Credit Card Customer Segmentation using K-Means Clustering

Importing Python Libraries

In [81]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import scipy.stats as stats
#import pandas_profiling

%matplotlib inline
plt.rcParams['figure.figsize'] = 10, 7.5
plt.rcParams['axes.grid'] = True

from matplotlib.backends.backend_pdf import PdfPages
from sklearn.cluster import KMeans

# center and scale the data
from sklearn.preprocessing import StandardScaler
```

Dataset Description

The are total 18 columns in the dataset given

Link to the dataset: https://www.kaggle.com/arjunbhasin2013/ccdata (https://www.kaggle.com/arjunbhasin2013/ccdata)

Following is the Data Dictionary for Credit Card dataset:

CUST_ID: Identification of Credit Card holder (Categorical) BALANCE: Balance amount left in their account to make purchases BALANCE_FREQUENCY: How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated) PURCHASES: Amount of purchases made from account ONEOFF_PURCHASES: Maximum purchase amount done in one-go INSTALLMENTS_PURCHASES: Amount of purchase done in installment CASH_ADVANCE: Cash in advance given by the user PURCHASES_FREQUENCY: How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased) ONEOFFPURCHASESFREQUENCY: How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased) PURCHASESINSTALLMENTSFREQUENCY: How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done) CASHADVANCEFREQUENCY: How frequently the cash in advance being paid CASHADVANCETRX: Number of Transactions made with "Cash in Advanced" PURCHASES_TRX: Number of purchase transactions made CREDIT_LIMIT: Limit of Credit Card for user PAYMENTS: Amount of Payment done by user MINIMUM_PAYMENTS: Minimum amount of payments made by user PRCFULLPAYMENT: Percent of full payment paid by user TENURE: Tenure of credit card service for user Two columns namely "CREDIT_LIMIT" and "MINIMUM_PAYMENTS" are having NULL values. "CREDIT_LIMIT" - NULL will be filled with median and for "MINIMUM_PAYMENTS" - NULL will be filled with ZERO.

In [2]:

```
credit= pd.read_csv(r'C:\Users\user\Desktop\BTECH\MAJOR PROJECT\FunalProject-edx\CC_GENERAL
```

In [3]:

```
credit.head(12)
```

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTA
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	
5	C10006	1809.828751	1.000000	1333.28	0.00	
6	C10007	627.260806	1.000000	7091.01	6402.63	
7	C10008	1823.652743	1.000000	436.20	0.00	
8	C10009	1014.926473	1.000000	861.49	661.49	
9	C10010	152.225975	0.545455	1281.60	1281.60	
10	C10011	1293.124939	1.000000	920.12	0.00	
11	C10012	630.794744	0.818182	1492.18	1492.18	

In [4]:

```
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
CUST_ID                             8950 non-null object
BALANCE                             8950 non-null float64
BALANCE_FREQUENCY                   8950 non-null float64
PURCHASES                           8950 non-null float64
ONEOFF_PURCHASES                    8950 non-null float64
INSTALLMENTS_PURCHASES              8950 non-null float64
CASH_ADVANCE                        8950 non-null float64
PURCHASES_FREQUENCY                 8950 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8950 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8950 non-null float64
CASH_ADVANCE_FREQUENCY              8950 non-null float64
CASH_ADVANCE_TRX                    8950 non-null int64
PURCHASES_TRX                       8950 non-null int64
CREDIT_LIMIT                        8949 non-null float64
PAYMENTS                            8950 non-null float64
MINIMUM_PAYMENTS                    8637 non-null float64
PRC_FULL_PAYMENT                    8950 non-null float64
TENURE                              8950 non-null int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [5]:

```
credit.columns
```

Out[5]:

```
Index(['CUST_ID', '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'],
      dtype='object')
```

In [6]:

```
credit.describe().T
```

Out[6]:

	count	mean	std	min	
BALANCE	8950.0	1564.474828	2081.531879	0.000000	128.28
BALANCE_FREQUENCY	8950.0	0.877271	0.236904	0.000000	0.88
PURCHASES	8950.0	1003.204834	2136.634782	0.000000	39.63
ONEOFF_PURCHASES	8950.0	592.437371	1659.887917	0.000000	0.00
INSTALLMENTS_PURCHASES	8950.0	411.067645	904.338115	0.000000	0.00
CASH_ADVANCE	8950.0	978.871112	2097.163877	0.000000	0.00
PURCHASES_FREQUENCY	8950.0	0.490351	0.401371	0.000000	0.08
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.202458	0.298336	0.000000	0.00
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.364437	0.397448	0.000000	0.00
CASH_ADVANCE_FREQUENCY	8950.0	0.135144	0.200121	0.000000	0.00
CASH_ADVANCE_TRX	8950.0	3.248827	6.824647	0.000000	0.00
PURCHASES_TRX	8950.0	14.709832	24.857649	0.000000	1.00
CREDIT_LIMIT	8949.0	4494.449450	3638.815725	50.000000	1600.00
PAYMENTS	8950.0	1733.143852	2895.063757	0.000000	383.27
MINIMUM_PAYMENTS	8637.0	864.206542	2372.446607	0.019163	169.12
PRC_FULL_PAYMENT	8950.0	0.153715	0.292499	0.000000	0.00
TENURE	8950.0	11.517318	1.338331	6.000000	12.00

NULL Handling -

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.isnull().sum()
```

Out[9]:

```
CUST_ID                               0
BALANCE                               0
BALANCE_FREQUENCY                     0
PURCHASES                             0
ONEOFF_PURCHASES                      0
INSTALLMENTS_PURCHASES                0
CASH_ADVANCE                          0
PURCHASES_FREQUENCY                   0
ONEOFF_PURCHASES_FREQUENCY            0
PURCHASES_INSTALLMENTS_FREQUENCY      0
CASH_ADVANCE_FREQUENCY                0
CASH_ADVANCE_TRX                      0
PURCHASES_TRX                         0
CREDIT_LIMIT                          1
PAYMENTS                              0
MINIMUM_PAYMENTS                    313
PRC_FULL_PAYMENT                      0
TENURE                                0
dtype: int64
```

In [10]:

```
credit['CREDIT_LIMIT'].isnull().value_counts()
```

Out[10]:

```
False    8949
True        1
Name: CREDIT_LIMIT, dtype: int64
```

In [11]:

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

Out[11]:

```
False    8637
True      313
Name: MINIMUM_PAYMENTS, dtype: int64
```

In [12]:

```
# For CREDIT_LIMIT - We will fill the NULL with the median of CREDIT_LIMIT
credit['CREDIT_LIMIT'].fillna(value=credit['CREDIT_LIMIT'].median(), inplace = True)
```

In [13]:

```
# For MINIMUM_PAYMENTS we will fill NULL with ZERO
credit['MINIMUM_PAYMENTS'] = credit['MINIMUM_PAYMENTS'].fillna(0)
```

In [14]:

```
credit.isnull().sum()
```

Out[14]:

```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        0
PAYMENTS                            0
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

Now we drop CUST_ID column, then normalize the input values using StandardScaler().

In [17]:

```
# drop ID col
data= credit.drop('CUST_ID', 1)

data.head()
```

Out[17]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_F
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

In [18]:

```
# normalize values
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
data_scaled.shape
print(data_scaled)
```

```
[[-0.73198937 -0.24943448 -0.42489974 ... -0.2973097  -0.52555097
   0.36067954]
 [ 0.78696085  0.13432467 -0.46955188 ...  0.10204243  0.2342269
   0.36067954]
 [ 0.44713513  0.51808382 -0.10766823 ... -0.08848934 -0.52555097
   0.36067954]
 ...
 [-0.7403981  -0.18547673 -0.40196519 ... -0.32175099  0.32919999
  -4.12276757]
 [-0.74517423 -0.18547673 -0.46955188 ... -0.33316552  0.32919999
  -4.12276757]
 [-0.57257511 -0.88903307  0.04214581 ... -0.31923775 -0.52555097
  -4.12276757]]
```

In [19]:

```
data_scaled.shape
```

Out[19]:

```
(8950, 17)
```

In [20]:

```
data_imputed = pd.DataFrame(data_scaled, columns=data.columns)
data_imputed.head()
```

Out[20]:

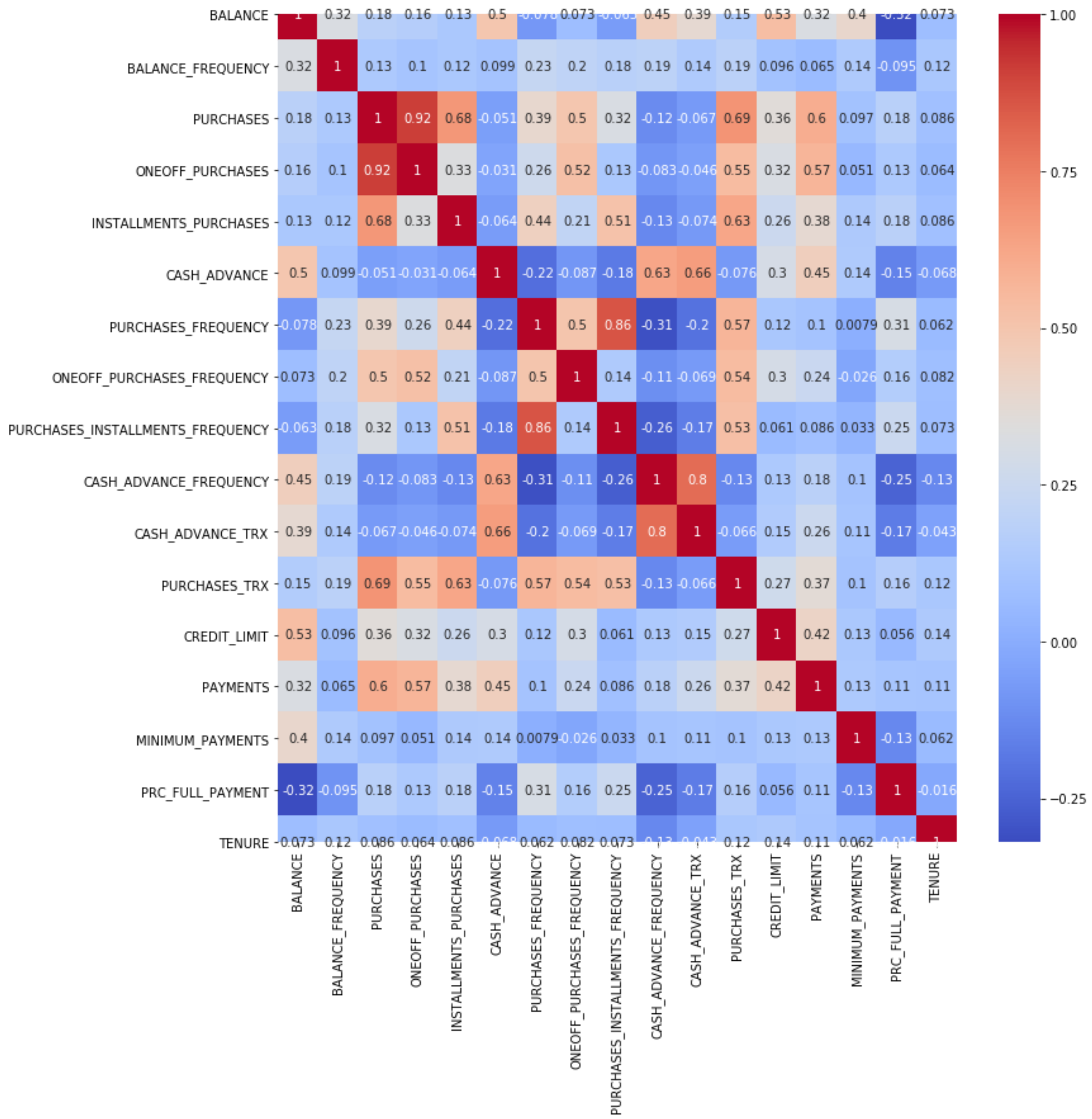
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
0	-0.731989	-0.249434	-0.424900	-0.356934	
1	0.786961	0.134325	-0.469552	-0.356934	
2	0.447135	0.518084	-0.107668	0.108889	
3	0.049099	-1.016953	0.232058	0.546189	
4	-0.358775	0.518084	-0.462063	-0.347294	

In [21]:

```
plt.figure(figsize = (12, 12))
sns.heatmap(data_imputed.corr(), annot=True, cmap='coolwarm',
            xticklabels=data_imputed.columns,
            yticklabels=data_imputed.columns)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe96bdb1e48>
```

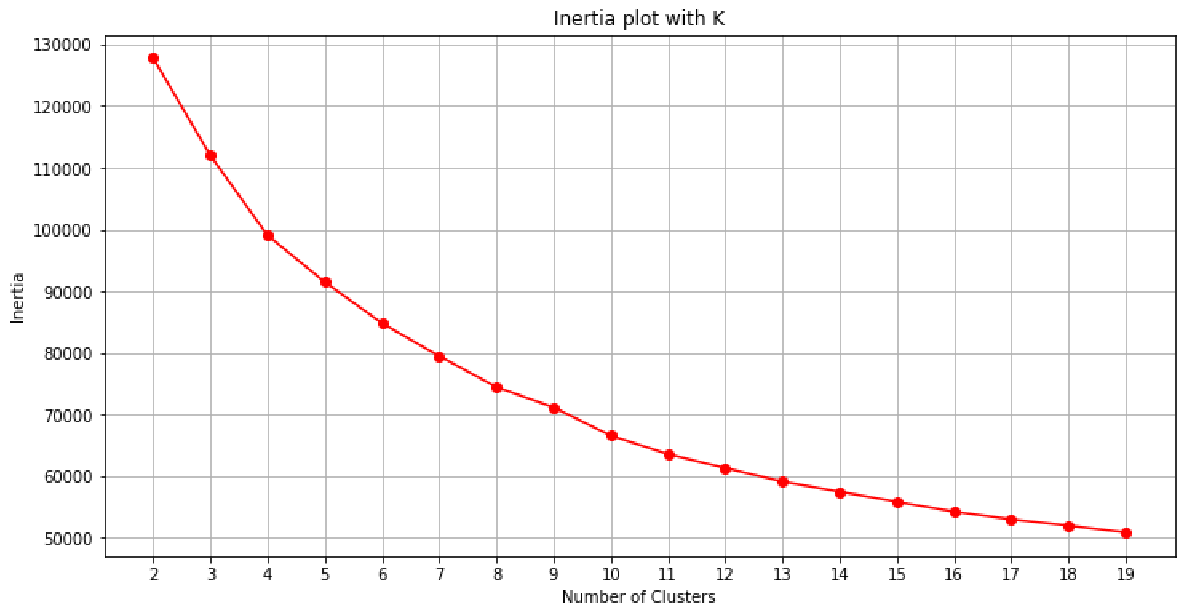


In [87]:

```
def inertia_plot(clust, X, start = 2, stop = 20):
    inertia = []
    for x in range(start,stop):
        km = clust(n_clusters = x)
        labels = km.fit_predict(X)
        inertia.append(km.inertia_)
    plt.figure(figsize = (12,6))
    plt.plot(range(start,stop), inertia, marker = 'o', color = 'red')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Inertia')
    plt.title('Inertia plot with K')
    plt.xticks(list(range(start, stop)))
    plt.show()
```

In [88]:

```
inertia_plot(KMeans, data_imputed)
```



In [39]:

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

In [40]:

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

In [41]:

```
# function for defining purchase type
#4 types of purchase behavior
def purchaagetyp(credit):
    if ((credit.ONEOFF_PURCHASES == 0) & (credit.INSTALLMENTS_PURCHASES == 0)):
        return 'NONE'
    if ((credit.ONEOFF_PURCHASES > 0) & (credit.INSTALLMENTS_PURCHASES == 0)):
        return 'ONE_OFF'
    if ((credit.ONEOFF_PURCHASES > 0) & (credit.INSTALLMENTS_PURCHASES > 0)):
        return 'BOTH_ONEOFF_INSTALLMENT'
    if ((credit.ONEOFF_PURCHASES == 0) & (credit.INSTALLMENTS_PURCHASES > 0)):
        return 'INSTALLMENTS'
```

In [42]:

```
#Purchase by Type
credit['PURCHASE_TYPE'] = credit.apply(purchaagetyp, axis=1)
```

In [75]:

```
#LIMIT USAGE (Credit Score - Lower implies customers are maintaining their balance properly
credit['LIMIT_USAGE'] = credit.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT'],axis =1)
```

In [45]:

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

Out[45]:

```
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
MNTHLY_AVG_PURCHASE                 False
MONTHLY_AVG_CASH_ADVANCE            False
PURCHASE_TYPE                       False
PAYMENT_MINPAYMENT                  False
dtype: bool
```

In [46]:

```
credit=credit.round(2)
```

In [47]:

```
credit.head()
```

Out[47]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLI
0	C10001	40.90	0.82	95.40	0.00	
1	C10002	3202.47	0.91	0.00	0.00	
2	C10003	2495.15	1.00	773.17	773.17	
3	C10004	1666.67	0.64	1499.00	1499.00	
4	C10005	817.71	1.00	16.00	16.00	

5 rows × 22 columns

In [48]:

```
credit.groupby('PURCHASE_TYPE').apply(lambda x: np.mean(x['PAYMENT_MINPAYMENT']))
```

Out[48]:

```
PURCHASE_TYPE
BOTH_ONEOFF_INSTALLMENT    10.067787
INSTALLMENTS               20.050496
NONE                       15.328521
ONE_OFF                    41.136110
dtype: float64
```

Insights

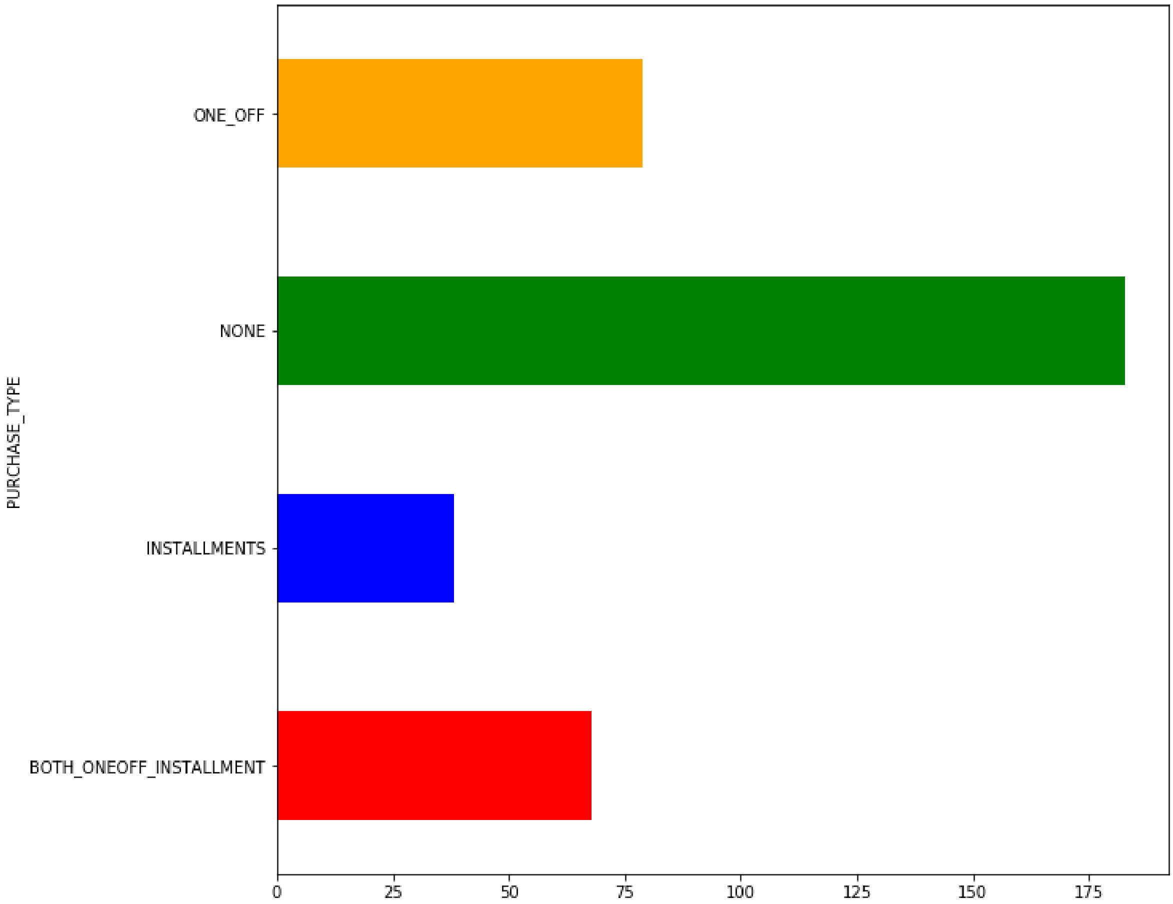
```
1 Customers with installment payments are paying dues
2 Customers who do not do ONOFF or INSTALLMENTS take more cash advance
3 Customers with installment purchases have good credit score
```

In [73]:

```
credit.groupby('PURCHASE_TYPE').apply(lambda x : np.mean(x['MONTHLY_AVG_CASH_ADVANCE'])).pl
```

Out[73]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe970af2d48>
```

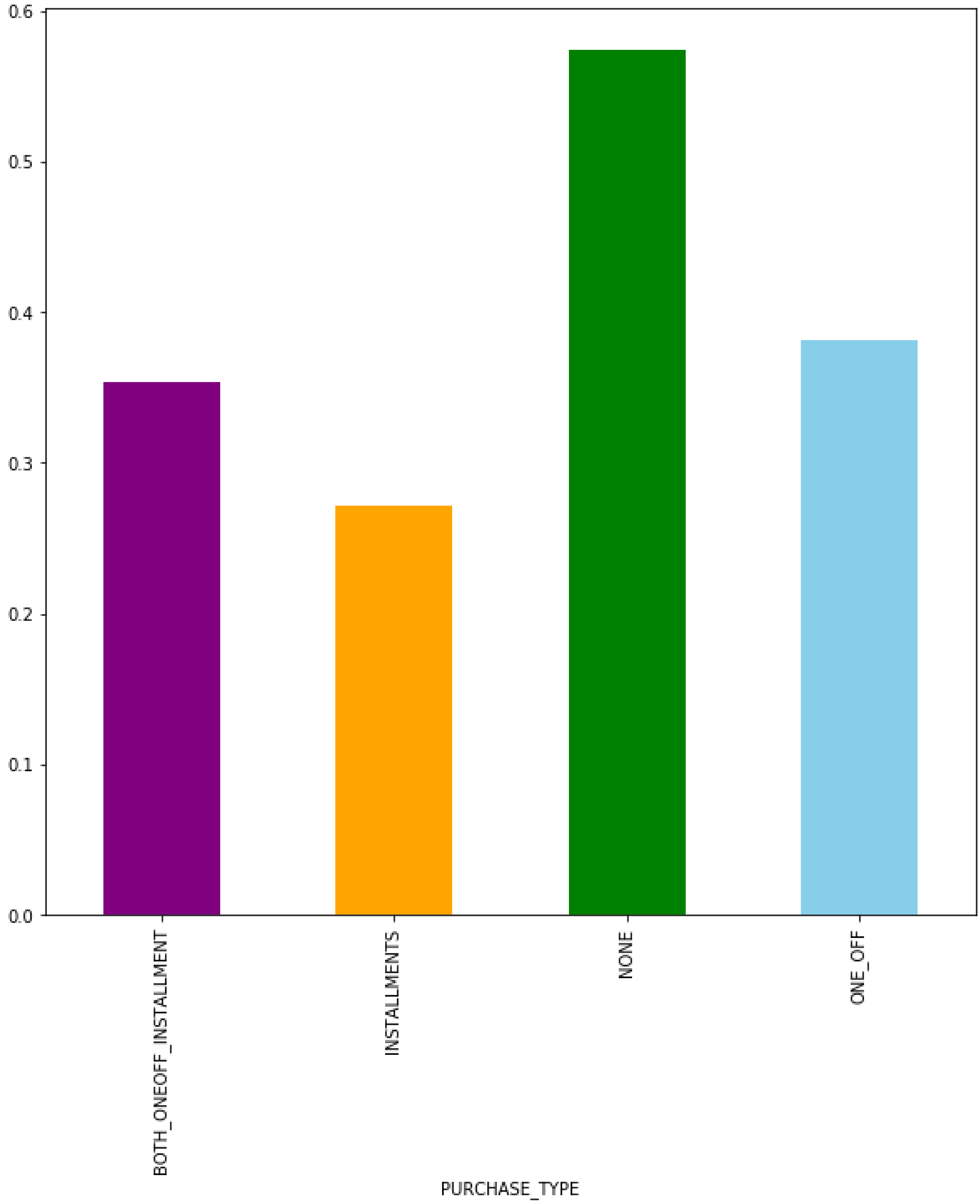


In [79]:

```
credit.groupby('PURCHASE_TYPE').apply(lambda x : np.mean(x['LIMIT_USAGE'])).plot.bar(grid =
```

Out[79]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe97207d048>
```



Clustering Using K-Means

For 6 cluster Solution behavior -

In [82]:

```
from sklearn.cluster import KMeans
```

In [83]:

```
km_6=KMeans(n_clusters=6,random_state=123)
```

In [84]:

```
km_6.fit(data_final)
km_6.labels_
```

Out[84]:

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

In [85]:

```
pd.Series(km_6.labels_).value_counts()
```

Out[85]:

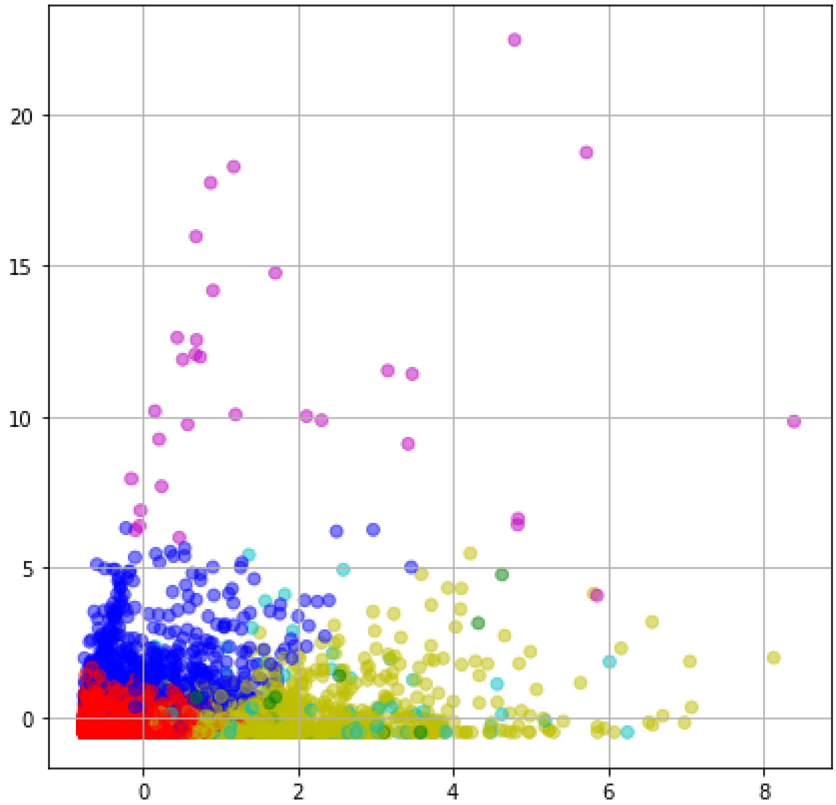
```
0    6124
1    1481
3    1145
4     132
2      37
5      31
dtype: int64
```

In [86]:

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

Out[86]:

```
<matplotlib.collections.PathCollection at 0xe972530448>
```



For 6 cluster Solution behavior -

In [24]:

```
# select best columns
best_cols = ["BALANCE", "PURCHASES", "CASH_ADVANCE", "CREDIT_LIMIT", "PAYMENTS", "MINIMUM_F

# dataframe with best columns
data_final = pd.DataFrame(data_imputed[best_cols])

print('New dataframe with best columns has just been created. Data shape: ' + str(data_fina
```

```
New dataframe with best columns has just been created. Data shape: (8950, 6)
```

In [25]:

```
alg = KMeans(n_clusters = 6, random_state=123)
label = alg.fit_predict(data_final)
pd.Series(label).value_counts()
```

Out[25]:

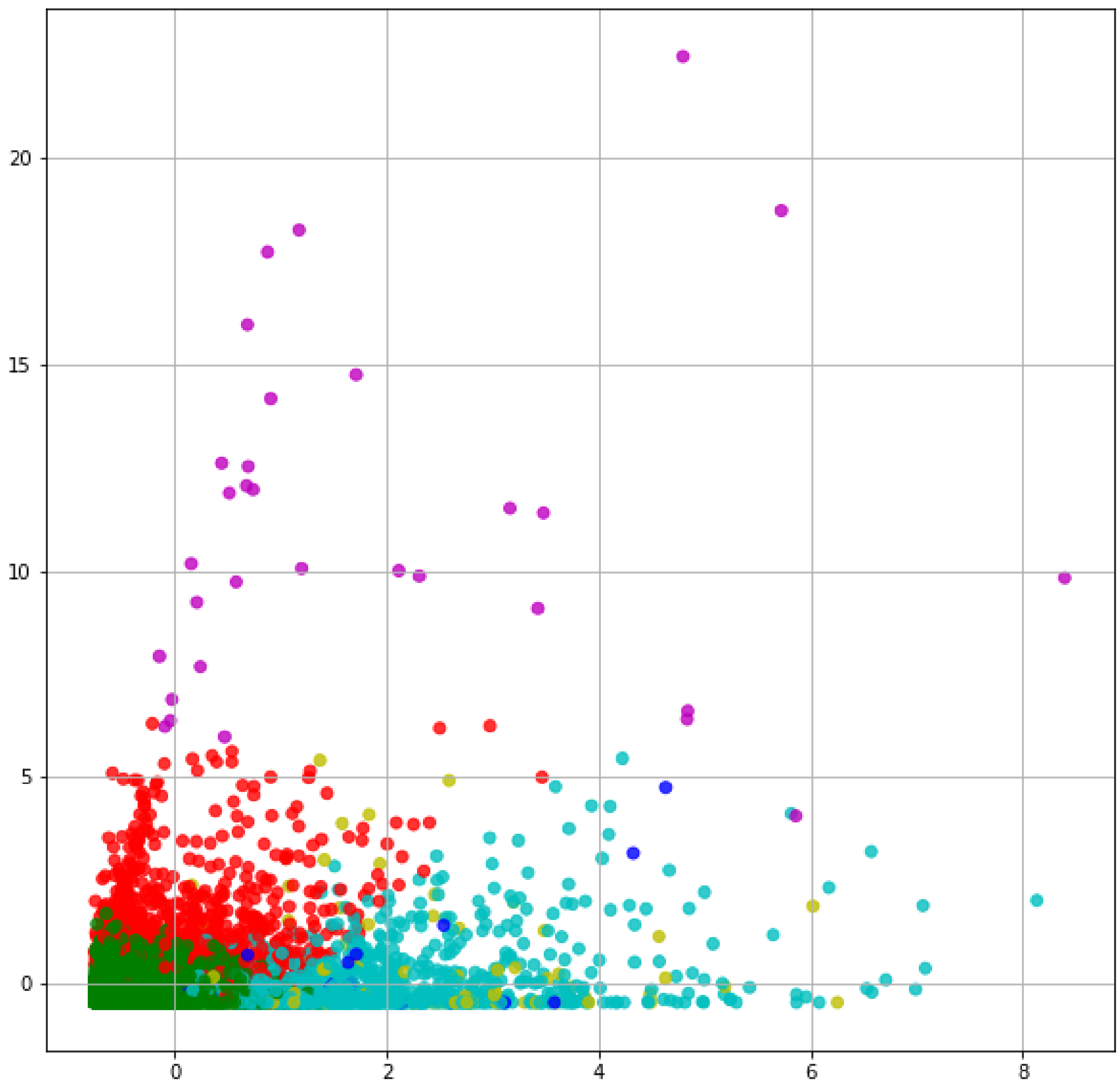
```
5    6124
0    1481
2    1145
1     132
4      37
3      31
dtype: int64
```

In [26]:

```
# apply KMeans clustering
#alg = KMeans(n_clusters = 6)
#label = alg.fit_predict(data_final)

# create a 'cluster' column
data_final['cluster'] = label
best_cols.append('cluster')

color_map={0:'r', 1:'y', 2:'c', 3:'m', 4:'b', 5:'g'}
label_color = [color_map[l] for l in label]
#color_map={0:'m', 1:'r', 2:'y', 3:'g', 4:'b', 5:'o'}
#label_color = [color_map[l] for l in label]
plt.figure(figsize=(10,10))
#plt.scatter(data_final.iloc[:,0], data_final.iloc[:,1],c=label,cmap='Oranges',alpha=1.0)
plt.scatter(data_final.iloc[:,0], data_final.iloc[:,1],c=label_color,alpha=0.8)
#plt.xlim(-1, 8)
#plt.ylim(-1, 10)

# make a Seaborn pairplot
#sns.pairplot(data_final[best_cols], hue='cluster')
#sns.scatterplot(data)
```

Out[26]:

```
<matplotlib.collections.PathCollection at 0xe96d432488>
```

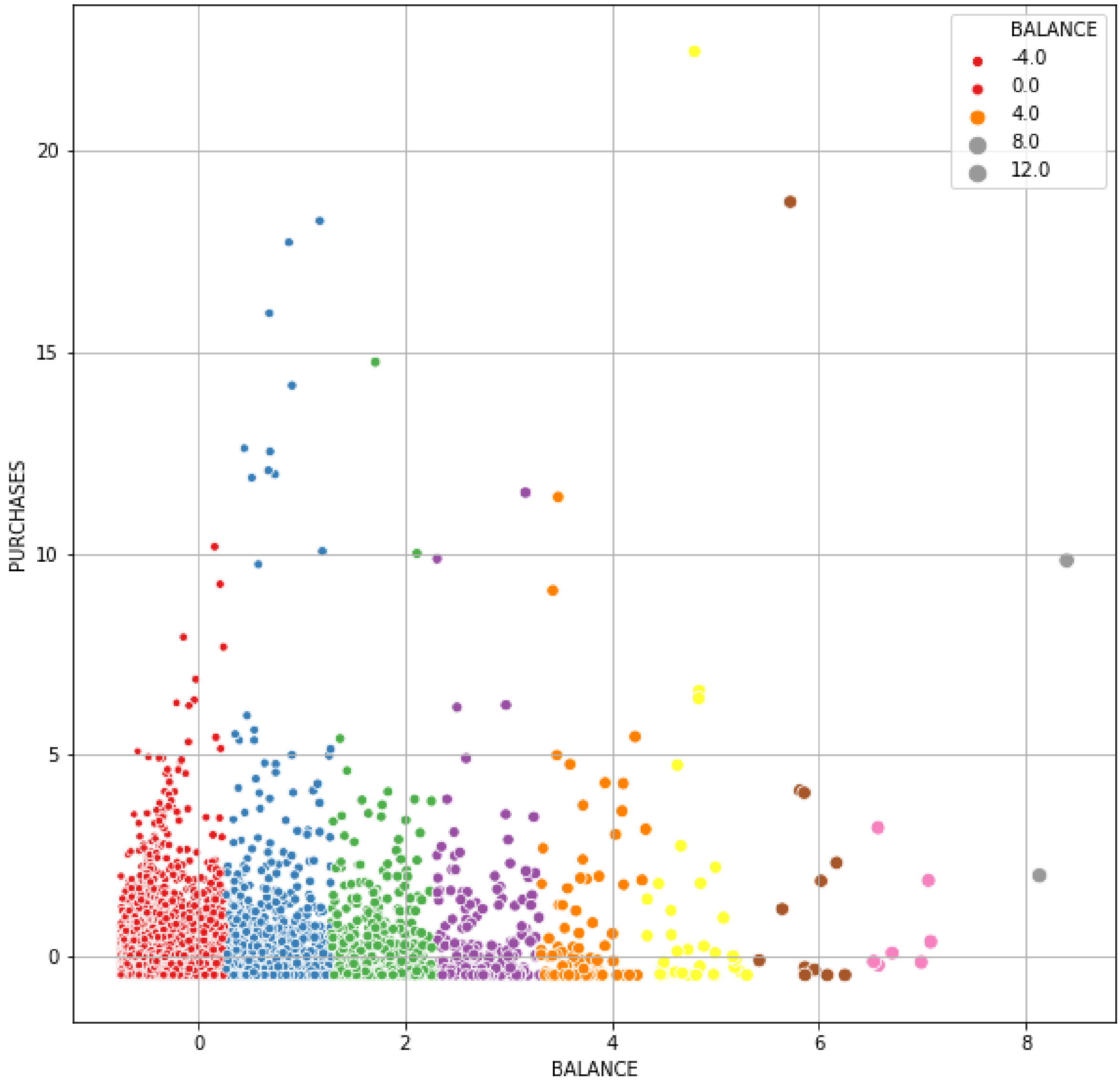


In [27]:

```
data_final['cluster'] = label
best_cols.append('cluster')
cmap = sns.cubehelix_palette(dark=.3, light=.8, as_cmap=True)
#sns.scatterplot(data_final.iloc[:,0], data_final.iloc[:,1])
plt.figure(figsize=(10,10))
sns.scatterplot(x=data_final.iloc[:,0], y=data_final.iloc[:,1],hue=data_final.iloc[:,0],pal
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe96bdf6c48>
```



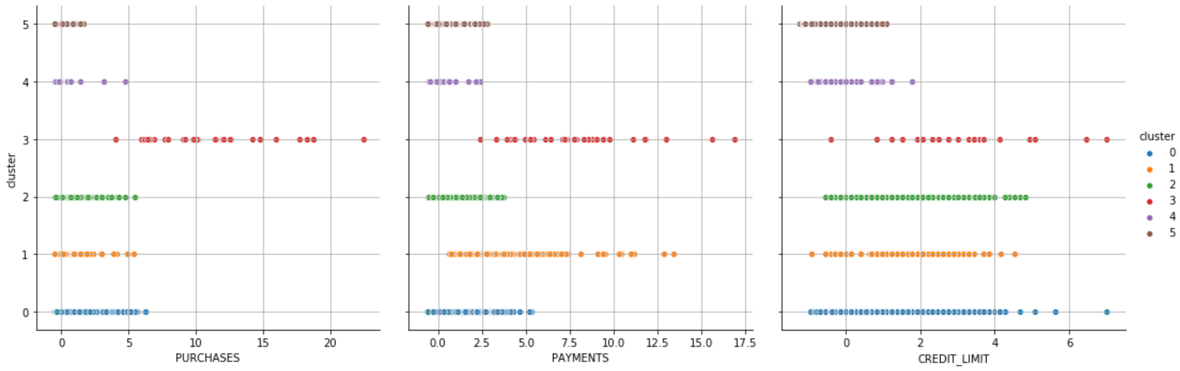
```
****************************************** THANK YOU ******************************
*********************
```

In [28]:

```python
sns.pairplot(data_final,hue='cluster', x_vars=['PURCHASES', 'PAYMENTS', 'CREDIT_LIMIT'],
            y_vars=['cluster'],
            height = 5, aspect=1)
```

Out[28]:

```
<seaborn.axisgrid.PairGrid at 0xe96be01fc8>
```



Cluster 0 (Blue) : This group of users, while having the highest number of users by far, is fairly frugal: they have lowest purchases, second lowest payments, and lowest credit limit. The bank would not make much profit from this group, so there should be some sorts of strategy to attract these people more.

Cluster 1 (Orange) : This group of users is very active in general: they have second highest purchases, third highest payments, and the most varied credit limit values. This type of credit card users is the type you should spend the least time and effort on, as they are already the ideal one.

Cluster 2 (Green) : The Big Spenders. This group is by far the most interesting to analyze, since they do not only have the highest number of purchases, highest payments, highest minimum payments, but the other features are also wildly varied in values.

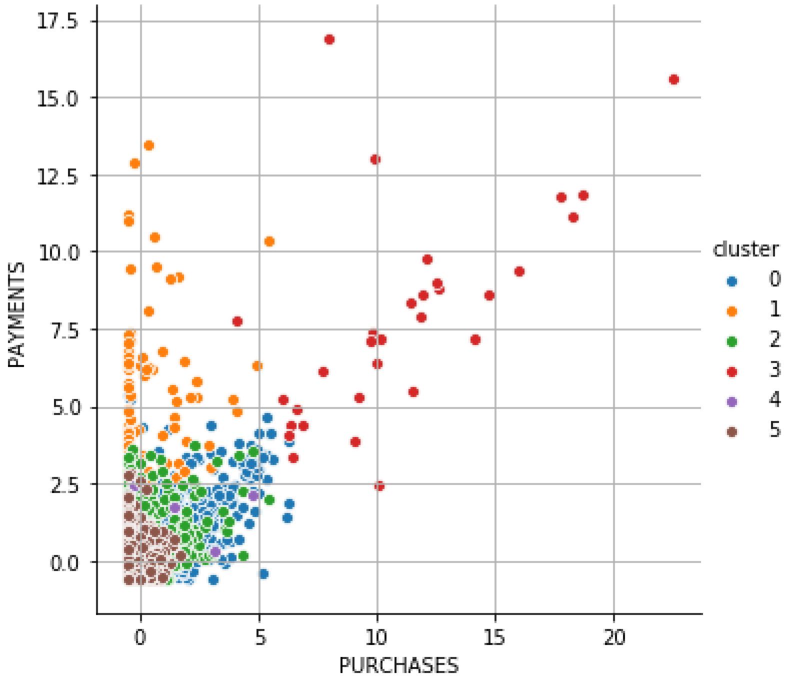
Cluster 3 (Red) : Wildly varied balance, second highest payments, average purchases. The special thing about this cluster is that these people have the highest cash advance by far - there is even one extreme case that has like 25 cash advance points. We call these people "The Money Borrowers".

In [32]:

```
sns.pairplot(data_final, hue='cluster', x_vars=['PURCHASES'], y_vars=['PAYMENTS'],height=5,
```

Out[32]:

```
<seaborn.axisgrid.PairGrid at 0xe9702561c8>
```

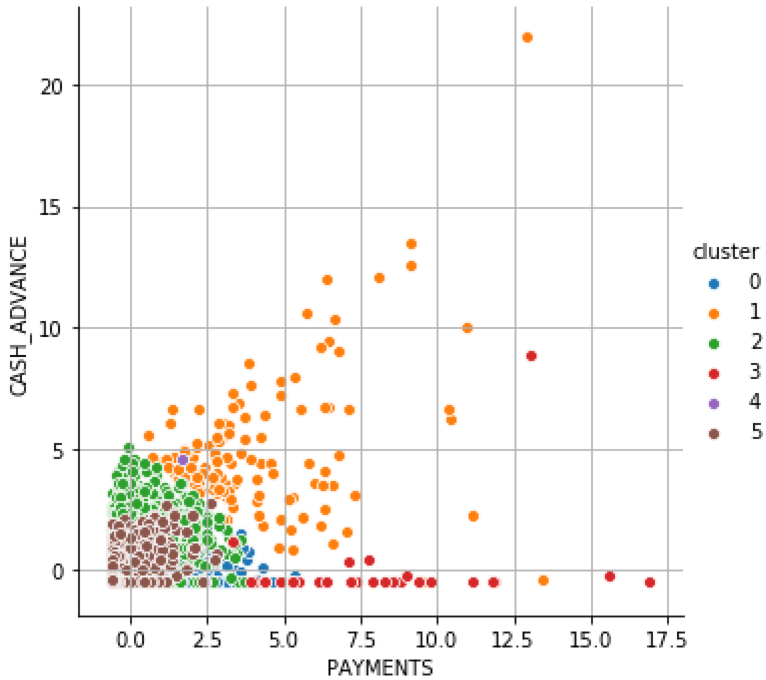


In [91]:

```
sns.pairplot(data_final, hue='cluster', x_vars=['PAYMENTS'], y_vars=['CASH_ADVANCE'],height
```

Out[91]:

```
<seaborn.axisgrid.PairGrid at 0xe9728c26c8>
```



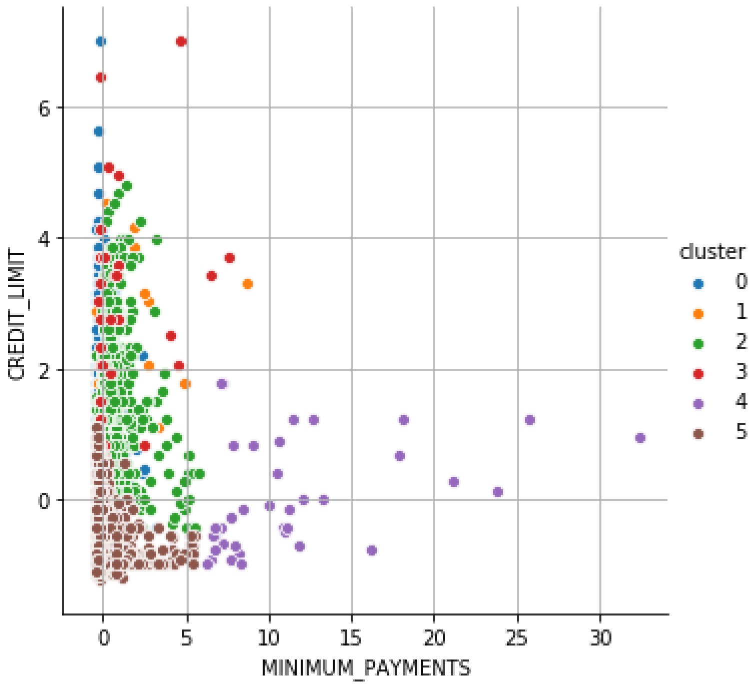
As a nature of the "Big Spenders", there are many outliers in this cluster: people who have/make abnormally high balance, purchases, cash advance, and payment. The graph below will give you an impression of how outlier-heavy this cluster is - almost all the green dots are outliers relatively compared to the rest of the whole dataset.

In [34]:

```
sns.pairplot(data_final, hue='cluster', x_vars=['MINIMUM_PAYMENTS'], y_vars=['CREDIT_LIMIT'
```

Out[34]:

```
<seaborn.axisgrid.PairGrid at 0xe97031eb08>
```



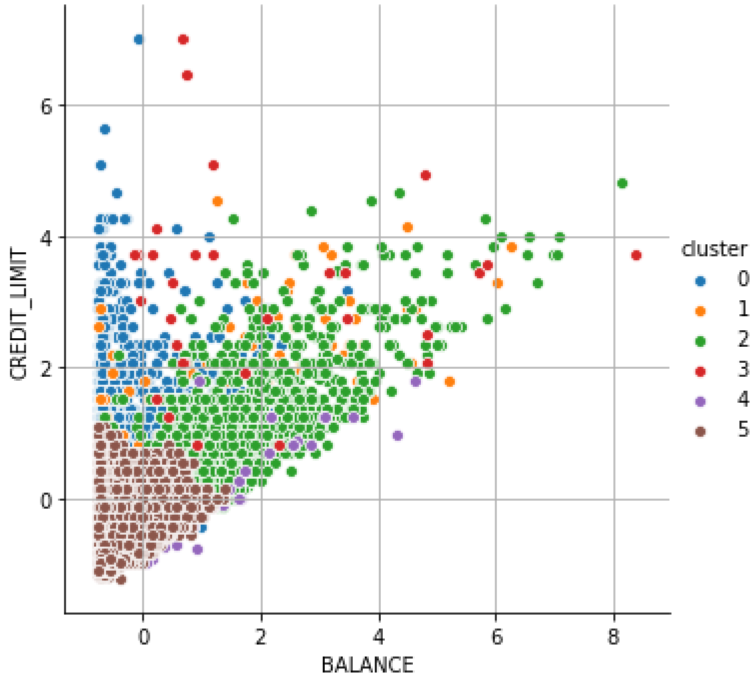
Cluster 4 (Purple) : This group has absurdly high minimum payments while having the second lowest credit limit. It looks like the bank has identified them as higher risk.

In [36]:

```
sns.pairplot(data_final, hue='cluster', x_vars=['BALANCE'], y_vars=['CREDIT_LIMIT'],height=
```

Out[36]:

```
<seaborn.axisgrid.PairGrid at 0xe96db3b788>
```



Cluster 5 (Brown) : This group is troublesome to analyze and to come up with a good marketing strategy towards, as both their credit limit and balance values are wildly varied. As you can see, the above graph looks like half of it was made of the color brown!

In [ ]:

```

```