENCY', axis=1, inplace=True)

# Plot to visualize linear relationship between CASH_ADVANCE_TRX and CASH_ADVANCE_FREQUENCY

sns.lmplot(x='CASH_ADVANCE_TRX', y='CASH_ADVANCE_FREQUENCY', data=df)



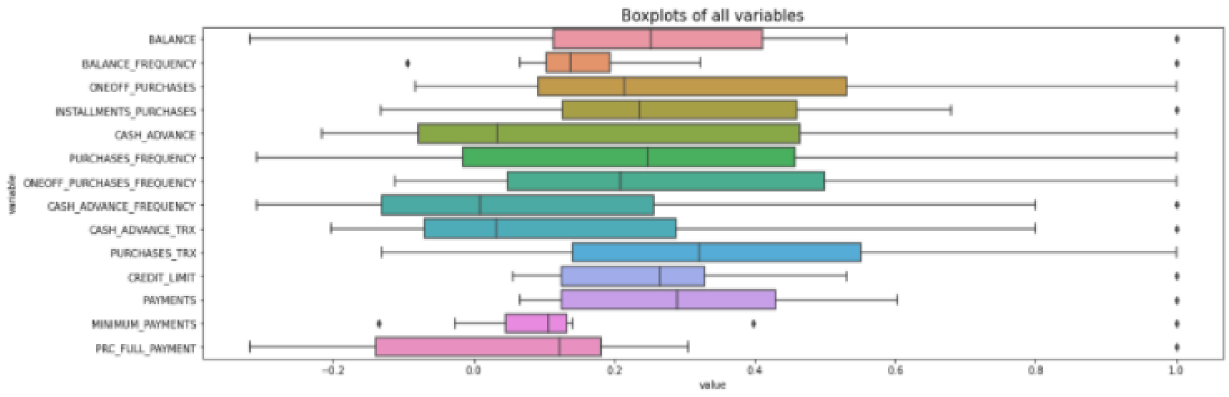
# Plotting Boxplots of all our features to get idea of distribution and outliers

plt.figure(figsize=(18,6))

sns.boxplot(x="value", y="variable", data=pd.melt(df))

plt.title('Boxplots of all variables', size=15)

plt.show()



```
from sklearn.preprocessing import StandardScaler


# Scaling the data
scaler = StandardScaler()
scaler.fit(df)
df = pd.DataFrame(scaler.fit_transform(df), columns= df.columns, index=df.index)
df.head()


# Replacing extreme outliers with 99th and 1st percentiles for each variable
df_kmeans = df.copy()
for i in df_kmeans.columns:
    ulimit = np.percentile(df_kmeans[i].values, 99)
    llimit = np.percentile(df_kmeans[i].values, 1)
    df_kmeans[i].loc[df_kmeans[i]>ulimit] = ulimit
    df_kmeans[i].loc[df_kmeans[i]<llimit] = llimit



# Comparison of Boxplots before and after treating outliers
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(18,12) )
```

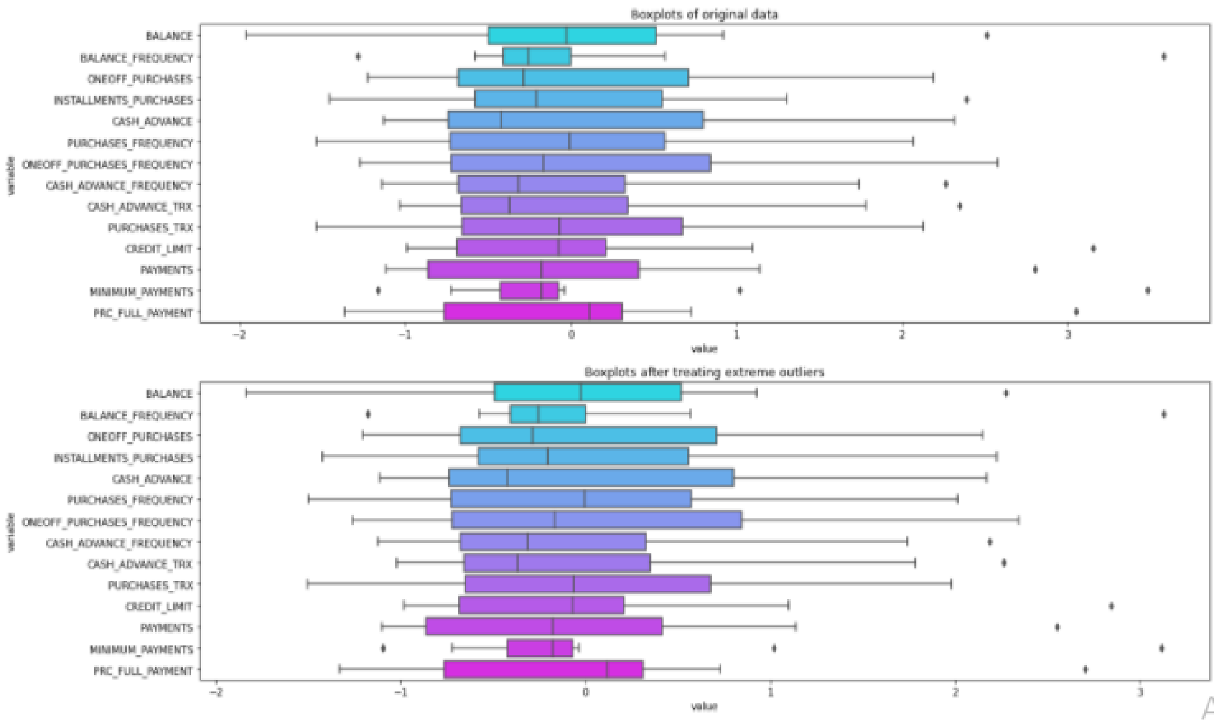
sns.boxplot(x="value", y="variable", data=pd.melt(df), ax=ax[0], palette='cool')

ax[0].title.set_text('Boxplots of original data')

sns.boxplot(x="value", y="variable", data=pd.melt(df_kmeans), ax=ax[1], palette='cool')

ax[1].title.set_text('Boxplots after treating extreme outliers')

plt.show()



#g. Standardize the data using appropriate methods. (2 points)

from sklearn.decomposition import PCA

scaler = StandardScaler()

scaler.fit(df_copy)

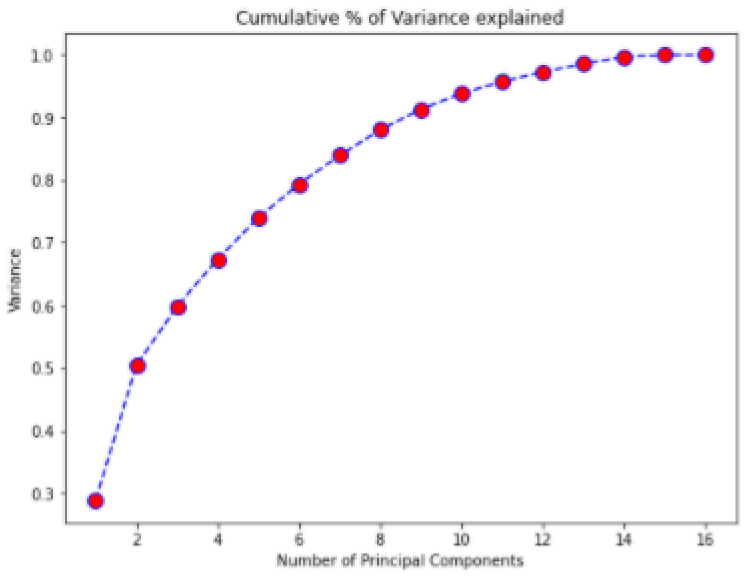
scaled_df = pd.DataFrame(scaler.fit_transform(df_copy), columns= df_copy.columns, index=df_copy.index)

scaled_df.head()

#3. Apply PCA to the dataset and perform all steps from Q2 on the new features generated using PCA. (15

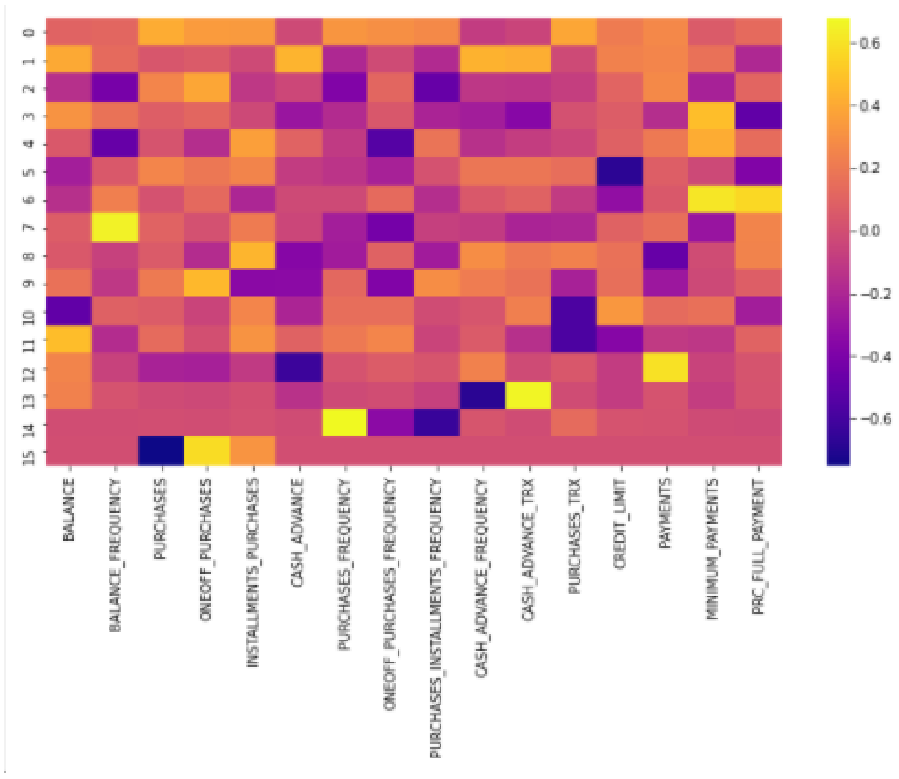
#points)

```
#Fitting the PCA algorithm to our data

pca = PCA().fit(scaled_df)


#Plotting the Cumulative Summation of the Explained Variance

plt.figure(figsize=(8,6))

plt.plot(list(range(1, df_copy.shape[1]+1)), np.cumsum(pca.explained_variance_ratio_),
color='blue', linestyle='dashed', marker='o',

        markerfacecolor='red', markersize=10)

plt.xlabel('Number of Principal Components')

plt.ylabel('Variance') #for each component

plt.title('Cumulative % of Variance explained')

plt.show()
```



# Cumulative Variance values

```
print(np.cumsum(pca.explained_variance_ratio_))


# PCA transformed dataset
cols=[]
for i in range(1,df_copy.shape[1]+1):
    cols.append('PC_' + str(i))


df_pca = pca.fit_transform(scaled_df)
df_pca = pd.DataFrame(df_pca, columns=cols, index= df_copy.index)
df_pca.head()



#Interpreting the share of each feature in forming the Principal Components
df_comp = pd.DataFrame(pca.components_, columns=df_copy.columns)
plt.figure(figsize=(12,6))
sns.heatmap(df_comp,cmap='plasma')
plt.show()
```



'''Build a k-means algorithm for clustering credit card data. Kindly follow the below steps and answer the

following. (10 points)

a. Build k means model on various k values and plot the inertia against various k values.

b. Evaluate the model using Silhouette coefficient

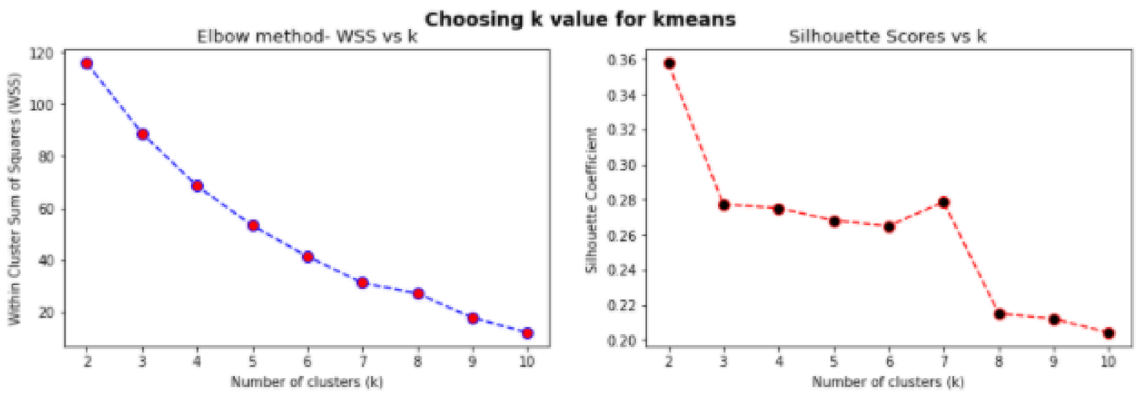
c. Plot an elbow plot to find the optimal value of k

d. Which k value gives the best result?'''

```
# Imports for kmeans

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.cm as cm
```

```
# k-means with 10 different centroid seeds
# (init= 'k-means++' : selects initial cluster centers in a smart way to speed up convergence)

# Function to help us do cluster analysis using Silhouette plots and calculate the WSS (Within cluster Sum of Squared error)
# and Silhouette scores for different values of k (number of clusters).

def kmeans_analysis(df_kmeans, random_state=101):

    range_n_clusters = list(range(2,11))
    silhouette_scores = []
    wss = []

    # Taking 2 Principal Components for the dataset for purpose of visualization
    pca = PCA().fit(df_kmeans)
    X = pca.fit_transform(df_kmeans)

    # Looping through the values of k
    for n_clusters in range_n_clusters:
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.set_size_inches(18, 7)
        ax1.set_xlim([-0.1, 1])
        ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

        clusterer = KMeans(n_clusters=n_clusters, random_state= random_state)
        cluster_labels = clusterer.fit_predict(df_kmeans)

        # The silhouette_score gives the average value for all the samples.
        silhouette_avg = silhouette_score(df_kmeans, cluster_labels)
        print("For n_clusters =", n_clusters,
              "The average silhouette_score is :", round(silhouette_avg,2))

        # Appending silhouette score and within cluster squared error to seperate lists for the
particular k value
        silhouette_scores.append(silhouette_avg)
        wss.append(clusterer.inertia_)

        # Silhouette scores for each sample
        sample_silhouette_values = silhouette_samples(df_kmeans, cluster_labels)

        y_lower = 10
        for i in range(n_clusters):
            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them
            ith_cluster_silhouette_values = \
                sample_silhouette_values[cluster_labels == i]

            ith_cluster_silhouette_values.sort()

            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i

            color = cm.nipy_spectral(float(i) / n_clusters)
            ax1.fill_betweenx(np.arange(y_lower, y_upper),
                              0, ith_cluster_silhouette_values,
                              facecolor=color, edgecolor=color, alpha=0.7)

            # Compute the new y_lower for next plot
            y_lower = y_upper + 10

        ax1.set_title("The silhouette plot for the clusters.")
        ax1.set_xlabel("The silhouette coefficient values")
        ax1.set_ylabel("Cluster label")



        # The vertical line for average silhouette score of all the values
        ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

        ax1.set_yticks([])
        ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])



        # 2nd Plot showing the actual clusters formed
        colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
        ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                    c=colors, edgecolor='k')

        ax2.set_title("Visualization of the clustered data on 1st and 2nd PC")
        ax2.set_xlabel("Feature space for PC_1")
        ax2.set_ylabel("Feature space for PC_2")

        plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                      "with n_clusters = %d" % n_clusters),
                     fontsize=14, fontweight='bold')

    plt.show()
```

```
    return silhouette_scores, wss


# Function to plot WSS and Silhouette scores vs No. of clusters
def make_plots(wss, silhouette_scores):
    fig, (ax1,ax2) = plt.subplots(nrows=1, ncols=2, figsize=(14,4))


    ax1.plot(range(2, 11), wss, color='blue', linestyle='dashed', marker='o', markerfacecolor='red',
markersize=8)
    ax1.set_title('Elbow method- WSS vs k', size=13)
    ax1.set_xlabel('Number of clusters (k)')
    ax1.set_ylabel('Within Cluster Sum of Squares (WSS)') #within cluster sum of squares


    ax2.plot(range(2, 11), silhouette_scores, color='red', linestyle='dashed', marker='o',
markerfacecolor='black', markersize=8)
    ax2.set_title('Silhouette Scores vs k', size=13)
    ax2.set_xlabel('Number of clusters (k)')
    ax2.set_ylabel('Silhouette Coefficient')


    plt.suptitle(('Choosing k value for kmeans'),
             fontsize=14, fontweight='bold')
    plt.show()



silhouette_scores, wss = kmeans_analysis(df_kmeans)


make_plots(wss, silhouette_scores)
```



d. Which k value gives the best result?

A peak appears at k=5 in the right graph.

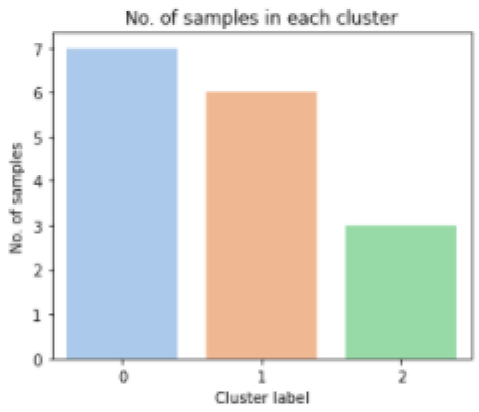
When we examine the Silhouette plot for k=5, we see that all clusters have points with above average Silhouette score, which is a good sign. But some clusters also have points with negative scores which represents poor clustering. We can also observe that we obtain one big cluster and four almost equal sized clusters.

k=3 is also an option. There is an elbow in the left graph and Silhouette score in the right graph is high too.

4. Create a new column as a cluster label in the original data frame and perform cluster analysis. Check the correlation of cluster labels with various features and mention your inferences. (Hint - Does cluster 1 have a high credit limit?) (5 points)

```
# Visualizing cluster sizes

km = KMeans(n_clusters=3, random_state=101)

km.fit(df_kmeans)

x = pd.Series(km.labels_).value_counts().sort_index().reset_index()

x.columns= ['Cluster label', 'No. of samples']
```

```
#x.set_index('Cluster label', inplace=True)
plt.figure(figsize=(5,4))
plt.title("No. of samples in each cluster")
sns.barplot(x='Cluster label', y='No. of samples', data=x, palette='pastel')
plt.show()
```

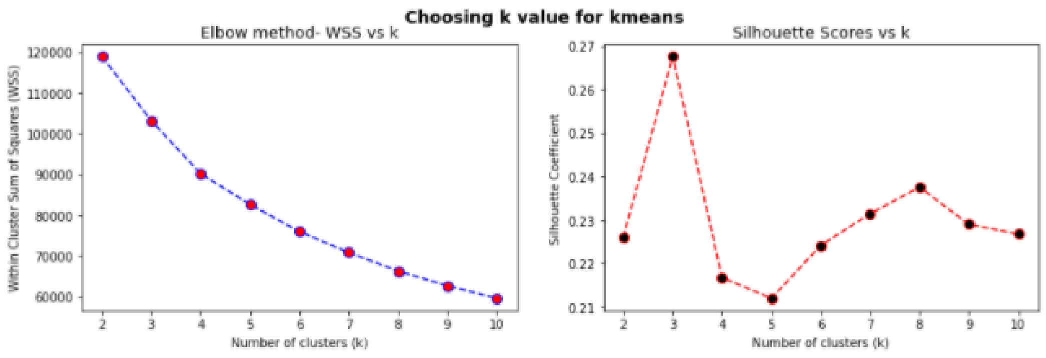


```
# Appending labels to the dataframe
df_kmeans['kmeans_label'] = km.labels_
df_kmeans.head()
```

```
df_kmeans.head()
```

```
df_pca.head()
```

```
silhouette_scores_pca, wss_pca = kmeans_analysis(df_pca)
```



5. Comment your findings and inferences and compare the performance. Does applying PCA give a better result in comparison to earlier? (5 points)

Kmeans algorithm is giving maximum Silhouette score for 3 clusters (k=3). Treatment of outliers and correlated variables was an important pre-processing step.

The algorithm performed efficiently and the need to do post-processing of the clusters did not arise.

Using PCA to make the features independent is an option to treat multi-collinearity and preserve information, but then interpretation of the clusters become difficult.

CONCLUSION:

The analysis made it easier for us to understand our customers and their needs more efficiently. The correlation and various models are helping us to do so to adapt a better strategy.