Credit Card Segmentation

Project Report

Problem Statement:

This case requires trainees 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.

But first, **what is customer segmentation?**

Segmentation is an integral part of the development of marketing objectives and strategies, where defining those objectives will generally include either [1,2] :

(a) an analysis of how products should be sold or developed, based on an analysis of current customer segments

(b) the identification of new segments as targets for existing products or the development of new products.

Segmentation is critical because a company has limited resources, and must focus on how to best identify and serve its customers.

Effective segmentation allows a company to determine which customer groups they should try to serve and how to best position their products and services for each group

Introduction:

Machine Learning is a study of computer algorithms that improve automatically through experience and by the use of data. It is a subfield of artificial intelligence. Machine Learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. These ML algorithms are widely used in email spam filtering, computer vision, sales forecasting…. Etc. where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

In addition, machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to field of machine learning. And also data mining is a related field of study, focusing on EDA through unsupervised learning.

Machine Learning approaches are tradionally divided into three broad cateogories, depending on the nature of the signal or feedback available to the learning system. There are,

- **Supervised Learning** is type of train the machine learning algorithms by giving a input features and target in train dataset for to develop a machine learning model, to make predictions.

- **Unsupervised Learning** is a type of train the machine learning algorithms by giving only input features to find the target while in training and deploying the machine learning model.
- **Semi Supervised Learning** is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning and supervised learning. It is a special instance of weak supervision.
- **Reinforcement Learning** is a model free train the machine learning algorithms by giving an actions and space in environment and agent objects for to make right predictions in real time.

Scikit-learn is a python module for machine learning built top os Scipy module and is distributed under the 3-clause BSD License. It is simple and efficient tools for predictive data analysis. It is accessible to everyone, and reusable in various contexts. It was bulit on Numpy, Scipy and Matplotlib.

Scikit-learn module mainly used for the Regression, Classification, Clustering, Dimensionality Reduction, Model Selection and Preprocessing of historical data to predict the futures, and classify the data.

System Requirements:

The requirement of the CPU or laptop for to run the project. The following are important requirement to run this project. They are:

1.Laptop or CPU with above i3 8^{th} gen processor.

2.RAM not less than 8GB

3.Python 3.8.8 64 bit

4.R 4.10 with R Studio

5.Windows 10 64bit

6.Required python Packages are listed in “Requirement.txt” file.

Advanced data preparation:

Build an ‘enriched’ customer profile by deriving “intelligent” KPIs such as:

1 – Monthly average purchase and cash advance amount

2 – Purchase by type (one-off, installments)

3 – Average amount per purchase and cash advance transaction,

4 – Limit usage (balance to credit limit ratio),

5 – Payments to minimum payments ratio etc.

6 – Advanced reporting: Use the derived KPIs to gain insight on the customer profiles.

7 – Identification of the relationships/affinities between service.

8 – Clustering: Apply a data reduction technique factor analysis for variable reduction technique

9 – Identify cluster characteristics of the cluster using detailed profiling.

10 – Provide the strategic insights and implementation of strategies for given set of cluster char.

Data Dictionary:

To understand better about each feature of the data mean's here is the data dictionary.

CUST_ID: Credit card holder ID

BALANCE: Monthly average balance (based on daily balance averages)

BALANCE_FREQUENCY: Ratio of last 12 months with balance

PURCHASES: Total purchase amount spent during last 12 months

ONEOFF_PURCHASES: Total amount of one-off purchases.

INSTALLMENTS_PURCHASES: Total amount of one-off purchases

CASH_ADVANCE: Total cash-advance amount

PURCHASES_FREQUENCY: Frequency of purchases (Percent of months with at least one purchase)

ONEOFF_PURCHASES_FREQUENCY: Frequency of one-off purchases.

PUCHASES_INSTALLMENTS_FREQUENCY: Frequency of installments purchases

CASH_ADVANCE_FREQUENCY: Cash-Advance frequency

AVERAGE_PURCHASE_TRX: Average amount per cash-advance transaction.

CASH_ADVANCE_TRX: Average amount per cash-advance transaction

PURCHASES_TRX: Average amount per purchase transaction

CREDIT_LIMIT: Credit limit

PAYMENTS: Total payments (due amount paid by the customer to decrease their statement balance) in the period

MINIMUM_PAYMNETS: Total minimum payments due in the period

PRC_FULL_PAYMENT: Percentage of months with full payment of the due statement balance

TENURE: Number of months as a customer

Preprocessing Data:

Credit Card Segmentation dataset is downloaded from the link given in the project problem statement. It contains the csv file with name of credit-card-data.csv and it is extracted to the working directory as credit in python and creditdf in R. The project folder is open with jupyter lab for python and with R studio for R coding and create a new file as Credit card segementation.py and credit card segementation.R.

After that, import a pandas module as "pd", by simply typing as "import pandas as pd" in credit card segmentation.py. Then load the dataset in python by using pandas data frame as credit. In R as creditdf with a code as follows as:

```
# reading data into dataframe
credit= pd.read_csv("credit-card-data.csv")
```

```
creditdf <- read.csv("credit-card-data.csv", sep=",", header = TRUE, stringsAsFactors = FALSE)
```

Above code will display the table format of data which is converted from csv file to pandas Dataframe. Now the right time to check the null values in dataframe or not by following syntax

```
"""### Information about data set"""

credit.head()

credit.info()

# Find the total number of missing values in the dataframe
print ("\nMissing values :  ", credit.isnull().sum().values.sum())

# printing total numbers of Unique value in the dataframe.
print ("\nUnique values :  \n",credit.nunique())

credit.shape

# Intital descriptive analysis of data.
credit.describe()
```

A little glance at the data:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083

```
(8950, 18)
```

The data consists of 8950 rows and 18 columns. Here is the summary of the data.

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371
min	0.000000	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	0.083333
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000

```
Missing values :   314

Unique values :
 CUST_ID                             8950
BALANCE                             8871
BALANCE_FREQUENCY                     43
PURCHASES                           6203
ONEOFF_PURCHASES                    4014
INSTALLMENTS_PURCHASES              4452
CASH_ADVANCE                        4323
PURCHASES_FREQUENCY                   47
ONEOFF_PURCHASES_FREQUENCY            47
PURCHASES_INSTALLMENTS_FREQUENCY      47
CASH_ADVANCE_FREQUENCY                54
CASH_ADVANCE_TRX                      65
PURCHASES_TRX                        173
CREDIT_LIMIT                         205
PAYMENTS                            8711
MINIMUM_PAYMENTS                    8636
PRC_FULL_PAYMENT                      47
TENURE                                 7
dtype: int64
```

Missing value treatment

Since there are 314 missing values in the data so we are imputing them with median.

```
83 credit.isnull().any()
```

Before Handling missing values

```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                         True
PAYMENTS                            False
MINIMUM_PAYMENTS                     True
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

From the above output we can say that CREDIT_LIMIT and MINIMUM_PAYMENTS has missing values but I didn’t dropped them. I handled these missing values by replacing them with by median.

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

credit['CREDIT_LIMIT'].count()


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

Again, checking for the missing values

```
credit.isnull().any()
```

After handling missing values

```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                        False
PAYMENTS                            False
MINIMUM_PAYMENTS                    False
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

Deriving New KPIs

1.Monthly average purchase and cash advance amount

```
credit['Monthly_avg_purchase']=credit['PURCHASES']/credit['TENURE']
```

```
credit['Monthly_cash_advance']=credit['CASH_ADVANCE']/credit['TENURE']

credit[credit['ONEOFF_PURCHASES']==0]['ONEOFF_PURCHASES'].count()
```

2.Purchases by type (one-off, installments)

Here we will find type of purchases customers are making on credit card and also customers ONEOFF_PURCHASES and INSTALLMENTS_PURCHASES details.

```
credit.loc[:,['ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES']]
```

	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	0.00	95.40
1	0.00	0.00
2	773.17	0.00
3	1499.00	0.00
4	16.00	0.00
...	...	...
8945	0.00	291.12
8946	0.00	300.00
8947	0.00	144.40
8948	0.00	0.00
8949	1093.25	0.00

8950 rows × 2 columns

```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```

```
(2042, 20)
```

```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

```
(2774, 20)
```

```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```

```
(1874, 20)
```

```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

```
(2260, 20)
```

As per above detail we found out that there are 4 types of purchases behavior in the data. So we need to derive a categorical variable based on their behavior.

```
def purchase(credit):
    if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0):
        return 'none'
    if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0):
         return 'both_oneoff_installment'
    if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0):
        return 'one_off'
    if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0):
        return 'istallment'

credit['purchase_type']=credit.apply(purchase,axis=1)

credit['purchase_type'].value_counts()
```

```
both_oneoff_installment    2774
istallment                 2260
none                       2042
one_off                    1874
Name: purchase_type, dtype: int64
```

4. Limit_usage (balance to credit limit ratio) credit card utilization

Lower value implies customers are maintain their balance properly. Lower value means good credit score.

```
credit['limit_usage']=credit.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT'], axis=1)

credit['limit_usage'].head()
```

```
0    0.040901
1    0.457495
2    0.332687
3    0.222223
4    0.681429
Name: limit_usage, dtype: float64
```

5. Payments to minimum payments ratio etc.

```
credit['PAYMENTS'].isnull().any()
credit['MINIMUM_PAYMENTS'].isnull().value_counts()
```

```
False    8950
Name: MINIMUM_PAYMENTS, dtype: int64
```

```
credit['MINIMUM_PAYMENTS'].describe()
```

```
count     8950.000000
mean       844.906767
std       2332.792322
min          0.019163
25%        170.857654
50%        312.343947
75%        788.713501
max      76406.207520
Name: MINIMUM_PAYMENTS, dtype: float64
```

```
credit['payment_minpay']=credit.apply(lambda x:x['PAYMENTS']/x['MINIMUM_PAYMENTS'],axis=1)
```

Exploratory Data Analysis:

Exploratory data analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

The most important code line in python is that credit.info(), credit.describe(), credit.shape() are intital steps to perform the Exploratory data analysis.

Next we are moving to check the extreme values in the data.

Since there are variables having extreme values so I am doing log-transformation on the data to remove the outlier effect.

Log Transformation

Logarithmic transformation is a convenient means of transforming a highly skewed variable into a more normalized dataset. When modeling variables with non-linear relationships, the chances of producing errors may also be skewed negatively.

Log transformation is a data transformation method in which it replaces each variable x with a log(x).

```
cr_log=credit.drop(['CUST_ID','purchase_type'],axis=1).applymap(lambda x: np.log(x+1))

cr_log.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000

```
cr_pre=cr_log[[x for x in cr_log.columns if x not in col ]]
```

```
cr_pre.columns
```

```
Index(['BALANCE_FREQUENCY', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'Monthly_avg_purchase',
       'Monthly_cash_advance', 'limit_usage', 'payment_minpay'],
      dtype='object')
```

```
cr_log.columns
```

```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
       'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT',
       'TENURE', 'Monthly_avg_purchase', 'Monthly_cash_advance', 'limit_usage',
       'payment_minpay'],
      dtype='object')
```

Insights from KPIs

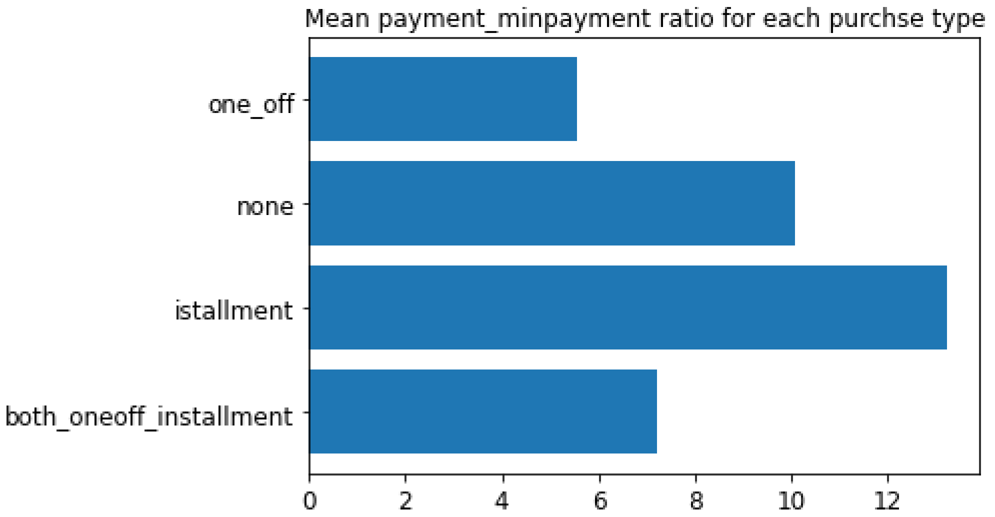
Average payment_minpayment ratio for each purchase type.

```
x=credit.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpay']))
type(x)
x.values
```

```
array([ 7.23698216, 13.2590037 , 10.08745106,  5.57108156])
```

```
ax.barh?

fig,ax=plt.subplots()
ax.barh(y=range(len(x)), width=x.values,align='center')
ax.set(yticks= np.arange(len(x)),yticklabels = x.index);
plt.title('Mean payment_minpayment ratio for each purchse type')
```

```
Text(0.5, 1.0, 'Mean payment_minpayment ratio for each purchse type')
```



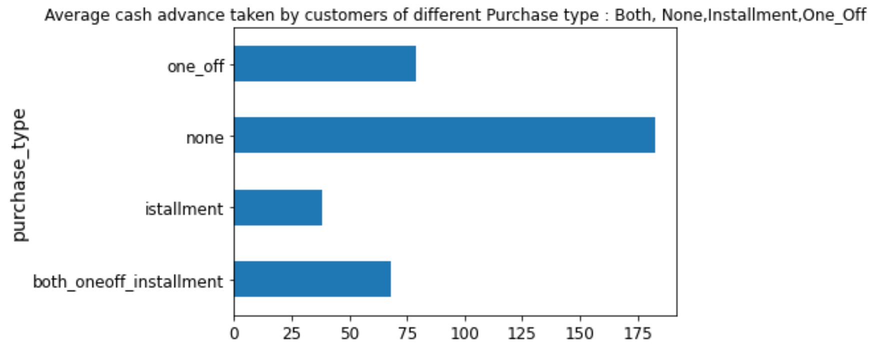

```
credit.describe()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371
min	0.000000	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	0.083333
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000

Customers with installment purchases are paying dues

```
credit[credit['purchase_type']=='n']

credit.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_cash_advance'])).plot.barh()

plt.title('Average cash advance taken by customers of different Purchase type : Both, None,Installment,One_Off')
```

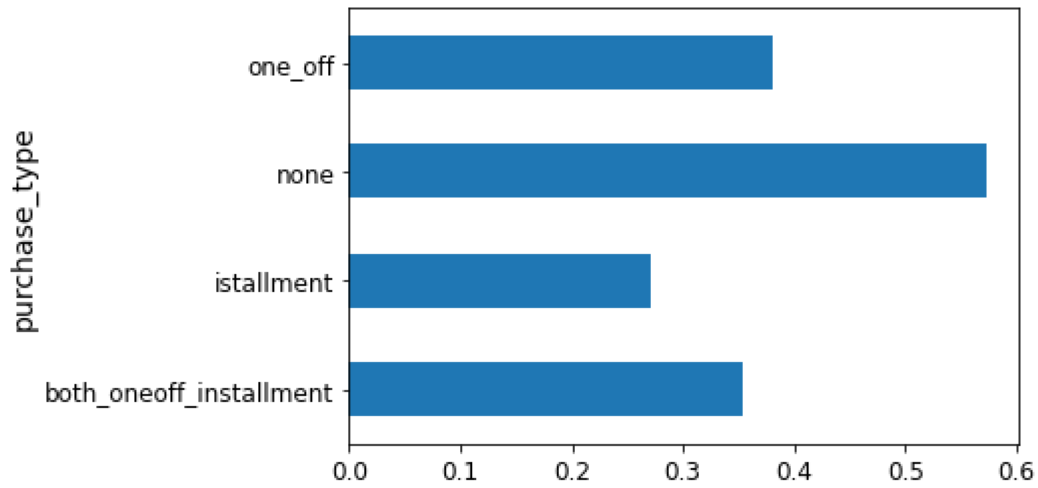
```
Text(0.5, 1.0, 'Average cash advance taken by customers of different Purchase type : Both, None,Installment,One_Off')
```



Customers who don't do either one-off or installment purchases take cash on advance

```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage'])).plot.barh()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x247931df9d0>
```



Original dataset with categorical column converted to number type

```
cre_original=pd.concat([credit,pd.get_dummies(credit['purchase_type'])],axis=1)
```

We do have some categorical data which need to convert with the help of dummy creation

```
# creating Dummies for categorical variable
cr_pre['purchase_type']=credit.loc[:,'purchase_type']
pd.get_dummies(cr_pre['purchase_type'])
```

	both_oneoff_installment	istallment	none	one_off
0	0	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	1
4	0	0	0	1
...	...	...	...	...

Now merge the created dummy with the original data frame

```
cr_dummy=pd.concat([cr_pre,pd.get_dummies(cr_pre['purchase_type'])],axis=1)

l=['purchase_type']

cr_dummy=cr_dummy.drop(l,axis=1)
cr_dummy.isnull().any()
```

```
BALANCE_FREQUENCY                   False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
Monthly_avg_purchase                False
Monthly_cash_advance                False
limit_usage                         False
payment_minpay                      False
both_oneoff_installment             False
istallment                          False
none                                False
one_off                             False
dtype: bool
```

234
```
cr_dummy.info()
```

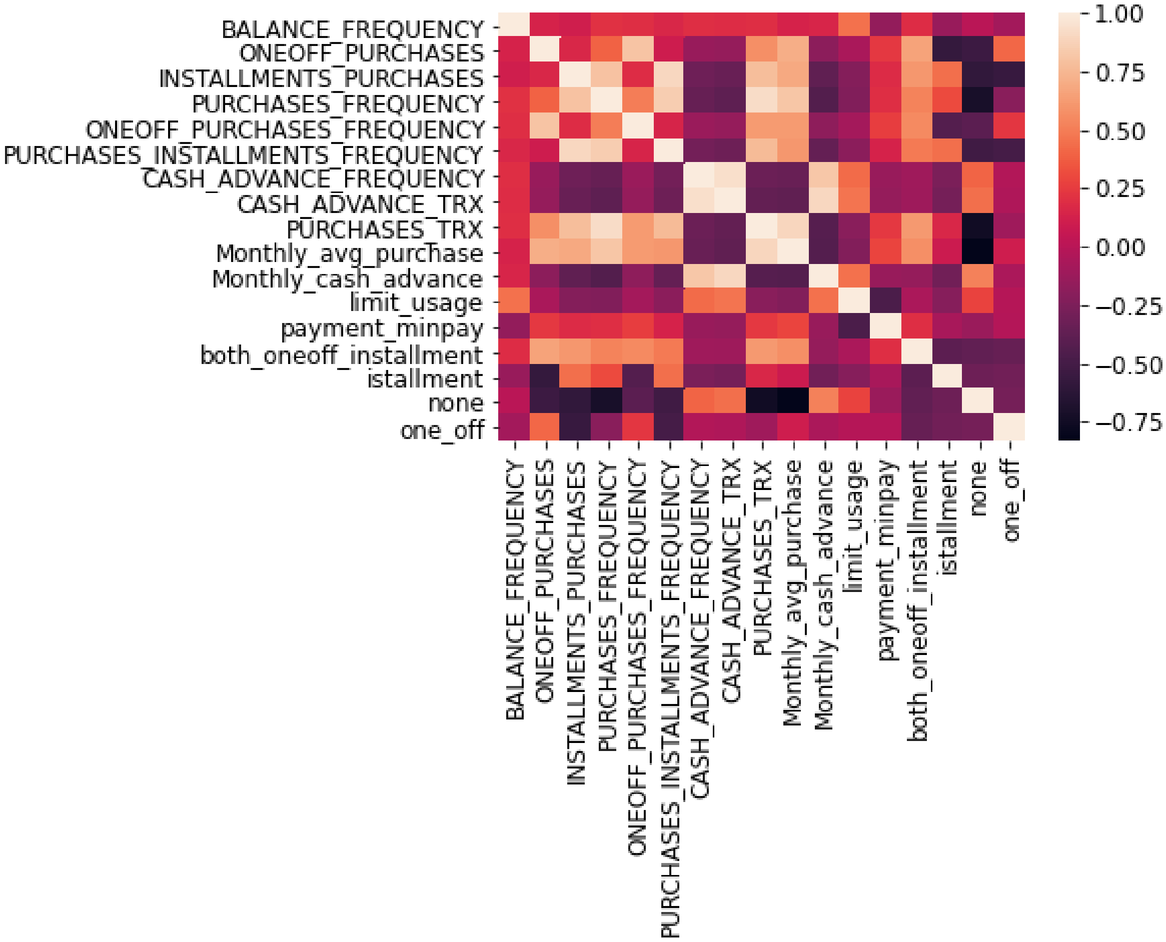
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 17 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   BALANCE_FREQUENCY                 8950 non-null   float64
 1   ONEOFF_PURCHASES                  8950 non-null   float64
 2   INSTALLMENTS_PURCHASES            8950 non-null   float64
 3   PURCHASES_FREQUENCY               8950 non-null   float64
 4   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 5   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 6   CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 7   CASH_ADVANCE_TRX                  8950 non-null   float64
 8   PURCHASES_TRX                     8950 non-null   float64
 9   Monthly_avg_purchase              8950 non-null   float64
 10  Monthly_cash_advance              8950 non-null   float64
 11  limit_usage                       8950 non-null   float64
 12  payment_minpay                    8950 non-null   float64
 13  both_oneoff_installment           8950 non-null   uint8  
 14  istallment                        8950 non-null   uint8  
 15  none                              8950 non-null   uint8  
 16  one_off                           8950 non-null   uint8  
dtypes: float64(13), uint8(4)
memory usage: 944.1 KB
```

236
```
cr_dummy.head(3)
```

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_I
0	0.597837	0.000000	4.568506	0.154151	0.000000	
1	0.646627	0.000000	0.000000	0.000000	0.000000	
2	0.693147	6.651791	0.000000	0.693147	0.693147	

238
```
sns.heatmap(cr_dummy.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x247935f2070>
```



Heat map shows that many features are co-related so applying dimensionality reduction will help negative multi-collinearity in data.

Before applying PCA we will standardize data to avoid effect of scale on our result. Centering and scaling will make all features with equal weight.

Standardization

Standardization is a technique often applied as part of data preparation for machine learning. The goal of standardization is to change the values. The values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

$$X' = \frac{X - \mu}{\sigma}$$

```
from sklearn.preprocessing import  StandardScaler

sc=StandardScaler()

cr_dummy.shape
```

```
(8950, 17)
```

```
cr_scaled=sc.fit_transform(cr_dummy)
cr_scaled
```

```
array([[-0.14875746, -0.98708958,  0.39447984, ...,  1.72051649,
        -0.54369045, -0.514625  ],
       [ 0.17961568, -0.98708958, -1.08745376, ..., -0.58122082,
         1.83928189, -0.514625  ],
       [ 0.49271003,  1.06202168, -1.08745376, ..., -0.58122082,
        -0.54369045,  1.94316249],
       ...,
       [-0.09290575, -0.98708958,  0.52779444, ...,  1.72051649,
        -0.54369045, -0.514625  ],
       [-0.09290575, -0.98708958, -1.08745376, ..., -0.58122082,
         1.83928189, -0.514625  ],
       [-0.73437135,  1.16861854, -1.08745376, ..., -0.58122082,
        -0.54369045,  1.94316249]])
```

Dimension Reduction using PCA

I applied prinicipal component analysis(PCA) to transform into 5 dimensions for visualization because we wont be able to visualize the data in 17 dimensions. PCA transforms a large set of variables into a smaller one that still contains most of the information in the large set. Reducing the number of variables of data.

With the help of prinicipal component analysis we will reduce features

```
from sklearn.decomposition import PCA

cr_dummy.shape
```

```
(8950, 17)
```

We have 17 features so our n_component will be 17

```
pc=PCA(n_components=17)
cr_pca=pc.fit(cr_scaled)
```

Lets check if we take 17 component then how much variance it explain. Ideally it should be i.e., 100%

```
sum(cr_pca.explained_variance_ratio_)
```

```
0.9999999999999999
```

```
var_ratio={}
for n in range(2,18):
    pc=PCA(n_components=n)
    cr_pca=pc.fit(cr_scaled)
    var_ratio[n]=sum(cr_pca.explained_variance_ratio_)

var_ratio
```

```
{2: 0.5826439793960276,
 3: 0.7299379309512696,
 4: 0.811544276235126,
 5: 0.8770555795291434,
 6: 0.9186492443512615,
 7: 0.9410925256030127,
 8: 0.9616114053683064,
 9: 0.9739787081990643,
 10: 0.9835896584630706,
 11: 0.9897248107341959,
 12: 0.9927550009135229,
 13: 0.995390756238542,
 14: 0.9979616898169593,
 15: 0.9996360473172954,
 16: 0.9999999999999999,
 17: 0.9999999999999999}
```

Since 6 components are explaining about 90% variance so we select 5 components

```
pc=PCA(n_components=6)

p=pc.fit(cr_scaled)

cr_scaled.shape
```

```
(8950, 17)
```

```
p.explained_variance_
```

```
array([6.83574755, 3.07030693, 2.50427698, 1.38746289, 1.1138166 ,
       0.70717132])
```

```
np.sum(p.explained_variance_)
```

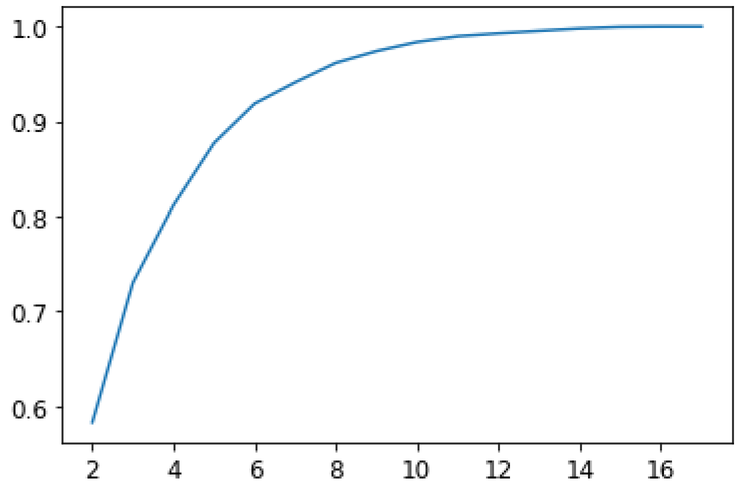
```
15.618782269308802
```

```
var_ratio
```

```
{2: 0.5826439793960276,
 3: 0.7299379309512696,
 4: 0.811544276235126,
 5: 0.8770555795291434,
 6: 0.9186492443512615,
 7: 0.9410925256030127,
 8: 0.9616114053683064,
 9: 0.9739787081990643,
 10: 0.9835896584630706,
 11: 0.9897248107341959,
 12: 0.9927550009135229,
 13: 0.995390756238542,
 14: 0.9979616898169593,
 15: 0.9996360473172954,
 16: 0.9999999999999999,
 17: 0.9999999999999999}
```

```
pd.Series(var_ratio).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x24794a0adc0>
```



Since 5 components are explaining about 87% variance so we select 5 components

```
cr_scaled.shape
```

```
(8950, 17)
```

```
pc_final=PCA(n_components=6).fit(cr_scaled)

reduced_cr=pc_final.fit_transform(cr_scaled)

dd=pd.DataFrame(reduced_cr)

dd.head()
```

	0	1	2	3	4	5
0	-0.242841	-2.759668	0.343061	-0.417359	-0.007100	0.019755
1	-3.975652	0.144625	-0.542989	1.023832	-0.428929	-0.572463
2	1.287396	1.508938	2.709966	-1.892252	0.010809	-0.599932
3	-1.047613	0.673103	2.501794	-1.306784	0.761348	1.408986
4	-1.451586	-0.176336	2.286074	-1.624896	-0.561969	-0.675214

So initially we had 17 variables now its 5 so our variable go reduced

```
dd.shape
```

```
(8950, 6)
```

```
col_list=cr_dummy.columns

col_list
```

```
Index(['BALANCE_FREQUENCY', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'Monthly_avg_purchase',
       'Monthly_cash_advance', 'limit_usage', 'payment_minpay',
       'both_oneoff_installment', 'istallment', 'none', 'one_off'],
      dtype='object')
```

```
pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in range(6)],index=col_list)
```

	PC_0	PC_1	PC_2	PC_3	PC_4	PC_5
BALANCE_FREQUENCY	0.029707	0.240072	-0.263140	-0.353549	-0.228681	-0.693816
ONEOFF_PURCHASES	0.214107	0.406078	0.239165	0.001520	-0.023197	0.129094
INSTALLMENTS_PURCHASES	0.312051	-0.098404	-0.315625	0.087983	-0.002181	0.115223
PURCHASES_FREQUENCY	0.345823	0.015813	-0.162843	-0.074617	0.115948	-0.081879
ONEOFF_PURCHASES_FREQUENCY	0.214702	0.362208	0.163222	0.036303	-0.051279	-0.097299
PURCHASES_INSTALLMENTS_FREQUENCY	0.295451	-0.112002	-0.330029	0.023502	0.025871	0.006731
CASH_ADVANCE_FREQUENCY	-0.214336	0.286074	-0.278586	0.096353	0.360132	0.066589
CASH_ADVANCE_TRX	-0.229393	0.291556	-0.285089	0.103484	0.332753	0.082307
PURCHASES_TRX	0.355503	0.106625	-0.102743	-0.054296	0.104971	-0.009402
Monthly_avg_purchase	0.345992	0.141635	0.023986	-0.079373	0.194147	0.015878
Monthly_cash_advance	-0.243861	0.264318	-0.257427	0.135292	0.268026	0.058258
limit_usage	-0.146302	0.235710	-0.251278	-0.431682	-0.181885	0.024298
payment_minpay	0.119632	0.021328	0.136357	0.591561	0.215446	-0.572467
both_oneoff_installment	0.241392	0.273676	-0.131935	0.254710	-0.340849	0.294708
istallment	0.082209	-0.443375	-0.208683	-0.190829	0.353821	-0.086087
none	-0.310283	-0.005214	-0.096911	0.245104	-0.342222	-0.176809
one_off	-0.042138	0.167737	0.472749	-0.338549	0.362585	-0.060698

So above data gave us eigen vector for each component we had all eigen vector value very small we can remove those variable but in our case its not.

Factor Analysis

```
328 pd.Series(pc_final.explained_variance_ratio_,index=['PC_'+ str(i) for i in range(6)])
```

```
PC_0    0.402058
PC_1    0.180586
PC_2    0.147294
PC_3    0.081606
PC_4    0.065511
PC_5    0.041594
dtype: float64
```

Machine learning Algorithm

Clustering is one of the most common exploratory data analysis techniques used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different.

A cluster refers to a collection of data points aggregated together because of certain similarities.

Here I used the K-means algorithm. K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. K-means algorithm is an iterative algorithm that tries to partition the dataset into *K* pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**.

K-means algorithm identifies *k* number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

The ***'means'*** in the K-means refers to averaging of the data; that is, finding the centroid.

It halts creating and optimizing clusters when either:

1.The centroids have stabilized — there is no change in their values because the clustering has been successful**.**

2.The defined number of iterations has been achieved.

Based on the intuition on type of purchase made by customers and their distinctive behavior exhibited based on the purchase_type (as visualized above in Insights from KPI), I am starting with 4 clusters

```
from sklearn.cluster import KMeans

km_4=KMeans(n_clusters=4,random_state=123)

km_4.fit(reduced_cr)
```

```
KMeans(n_clusters=4, random_state=123)
```

```
km_4.labels_
```

```
array([0, 1, 3, ..., 0, 1, 3])
```

```
pd.Series(km_4.labels_).value_counts()
```

```
2    2769
0    2224
1    2088
3    1869
dtype: int64
```

Here we do not know k value so we will find the k. To do that we need to take a cluster range between 1 and 21

Identify cluster error

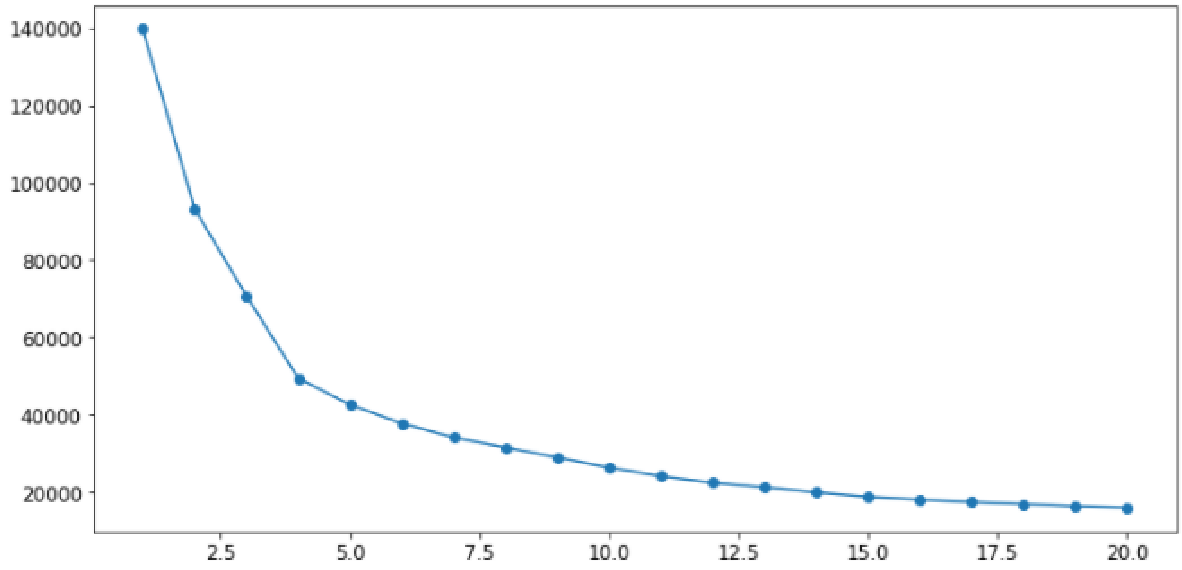
```
for num_clusters in cluster_range:
    clusters = KMeans( num_clusters )
    clusters.fit( reduced_cr )
    cluster_errors.append( clusters.inertia_ )# clusters.inertia_ is basically cluster error here.

clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors": cluster_errors } )

clusters_df[0:21]
```

	num_clusters	cluster_errors
0	1	139772.482528
1	2	93307.383469
2	3	70745.193400
3	4	49446.066485
4	5	42548.525149
5	6	37713.064124
6	7	34124.614172
7	8	31507.199184
8	9	28866.676186
9	10	26302.872695
10	11	24020.100006
11	12	22364.040541
12	13	21212.660251
13	14	19857.818179
14	15	18729.322403
15	16	18043.674561
16	17	17398.192095
17	18	16878.413100
18	19	16308.954087
19	20	15863.716399

```
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

To do this, we must first specify the number of clusters *K*. Here I used the elbow method to specify the best *K*. Elbow is a very simple method that gives us plots like elbow shape. And we can easily guess the optimal number of *K* from the plot.

```
[<matplotlib.lines.Line2D at 0x247962a7bb0>]
```



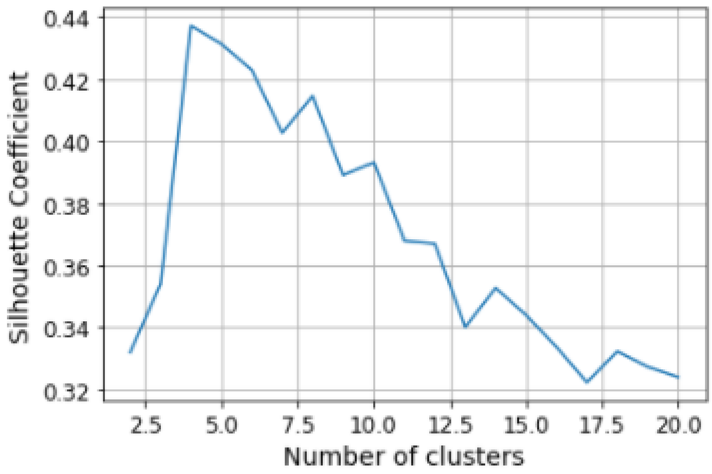
From the above graph we will find elbow range, here it is 4,5,6

Silhoutte Coefficient

```python
k_range = range(2, 21)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(reduced_cr)
    scores.append(metrics.silhouette_score(reduced_cr, km.labels_))

scores
```

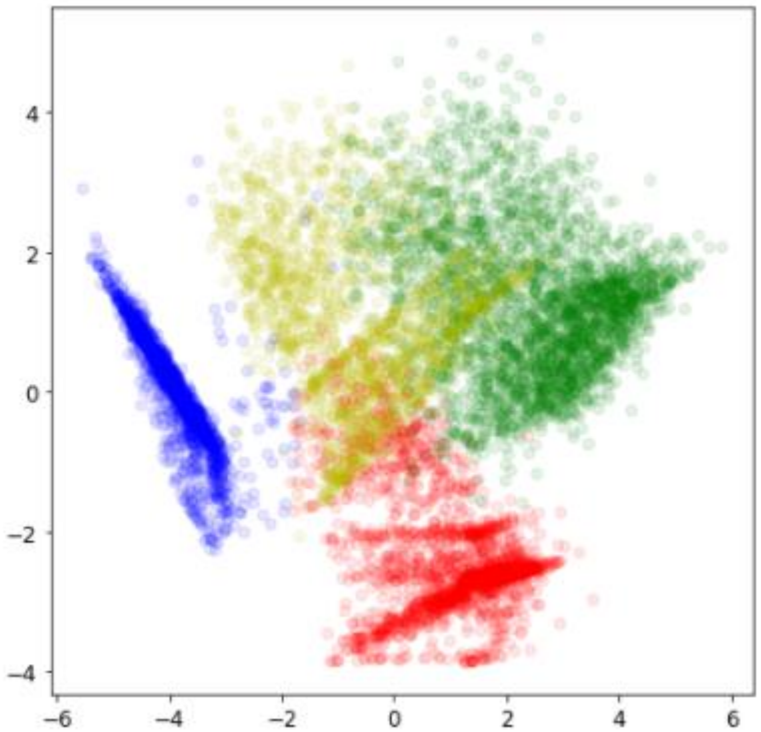
```
[0.3319452179234266,
 0.35401650944838775,
 0.43708577439659474,
 0.4312114520971776,
 0.4228144914653745,
 0.402584287627597,
 0.41445372986226175,
 0.3889287962123451,
 0.3929913554746213,
 0.36787983742685676,
 0.3669766371659528,
 0.34010431910388955,
 0.35271636570412507,
 0.343964162296631,
 0.3336956707851422,
 0.3222956561386571,
 0.3322743918699832,
 0.3272950756025998,
 0.32397923468919276]
```

```python
plt.plot(k_range, scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.grid(True)
```




```python
color_map={0:'r',1:'b',2:'g',3:'y'}
label_color=[color_map[l] for l in km_4.labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=label_color,cmap='Spectral',alpha=0.1)
```

```
<matplotlib.collections.PathCollection at 0x24796366ac0>
```

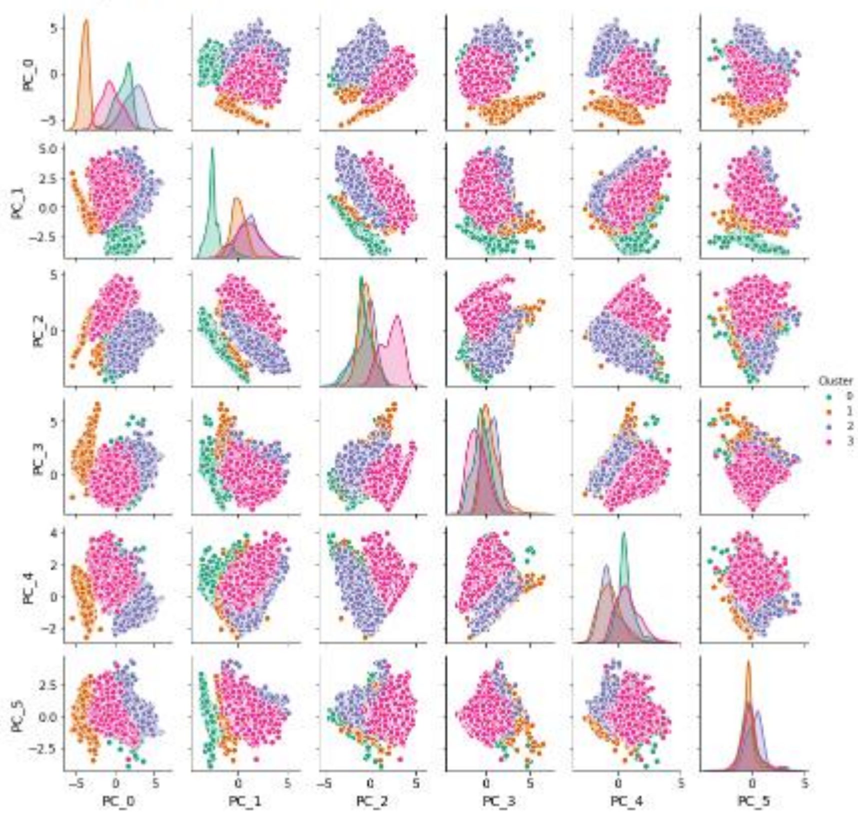



```python
df_pair_plot=pd.DataFrame(reduced_cr,columns=['PC_' +str(i) for i in range(6)])

df_pair_plot['Cluster']=km_4.labels_ #Add cluster column in the data frame

df_pair_plot.head()
```

	PC_0	PC_1	PC_2	PC_3	PC_4	PC_5	Cluster
0	-0.242841	-2.759668	0.343061	-0.417359	-0.007100	0.019755	0
1	-3.975652	0.144625	-0.542989	1.023832	-0.428929	-0.572463	1
2	1.287396	1.508938	2.709966	-1.892252	0.010809	-0.599932	3
3	-1.047613	0.673103	2.501794	-1.306784	0.761348	1.408986	3
4	-1.451586	-0.176336	2.286074	-1.624896	-0.561969	-0.675214	3

```
sns.pairplot(df_pair_plot,hue='Cluster', palette= 'Dark2', diag_kind='kde',size=1.85)
```

<seaborn.axisgrid.PairGrid at 0x24796396370>



Key performance variable selection, here I am taking variables which we will use in deriving a new KPI. We can take all 17 variables but it will be difficult to interpret. So we are selecting less no of variable

```
col_kpi=['PURCHASES_TRX','Monthly_avg_purchase','Monthly_cash_advance','limit_usage','CASH_ADVANCE_TRX',
         'payment_minpay','both_oneoff_installment','istallment','one_off','none','CREDIT_LIMIT']

cr_pre.describe()
```

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASI
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
mean	0.619940	3.204274	3.352403	0.361268	0.158699	
std	0.148590	3.246365	3.082973	0.277317	0.216672	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.635989	0.000000	0.000000	0.080042	0.000000	
50%	0.693147	3.663562	4.499810	0.405465	0.080042	
75%	0.693147	6.360274	6.151961	0.650588	0.262364	
max	0.693147	10.615512	10.021315	0.693147	0.693147	

Concatenating labels found through Kmeans with data

```
cluster_df_4=pd.concat([cre_original[col_kpi],pd.Series(km_4.labels_,name='Cluster_4')],axis=1)

cluster_df_4.head()
```

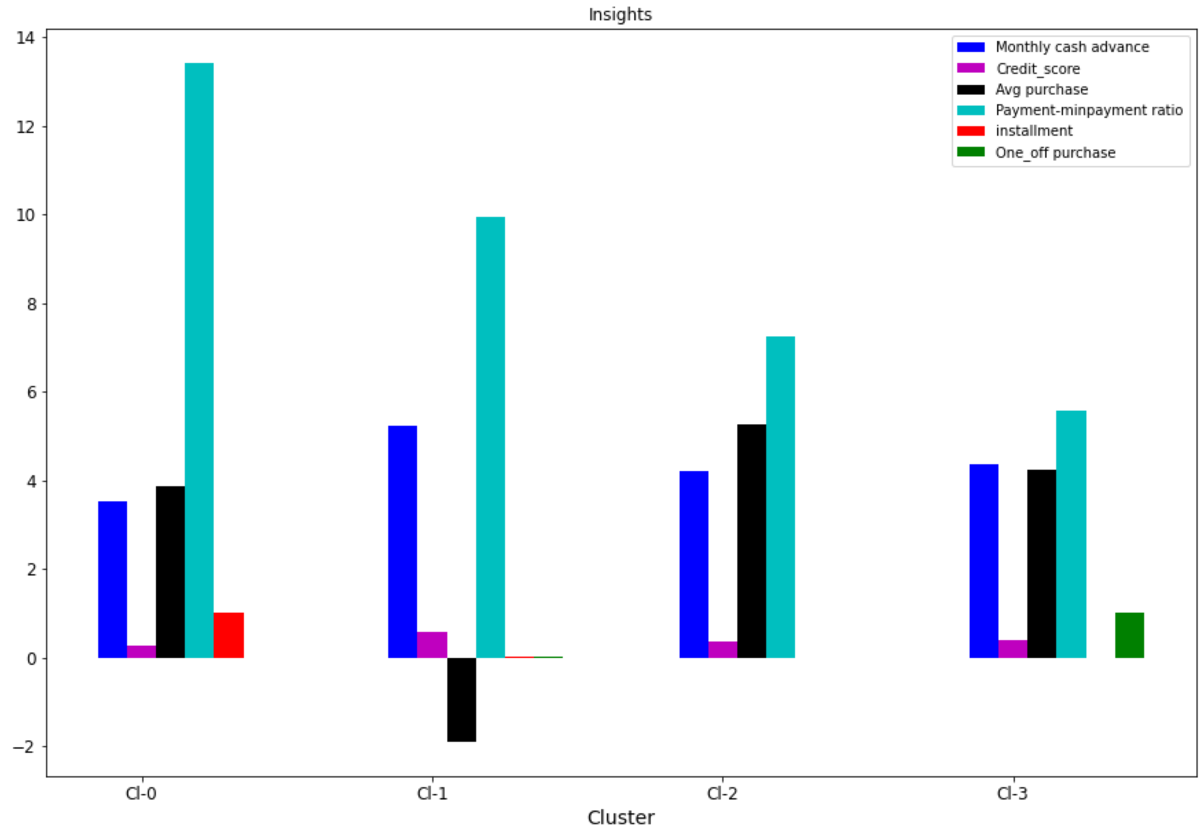
	PURCHASES_TRX	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	CASH_ADVANCE_TRX	payment_minpay	both_oneoff_installment	istallment	o
0	2	7.950000	0.000000	0.040901	0	1.446508	0	1	
1	0	0.000000	536.912124	0.457495	4	3.826241	0	0	
2	12	64.430833	0.000000	0.332687	0	0.991682	0	0	
3	1	124.916667	17.149001	0.222223	1	0.000000	0	0	
4	1	1.333333	0.000000	0.681429	0	2.771075	0	0	

Mean value gives a good indication of the distribution of data. So we are finding mean value for each variable for each cluster

```
cluster_4=cluster_df_4.groupby('Cluster_4').apply(lambda x: x[col_kpi].mean()).T
cluster_4
```

Cluster_4	0	1	2	3
PURCHASES_TRX	12.062050	0.043582	33.013723	7.127341
Monthly_avg_purchase	47.626256	0.148297	193.008043	69.875917
Monthly_cash_advance	33.550080	186.281319	67.466910	78.098613
limit_usage	0.264745	0.576076	0.353591	0.379761
CASH_ADVANCE_TRX	1.021133	6.540230	2.804261	2.881220
payment_minpay	13.422420	9.936617	7.245651	5.573672
both_oneoff_installment	0.000000	0.001916	1.000000	0.000535
istallment	1.000000	0.017241	0.000000	0.000000
one_off	0.000000	0.002874	0.000000	0.999465
none	0.000000	0.977969	0.000000	0.000000
CREDIT_LIMIT	3338.270406	4055.156450	5736.732730	4519.708481

```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(cluster_4.columns))

cash_advance=np.log(cluster_4.loc['Monthly_cash_advance',:].values)
credit_score=(cluster_4.loc['limit_usage',:].values)
purchase= np.log(cluster_4.loc['Monthly_avg_purchase',:].values)
payment=cluster_4.loc['payment_minpay',:].values
installment=cluster_4.loc['istallment',:].values
one_off=cluster_4.loc['one_off',:].values


bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash advance',width=bar_width)
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_score',width=bar_width)
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purchase',width=bar_width)
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-minpayment ratio',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=bar_width)
b6=plt.bar(index+5*bar_width,one_off,color='g',label='One_off purchase',width=bar_width)

plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0', 'Cl-1', 'Cl-2', 'Cl-3'))
plt.legend()
```



Insights

Clusters are clearly distinguishing behavior within customers

Cluster 2 is the group of customers who have highest Monthly_avg purchases and doing both installment as well as one_off purchases, have comparatively good credit score.

This group is about 31% of the total customer base.

Cluster 1 is taking maximum advance_cash and is paying comparatively less minimum payment and poor credit_score & doing no purchase transaction.

This group is about 23% of the total customer base

Cluster 0 customers are doing maximum One_off transactions and least payment ratio

This group is about 21% of the total customer base

Cluster 3 customers have maximum credit score and are apying dues and are doing maximum installment purchases.

This group is about 25% of the total customer base

```
s=cluster_df_4.groupby('Cluster_4').apply(lambda x: x['Cluster_4'].value_counts())
print (s),'\n'

per=pd.Series((s.values.astype('float')/ cluster_df_4.shape[0])*100,name='Percentage')
print ("Cluster -4 "),'\n'
print (pd.concat([pd.Series(s.values,name='Size'),per],axis=1))
```

```
Cluster_4
0          0    2224
1          1    2088
2          2    2769
3          3    1869
Name: Cluster_4, dtype: int64
Cluster -4
   Size  Percentage
0  2224   24.849162
1  2088   23.329609
2  2769   30.938547
3  1869   20.882682
```

Finding behaviour with 5 clusters

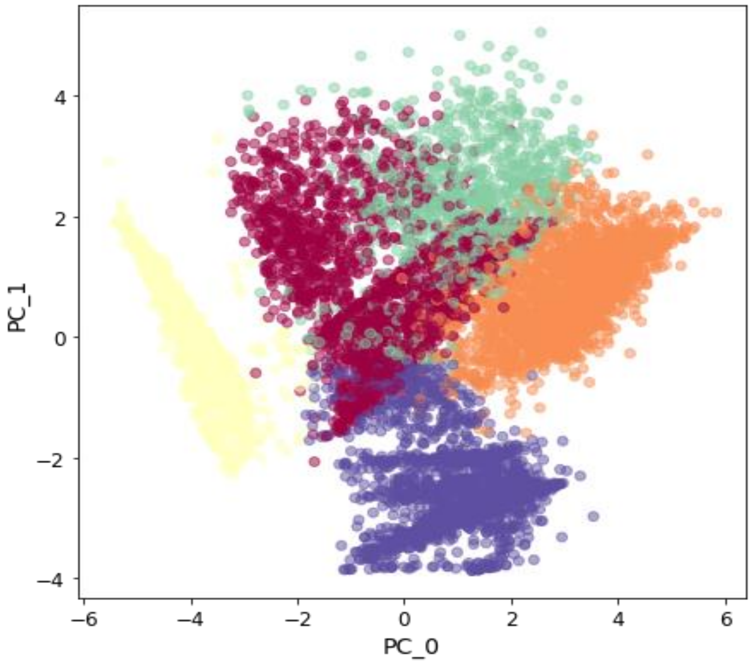
```
km_5=KMeans(n_clusters=5,random_state=123)
km_5=km_5.fit(reduced_cr)
km_5.labels_
```

```
array([4, 2, 0, ..., 4, 2, 0])
```

```
pd.Series(km_5.labels_).value_counts()
```

```
4    2149
2    2081
1    1977
0    1862
3     881
dtype: int64
```

```
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=km_5.labels_,cmap='Spectral',alpha=0.5)
plt.xlabel('PC_0')
plt.ylabel('PC_1')
```



```
cluster_df_5=pd.concat([cre_original[col_kpi],pd.Series(km_5.labels_,name='Cluster_5')],axis=1)
```

```
cluster_df_5.groupby('Cluster_5').apply(lambda x: x[col_kpi].mean()).T
```

Cluster_5	0	1	2	3	4
PURCHASES_TRX	7.096670	34.587759	0.032196	27.703746	11.905537
Monthly_avg_purchase	68.917645	210.536468	0.086126	141.584086	47.369817
Monthly_cash_advance	74.517541	4.040708	185.038534	249.942101	20.636870
limit_usage	0.377959	0.258931	0.576110	0.600096	0.250011
CASH_ADVANCE_TRX	2.697637	0.152757	6.448823	10.384790	0.550489
payment_minpay	5.562287	8.675499	9.963172	3.651686	13.783426
both_oneoff_installment	0.002148	1.000000	0.000000	0.900114	0.000000
istallment	0.000000	0.000000	0.015858	0.088536	1.000000
one_off	0.997852	0.000000	0.002883	0.011351	0.000000
none	0.000000	0.000000	0.981259	0.000000	0.000000
CREDIT_LIMIT	4497.951209	5722.970627	4046.692295	5873.041998	3228.949923

Conclusion with 5 clusters

We have a group of customers (cluster2) having highest average purchases but there is cluster 4 also having highest cash advance & second highest purchase behaviour but their type of purchases are same.

Cluster 0 and cluster 4 are behaving similar in terms of Credit_limit and have cash transactions is on higher side

So we don’t have quite distinguishable characteristics with 5 clusters

```
s1=cluster_df_5.groupby('Cluster_5').apply(lambda x: x['Cluster_5'].value_counts())
print (s1)
```

```
Cluster_5
0          0    1862
1          1    1977
2          2    2081
3          3     881
4          4    2149
Name: Cluster_5, dtype: int64
```

```
print ("Cluster-5"),'\n'
per_5=pd.Series((s1.values.astype('float')/ cluster_df_5.shape[0])*100,name='Percentage')
print (pd.concat([pd.Series(s1.values,name='Size'),per_5],axis=1))
```

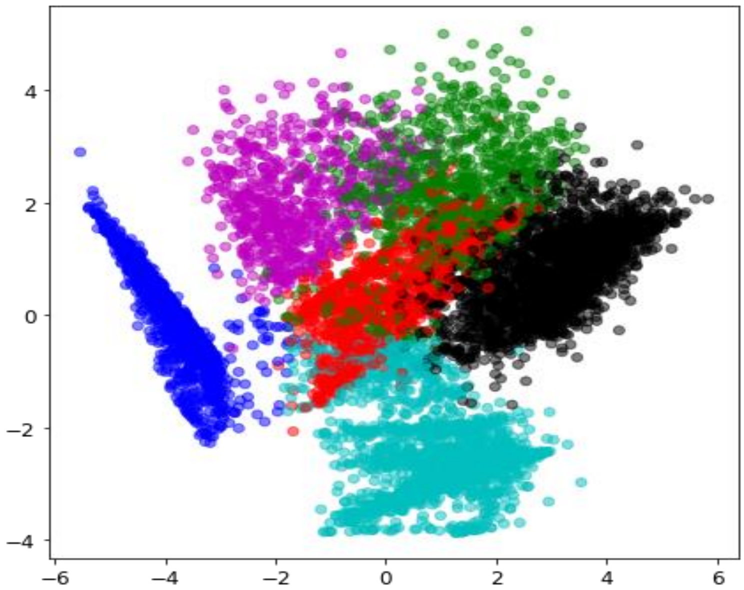
```
Cluster-5
   Size  Percentage
0  1862   20.804469
1  1977   22.089385
2  2081   23.251397
3   881    9.843575
4  2149   24.011173
```

Finding behaviour with 6 clusters

```
km_6=KMeans(n_clusters=6).fit(reduced_cr)
km_6.labels_
```

```
array([3, 1, 0, ..., 3, 1, 4])
```

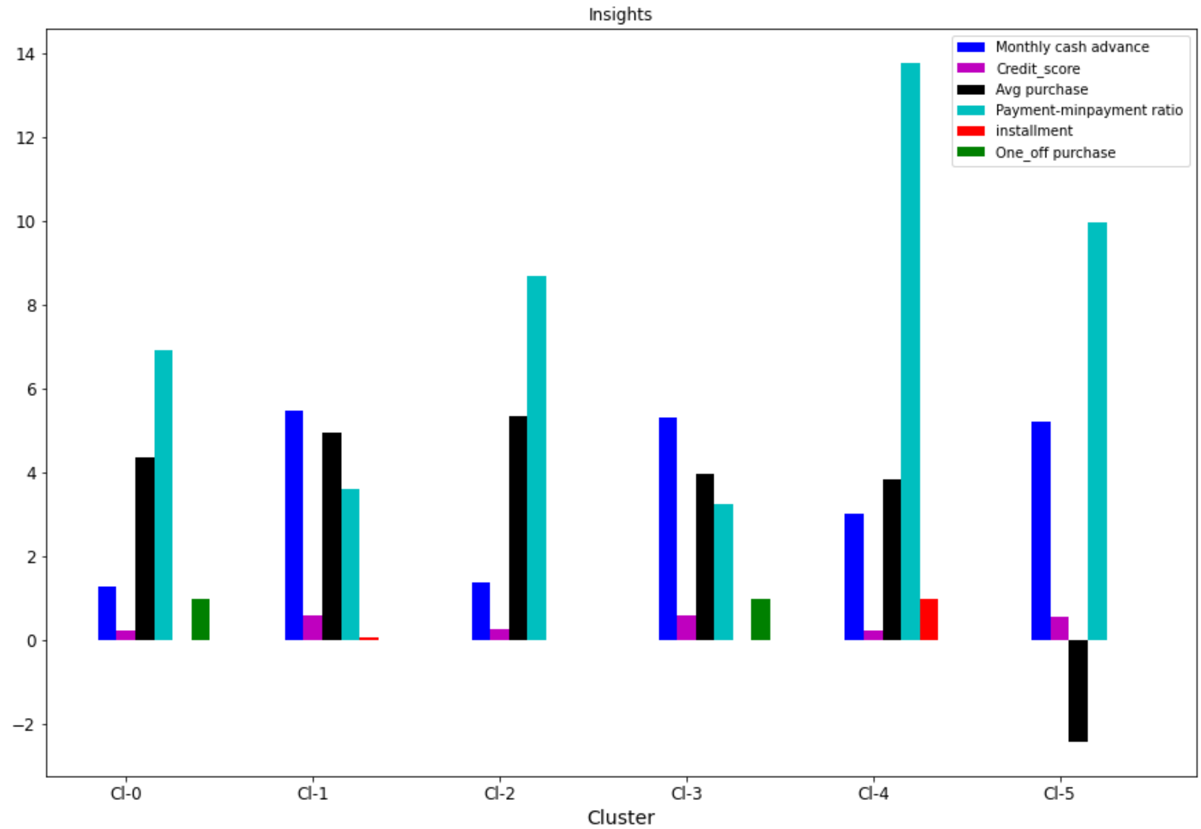
```
color_map={0:'r',1:'b',2:'g',3:'c',4:'m',5:'k'}
label_color=[color_map[l] for l in km_6.labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=label_color,cmap='Spectral',alpha=0.5)
```




```
cluster_df_6 = pd.concat([cre_original[col_kpi],pd.Series(km_6.labels_,name='Cluster_6')],axis=1)

six_cluster=cluster_df_6.groupby('Cluster_6').apply(lambda x: x[col_kpi].mean()).T
six_cluster
```

Cluster_6	0	1	2	3	4	5
PURCHASES_TRX	7.760575	0.030347	27.919908	11.905537	5.967143	34.663789
Monthly_avg_purchase	78.585295	0.088891	140.374727	47.369817	54.091602	211.196582
Monthly_cash_advance	3.603272	184.829434	242.856971	20.636870	205.502536	4.027720
limit_usage	0.245772	0.575724	0.600654	0.250011	0.605930	0.258206
CASH_ADVANCE_TRX	0.125212	6.434971	10.000000	0.550489	7.642857	0.150838
payment_minpay	6.911822	9.976487	3.616973	13.783426	3.257979	8.702974
both_oneoff_installment	0.006768	0.000000	0.911899	0.000000	0.000000	1.000000
istallment	0.000000	0.016378	0.088101	1.000000	0.000000	0.000000
one_off	0.993232	0.000000	0.000000	0.000000	1.000000	0.000000
none	0.000000	0.983622	0.000000	0.000000	0.000000	0.000000
CREDIT_LIMIT	4471.701020	4047.527296	5834.610984	3228.949923	4577.649351	5735.293514

```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(six_cluster.columns))

cash_advance=np.log(six_cluster.loc['Monthly_cash_advance',:].values)
credit_score=(six_cluster.loc['limit_usage',:].values)
purchase= np.log(six_cluster.loc['Monthly_avg_purchase',:].values)
payment=six_cluster.loc['payment_minpay',:].values
installment=six_cluster.loc['istallment',:].values
one_off=six_cluster.loc['one_off',:].values
bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash advance',width=bar_width)
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_score',width=bar_width)
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purchase',width=bar_width)
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-minpayment ratio',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=bar_width)
b6=plt.bar(index+5*bar_width,one_off,color='g',label='One_off purchase',width=bar_width)

plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0', 'Cl-1', 'Cl-2', 'Cl-3','Cl-4','Cl-5'))

plt.legend()
```




```python
cash_advance=np.log(six_cluster.loc['Monthly_cash_advance',:].values)
credit_score=list(six_cluster.loc['limit_usage',:].values)
cash_advance
```

```
array([1.28184245, 5.49247267, 1.39320045, 5.32545837, 3.02707927,
       5.21943342])
```

Conclusion with 6 Clusters:

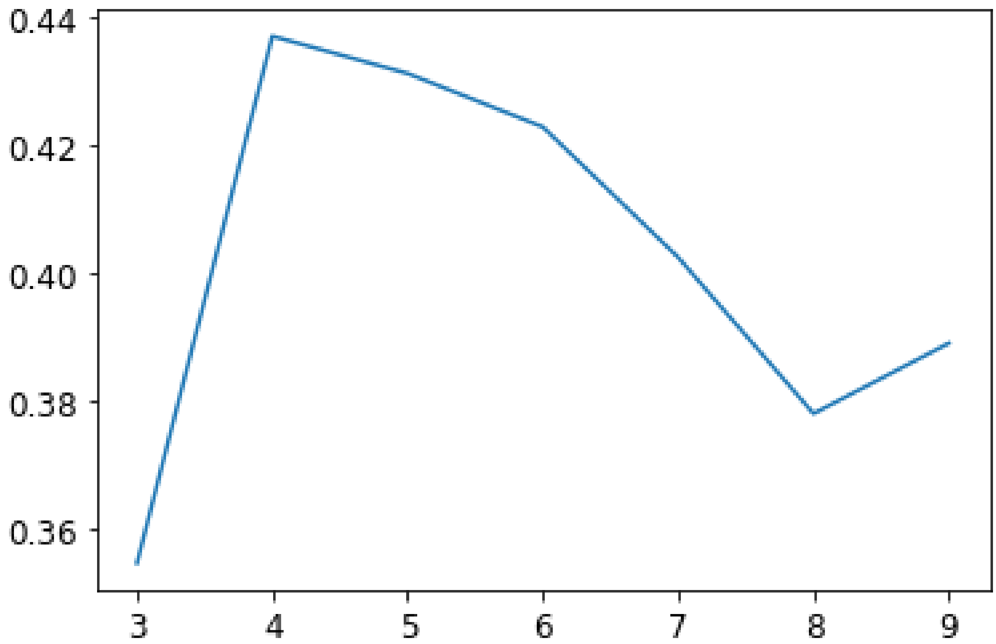
Here also groups are overlapping.

CL-0 and CL-2 behaving same.

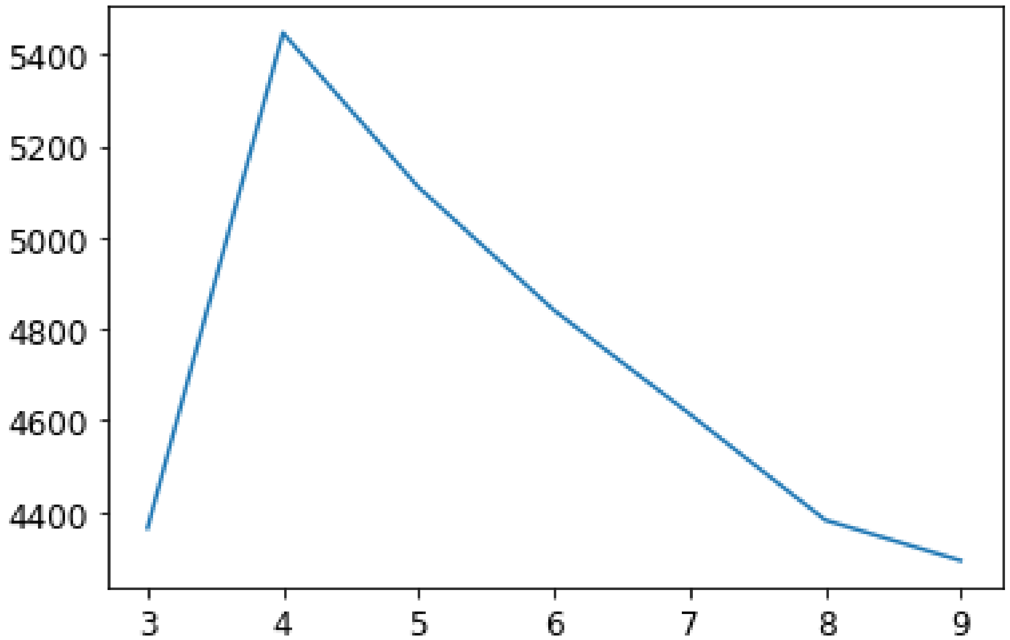
Checking performance metrics for Kmeans

I am validating performance with 2 metrics Calinski harabaz and Silhouette score.

```python
from sklearn.metrics import silhouette_score, calinski_harabasz_score

score={}
score_c={}
for n in range(3,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_cr)
    score_c[n]=calinski_harabasz_score(reduced_cr,km_score.labels_)
    score[n]=silhouette_score(reduced_cr,km_score.labels_)

pd.Series(score).plot()
```




```
599 pd.Series(score_c).plot()
```



Performance metrics also suggest that k-means with 4 cluster is able to show distinguished characteristics of each cluster.

Insights with 4 clusters

Cluster2 is the group of customers who have highest Monthly_avg purchases and doing both installment as well as one_off purchases, have comparatively good credit score.

This group is about 31% of the total customer base.

Cluster 1 is taking maximum advance_cash and is paying comparatively less minimum payment and poor credit_score & doing no purchase transaction .

This group is about 23% of the total customer base.

Cluster 0 customers are doing maximum One_off transactions and least payment ratio and credit_score on lower side.

This group is about 21% of the total customer base.

Cluster 3 customers having maximum credit score and are paying dues and are doing maximum installment purchases

This group is about 25% of the total customer base.

Marketing Strategy suggested:

Group 2

They are potential target customers who are paying dues and doing purchases and maintaining comparatively good credit score. We can increase credit limit or lower down interest rate – Can be given premium card/loyality cards to increase transactions

Group 1

They are poor credit score and taking only cash on advance. We can target them by providing less interest rate on purchase transaction

Group 0

This group is has minimum paying ratio and using card for just one off transactions (may be for utility bills only). This group seems to be risky group.

Group 3

This group is performing best among all as customers are maintaining good credit score and paying dues on time. – Giving reward point will make them perform more purchases.