Project Name – Credit Card Segmentation

Deadline - 15 Days

Problem Statement -

This case requires trainees to develop a customer segmentation to define marketing strategy. The sample dataset summarizes the usage behaviour of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioural variables.

Expectations from the student:

1. Advanced data preparation. Build an 'enriched' customer profile by deriving 'intelligent' KPI's such as monthly average purchase and cash advance amount, purchases by type (one-off, instalments), average amount per purchase and cash advance transaction, limit usage (balance to credit limit ratio), payments to minimum payments ratio etc.
2. Advanced reporting. Use the derived KPI's to gain insight on the customer profiles.
3. Clustering. Apply a data reduction technique factor analysis for variable reduction technique and a clustering algorithm to reveal the behavioural segments of credit card holders

Data Set :

1) credit-card-data.csv

Number of attributes:

- 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 installment purchases

- CASH_ADVANCE Total cash-advance amount
- PURCHASES_ FREQUENCY-Frequency of purchases (percentage of months with at least on purchase)
- ONEOFF_PURCHASES_FREQUENCY Frequency of one-off-purchases
- PURCHASES_INSTALLMENTS_FREQUENCY Frequency of installment purchases
- CASH_ADVANCE_ FREQUENCY Cash-Advance frequency
- AVERAGE_PURCHASE_TRX Average amount per purchase 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_PAYMENTS 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

Overview :-*Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters. Let's understand this with an example. Suppose, we are the head of a rental store and wish to understand preferences of our customers to scale up our business. Is it possible for us to look at details of each costumer and devise a unique business strategy for each one of them? Definitely not. But, what we can do is to cluster all of our customers into say 10 groups based on their purchasing habits and use a separate strategy for costumers in each of these 10 groups. And this is what we call clustering.*

Types of clustering algorithms :-

Since the task of clustering is subjective, the means that can be used for achieving this goal are plenty. Every methodology follows a different set of rules for defining the 'similarity' among data points. In fact, there are more than 100 clustering algorithms known. But few of the algorithms are used popularly, let's look at them in detail….

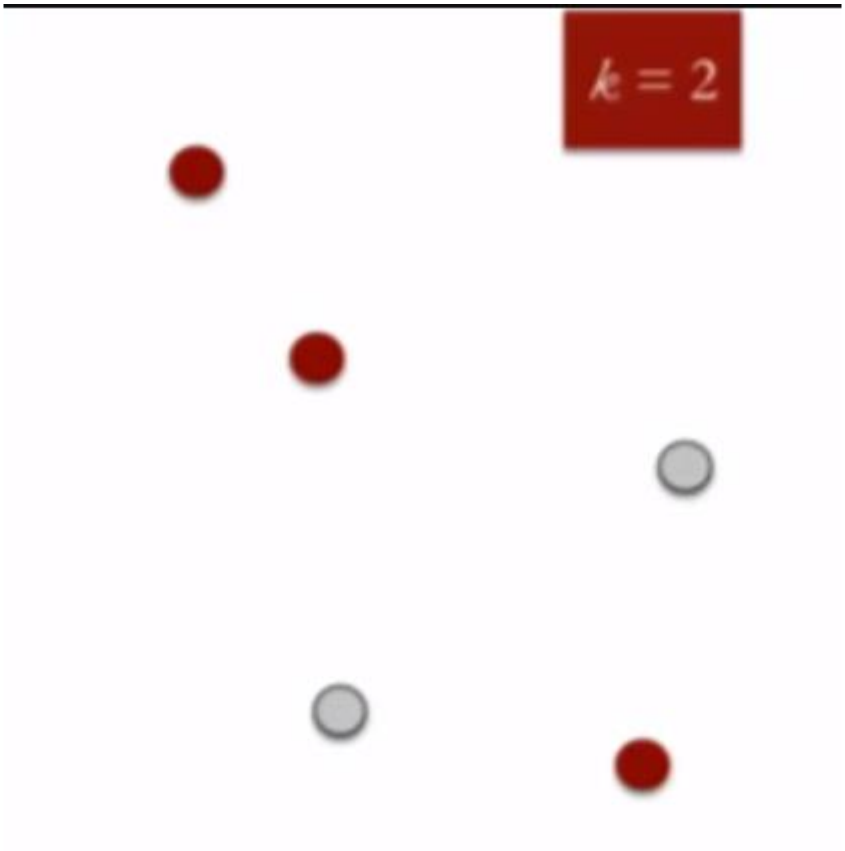
- ***Connectivity models***: *As the name suggests, these models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters & then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. These models are very easy to interpret but lacks scalability for handling big datasets. Examples of these models are hierarchical clustering algorithm and its variants.*
- ***Centroid models***: *These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. K-Means clustering algorithm is a popular algorithm that falls into this category. In these models, the no. of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima*.
- ***Distribution models***: *These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. A popular example of these models is Expectation-maximization algorithm which uses multivariate normal distributions.*
- *Density Models: These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of density models are DBSCAN and OPTICS..*

Now I will be taking you through two of the most popular clustering algorithms in detail – K Means clustering and Hierarchical clustering. Let's begin.

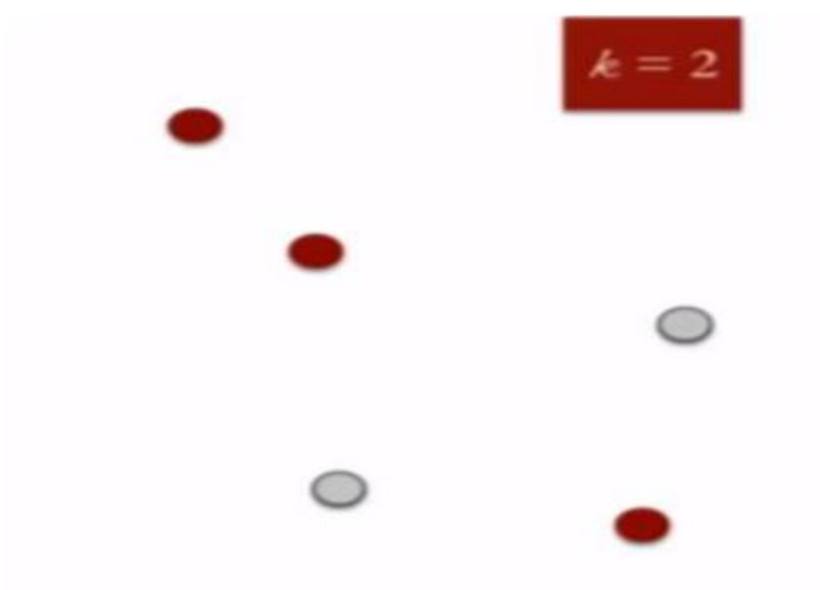
K Means Clustering :

K means is an iterative clustering algorithm that aims to find local maxima in each iteration. This algorithm works in these 5 steps :

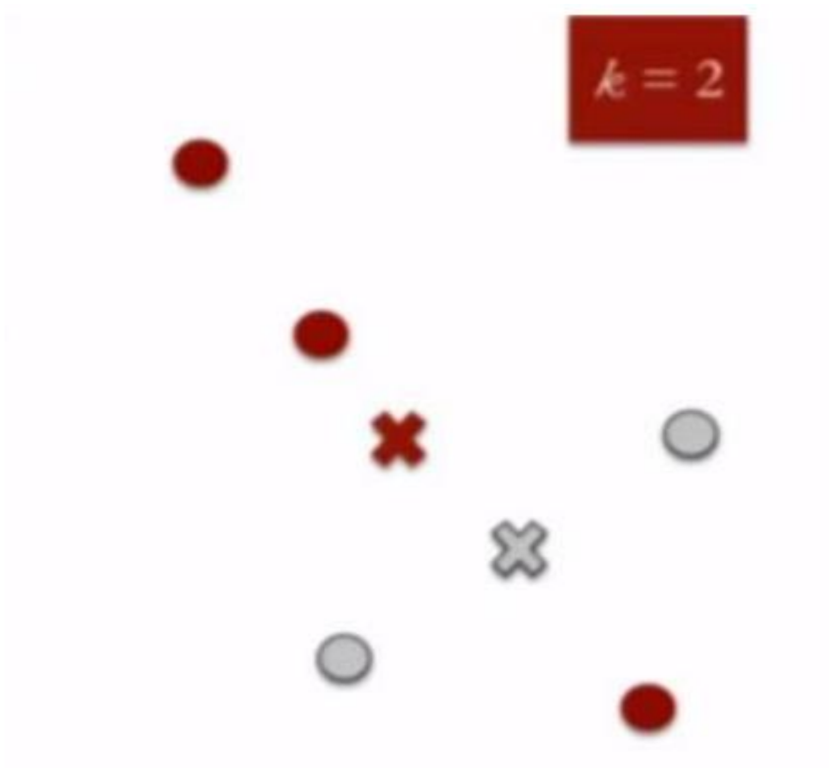
1. Specify the desired number of clusters K : Let us choose k=2 for these 5 data points in 2-D space.



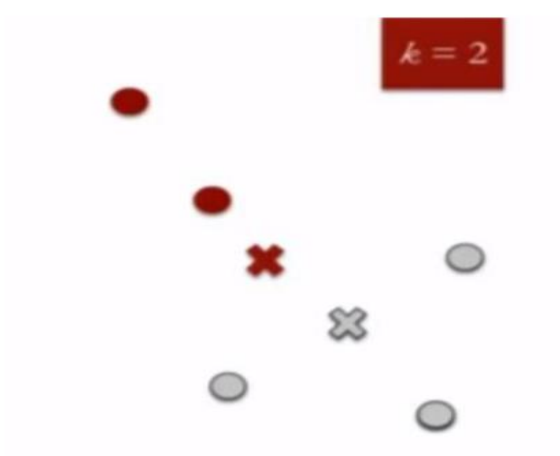
2. *Randomly assign each data point to a cluster . Let's assign three points in cluster 1 shown using red color and two points in cluster 2 shown using grey color.*



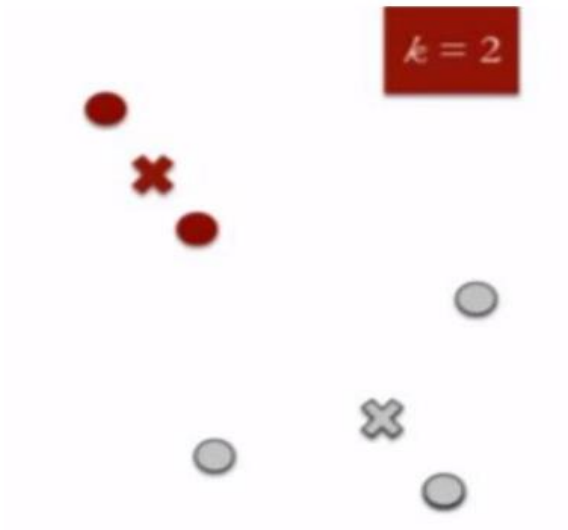
3. Compute cluster centroids : The centroid of data points in the red cluster is shown using red cross and those in grey cluster using grey cross.



4. *Re-assign each point to the closest cluster centroid : Note that only the data point at the bottom is assigned to the red cluster even though its closer to the centroid of grey cluster. Thus, we assign that data point into grey cluster..*



5. *Re-compute cluster centroids : Now, re-computing the centroids for both the clusters*.



6. *Repeat steps 4 and 5 until no improvements are possible : Similarly, we'll repeat the 4th and 5th steps until we'll reach global optima. When there will be no further switching of data points between two clusters for two successive repeats. It will mark the termination of the algorithm if not explicitly mentioned*.

PREVIEW OF OUR PROJECT: *We intend to segment the customer who are using credit cards, by using K Mean model as it a clustering project and comes under unsupervised learning. We will analyse the customer insights and derive the KPI's which would enable the organization to focus on the key areas. To start with, we will be using Python and later on R.*

Buisness Problem: Credit Card Segmentation :

In [1]:

```
## import all necessarry or use full libraries which we are going to use in project work

import pandas as pd
import numpy as np

from datetime import datetime,timedelta

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import statsmodels.formula.api  as sn

import scipy.stats as stats

import statsmodels.api as sm

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

# a library to remove all warnings occuring during project work
import warnings
warnings.filterwarnings('ignore')


```

Load Data

In [2]:

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

In [3]:

```
credit.head()
```

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUE
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.16
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.00
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.00
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.08
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.08

In [4]:

```
credit.info()
```

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

In [5]:

```
# 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())
```

```
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
```

In [6]:
```
credit.shape
```
Out[6]: (8950, 18)

In [7]:
```
# Intital descriptive analysis of data.
credit.describe()
```
Out[7]:

	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 value analysis

here we have some missing value we have to impute this value with the help of median

In [8]:
```
credit.isnull().any()
```
Out[8]:
```
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
```

In [9]:
```
# CREDIT_LIMIT  and MINIMUM_PAYMENTS has missing values so we need to remove with 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)
```

In [10]:
```
# Now again check the missing values.

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

Out[10]:
```
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
```

now we have to work according to the problem statement

Deriving new KPI

1. Monthly average purchase and cash advance amount

Monthly_avg_purchase

In [11]:
```

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

In [12]:
```
print(credit['Monthly_avg_purchase'].head(),'\n ',
credit['TENURE'].head(),'\n', credit['PURCHASES'].head())

```

```
0      7.950000
1      0.000000
2     64.430833
3    124.916667
4      1.333333
Name: Monthly_avg_purchase, dtype: float64
  0    12
1    12
2    12
3    12
4    12
Name: TENURE, dtype: int64
 0      95.40
1       0.00
2     773.17
3    1499.00
4      16.00
Name: PURCHASES, dtype: float64
```

Monthly_cash_advance Amount

In [13]:
```
credit['Monthly_cash_advance']=credit['CASH_ADVANCE']/credit['TENURE']
```

In [14]:
```
credit[credit['ONEOFF_PURCHASES']==0]['ONEOFF_PURCHASES'].count()
```

Out[14]: 4302

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

- To find what type of purchases customers are making on credit card

In [15]:
```
credit.loc[:,['ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES']]
```

Out[15]:

	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

In [16]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```

Out[16]: (2042, 20)

In [17]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

Out[17]: (2774, 20)

In [18]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```

Out[18]: (1874, 20)

In [19]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

Out[19]: (2260, 20)

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

In [20]:
```python
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 'installment'
```

In [21]:
```python
credit['purchase_type']=credit.apply(purchase,axis=1)
```

In [22]:
```python
credit['purchase_type'].value_counts()
```

Out[22]:
```
both_oneoff_installment    2774
installment                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 cutomers are maintaing thier balance properly. Lower value means good credit score

In [23]:
```python
credit['limit_usage']=credit.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT'], axis=1)
```

In [24]:
```python
credit['limit_usage'].head()
```

Out[24]:
```
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

In [25]:
```python
credit['PAYMENTS'].isnull().any()
credit['MINIMUM_PAYMENTS'].isnull().value_counts()
```

Out[25]:
```
False    8950
Name: MINIMUM_PAYMENTS, dtype: int64
```

In [26]:
```python
credit['MINIMUM_PAYMENTS'].describe()
```

Out[26]:
```
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
```

In [27]:
```
credit['payment_minpay']=credit.apply(lambda x:x['PAYMENTS']/x['MINIMUM_PAYMENTS'],axis=1)
```

In [28]:
```
credit['payment_minpay']
```

Out[28]:
```
0       1.446508
1       3.826241
2       0.991682
3       0.000000
4       2.771075
          ...
8945    6.660231
8946    0.883197
8947    0.986076
8948    0.942505
8949    0.715439
Name: payment_minpay, Length: 8950, dtype: float64
```

Extreme value Treatment

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

In [29]:
```
# log tranformation
cr_log=credit.drop(['CUST_ID','purchase_type'],axis=1).applymap(lambda x: np.log(x+1))
```

In [30]:
```
cr_log.describe()

```

Out[30]:

	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	6.161637	0.619940	4.899647	3.204274	3.352403	3.319086	0.361268
std	2.013303	0.148590	2.916872	3.246365	3.082973	3.566298	0.277317
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4.861995	0.635989	3.704627	0.000000	0.000000	0.000000	0.080042
50%	6.773521	0.693147	5.892417	3.663562	4.499810	0.000000	0.405465
75%	7.628099	0.693147	7.013133	6.360274	6.151961	7.016449	0.650588
max	9.854515	0.693147	10.800403	10.615512	10.021315	10.760839	0.693147

8 rows × 21 columns

In [31]:
```
col=['BALANCE','PURCHASES','CASH_ADVANCE','TENURE','PAYMENTS','MINIMUM_PAYMENTS','PRC_FULL_PAYMENT','CREDIT_LIMIT']
cr_pre=cr_log[[x for x in cr_log.columns if x not in col ]]
```

In [32]:
```
cr_pre.columns
```

Out[32]:
```
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')
```

In [33]:
```
cr_log.columns
```

Out[33]:
```
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')
```

2 . Insights from KPIs

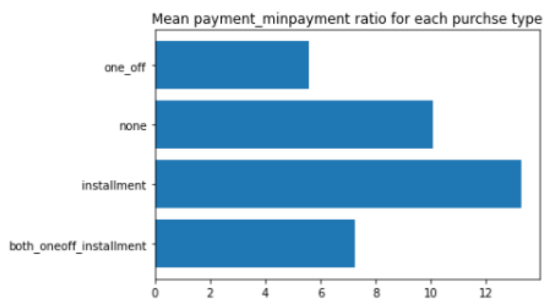
Average payment_minpayment ratio for each purchse type

In [34]:
```
x=credit.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpay']))
type(x)
x.values
```

Out[34]:
```
array([ 7.23698216, 13.2590037 , 10.08745106,  5.57108156])
```

In [35]:
```
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')
```

Out[35]:
```
Text(0.5, 1.0, 'Mean payment_minpayment ratio for each purchse type')
```



customers with installment purchases are paying dues

In [36]:
```
credit[credit['purchase_type']=='n']
```

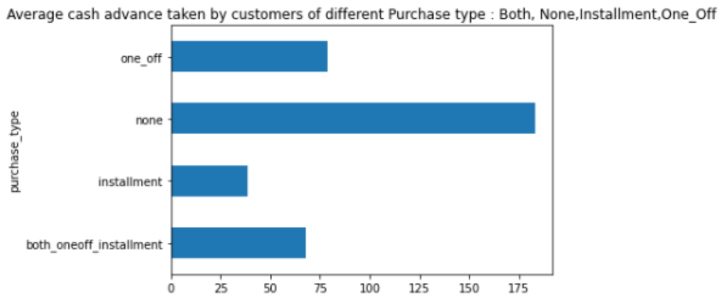
Out[36]:

CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENC

0 rows × 23 columns

In [37]:
```
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')
```

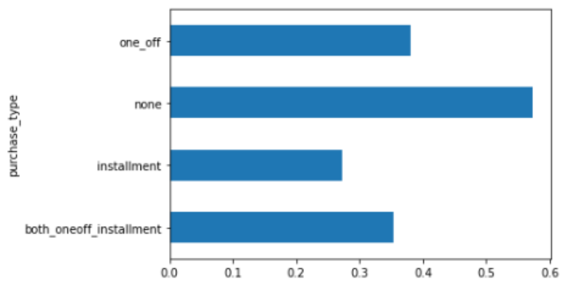
Out[37]:
```
Text(0.5, 1.0, 'Average cash advance taken by customers of different Purchase type : Both, None,Installment,One_Off')
```



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

In [38]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage'])).plot.barh()
```

Out[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x16a8dfd1cd0>
```



Original dataset with categorical column converted to number type

In [39]:
```
cre_original=pd.concat([credit,pd.get_dummies(credit['purchase_type'])],axis=1)
```

Now we are working on Machine Learning algorithm

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

In [40]:

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

Out[40]:

	both_oneoff_installment	installment	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
...	...	...	...	...
8945	0	1	0	0
8946	0	1	0	0
8947	0	1	0	0
8948	0	0	1	0
8949	0	0	0	1

8950 rows × 4 columns

Now merge the created dummy with the original data frame

In [41]:

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

In [42]:

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

Out[42]:

```
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
installment                             False
none                                    False
one_off                                 False
dtype: bool
```

In [43]:
```
1 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  installment                       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
```

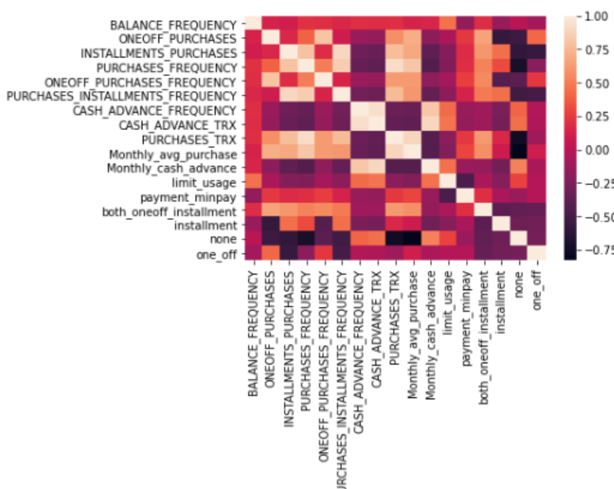
In [44]:
```
1 cr_dummy.head(3)
```

Out[44]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_
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	

In [45]:
```
1 sns.heatmap(cr_dummy.corr())
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x16a8e03c3a0>



- Heat map shows that many features are co-related so applying dimensionality reduction will help negating multi-colinearity 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.

  Standardrizing data
  - To put data on the same scale

In [46]:
```
from sklearn.preprocessing import  StandardScaler
```

In [47]:
```
sc=StandardScaler()
```

In [48]:
```
cr_dummy.shape
```

Out[48]:
```
(8950, 17)
```

In [49]:
```
cr_scaled=sc.fit_transform(cr_dummy)
```

In [50]:
```
cr_scaled
```

Out[50]:
```
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]])
```

Applying PCA

With the help of principal component analysis we will reduce features

In [51]:
```
from sklearn.decomposition import PCA
```

In [52]:
```
cr_dummy.shape
```

Out[52]:
```
(8950, 17)
```

In [53]:
```
#We have 17 features so our n_component will be 17.
pc=PCA(n_components=17)
cr_pca=pc.fit(cr_scaled)
```

In [54]:
```
#Lets check if we will take 17 component then how much varience it explain. Ideally it should be 1 i.e 100%
sum(cr_pca.explained_variance_ratio_)
```

Out[54]:
```
1.0
```

In [55]:
```
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_)
```

In [56]:
```
var_ratio
```

Out[56]:
```
{2: 0.5826439793960279,
 3: 0.7299379309512694,
 4: 0.8115442762351258,
 5: 0.8770555795291433,
 6: 0.9186492443512614,
 7: 0.941092525603013,
 8: 0.9616114053683066,
 9: 0.9739787081990646,
 10: 0.9835896584630706,
 11: 0.989724810734195,
 12: 0.9927550009135223,
 13: 0.9953907562385423,
 14: 0.9979616898169594,
 15: 0.9996360473172955,
 16: 1.0,
 17: 1.0}
```

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

```
In [56]: var_ratio
```

```
Out[56]: {2: 0.5826439793960279,
 3: 0.7299379309512694,
 4: 0.8115442762351258,
 5: 0.8770555795291433,
 6: 0.9186492443512614,
 7: 0.941092525603013,
 8: 0.9616114053683066,
 9: 0.9739787081990646,
 10: 0.9835896584630706,
 11: 0.989724810734195,
 12: 0.9927550009135223,
 13: 0.9953907562385423,
 14: 0.9979616898169594,
 15: 0.9996360473172955,
 16: 1.0,
 17: 1.0}
```

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

```
In [57]: pc=PCA(n_components=6)
```

```
In [58]: p=pc.fit(cr_scaled)
```

```
In [59]: cr_scaled.shape
```

```
Out[59]: (8950, 17)
```

```
In [60]: p.explained_variance_
```

```
Out[60]: array([6.83574755, 3.07030693, 2.50427698, 1.38746289, 1.1138166 ,
       0.70717132])
```

```
In [61]: np.sum(p.explained_variance_)
```

```
Out[61]: 15.618782269308792
```

```
In [62]: np.sum(p.explained_variance_)
```

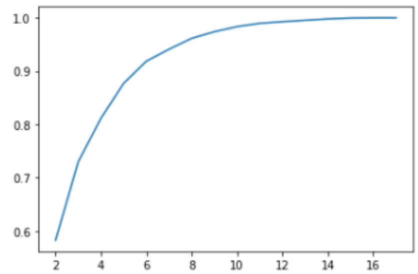
```
Out[62]: 15.618782269308792
```

```
In [63]: var_ratio
```

```
Out[63]: {2: 0.5826439793960279,
 3: 0.7299379309512694,
 4: 0.8115442762351258,
 5: 0.8770555795291433,
 6: 0.9186492443512614,
 7: 0.941092525603013,
 8: 0.9616114053683066,
 9: 0.9739787081990646,
 10: 0.9835896584630706,
 11: 0.989724810734195,
 12: 0.9927550009135223,
 13: 0.9953907562385423,
 14: 0.9979616898169594,
 15: 0.9996360473172955,
 16: 1.0,
 17: 1.0}
```

In [64]:
```
pd.Series(var_ratio).plot()
```

Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x16a8e113d90>



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

In [65]:
```
cr_scaled.shape
```

Out[65]: (8950, 17)

In [66]:
```
pc_final=PCA(n_components=6).fit(cr_scaled)

reduced_cr=pc_final.fit_transform(cr_scaled)
```

In [67]:
```
dd=pd.DataFrame(reduced_cr)
```

In [68]:
```
dd.head()
```

Out[68]:

	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

In [69]:
```
dd.shape

```

Out[69]: (8950, 6)

In [70]:
```
col_list=cr_dummy.columns
```

In [71]:
```
col_list
```

Out[71]:
```
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', 'installment', 'none', 'one_off'],
      dtype='object')
```

In [72]:
```
pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in range(6)],index=col_list)
```
Out[72]:

	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
installment	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 bur in our case its not

In [73]:
```
# Factor Analysis : variance explained by each component-
pd.Series(pc_final.explained_variance_ratio_,index=['PC_'+ str(i) for i in range(6)])
```
Out[73]:
```
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
```

3. Clustering

Based on the intuition on type of purchases 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.

In [74]:
```
from sklearn.cluster import KMeans
```
In [75]:
```
km_4=KMeans(n_clusters=4,random_state=123)
```
In [76]:
```
km_4.fit(reduced_cr)
```
Out[76]:
```
KMeans(n_clusters=4, random_state=123)
```
In [77]:
```
km_4.labels_
```
Out[77]:
```
array([0, 1, 3, ..., 0, 1, 3])
```

In [78]:
```
pd.Series(km_4.labels_).value_counts()
```

Out[78]:
```
2    2769
0    2224
1    2088
3    1869
dtype: int64
```

Here we donot have known 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

In [79]:
```
cluster_range = range( 1, 21 )
cluster_errors = []

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.
```

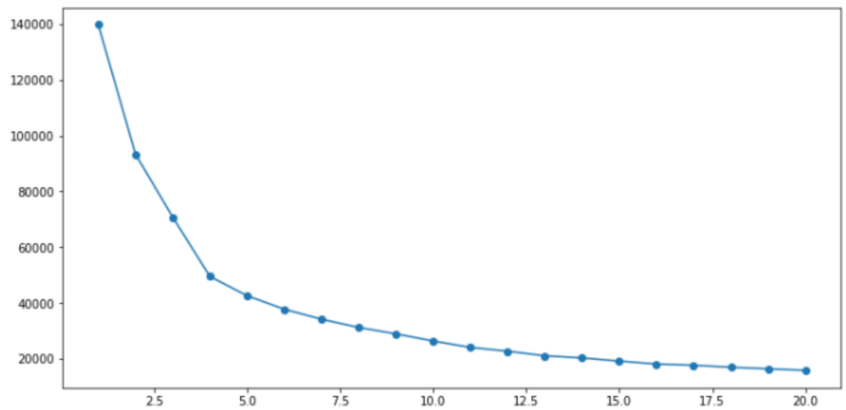
In [80]:
```
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors": cluster_errors } )

clusters_df[0:21]
```

Out[80]:

	num_clusters	cluster_errors
0	1	139772.482528
1	2	93308.610038
2	3	70745.678848
3	4	49446.078418
4	5	42548.771821
5	6	37713.103303
6	7	34124.957046
7	8	31164.910317
8	9	28866.132797
9	10	26318.428828
10	11	24020.681019
11	12	22663.032475
12	13	21006.767673
13	14	20247.490577
14	15	19115.795261
15	16	18059.588652
16	17	17623.240905
17	18	16897.426603
18	19	16351.746420
19	20	15796.268162

In [81]:
```
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

Out[81]: [<matplotlib.lines.Line2D at 0x16a8f7e1af0>]



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

Silhouette Coefficient

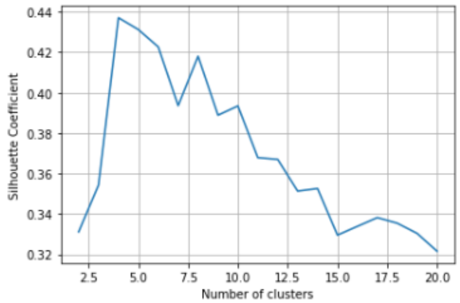
In [82]:
```
# calculate SC for K=3 through K=12
k_range = range(2, 21)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(reduced_cr)
    scores.append(metrics.silhouette_score(reduced_cr, km.labels_))
```

In [83]:
```
scores
```

Out[83]:
```
[0.33113628388878247,
 0.3543181116120539,
 0.4370857743965948,
 0.4312227676966943,
 0.4226353763133636,
 0.3935854567599129,
 0.4180523679657623,
 0.38887744216754966,
 0.3934512405824182,
 0.36787721306485616,
 0.36700015624006194,
 0.3513293979455327,
 0.3525965567774654,
 0.32956074855724693,
 0.33389461989573,
 0.33820715365121434,
 0.33546455056731656,
 0.33040681324630283,
 0.32159407885673597]
```

In [84]:

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

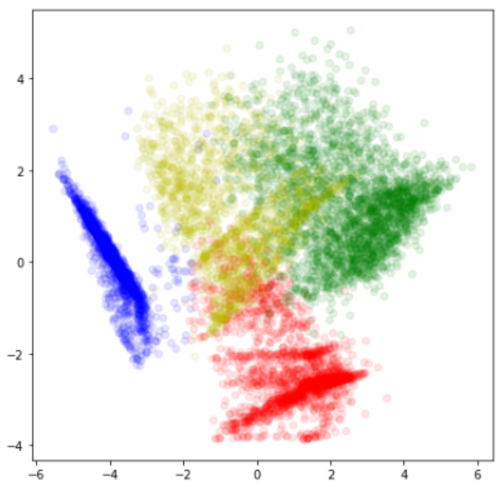


In [85]:

```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)
```

Out[85]:

```
<matplotlib.collections.PathCollection at 0x16a91a90640>
```



It is very difficult to draw iddividual plot for cluster, so we will use pair plot which will provide us all graph in one shot. To do that we need to take following steps

In [86]:

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

In [87]:

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

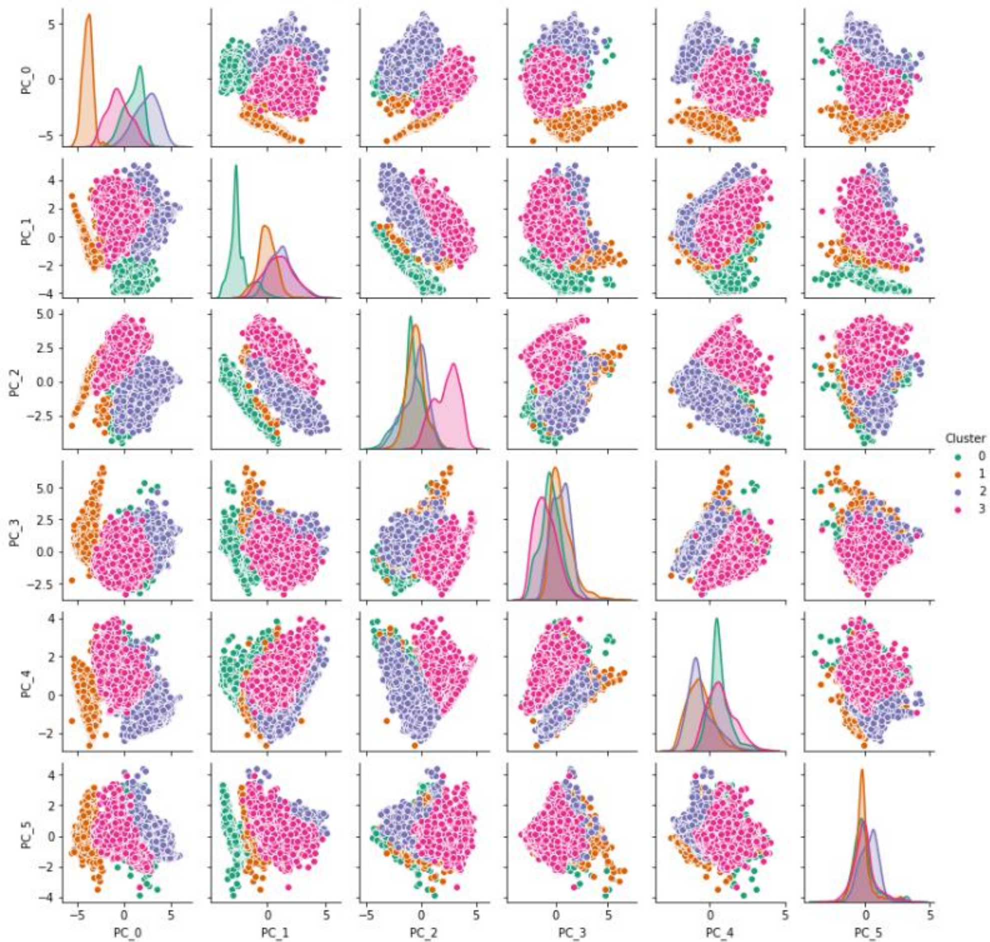
In [88]:
```
df_pair_plot.head()
```

Out[88]:

	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

In [89]:
```
#pairwise relationship of components on the data
sns.pairplot(df_pair_plot,hue='Cluster', palette= 'Dark2', diag_kind='kde',size=1.85)
```

Out[89]: <seaborn.axisgrid.PairGrid at 0x16a91b08d00>



It shows that first two components are able to indentify clusters

Now we have done here with priciple component now we need to come bring our original data frame and we will merge the cluster with them.

To interprate result we need to use our data frame

In [90]:
```
# Key performace variable selection . here i am taking varibales which we will use in derving new KPI.
#We can take all 17 variables but it will be difficult to interprate.So are are selecting less no of variables.

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

In [91]:
```
cr_pre.describe()
```

Out[91]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHAS
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	

In [92]:
```
cluster_df_4=pd.concat([cre_original[col_kpi],pd.Series(km_4.labels_,name='Cluster_4')],axis=1)
```

In [93]:
```
cluster_df_4.head()
```

Out[93]:

	PURCHASES_TRX	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	CASH_ADVANCE_TRX	payment_minpay	both_oneoff_installment	installment
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

In [94]:
```
# Mean value gives a good indication of the distribution of data. So we are finding mean value for each variable for each cl
cluster_4=cluster_df_4.groupby('Cluster_4')\
.apply(lambda x: x[col_kpi].mean()).T
cluster_4
```

Out[94]:

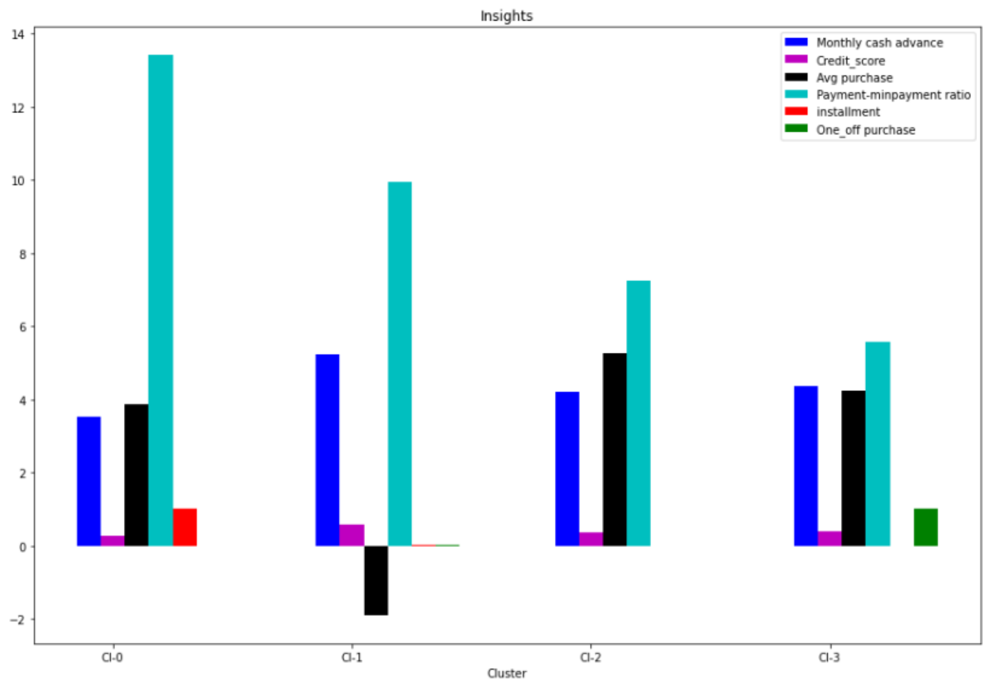
Cluster_4	0	1	2	3
PURCHASES_TRX	12.062050	0.043582	33.013723	7.127341
Monthly_avg_purchase	47.626256	0.148297	193.008043	69.875917
Monthly_cash_advance	33.550080	186.281319	67.466910	78.098613
limit_usage	0.264745	0.576076	0.353591	0.379761
CASH_ADVANCE_TRX	1.021133	6.540230	2.804261	2.881220
payment_minpay	13.422420	9.936617	7.245651	5.573672
both_oneoff_installment	0.000000	0.001916	1.000000	0.000535
installment	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

In [95]:

```python
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['installment',:].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()
```

Out[95]:

```
<matplotlib.legend.Legend at 0x16a9392fe50>
```



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 cus
tomer 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 abo
ut 21% of the total customer base
  *  Cluster 3 customers have maximum credit score and are paying dues and are doing maximum                installment
purchases. This group is about 25% of the total customer base
```

In [96]:
```python
# Percentage of each cluster in 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:

In [97]:
```python
km_5=KMeans(n_clusters=5,random_state=123)
km_5=km_5.fit(reduced_cr)
km_5.labels_
```

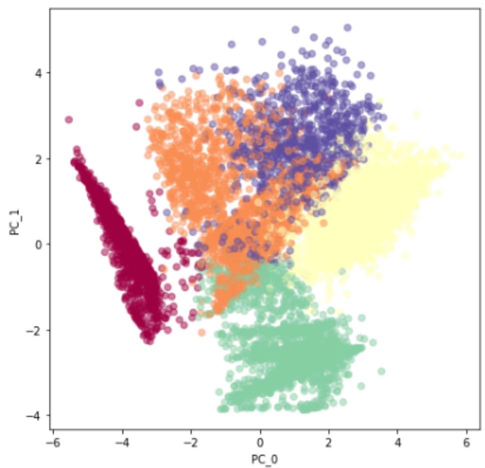
Out[97]: `array([3, 0, 1, ..., 3, 0, 1])`

In [98]:
```python
pd.Series(km_5.labels_).value_counts()
```

Out[98]:
```
3    2147
0    2082
2    1977
1    1862
4     882
dtype: int64
```

In [99]:
```
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')
```

Out[99]:
```
Text(0, 0.5, 'PC_1')
```



In [100]:
```
cluster_df_5=pd.concat([cre_original[col_kpi],pd.Series(km_5.labels_,name='Cluster_5')],axis=1)
```

In [101]:
```
# Finding Mean of features for each cluster
cluster_df_5.groupby('Cluster_5')\
.apply(lambda x: x[col_kpi].mean()).T
```

Out[101]:

Cluster_5	0	1	2	3	4
PURCHASES_TRX	0.033141	7.096670	34.587759	11.910107	27.685941
Monthly_avg_purchase	0.092654	68.917645	210.536468	47.400083	141.441791
Monthly_cash_advance	185.097327	74.517541	4.040708	20.477295	249.745380
limit_usage	0.576097	0.377959	0.258931	0.249543	0.600498
CASH_ADVANCE_TRX	6.449087	2.697637	0.152757	0.543083	10.384354
payment_minpay	9.959037	5.562287	8.675499	13.795305	3.648349
both_oneoff_installment	0.000000	0.002148	1.000000	0.000000	0.899093
installment	0.016330	0.000000	0.000000	1.000000	0.089569
one_off	0.002882	0.997852	0.000000	0.000000	0.011338
none	0.980788	0.000000	0.000000	0.000000	0.000000
CREDIT_LIMIT	4047.870637	4497.951209	5722.970627	3227.300132	5870.351474

Conclusion With 5 clusters :

- we have a group of customers (cluster 2) having highest avergae purchases but there is Cluster 4 also having highest cash advance & secong 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,

In [102]:
```
s1=cluster_df_5.groupby('Cluster_5').apply(lambda x: x['Cluster_5'].value_counts())
print (s1)
```

```
Cluster_5
0          0    2082
1          1    1862
2          2    1977
3          3    2147
4          4     882
Name: Cluster_5, dtype: int64
```

In [103]:
```
# percentage of each cluster

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  2082   23.262570
1  1862   20.804469
2  1977   22.089385
3  2147   23.988827
4   882    9.854749
```

Finding behavior with 6 clusters

In [104]:
```
km_6=KMeans(n_clusters=6).fit(reduced_cr)
km_6.labels_
```

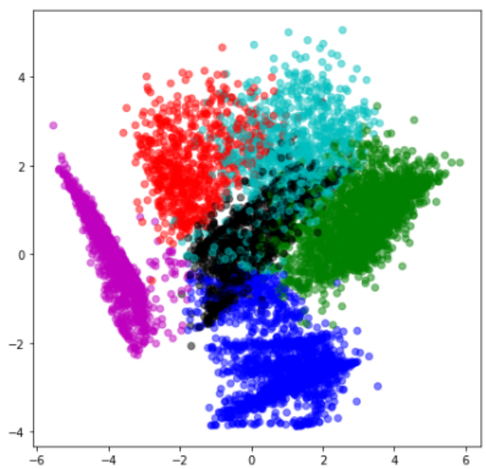
Out[104]:
```
array([1, 4, 5, ..., 1, 4, 0])
```

In [105]:

```python
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)
```

Out[105]:

```
<matplotlib.collections.PathCollection at 0x16a93f51c10>
```



In [106]:

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

In [107]:

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

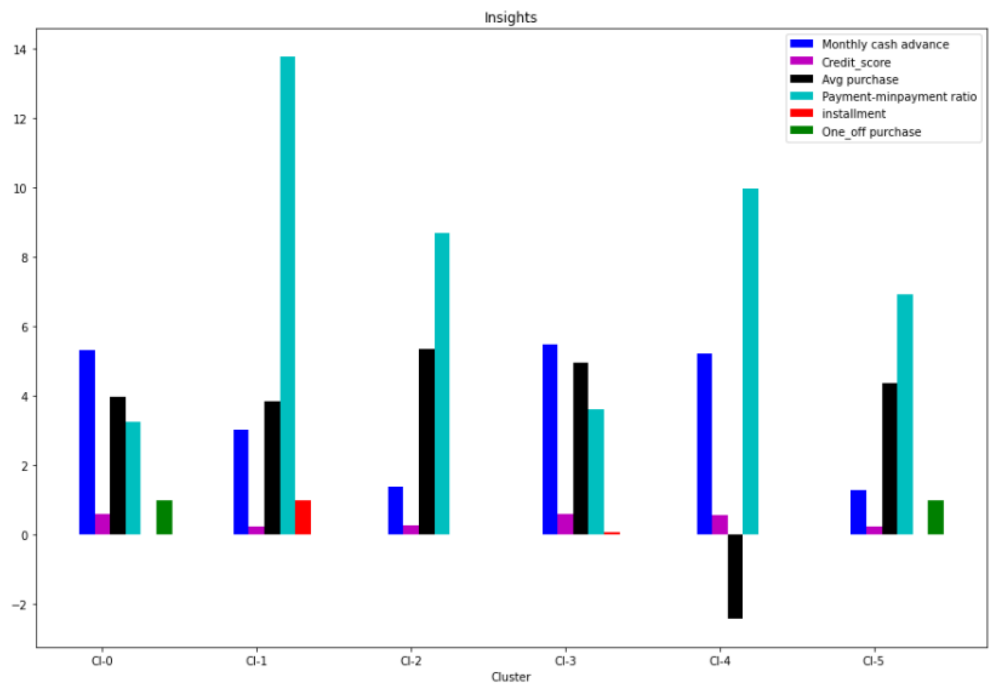
Out[107]:

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

In [108]:

```
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['installment',:].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()
```

Out[108]: <matplotlib.legend.Legend at 0x16a93fd8a60>



In [109]:

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

Out[109]:

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

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**

In [110]:

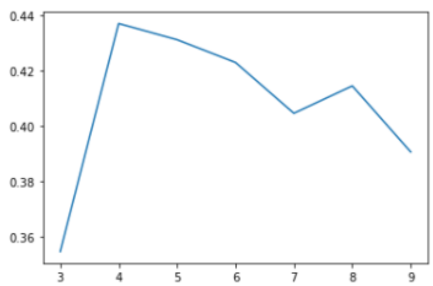
```
score={}
score_c={}
for n in range(3,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_cr)
    score_c[n]=calinski_harabasz_score(reduced_cr,km_score.labels_)
    score[n]=silhouette_score(reduced_cr,km_score.labels_)
```

In [111]:

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

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16a941a5d90>
```

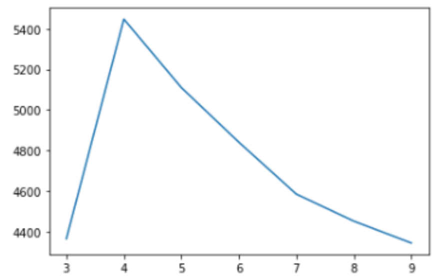


In [117]:

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

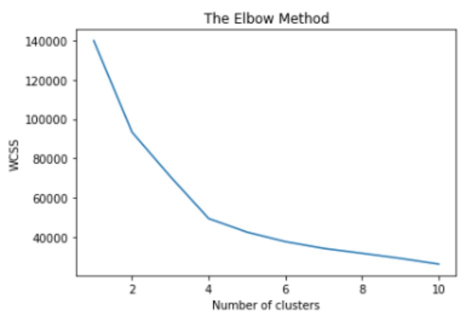
Out[117]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x16a94a232e0>
```



In [121]:

```python
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(reduced_cr)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



Observations:

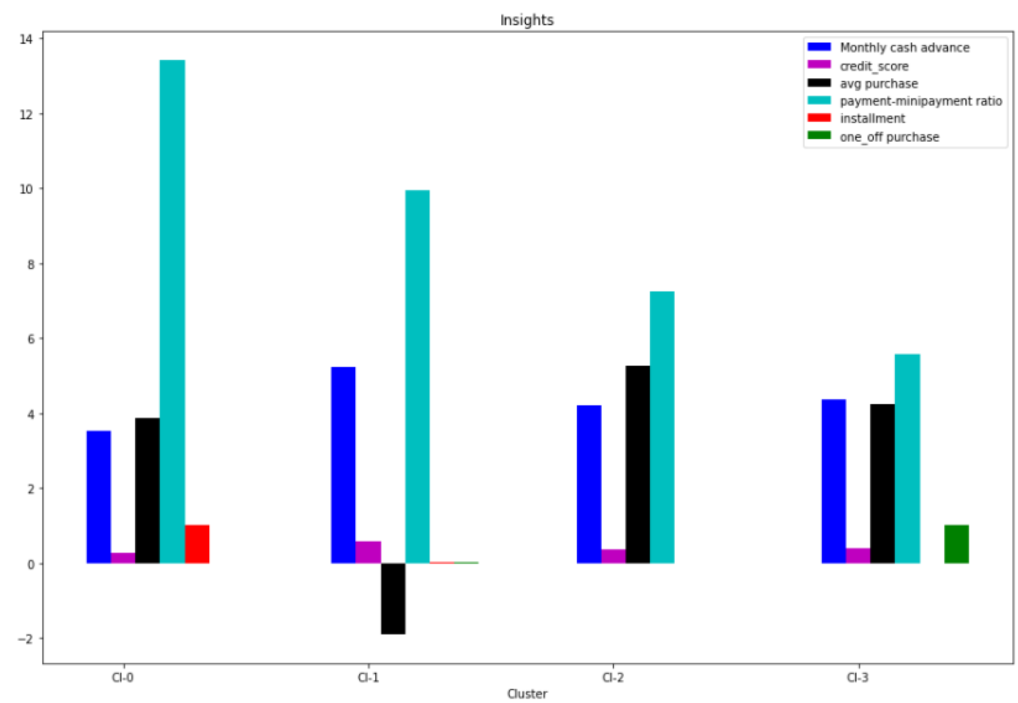
From all the above graphs we can conclude the performance of the KMeans Model regarding the explanation of data distribution and measure of spread is highest when we consider the number of cluster as four.

Final K-Means Model

In [116]:

```
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['installment',:].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-minipayment 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()
```

Out[116]: `<matplotlib.legend.Legend at 0x16a94c64850>`



Marketing Strategies cluster:

CLUSTER 0:

Customers belong to this cluster must be the primary focus regarding the marketing strategy because the customers under this cluster are making frequent purchases and also paying the dues on time thus maintaining good credit score. Customers in this cluster must be given with good reward points and provided with increased credit limit or the premium credit cards with some exciting offers make them do more transactions in the future

CLUSTER 1:

Customers who fall under this category of cluster are having the best credit card and also paying the dues on time without defaults. Hence these group of customers must rewarded with reward points and thus make them do more transactions in future.

CLUSTER 2:

Customers belong to this category of cluster having the highest cash advance and poor avg purchase score yet these customers pay the due amounts of the installments on time.Hence these customers may be given with the loan amounts at less interest charges, thus help the banks providing continuous services to these group of customers in future

CLUSTER 3:

Customers belong to this cluster has the least minimum payment ratio and always does the one off payment transactions, hence no bank offers can excite these kind of cutomers. The marketing to this group of customers is hard and when the usage is minimum, this group can be ignored from the marketing strategy.Further the customers falling under this category can be rejected from issuing the credit cards in future.

THE SAME THINGS WE DO IN R :-

```
rm(list = ls(all=T))
credit = read.csv("C:/Users/adars/Downloads/credit-card-data.csv")
View(credit)
sum(is.na(credit$CUST_ID))
sum(is.na(credit$BALANCE))
sum(is.na(credit$BALANCE_FREQUENCY))
sum(is.na(credit$PURCHASES))
sum(is.na(credit$ONEOFF_PURCHASES))
sum(is.na(credit$INSTALLMENTS_PURCHASES))
sum(is.na(credit$CASH_ADVANCE))
sum(is.na(credit$PURCHASES_FREQUENCY))
sum(is.na(credit$ONEOFF_PURCHASES_FREQUENCY))
sum(is.na(credit$PURCHASES_INSTALLMENTS_FREQUENCY))
sum(is.na(credit$CASH_ADVANCE_FREQUENCY))
sum(is.na(credit$CASH_ADVANCE_TRX))
sum(is.na(credit$PURCHASES_TRX))
sum(is.na(credit$CREDIT_LIMIT))##1
sum(is.na(credit$PAYMENTS))
sum(is.na(credit$MINIMUM_PAYMENTS))##313
```

```
sum(is.na(credit$PRC_FULL_PAYMENT))
sum(is.na(credit$TENURE))
######### Identifying Outliers##########
mystats = function(x) {
  nmiss=sum(is.na(x))
  a = x[!is.na(x)]
  m = mean(a)
  n = length(a)
  s = sd(a)
  min = min(a)
  p1=quantile(a,0.01)
  p5=quantile(a,0.05)
  p10=quantile(a,0.10)
  q1=quantile(a,0.25)
  q2=quantile(a,0.5)
  q3=quantile(a,0.75)
  p90=quantile(a,0.90)
  p95=quantile(a,0.95)
  p99=quantile(a,0.99)
  max = max(a)
  UC = m+2*s
  LC = m-2*s
  outlier_flag= max>UC | min<LC
  return(c(n=n, nmiss=nmiss, outlier_flag=outlier_flag, mean=m,
stdev=s,min = min,
p1=p1,p5=p5,p10=p10,q1=q1,q2=q2,q3=q3,p90=p90,p95=p95,p99=p99,ma
x=max, UC=UC, LC=LC ))
}
##########New Variables creation#########
credit$Monthly_Avg_PURCHASES =
credit$PURCHASES/(credit$PURCHASES_FREQUENCY*credit$TENURE
)
credit$Monthly_CASH_ADVANCE =
credit$CASH_ADVANCE/(credit$CASH_ADVANCE_FREQUENCY*credit$
TENURE)
```

```
credit$LIMIT_USAGE = credit$BALANCE/credit$CREDIT_LIMIT
credit$MIN_PAYMENTS_RATIO =
credit$PAYMENTS/credit$MINIMUM_PAYMENTS


write.csv(credit,"New_variables_creation.csv")




Num_Vars =
c("BALANCE","BALANCE_FREQUENCY","PURCHASES","Monthly_Avg_P
URCHASES","ONEOFF_PURCHASES","INSTALLMENTS_PURCHASES",

"CASH_ADVANCE","Monthly_CASH_ADVANCE","PURCHASES_FREQU
ENCY","ONEOFF_PURCHASES_FREQUENCY","PURCHASES_INSTALL
MENTS_FREQUENCY",

"CASH_ADVANCE_FREQUENCY","CASH_ADVANCE_TRX","PURCHASE
S_TRX","CREDIT_LIMIT","LIMIT_USAGE","PAYMENTS",

"MINIMUM_PAYMENTS","MIN_PAYMENTS_RATIO","PRC_FULL_PAYME
NT","TENURE")



Outliers=t(data.frame(apply(credit[Num_Vars], 2, mystats)))
View(Outliers)


write.csv(Outliers,"Outliers.csv")


credit$BALANCE[credit$BALANCE>5727.53]=5727.53
credit$BALANCE_FREQUENCY[credit$BALANCE_FREQUENCY>1.35107
87]=1.3510787
```

```
credit$PURCHASES[credit$PURCHASES>5276.46]=5276.46
credit$Monthly_Avg_PURCHASES[credit$Monthly_Avg_PURCHASES>800.03] = 800.03
credit$ONEOFF_PURCHASES[credit$ONEOFF_PURCHASES>3912.2173709]=3912.2173709
credit$INSTALLMENTS_PURCHASES[credit$INSTALLMENTS_PURCHASES>2219.7438751]=2219.7438751
credit$CASH_ADVANCE[credit$CASH_ADVANCE>5173.1911125]=5173.1911125
credit$Monthly_CASH_ADVANCE[credit$Monthly_CASH_ADVANCE>2558.53] = 2558.53
credit$PURCHASES_FREQUENCY[credit$PURCHASES_FREQUENCY>1.2930919]=1.2930919
credit$ONEOFF_PURCHASES_FREQUENCY[credit$ONEOFF_PURCHASES_FREQUENCY>0.7991299]=0.7991299
credit$PURCHASES_INSTALLMENTS_FREQUENCY[credit$PURCHASES_INSTALLMENTS_FREQUENCY>1.1593329]=1.1593329
credit$CASH_ADVANCE_FREQUENCY[credit$CASH_ADVANCE_FREQUENCY>0.535387]=0.535387
credit$CASH_ADVANCE_TRX[credit$CASH_ADVANCE_TRX>16.8981202]=16.8981202
credit$PURCHASES_TRX[credit$PURCHASES_TRX>64.4251306]=64.4251306
credit$CREDIT_LIMIT[credit$CREDIT_LIMIT>11772.09]=11772.09
credit$LIMIT_USAGE[credit$LIMIT_USAGE>1.1683] = 1.1683
credit$PAYMENTS[credit$PAYMENTS>7523.26]=7523.26
credit$MINIMUM_PAYMENTS[credit$MINIMUM_PAYMENTS>5609.1065423]=5609.1065423
credit$MIN_PAYMENTS_RATIO[credit$MIN_PAYMENTS_RATIO>249.9239] = 249.9239
credit$PRC_FULL_PAYMENT[credit$PRC_FULL_PAYMENT>0.738713]=0.738713
credit$TENURE[credit$TENURE>14.19398]=14.19398
```

```
###### Missing value analysis using mean
credit$MINIMUM_PAYMENTS[which(is.na(credit$MINIMUM_PAYMENTS))
] = 721.9256368
credit$CREDIT_LIMIT[which(is.na(credit$CREDIT_LIMIT))] = 4343.62
credit$Monthly_Avg_PURCHASES[which(is.na(credit$Monthly_Avg_PURC
HASES))] =184.8991609
credit$Monthly_CASH_ADVANCE[which(is.na(credit$Monthly_CASH_ADV
ANCE))] = 717.7235629
credit$LIMIT_USAGE[which(is.na(credit$LIMIT_USAGE))] =0.3889264
credit$MIN_PAYMENTS_RATIO[which(is.na(credit$MIN_PAYMENTS_RAT
IO))] = 9.3500701


########### Checking Missing Value  ##########
check_Missing_Values=t(data.frame(apply(credit[Num_Vars], 2, mystats)))
View(check_Missing_Values)
write.csv(credit,"Missing_value_treatment.csv")
# Variable Reduction (Factor Analysis)
Step_nums = credit[Num_Vars]
corrm= cor(Step_nums)
View(corrm)
write.csv(corrm, "Correlation_matrix.csv")
eigen(corrm)$values


install.packages(c('dplyr','psych','tables'))
library(dplyr)
eigen_values = mutate(data.frame(eigen(corrm)$values)
                 ,cum_sum_eigen=cumsum(eigen.corrm..values)
                 , pct_var=eigen.corrm..values/sum(eigen.corrm..values)
                 , cum_pct_var=cum_sum_eigen/sum(eigen.corrm..values))
write.csv(eigen_values, "EigenValues2.csv")
```

```
############# standardizing the data ###########
credit_prepared =credit[Num_Vars]
credit_prepared = scale(credit_prepared)
write.csv(credit_prepared, "standardized data.csv")


#building clusters using k-means clustering
cluster_three = kmeans(credit_prepared,3)
cluster_four = kmeans(credit_prepared,4)
cluster_five = kmeans(credit_prepared,5)
cluster_six = kmeans(credit_prepared,6)
credit_new=cbind(credit,km_clust_3=cluster_three$cluster,km_clust_4=clus
ter_four$cluster,
                 km_clust_5=cluster_five$cluster ,km_clust_6=cluster_six$cluster
)
View(credit_new)
# Profiling
Num_Vars2 = c(
  "Monthly_Avg_PURCHASES",
  "Monthly_CASH_ADVANCE",
  "CASH_ADVANCE",
  "CASH_ADVANCE_TRX",
  "CASH_ADVANCE_FREQUENCY",
  "ONEOFF_PURCHASES",
  "ONEOFF_PURCHASES_FREQUENCY",
  "PAYMENTS",
  "CREDIT_LIMIT",
  "LIMIT_USAGE",
  "PURCHASES_INSTALLMENTS_FREQUENCY",
  "PURCHASES_FREQUENCY",
  "INSTALLMENTS_PURCHASES",
  "PURCHASES_TRX",
  "MINIMUM_PAYMENTS",
  "MIN_PAYMENTS_RATIO",
```

```
  "BALANCE",
  "TENURE")

library(tables)

tt
=cbind(tabular(1+factor(km_clust_3)+factor(km_clust_4)+factor(km_clust_5
)+
            factor(km_clust_6)~Heading()*length*All(credit[1]),


data=credit_new),tabular(1+factor(km_clust_3)+factor(km_clust_4)+factor(k
m_clust_5)+


factor(km_clust_6)~Heading()*mean*All(credit[Num_Vars2]),
                        data=credit_new))
tt2 = as.data.frame.matrix(tt)
View(tt2)

rownames(tt2)=c(
  "ALL",
  "KM3_1",
  "KM3_2",
  "KM3_3",
  "KM4_1",
  "KM4_2",
  "KM4_3",
  "KM4_4",
  "KM5_1",
  "KM5_2",
  "KM5_3",
  "KM5_4",
  "KM5_5",
  "KM6_1",
```

```
  "KM6_2",
  "KM6_3",
  "KM6_4",
  "KM6_5",
  "KM6_6")
colnames(tt2)=c(
  "SEGMENT_SIZE",
  "Monthly_Avg_PURCHASES",
  "Monthly_CASH_ADVANCE",
  "CASH_ADVANCE",
  "CASH_ADVANCE_TRX",
  "CASH_ADVANCE_FREQUENCY",
  "ONEOFF_PURCHASES",
  "ONEOFF_PURCHASES_FREQUENCY",
  "PAYMENTS",
  "CREDIT_LIMIT",
  "LIMIT_USAGE",
  "PURCHASES_INSTALLMENTS_FREQUENCY",
  "PURCHASES_FREQUENCY",
  "INSTALLMENTS_PURCHASES",
  "PURCHASES_TRX",
  "MINIMUM_PAYMENTS",
  "MIN_PAYMENTS_RATIO",
  "BALANCE",
  "TENURE"
)
cluster_profiling2 = t(tt2)
write.csv(cluster_profiling2,'cluster_profiling2.csv')
```