Customer Segmentaion

In [167]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
import seaborn as sns
```

In [168]:
```
# reading data into dataframe

credit= pd.read_csv("CC_GENERAL.csv")
```

In [169]:
```
credit.head()
```

Out[169]:

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

In [170]:
```
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 [171]:
```
credit.shape
```

Out[171]:
```
(8950, 18)
```

In [172]:
```
# Intital descriptive analysis of data.
credit.describe()
```

Out[172]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FRI
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	89
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	
min	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	
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	

a) Missing Value Treatment

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

In [173]:
```
credit.isnull().any()
```

Out[173]:
```
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 [174]:
```
credit['CREDIT_LIMIT'].fillna(credit['CREDIT_LIMIT'].median(),inplace=True)
credit['MINIMUM_PAYMENTS'].fillna(credit['MINIMUM_PAYMENTS'].median(),inplace=True)
```

In [175]:
```
credit.isnull().any()
```

Out[175]:
```
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 KPI(KEY PERFORMANCE INDICATORS)

---

1. Monthly_avg_purchase

In [176]:
```
credit['Monthly_avg_purchase']=credit['PURCHASES']/credit['TENURE']
```

In [177]:
```
print('\n\n Monthly_avg_purchase:\n\n',credit['Monthly_avg_purchase'].head(),'\n\n Tenure:\n\n' ,
credit['TENURE'].head(),'\n\n Average purchase\n\n', credit['PURCHASES'].head())
```

```
 Monthly_avg_purchase:

 0      7.950000
1      0.000000
2     64.430833
3    124.916667
4      1.333333
Name: Monthly_avg_purchase, dtype: float64

 Tenure:

 0    12
1    12
2    12
3    12
4    12
Name: TENURE, dtype: int64

 Average purchase

 0      95.40
1       0.00
2     773.17
3    1499.00
4      16.00
Name: PURCHASES, dtype: float64
```

2. Monthly_cash_advance

In [178]:
```
credit['Monthly_cash_advance']=credit['CASH_ADVANCE']/credit['TENURE']
```

In [179]:
```
credit[credit['ONEOFF_PURCHASES']==0]['ONEOFF_PURCHASES'].count()
```
Out[179]:
```
4302
```

3. Purchase_type

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

In [180]:
```
credit.loc[:,['ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES']]
```
Out[180]:

	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 [181]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```
Out[181]:
```
(2042, 20)
```

In [182]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

Out[182]:
```
(2774, 20)
```

In [183]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
```

Out[183]:
```
(1874, 20)
```

In [184]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
```

Out[184]:
```
(2260, 20)
```

We can spot that there are 4 types of purchase behaviour in the data set. So deriving a categorical variable based on the behaviour

In [185]:
```
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'
```

In [186]:
```
credit['purchase_type']=credit.apply(purchase,axis=1)
```

In [187]:
```
credit['purchase_type'].value_counts()
```

Out[187]:
```
both_oneoff_installment    2774
istallment                 2260
none                       2042
one_off                    1874
Name: purchase_type, dtype: int64
```

---

4. Limit_usage (shows credit-score) credit card utilization

- Lower value implies cutomers are maintaing thier balance properly. Lower value means good credit score

In [188]:
```
credit['limit_usage']=credit.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT'], axis=1)
```

In [189]:
```
credit['payment_minpay']=credit.apply(lambda x:x['PAYMENTS']/x['MINIMUM_PAYMENTS'],axis=1)
```

In [190]:
```
credit.dtypes
```

Out[190]:
```
CUST_ID                              object
BALANCE                             float64
BALANCE_FREQUENCY                   float64
PURCHASES                           float64
ONEOFF_PURCHASES                    float64
INSTALLMENTS_PURCHASES              float64
CASH_ADVANCE                        float64
PURCHASES_FREQUENCY                 float64
ONEOFF_PURCHASES_FREQUENCY          float64
PURCHASES_INSTALLMENTS_FREQUENCY    float64
CASH_ADVANCE_FREQUENCY              float64
CASH_ADVANCE_TRX                      int64
PURCHASES_TRX                         int64
CREDIT_LIMIT                        float64
PAYMENTS                            float64
MINIMUM_PAYMENTS                    float64
PRC_FULL_PAYMENT                    float64
TENURE                                int64
Monthly_avg_purchase                float64
Monthly_cash_advance                float64
purchase_type                        object
limit_usage                         float64
payment_minpay                      float64
dtype: object
```

b) Extreme value Treatment

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

In [191]:
```
# log tranformation
cr_log=credit.drop(['CUST_ID','purchase_type'],axis=1).applymap(lambda x: np.log(x+1))
```

In [192]:
```
cr_log.describe()
```

Out[192]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQU
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.0
mean	6.161637	0.619940	4.899647	3.204274	3.352403	3.319086	0.3
std	2.013303	0.148590	2.916872	3.246365	3.082973	3.566298	0.2
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	4.861995	0.635989	3.704627	0.000000	0.000000	0.000000	0.0
50%	6.773521	0.693147	5.892417	3.663562	4.499810	0.000000	0.4
75%	7.628099	0.693147	7.013133	6.360274	6.151961	7.016449	0.6
max	9.854515	0.693147	10.800403	10.615512	10.021315	10.760839	0.6

rows × 21 columns

In [193]:
```
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 ]]
```

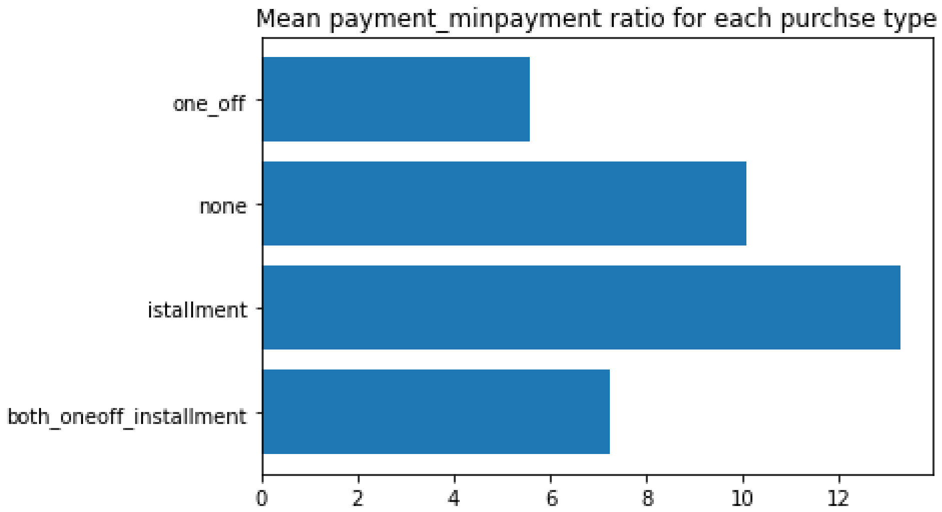
Insights from new KPIs

In [194]:
```
# Average payment_minpayment ratio for each purchse type.
x=credit.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpay']))
type(x)
x.values
```

Out[194]:
```
array([ 7.23698216, 13.2590037 , 10.08745106,  5.57108156])
```

In [195]:
```
#plt.barh(left=np.arange(len(x)),bottom=x.values)
fig,ax=plt.subplots()
ax.barh(y=range(len(x)),width=x.values)
ax.set(yticks= np.arange(len(x)),yticklabels=x.index);
plt.title('Mean payment_minpayment ratio for each purchse type')
```

Out[195]:
```
Text(0.5, 1.0, 'Mean payment_minpayment ratio for each purchse type')
```



In [196]:
```
credit.describe()
```

Out[196]:

‚SES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FI
0000	8950.000000	8950.000000	8950.000000	8950.000000	8
7645	978.871112	0.490351	0.202458	0.364437	
8115	2097.163877	0.401371	0.298336	0.397448	
0000	0.000000	0.000000	0.000000	0.000000	
0000	0.000000	0.083333	0.000000	0.000000	
0000	0.000000	0.500000	0.083333	0.166667	
7500	1113.821139	0.916667	0.300000	0.750000	
0000	47137.211760	1.000000	1.000000	1.000000	

customers with installment purchases are paying dues

In [197]:
```
credit[credit['purchase_type']=='none']
```

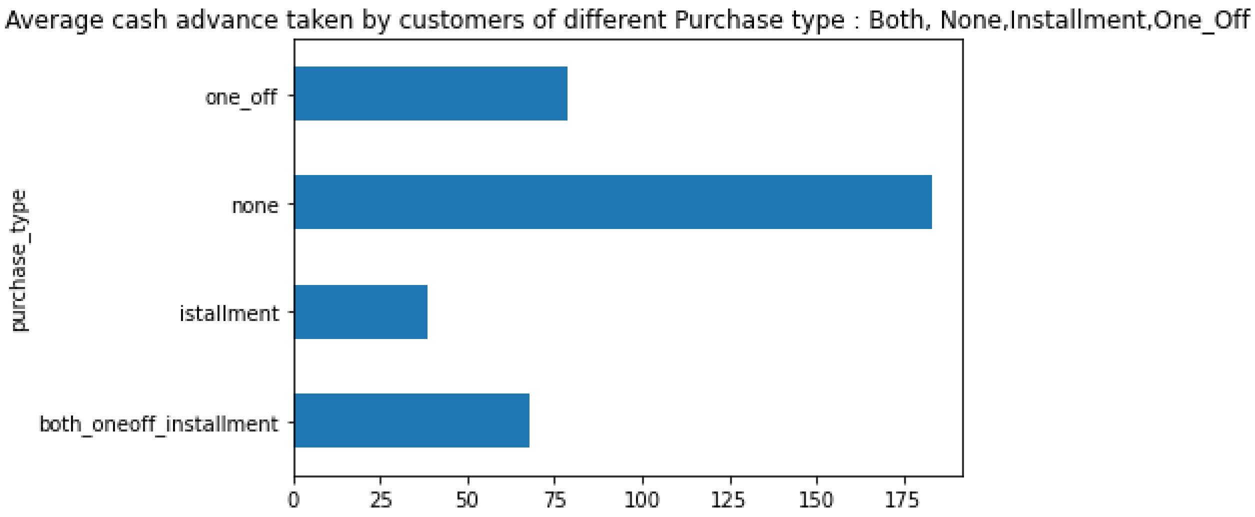
Out[197]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHA
1	C10002	3202.467416	0.909091	0.0	0.0	0.0	6442.945483	
14	C10015	2772.772734	1.000000	0.0	0.0	0.0	346.811390	
16	C10017	2072.074354	0.875000	0.0	0.0	0.0	2784.274703	
24	C10025	5368.571219	1.000000	0.0	0.0	0.0	798.949863	
35	C10036	1656.350781	1.000000	0.0	0.0	0.0	99.264367	
...	...	...	...	...	...	...	...	
8920	C19161	1055.087681	0.666667	0.0	0.0	0.0	1820.116200	
8929	C19170	371.527312	0.333333	0.0	0.0	0.0	1465.407927	
8937	C19178	163.001629	0.666667	0.0	0.0	0.0	274.440466	
8938	C19179	78.818407	0.500000	0.0	0.0	0.0	1113.186078	
8948	C19189	13.457564	0.833333	0.0	0.0	0.0	36.558778	

2042 rows × 23 columns

In [198]:
```
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')
```

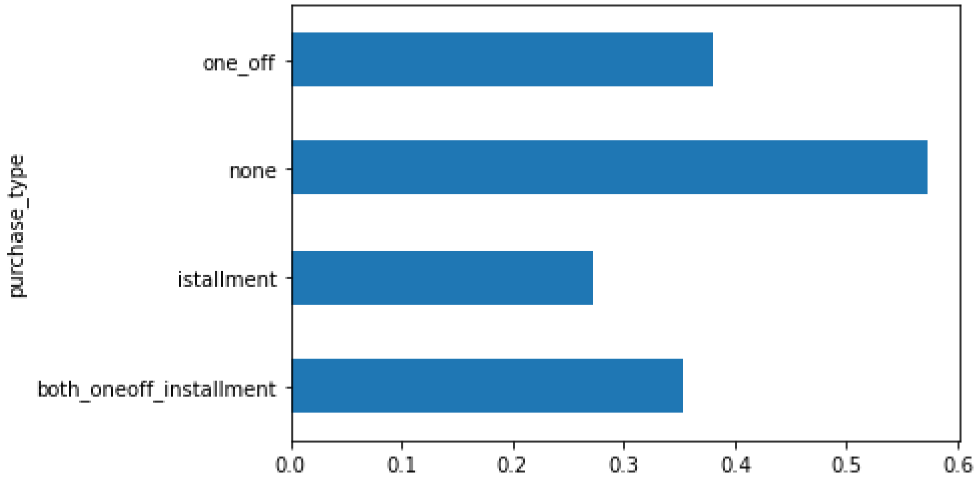
Out[198]:
```
Text(0.5, 1.0, 'Average cash advance taken by customers of different Purchase type : Both, None,Installment,One_Off')
```



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

In [199]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage'])).plot.barh()
```

Out[199]: <AxesSubplot:ylabel='purchase_type'>



Customers with installment purchases have good credit score

In [200]:
```
# Original dataset with categorical column converted to number type.
cre_original=pd.concat([credit,pd.get_dummies(credit['purchase_type'])],axis=1)
```

c) Preparing for Machine learning

In [201]:
```
# creating Dummies for categorical variable
cr_pre['purchase_type']=credit.loc[:,'purchase_type']
pd.get_dummies(cr_pre['purchase_type'])
```

```
<ipython-input-201-b4513e35aa10>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py)
  cr_pre['purchase_type']=credit.loc[:,'purchase_type']
```

Out[201]:

	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
...	...	...	...	...
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

In [202]:
```
cr_dummy=pd.concat([cr_pre,pd.get_dummies(cr_pre['purchase_type'])],axis=1)
```

In [203]:
```
l=['purchase_type']
```

In [204]:
```
cr_dummy=cr_dummy.drop(l,axis=1)
cr_dummy.isnull().any()
```

Out[204]:
```
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
```

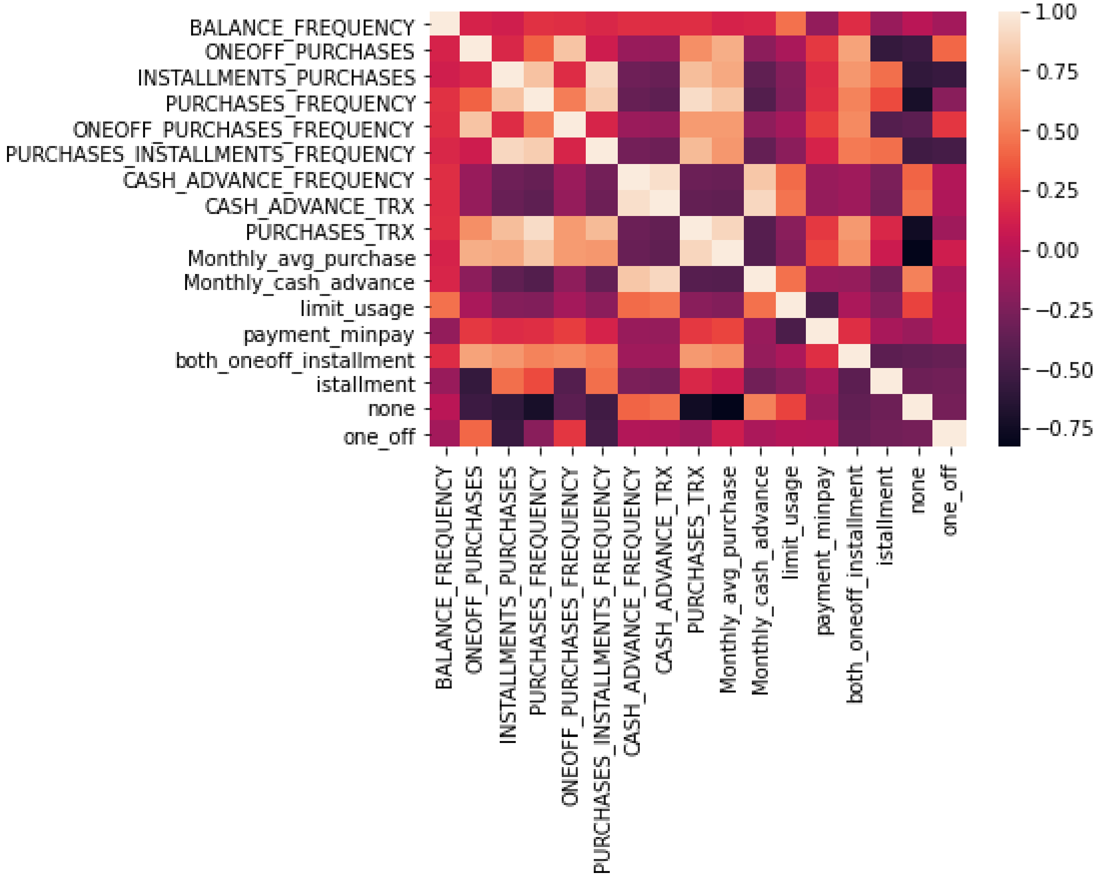
In [205]:
```
cr_dummy.describe()
```

Out[205]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY
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 [206]:
```
sns.heatmap(cr_dummy.corr())
```

Out[206]: <AxesSubplot:>



- 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.

d). Standardrizing data

- To put data on the same scale

In [207]:
```
from sklearn.preprocessing import  StandardScaler
```

In [249]:
```
sc=StandardScaler()
```

In [251]:
```
cr_scaled=sc.fit_transform(cr_dummy)
```

e) Applying PCA

In [210]:
```
from sklearn.decomposition import PCA
```

In [211]:
```
var_ratio={}
for n in range(4,15):
    pc=PCA(n_components=n)
    cr_pca=pc.fit(cr_scaled)
    var_ratio[n]=sum(cr_pca.explained_variance_ratio_)
```

In [212]:
```
pc=PCA(n_components=5)
```

In [213]:
```
p=pc.fit(cr_scaled)
```

In [214]:
```
cr_scaled.shape
```

Out[214]:
```
(8950, 17)
```

In [215]:
```
p.explained_variance_
```

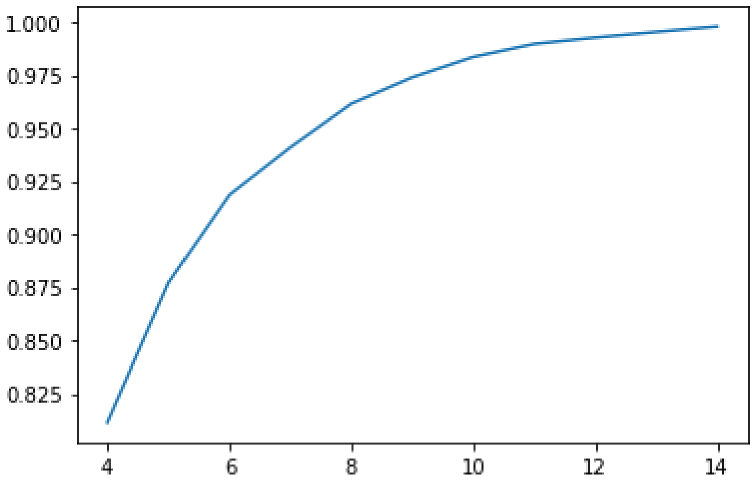
Out[215]:
```
array([6.83574755, 3.07030693, 2.50427698, 1.38746289, 1.1138166 ])
```

In [216]:
```
var_ratio
```

Out[216]:
```
{4: 0.8115442762351257,
 5: 0.8770555795291428,
 6: 0.918649244351261,
 7: 0.9410925256030138,
 8: 0.9616114053683061,
 9: 0.9739787081990643,
 10: 0.9835896584630709,
 11: 0.9897248107341962,
 12: 0.9927550009135233,
 13: 0.9953907562385427,
 14: 0.9979616898169593}
```

In [217]:
```
pd.Series(var_ratio).plot()
```

Out[217]: <AxesSubplot:>



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

In [218]:
```
pc_final=PCA(n_components=5).fit(cr_scaled)

reduced_cr=pc_final.fit_transform(cr_scaled)
```

In [219]:
```
dd=pd.DataFrame(reduced_cr)
```

In [220]:
```
dd.shape
dd
```

Out[220]:

	0	1	2	3	4
0	-0.242841	-2.759668	0.343061	-0.417359	-0.007100
1	-3.975652	0.144625	-0.542989	1.023832	-0.428929
2	1.287396	1.508938	2.709966	-1.892252	0.010809
3	-1.047613	0.673103	2.501794	-1.306784	0.761348
4	-1.451586	-0.176336	2.286074	-1.624896	-0.561969
...	...	...	...	...	...
8945	1.779193	-2.618043	-0.737105	-0.076058	0.619959
8946	1.614080	-2.657311	-0.934198	-0.943577	0.307854
8947	1.156359	-2.798864	-0.536306	-0.681240	0.325711
8948	-3.249950	-1.015633	0.473838	0.815603	-1.125202
8949	0.238814	2.223378	1.839580	-1.107814	1.745657

8950 rows × 5 columns

In [221]:
```
col_list=cr_dummy.columns
```

In [222]:
```
col_list
```

Out[222]:
```
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')
```

In [253]:
```
pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in range(5)],index=col_list)
```

Out[253]:

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

In [224]:
```
# Factor Analysis : variance explained by each component-
pd.Series(pc_final.explained_variance_ratio_,index=['PC_'+ str(i) for i in range(5)])
```

Out[224]:
```
PC_0    0.402058
PC_1    0.180586
PC_2    0.147294
PC_3    0.081606
PC_4    0.065511
dtype: float64
```

In [225]:
```
type(cr_pca)
```

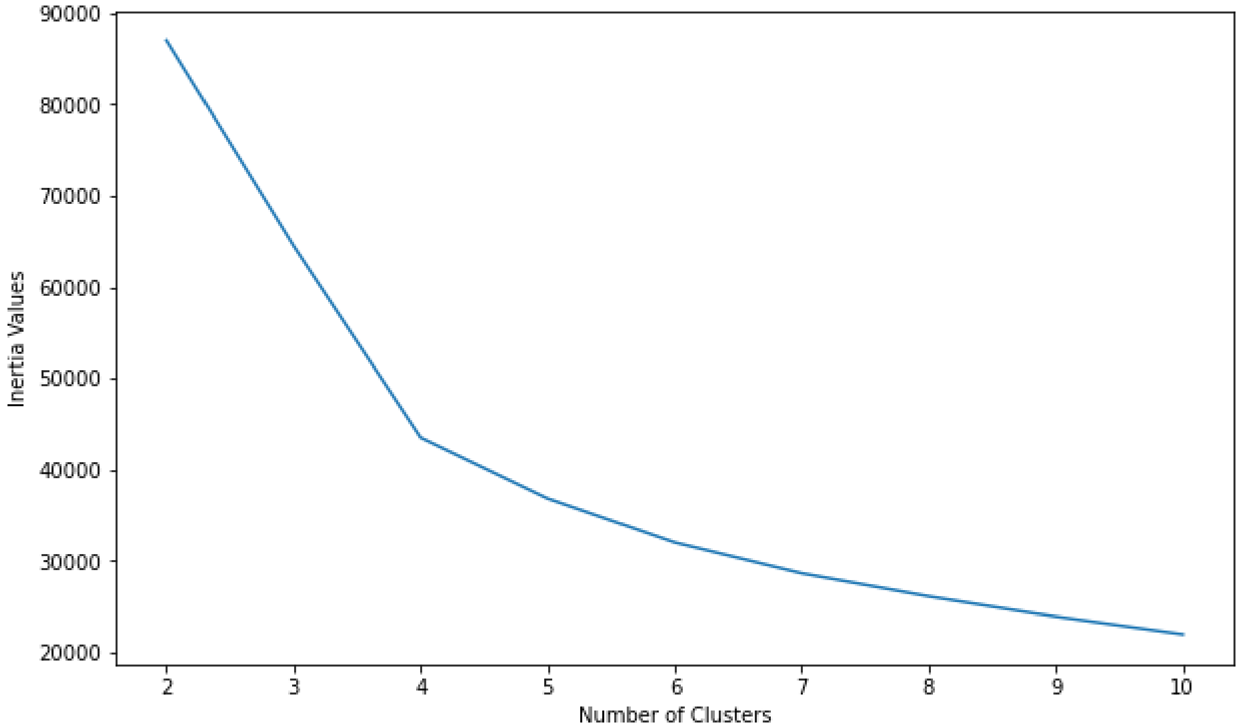
Out[225]:
```
sklearn.decomposition._pca.PCA
```

f). Clustering

Elbow Method for optimal value of k in KMeans

In [226]:
```
from sklearn.cluster import KMeans
from sklearn import metrics

def KMeans_Algorithm(dataset, n):
    clustering_KMeans = KMeans(n_clusters= n,init='k-means++', max_iter=300, random_state=0, algorithm = "elkan")
    clustering_KMeans.fit(dataset)

    # create data frame to store centroids
    centroids  = clustering_KMeans.cluster_centers_

    # add cluster label for each data point
    label = clustering_KMeans.labels_
    credit["label"] = label

    # evaluation metrics for clustering - inertia
    inertia = clustering_KMeans.inertia_

    return inertia, label, centroids

X2=dd
X2_inertia_values = []

fig2 = plt.figure(figsize=(20,20))
for i in range (2,11):
    X2_inertia, X2_label, X2_centroids = KMeans_Algorithm(X2, i)
    X2_inertia_values.append(X2_inertia)


# plot inertia values against number of clusters
plt.figure(figsize = (10 ,6))
plt.plot(np.arange(2, 11) , X2_inertia_values , '-')
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia Values")
```

Out[226]:
```
Text(0, 0.5, 'Inertia Values')

<Figure size 1440x1440 with 0 Axes>
```



To determine the optimal number of clusters, we have to select the value of k at the “elbow” ie the point after which the distortion/inertia start decreasing in a linear fashion. Thus for the given data, we conclude that the optimal number of clusters for the data is 4.

K-Means Clustering

In [227]:
```
km_4=KMeans(n_clusters=4,random_state=123)
```

In [228]:
```
km_4.fit(reduced_cr)
km_4.labels_
```

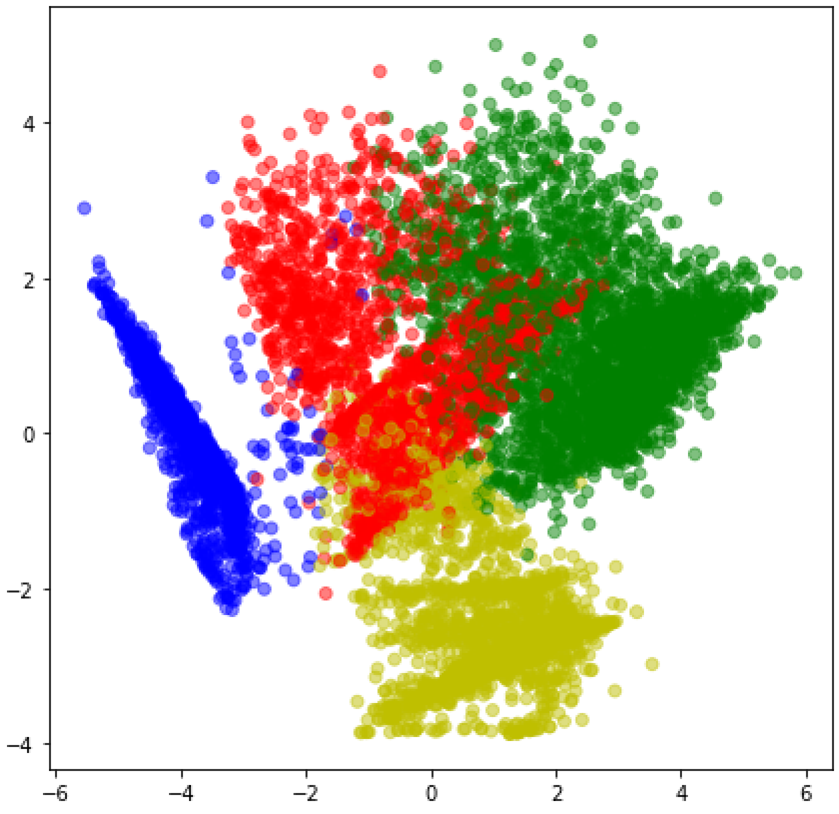
Out[228]:
```
array([3, 1, 0, ..., 3, 1, 0])
```

In [229]:
```
pd.Series(km_4.labels_).value_counts()
```

Out[229]:
```
2    2758
3    2228
1    2090
0    1874
dtype: int64
```

In [230]:
```
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=0.5)
```

Out[230]:
```
<matplotlib.collections.PathCollection at 0x12e09186670>
```



In [231]:
```
cr_dummy.dtypes
```

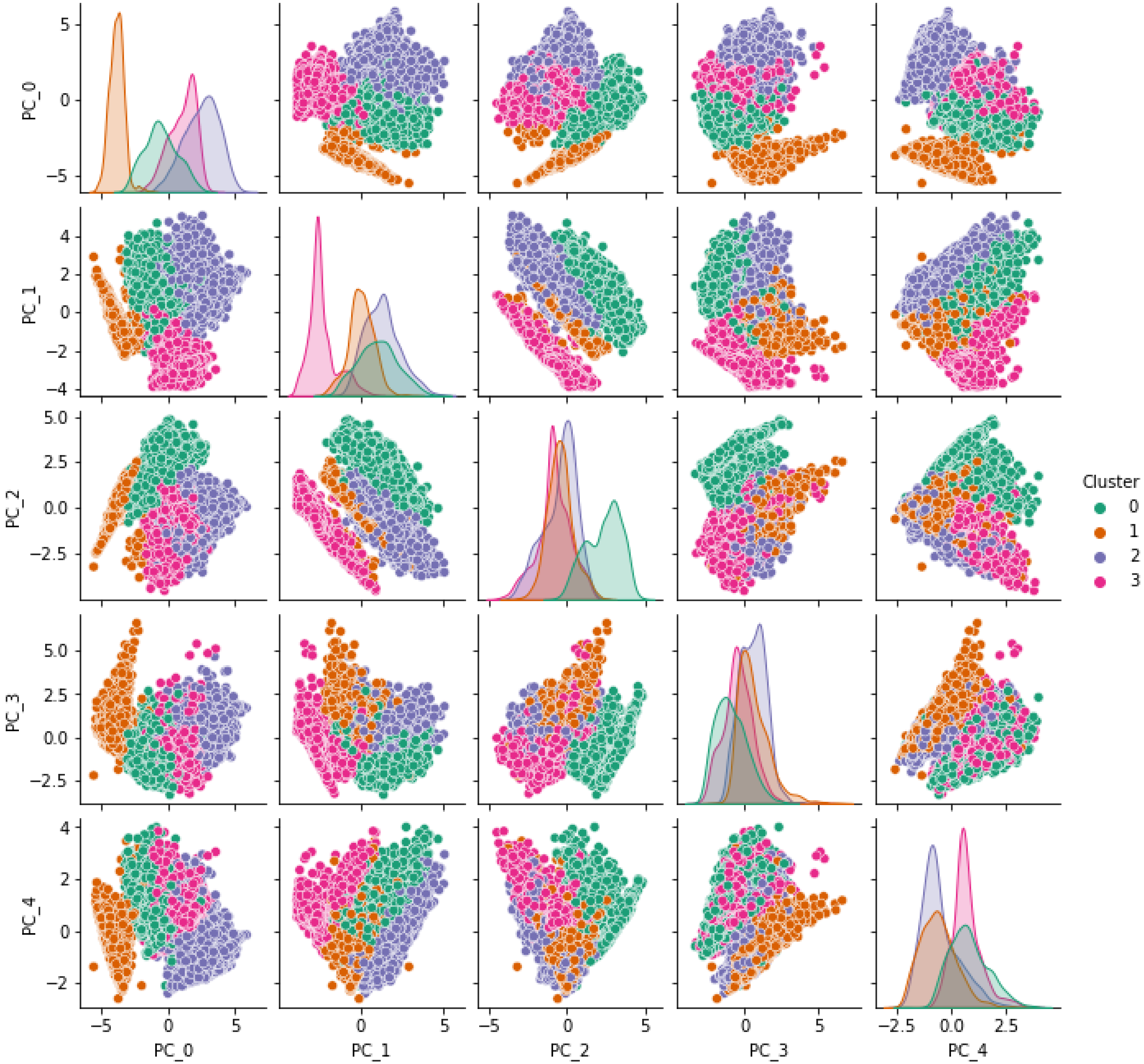
Out[231]:
```
BALANCE_FREQUENCY                    float64
ONEOFF_PURCHASES                     float64
INSTALLMENTS_PURCHASES               float64
PURCHASES_FREQUENCY                  float64
ONEOFF_PURCHASES_FREQUENCY           float64
PURCHASES_INSTALLMENTS_FREQUENCY     float64
CASH_ADVANCE_FREQUENCY               float64
CASH_ADVANCE_TRX                     float64
PURCHASES_TRX                        float64
Monthly_avg_purchase                 float64
Monthly_cash_advance                 float64
limit_usage                          float64
payment_minpay                       float64
both_oneoff_installment                uint8
istallment                             uint8
none                                   uint8
one_off                                uint8
dtype: object
```

In [232]:
```
df_pair_plot=pd.DataFrame(reduced_cr,columns=['PC_' +str(i) for i in range(5)])
```

In [233]:
```
df_pair_plot['Cluster']=km_4.labels_
```

In [234]:
```
#pairwise relationship of components on the data
sns.pairplot(df_pair_plot,hue='Cluster', palette= 'Dark2', diag_kind='kde',height=1.85)
```

Out[234]:
```
<seaborn.axisgrid.PairGrid at 0x12e091a1700>
```



It shows that first two components are able to indentify clusters

In [235]:
```
# Key performace variable selection . here i am dropping varibales which are used in derving new KPI
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']
```

In [236]:
```
cr_pre.describe()
```

Out[236]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY
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 [237]:
```
# Conactenating 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)
```

In [238]:
```
cluster_df_4.head()
```

Out[238]:

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

In [239]:
```
# Mean value gives a good indication of the distribution of data. So we are finding mean value for each variable for eacl
cluster_4=cluster_df_4.groupby('Cluster_4')\
.apply(lambda x: x[col_kpi].mean()).T
cluster_4
```

Out[239]:

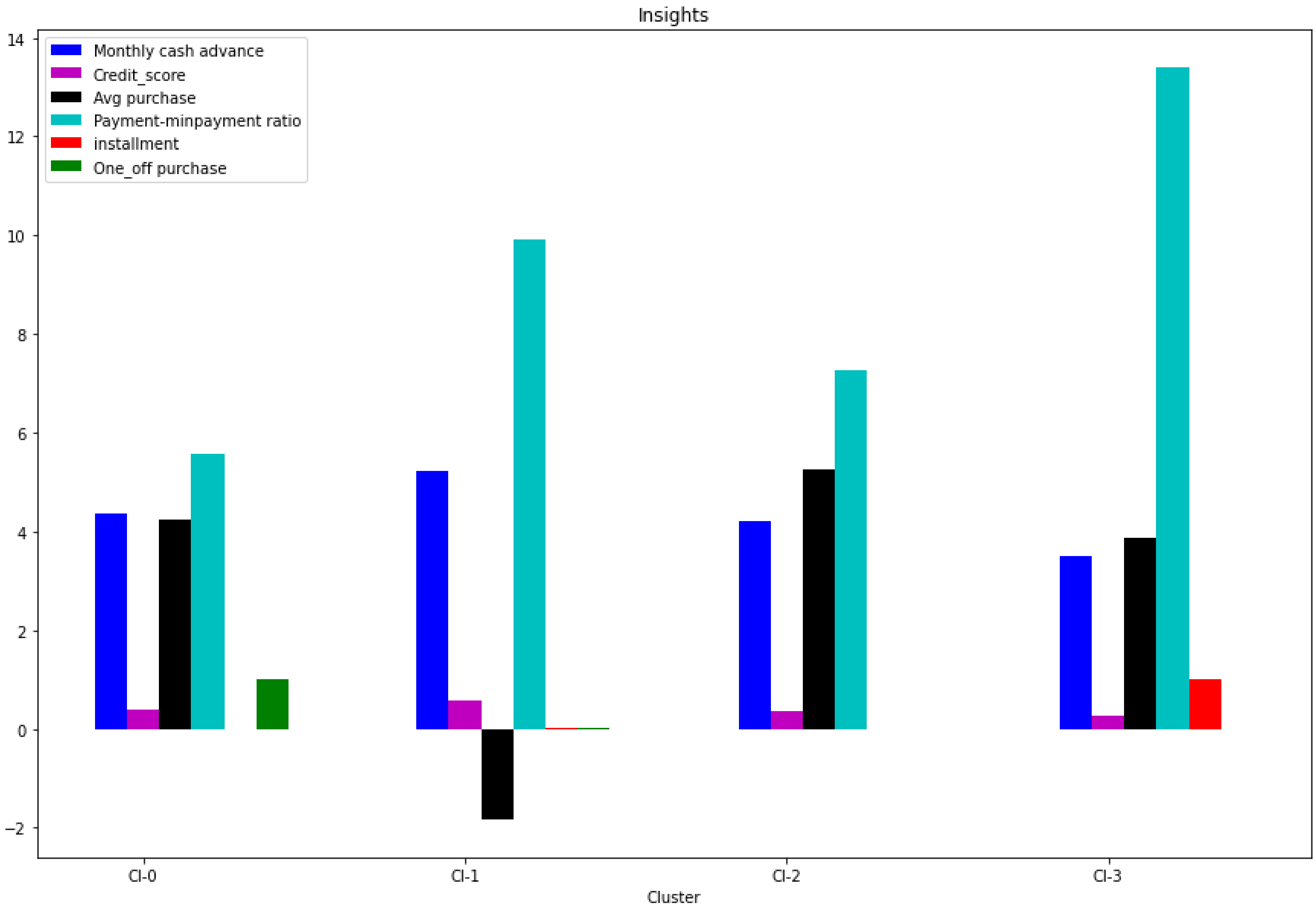
Cluster_4	0	1	2	3
PURCHASES_TRX	7.118997	0.045933	33.125453	12.053860
Monthly_avg_purchase	69.758276	0.159337	193.696083	47.573598
Monthly_cash_advance	77.843485	186.298043	67.620006	33.489846
limit_usage	0.378727	0.576217	0.354487	0.264275
CASH_ADVANCE_TRX	2.864995	6.552632	2.807107	1.019300
payment_minpay	5.561421	9.927979	7.268605	13.402660
both_oneoff_installment	0.003735	0.002392	1.000000	0.001795
istallment	0.000000	0.017225	0.000000	0.998205
one_off	0.996265	0.003349	0.000000	0.000000
none	0.000000	0.977033	0.000000	0.000000
CREDIT_LIMIT	4512.905630	4055.582137	5750.015565	3335.697210

In [240]:

```
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()
```

Out[240]:

```
<matplotlib.legend.Legend at 0x12e17ddfe80>
```



Findings through clustering is validating Insights dervied from KPI. (as shown above in Insights from KPI

In [242]:
```
# Percentage of each cluster in the total customer base
s=cluster_df_4.groupby('Cluster_4').apply(lambda x: x['Cluster_4'].value_counts())

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),'\n')
```

```
Cluster -4 

   Size  Percentage
0  1874   20.938547
1  2090   23.351955
2  2758   30.815642
3  2228   24.893855
```

** Insights**

Clusters are clearly distinguishing behavior within customers

- CLUSTER 0 customers are doing maximum One_Off transactions and least payment ratio. * **This group is about 21% 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 2 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** *
- CLUSTER 3 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** *

Checking performance metrics for Kmeans

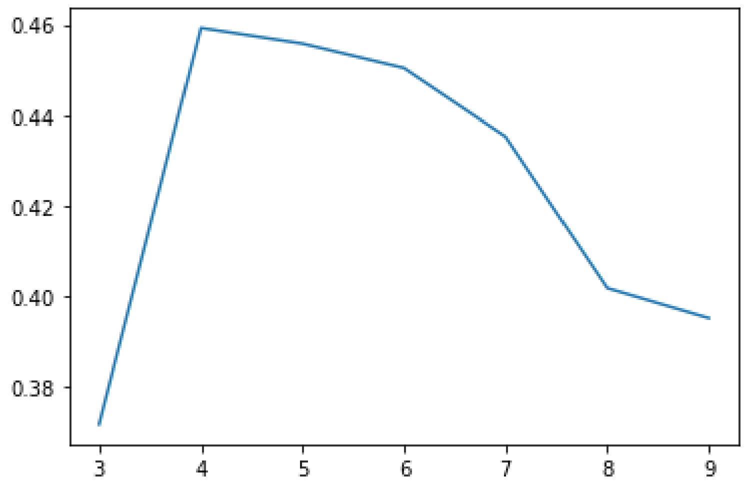
- validating performance with Silhouette score

In [244]:
```
from sklearn.metrics import silhouette_score
```

In [245]:
```
score={}
for n in range(3,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_cr)
    score[n]=silhouette_score(reduced_cr,km_score.labels_)
```

In [246]:
```
pd.Series(score).plot()
```

Out[246]:
```
<AxesSubplot:>
```



Performance metrics also suggest that K-means with 4 cluster is able to show

distinguished characteristics of each cluster.

Marketing Strategy Suggested:

i) Group 0:

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

ii) Group 1:

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

iii) 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 can lower down interest rate -- Can be given premium card /loyality cards to increase transactions

iv) Group 3:

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

END