Market Segmentation

# IMPORT LIBRARIES AND DATASETS

**# This need to execute in tensor flow environment**

**Import Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, normalize

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

import os

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

**Check Current Directory**

os.getcwd()

**Change the directory**

os.chdir ('C:\\Noble\\Training\\Top Mentor\\Training\\Presentation\\Project\\Project -5 Marketting Department\\')

os.getcwd()

**Read Sales Data, display top 5 records**

creditcard\_df = pd.read\_csv('Marketing\_data.csv')

display (creditcard\_df )

# CUSTID: Identification of Credit Card holder

# BALANCE: Balance amount left in customer's 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

# ONEOFFPURCHASES: Maximum purchase amount done in one-go

# INSTALLMENTS\_PURCHASES: Amount of purchase done in instalment

# CASH\_ADVANCE: 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: Number 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

**Data Set Details - Info**

creditcard\_df.info()

# 18 features with 8950 points

**Data Set Details – Describe**

creditcard\_df.describe()

# Mean balance is $1564

# Balance frequency is frequently updated on average ~0.9

# Purchases average is $1000

# one off purchase average is ~$600

# Average purchases frequency is around 0.5

# average ONEOFF\_PURCHASES\_FREQUENCY, PURCHASES\_INSTALLMENTS\_FREQUENCY, and CASH\_ADVANCE\_FREQUENCY are generally low

# Average credit limit ~ 4500

# Percent of full payment is 15%

# Average tenure is 11 years

**Customer with maximum 'ONEOFF\_PURCHASES'**

# Check who made one off purchase of $40761 ie maximum ONEOFF\_PURCHASES

creditcard\_df[creditcard\_df['ONEOFF\_PURCHASES'] == 40761.25]

**Customer with maximum Cash Advance**

creditcard\_df['CASH\_ADVANCE'].max()

# Check who made cash advance of $47137

# This customer made 123 cash advance transactions

# Never paid credit card in full

creditcard\_df[creditcard\_df['CASH\_ADVANCE'] == 47137.211760000006]

# VISUALIZE AND EXPLORE DATASET

**Check missing values**

# # Check for missing Data

# creditcard\_df.isnull().sum()

# Heat map for missing data

# sns.heatmap(creditcard\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")

**Fill the missing values - Column MINIMUM\_PAYMENTS and CREDIT\_LIMIT**

# Fill up the missing elements with mean of the 'MINIMUM\_PAYMENT'

creditcard\_df.loc[(creditcard\_df['MINIMUM\_PAYMENTS'].isnull() == True), 'MINIMUM\_PAYMENTS'] = creditcard\_df['MINIMUM\_PAYMENTS'].mean()

# Fill up the missing elements with mean of the 'CREDIT\_LIMIT'

creditcard\_df.loc[(creditcard\_df['CREDIT\_LIMIT'].isnull() == True), 'CREDIT\_LIMIT'] = creditcard\_df['CREDIT\_LIMIT'].mean()

**Check missing values**

creditcard\_df.isnull().sum()

**Missing values Heatmap**

sns.heatmap(creditcard\_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")

**Check for Duplicate Data**

# Check for duplicated entries in the data

creditcard\_df.duplicated().sum()

**Drop customer id column**

# Drop Customer ID since it has no meaning here

creditcard\_df.drop("CUST\_ID", axis = 1, inplace= True)

display(creditcard\_df)

**Number of Columns**

n = len(creditcard\_df.columns)

display(n)

**Display Column Names**

display (creditcard\_df.columns)

**Create dist plot**

# distplot combines the matplotlib.hist function with seaborn kdeplot()

# KDE Plot represents the Kernel Density Estimate

# KDE is used for visualizing the Probability Density of a continuous variable.

# KDE demonstrates the probability density at different values in a continuous variable.

# Mean of balance is $1500

# 'Balance\_Frequency' for most customers is updated frequently ~1

# For 'PURCHASES\_FREQUENCY', there are two distinct group of customers

# For 'ONEOFF\_PURCHASES\_FREQUENCY' and 'PURCHASES\_INSTALLMENT\_FREQUENCY' most users don't do one off puchases or installment purchases frequently

# Very small number of customers pay their balance in full 'PRC\_FULL\_PAYMENT'~0

# Credit limit average is around $4500

# Most customers are ~11 years tenure

plt.figure(figsize=(20,80))

for i in range(len(creditcard\_df.columns)):

plt.subplot(17, 1, i+1)

sns.distplot(creditcard\_df[creditcard\_df.columns[i]], kde\_kws={"color": "b", "lw": 3, "label": "KDE"}, hist\_kws={"color": "g"})

plt.title(creditcard\_df.columns[i])

plt.tight\_layout()

**Create Pair Plot**

sns.pairplot(creditcard\_df)

# Correlation between 'PURCHASES' and ONEOFF\_PURCHASES & INSTALMENT\_PURCHASES

# Trend between 'PURCHASES' and 'CREDIT\_LIMIT' & 'PAYMENTS'

**Display Co relation Matrix**

correlations = creditcard\_df.corr()

display (correlations )

**Create Heat Map**

f, ax = plt.subplots(figsize = (20, 20))

sns.heatmap(correlations, annot = True)

# 'PURCHASES' have high correlation between one-off purchases, 'installment purchases, purchase transactions, credit limit and payments.

# Strong Positive Correlation between 'PURCHASES\_FREQUENCY' and 'PURCHASES\_INSTALLMENT\_FREQUENCY'

OPTIMAL NUMBER OF CLUSTERS

**Display the Data Set**

display (creditcard\_df)

**Standardise the Data**

# Let's scale the data first

scaler = StandardScaler()

creditcard\_df\_scaled = scaler.fit\_transform(creditcard\_df)

**Display the Shape**

creditcard\_df\_scaled.shape

**Display Standardised Data Set**

display(pd.DataFrame(creditcard\_df\_scaled))

**Create Elbow Graph**

scores\_1 = []

range\_values = range(1, 20)

for i in range\_values:

kmeans = KMeans(n\_clusters = i)

kmeans.fit(creditcard\_df\_scaled)

scores\_1.append(kmeans.inertia\_)

plt.plot(scores\_1, 'bx-')

plt.title('Finding the right number of clusters')

plt.xlabel('Clusters')

plt.ylabel('Scores')

plt.show()

# From this we can observe that, 4th cluster seems to be forming the elbow of the curve.

# However, the values does not reduce linearly until 8th cluster.

# Let's choose the number of clusters to be 7.

**APPLY K -Means**

kmeans = KMeans(8)

kmeans.fit(creditcard\_df\_scaled)

labels = kmeans.labels\_

**Number of Clusters**

kmeans.cluster\_centers\_.shape

**Cluster Canters**

cluster\_centers = pd.DataFrame(data = kmeans.cluster\_centers\_, columns = [creditcard\_df.columns])

display(cluster\_centers )

**Inverse Transformation- Convert to original Data**

# In order to understand what these numbers mean, perform inverse transformation

cluster\_centers = scaler.inverse\_transform(cluster\_centers)

cluster\_centers = pd.DataFrame(data = cluster\_centers, columns = [creditcard\_df.columns])

display(cluster\_centers)

# First Customers cluster (Transactors): Those are customers who pay least amount of intrerest charges and careful with their money, Cluster with lowest balance ($104) and cash advance ($303), Percentage of full payment = 23%

# Second customers cluster (revolvers) who use credit card as a loan (most lucrative sector): highest balance ($5000) and cash advance (~$5000), low purchase frequency, high cash advance frequency (0.5), high cash advance transactions (16) and low percentage of full payment (3%)

# Third customer cluster (VIP/Prime): high credit limit $16K and highest percentage of full payment, target for increase credit limit and increase spending habits

# Fourth customer cluster (low tenure): these are customers with low tenure (7 years), low balance

**Display Cluster Details**

display(labels.shape) # Labels associated to each data point

display (labels.max())

display (labels.min())

**Display Cluster numbers**

y\_kmeans = kmeans.fit\_predict(creditcard\_df\_scaled)

display(y\_kmeans)

**Concatenate the clusters labels to our original data frame**

creditcard\_df\_cluster = pd.concat([creditcard\_df, pd.DataFrame({'cluster':labels})], axis = 1)

creditcard\_df\_cluster.head()

**Plot the histogram of various clusters**

for i in creditcard\_df.columns:

plt.figure(figsize = (35, 5))

for j in range(8):

plt.subplot(1,8,j+1)

cluster = creditcard\_df\_cluster[creditcard\_df\_cluster['cluster'] == j]

cluster[i].hist(bins = 20)

plt.title('{} \nCluster {} '.format(i,j))

plt.show()

PRINCIPAL COMPONENT ANALYSIS

**PCA with n components = 2**

pca = PCA(n\_components=2)

principal\_comp = pca.fit\_transform(creditcard\_df\_scaled)

display(principal\_comp)

**Create a data frame with the two components**

pca\_df = pd.DataFrame(data = principal\_comp, columns =['pca1','pca2'])

pca\_df.head()

**Concatenate the clusters labels to the data frame**

pca\_df = pd.concat([pca\_df,pd.DataFrame({'cluster':labels})], axis = 1)

display(pca\_df)

**Number of records in each cluster**

pca\_df.value\_counts(pca\_df.cluster)

**Plot the graph with two components**

plt.figure(figsize=(20,10))

ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca\_df, palette =['red','green','blue','pink','yellow','gray','purple', 'black'])

plt.show()

APPLY AUTO ENCODERS

**Create the Layers**

from tensorflow.keras.layers import Input, Add, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D, MaxPooling2D, Dropout

from tensorflow.keras.models import Model, load\_model

from tensorflow.keras.initializers import glorot\_uniform # This is normalizer

from keras.optimizers import SGD

encoding\_dim = 7

input\_df = Input(shape=(17,)) # 17 Features

# Glorot normal initializer (Xavier normal initializer) draws samples from a truncated normal distribution

x = Dense(encoding\_dim, activation='relu')(input\_df)

x = Dense(500, activation='relu', kernel\_initializer = 'glorot\_uniform')(x)

x = Dense(500, activation='relu', kernel\_initializer = 'glorot\_uniform')(x)

x = Dense(2000, activation='relu', kernel\_initializer = 'glorot\_uniform')(x)

encoded = Dense(10, activation='relu', kernel\_initializer = 'glorot\_uniform')(x)

x = Dense(2000, activation='relu', kernel\_initializer = 'glorot\_uniform')(encoded)

x = Dense(500, activation='relu', kernel\_initializer = 'glorot\_uniform')(x)

decoded = Dense(17, kernel\_initializer = 'glorot\_uniform')(x)

# autoencoder

autoencoder = Model(input\_df, decoded)

#encoder - used for our dimention reduction

encoder = Model(input\_df, encoded)

autoencoder.compile(optimizer= 'adam', loss='mean\_squared\_error')

**Display the shape**

display (creditcard\_df\_scaled.shape)

**Create Auto Encoder – Fit model**

autoencoder.fit(creditcard\_df\_scaled, creditcard\_df\_scaled, batch\_size = 128, epochs = 25, verbose = 1)

**Auto Encoder Summary**

autoencoder.summary()

**Weights**

autoencoder.save\_weights('autoencoder.h5')

**Generate autoencoder values.**

#After auto encoder number of columns reduced to 10

pred = encoder.predict(creditcard\_df\_scaled)

display (pd.DataFrame(pred))

**Display Shape**

pred.shape

**Create Elbow graph**

scores\_2 = []

range\_values = range(1, 20)

for i in range\_values:

kmeans = KMeans(n\_clusters= i)

kmeans.fit(pred)

scores\_2.append(kmeans.inertia\_)

plt.plot(scores\_2, 'bx-')

plt.title('Finding right number of clusters')

plt.xlabel('Clusters')

plt.ylabel('scores')

plt.show()

**Display the graph, current and earlier wcss values**

# Scores\_1 – Earlier wcss values

# Scores\_2 – Current wcss values

plt.plot(scores\_1, 'bx-', color = 'r')

plt.plot(scores\_2, 'bx-', color = 'g')

**Create K Means Cluster with optimal number of clusters. In this case it is 4 clusters**

kmeans = KMeans(4)

kmeans.fit(pred)

labels = kmeans.labels\_

y\_kmeans = kmeans.fit\_predict(creditcard\_df\_scaled)

**Display Cluster Details**

display(labels.shape) # Labels associated to each data point

display (labels.max())

display (labels.min())

**Display individual cluster numbers**

y\_kmeans = kmeans.fit\_predict(pred)

display(y\_kmeans)

**Concatenate Cluster number with Original Data Set**

df\_cluster\_dr = pd.concat([creditcard\_df, pd.DataFrame({'cluster':labels})], axis = 1)

display(df\_cluster\_dr.head())

**Apply PCA on auto encoded data set**

pca = PCA(n\_components=2)

prin\_comp = pca.fit\_transform(pred)

pca\_df = pd.DataFrame(data = prin\_comp, columns =['pca1','pca2'])

display (pca\_df.head())

**Concatenate Cluster number with PCA Data Set**

pca\_df = pd.concat([pca\_df,pd.DataFrame({'cluster':labels})], axis = 1)

pca\_df.head()

**Print Clusters and number records in each cluster**

pca\_df.value\_counts(pca\_df.cluster)

**Plot Graph**

plt.figure(figsize=(10,10))

ax = sns.scatterplot(x="pca1", y="pca2", hue = "cluster", data = pca\_df, palette =['red','green','blue','yellow'])

plt.show()