In [48]:
```
import os
os.chdir("D:\Data Science\Python")
```

In [49]:
```
os.getcwd()
```

Out[49]: 'D:\\Data Science\\Python'

In [50]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
%matplotlib inline
```

In [66]:
```
data = pd.read_csv("credit-card-data.csv", encoding = "unicode_escape")
```

In [52]:
```
data.head()
```

Out[52]:

	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 [53]:
```
data.describe()
```

Out[53]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FRE
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	

In [54]:
```
data[data.columns[0]].isnull().value_counts()
```

Out[54]:
```
False    8950
Name: CUST_ID, dtype: int64
```

In [55]:
```
data['CREDIT_LIMIT'].isnull().value_counts()
```

Out[55]:
```
False    8949
True        1
Name: CREDIT_LIMIT, dtype: int64
```

In [63]:
```
data['CREDIT_LIMIT'].describe()
```

Out[63]:
```
count     8949.000000
mean      4494.449450
std       3638.815725
min         50.000000
25%       1600.000000
50%       3000.000000
75%       6500.000000
max      30000.000000
Name: CREDIT_LIMIT, dtype: float64
```

In [ ]:
```
#******** Missing value treatment********#
```

In [67]:
```
data[data['CREDIT_LIMIT'].isnull()]
```

Out[67]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASI
5203	C15349	18.400472	0.166667	0.0	0.0	0.0	186.853063	

In [68]:
```
data.drop(5203,axis=0,inplace=True)
```

In [69]:
```
data.isnull().any()
```

Out[69]:
```
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                     True
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

In [70]:
```
data['MINIMUM_PAYMENTS'].isnull().sum()

#since the number of outliers are many so we cananot eliminate the data therefore we'll impute the missing values.
```

Out[70]:
```
313
```

In [71]:
```
data['MINIMUM_PAYMENTS'].fillna(data['MINIMUM_PAYMENTS'].median(),inplace=True)
```

In [72]:
```
data.isnull().any()      #claerly we can see that there is no missing values after the imputation.
```

Out[72]:
```
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
```

In [73]:
```
data['Monthly_avg_purchase']=data['PURCHASES']/data['TENURE']
```

In [74]:
```
data['Monthly_avg_purchase'].head()
```

Out[74]:
```
0      7.950000
1      0.000000
2     64.430833
3    124.916667
4      1.333333
Name: Monthly_avg_purchase, dtype: float64
```

In [75]:
```
data['Monthly_cash_advance']=data['CASH_ADVANCE']/data['TENURE']
```

In [76]:
```
data['Monthly_cash_advance'].head()
```

Out[76]:
```
0      0.000000
1    536.912124
2      0.000000
3     17.149001
4      0.000000
Name: Monthly_cash_advance, dtype: float64
```

In [ ]:
```
# In the data given, we see there are columns based on the nature of purchase there we'll consider nature of purchase so
```

In [77]:
```
data[data['ONEOFF_PURCHASES']==0]['ONEOFF_PURCHASES'].count()
```

Out[77]:
```
4301
```

In [45]:
```
# Number of customers with ONEOFF_PURCHASES and INSTALLMENTS_PURCHASES
```

In [78]:
```
def purchase(data):
    if (data['ONEOFF_PURCHASES']==0) & (data['INSTALLMENTS_PURCHASES']==0):
        return 'do not prefer card'
    if (data['ONEOFF_PURCHASES']>0) & (data['INSTALLMENTS_PURCHASES']>0):
         return ' prefer both'
    if (data['ONEOFF_PURCHASES']>0) & (data['INSTALLMENTS_PURCHASES']==0):
        return 'prefer one time purchase'
    if (data['ONEOFF_PURCHASES']==0) & (data['INSTALLMENTS_PURCHASES']>0):
        return 'prefer EMI'
```

In [79]:
```

data['purchase_nature']=data.apply(purchase, axis=1)
```

In [80]:
```
data['purchase_nature'].count()
```

Out[80]:
```
8949
```

In [85]:
```
#Now, we'll proceed towards the credit score using balance and limit of the card.
```

In [86]:
```
print ("**Considering the above details, there may be many potential customers who may opt for the card if given better
```

```
**Considering the above details, there may be many potential customers who may opt for the card if given better deal ap
art from potential cutomer we may also face some fraudsters
```

In [81]:
```
data['limit_usage']=data.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT'], axis=1)
```

In [82]:
```
data['limit_usage'].head()
```

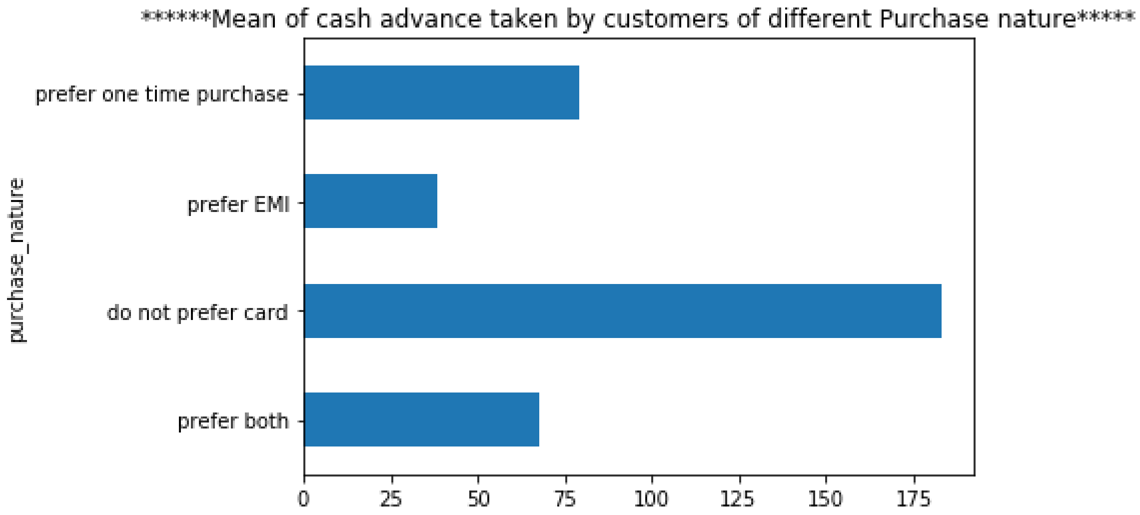
Out[82]:
```
0    0.040901
1    0.457495
2    0.332687
3    0.222223
4    0.681429
Name: limit_usage, dtype: float64
```

In [28]:
```
print ("******* LOWER THE LIMIT USAGE BETTER THE CREDIT SCORE******")
```

```
******* LOWER THE LIMIT USAGE BETTER THE CREDIT SCORE******
```

In [83]:
```
data.groupby('purchase_nature').apply(lambda x: np.mean(x['Monthly_cash_advance'])).plot.barh()
plt.title('******Mean of cash advance taken by customers of different Purchase nature*****')
```

Out[83]:
```
Text(0.5, 1.0, '******Mean of cash advance taken by customers of different Purchase nature*****')
```

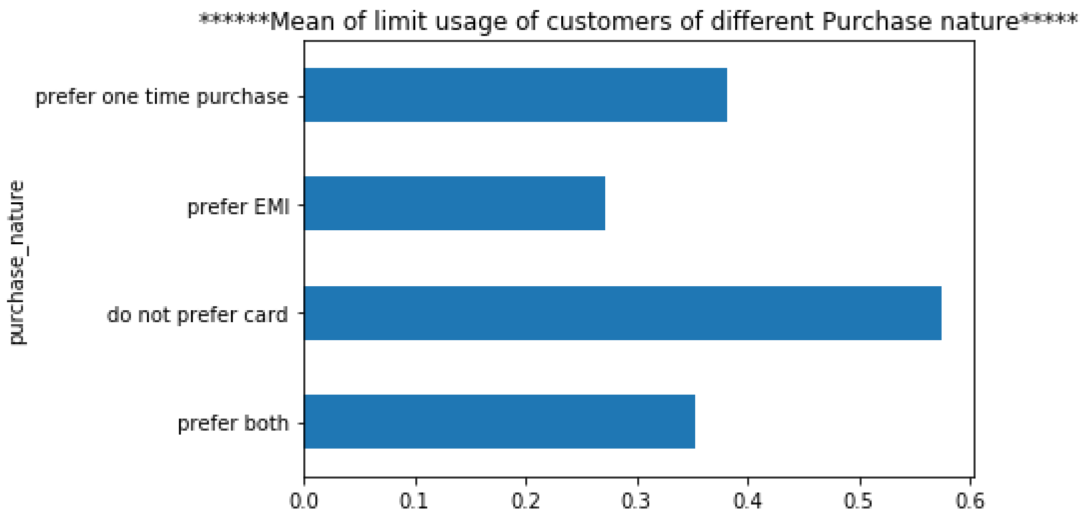


In [30]:
```
print("******Marketing strategy Suggested: We can encourage the **CATEGORY 3** to use card by providing some loyalty car
```

```
******Marketing strategy Suggested: We can encourage the **CATEGORY 3** to use card by providing some loyalty cards & o
ffers*******
```

In [84]:
```
data.groupby('purchase_nature').apply(lambda x: np.mean(x['limit_usage'])).plot.barh()
plt.title('******Mean of limit usage of customers of different Purchase nature*****')
```

Out[84]:
```
Text(0.5, 1.0, '******Mean of limit usage of customers of different Purchase nature*****')
```



In [33]:
```
print ("**Marketing strategy Suggested: As the credit score of *CATEGORY 2 is excellent in comparison to other so they c
```

```
**Marketing strategy Suggested: As the credit score of *CATEGORY 2 is excellent in comparison to other so they can our
potential customer for upcoming products****
```

In [85]:
```
data1 = data.drop(['CUST_ID','purchase_nature' ],axis=1)    #as they contain string
```

In [86]:
```
data1
```

Out[86]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FI
0	40.900749	0.818182	95.40	0.00	95.40	0.000000	
1	3202.467416	0.909091	0.00	0.00	0.00	6442.945483	
2	2495.148862	1.000000	773.17	773.17	0.00	0.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.00	205.788017	
4	817.714335	1.000000	16.00	16.00	0.00	0.000000	
...	...	...	...	...	...	...	
8945	28.493517	1.000000	291.12	0.00	291.12	0.000000	
8946	19.183215	1.000000	300.00	0.00	300.00	0.000000	
8947	23.398673	0.833333	144.40	0.00	144.40	0.000000	
8948	13.457564	0.833333	0.00	0.00	0.00	36.558778	
8949	372.708075	0.666667	1093.25	1093.25	0.00	127.040008	

8949 rows × 20 columns

In [87]:
```
from sklearn.preprocessing import  StandardScaler
```

In [88]:
```
sc=StandardScaler()
```

In [89]:
```
data_scaled=sc.fit_transform(data1)
```

In [90]:
```
data_scaled              #standarisation so to bring data on same scale
```

Out[90]:
```
array([[-0.73205404, -0.24988139, -0.4249337 , ..., -0.43341823,
        -0.46073668, -0.89305917],
       [ 0.78685815,  0.1340494 , -0.4695839 , ..., -0.47746097,
         2.31924476,  0.17595288],
       [ 0.44704093,  0.51798018, -0.10771601, ..., -0.12051627,
        -0.46073668, -0.14431566],
       ...,
       [-0.74046257, -0.18589504, -0.40200016, ..., -0.34413243,
        -0.46073668, -0.93797077],
       [-0.74523857, -0.18589504, -0.4695839 , ..., -0.47746097,
        -0.42918815, -0.92894729],
       [-0.57264377, -0.88976603,  0.0420915 , ...,  0.5319672 ,
        -0.35110705, -0.20101676]])
```

In [91]:
```
from sklearn.decomposition import PCA
```

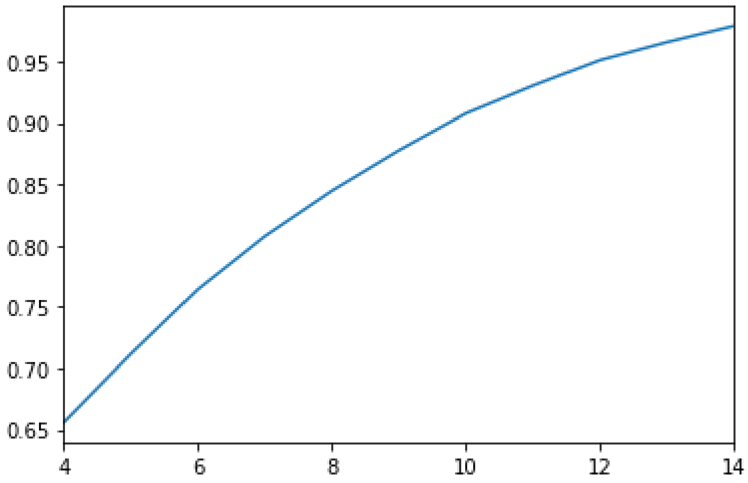
In [92]:
```
var_ratio={}
for n in range(4,15):
    pc=PCA(n_components=n)
    data_pca=pc.fit(data_scaled)
    var_ratio[n]=sum(data_pca.explained_variance_ratio_)
```

In [93]:
```
var_ratio
```

Out[93]:
```
{4: 0.6559915798250473,
 5: 0.7116269865862184,
 6: 0.7644203636218111,
 7: 0.807776584322305,
 8: 0.8448087591646123,
 9: 0.8777038512005656,
 10: 0.9078383206523953,
 11: 0.9303593969997241,
 12: 0.9511651686999866,
 13: 0.9658253074912885,
 14: 0.9790302970911725}
```

In [110]:
```
pd.Series(var_ratio).plot()     #Since 9 components are explaining about 87% variance so we select 9 components
```

Out[110]:
```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x14f73036d08&gt;
```



In [113]:
```
from sklearn.decomposition import FactorAnalysis
```

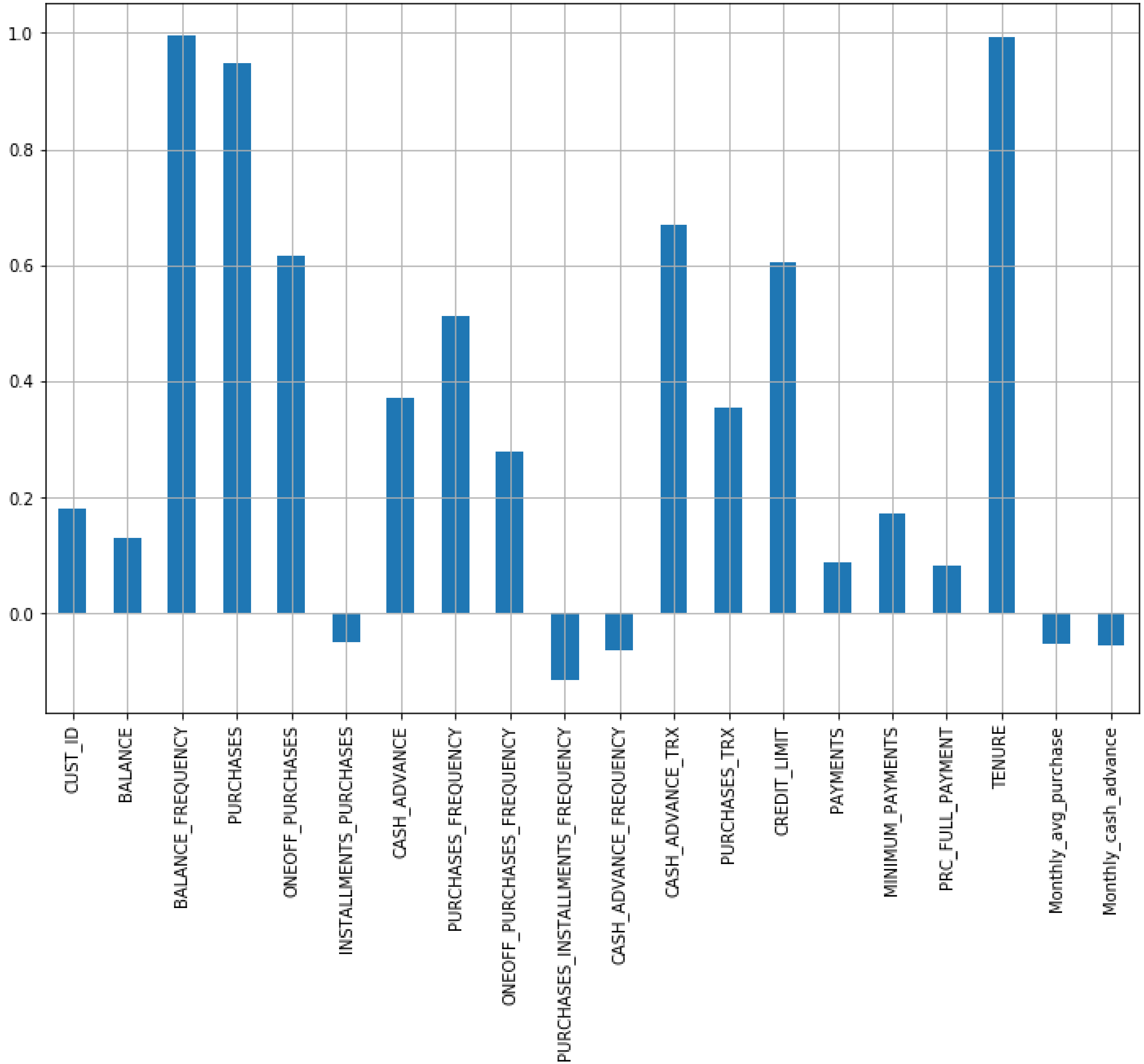
In [114]:
```
transformer = FactorAnalysis(n_components=9, random_state=0)
```

In [115]:
```
transformer.fit(data_scaled)
```

Out[115]:
```
FactorAnalysis(copy=True, iterated_power=3, max_iter=1000, n_components=9,
               noise_variance_init=None, random_state=0,
               svd_method='randomized', tol=0.01)
```

In [117]:
```
components_data=pd.DataFrame(transformer.components_)
for i in range(len(data.columns)):
    components_data.rename(columns={i:data.columns[i]},inplace=True)
```

In [118]:
```
plt.figure(figsize=(12,8))
components_data.loc[0].plot(kind="bar")
plt.grid(True)
```


1.0
0.8
0.6
0.4
0.2
0.0
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
Monthly_avg_purchase
Monthly_cash_advance

In [120]:
```
components_data.loc[0].sort_values()
```

Out[120]:
```
PURCHASES_INSTALLMENTS_FREQUENCY   -0.114006
CASH_ADVANCE_FREQUENCY             -0.063797
Monthly_cash_advance               -0.055112
Monthly_avg_purchase               -0.051958
INSTALLMENTS_PURCHASES             -0.048076
PRC_FULL_PAYMENT                    0.082698
PAYMENTS                            0.087503
BALANCE                             0.129306
MINIMUM_PAYMENTS                    0.172902
CUST_ID                             0.180151
ONEOFF_PURCHASES_FREQUENCY          0.280533
PURCHASES_TRX                       0.354104
CASH_ADVANCE                        0.371684
PURCHASES_FREQUENCY                 0.511065
CREDIT_LIMIT                        0.604049
ONEOFF_PURCHASES                    0.615899
CASH_ADVANCE_TRX                    0.668945
PURCHASES                           0.947138
TENURE                              0.992334
BALANCE_FREQUENCY                   0.996463
Name: 0, dtype: float64
```

In [ ]:
```
#****Considering only those features which has more variance******
PURCHASES
CASH_ADVANCE_TRX
CREDIT_LIMIT
PURCHASES_FREQUENCY
CASH_ADVANCE
PURCHASES_TRX
```

In [122]:
```
data_cluster = data[['PURCHASES', 'CASH_ADVANCE_TRX', 'PURCHASES_FREQUENCY','CREDIT_LIMIT', 'CASH_ADVANCE_TRX','PURCHASE
```

In [123]:
```
data_cluster.head()
```

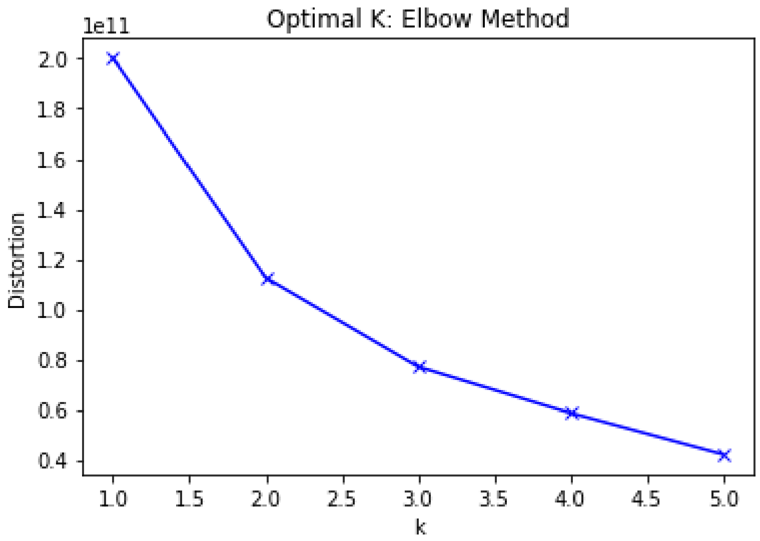
Out[123]:

	PURCHASES	CASH_ADVANCE_TRX	PURCHASES_FREQUENCY	CREDIT_LIMIT	CASH_ADVANCE_TRX	PURCHASES
0	95.40	0	0.166667	1000.0	0	95.40
1	0.00	4	0.000000	7000.0	4	0.00
2	773.17	0	1.000000	7500.0	0	773.17
3	1499.00	1	0.083333	7500.0	1	1499.00
4	16.00	0	0.083333	1200.0	0	16.00

In [124]:
```
from sklearn.cluster import KMeans
import sklearn.cluster as cluster
import time
```

In [126]:
```
distortions = []
K = range(1,6)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(data_cluster)
    distortions.append(kmeanModel.inertia_)
```

In [128]:
```
# Plot the elbow
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('Optimal K: Elbow Method')
plt.show()
```



In [ ]:
```
# As the above graph, we can clearly see the formation of arm like structure is at 2 therefore the value of optimal k=2
```

In [129]:
```
km = KMeans(init="random", n_clusters=2)
km.fit(data_cluster)
```

Out[129]:
```
KMeans(algorithm='auto', copy_x=True, init='random', max_iter=300, n_clusters=2,
       n_init=10, n_jobs=None, precompute_distances='auto', random_state=None,
       tol=0.0001, verbose=0)
```

In [130]:
```
labels=km.labels_
```

In [131]:
```
labels=labels.tolist()
```

In [132]:
```
labels=pd.Series(data=labels,index=range(len(labels)))
```

In [133]:
```
# Now, we will be extracting the data from the clusters.
```

In [135]:
```
labels_x=list()
for i in range(10):
    labels_x.append(labels[labels.values==i])
```

In [137]:
```
data_cluster.iloc[labels_x[0].index,:].describe().T          #cluster 1
```

Out[137]:

	count	mean	std	min	25%	50%	75%	max
PURCHASES	2324.0	2247.979583	3687.777516	0.0	219.080000	1085.125	2893.6875	49039.57
CASH_ADVANCE_TRX	2324.0	4.690189	9.401297	0.0	0.000000	0.000	6.0000	123.00
PURCHASES_FREQUENCY	2324.0	0.602515	0.397845	0.0	0.166667	0.750	1.0000	1.00
CREDIT_LIMIT	2324.0	9476.552965	3211.597391	2800.0	7200.000000	8500.000	11000.0000	30000.00
CASH_ADVANCE_TRX	2324.0	4.690189	9.401297	0.0	0.000000	0.000	6.0000	123.00
PURCHASES	2324.0	2247.979583	3687.777516	0.0	219.080000	1085.125	2893.6875	49039.57

In [141]:
```
data_cluster.iloc[labels_x[1].index,:].describe().T          ##cluster 2
```

Out[141]:

	count	mean	std	min	25%	50%	75%	max
PURCHASES	6625.0	566.698673	814.799799	0.0	0.0	267.750000	756.540000	8591.31
CASH_ADVANCE_TRX	6625.0	2.743547	5.562628	0.0	0.0	0.000000	3.000000	123.00
PURCHASES_FREQUENCY	6625.0	0.451078	0.395149	0.0	0.0	0.416667	0.888889	1.00
CREDIT_LIMIT	6625.0	2746.765138	1583.037815	50.0	1500.0	2500.000000	4000.000000	6700.00
CASH_ADVANCE_TRX	6625.0	2.743547	5.562628	0.0	0.0	0.000000	3.000000	123.00
PURCHASES	6625.0	566.698673	814.799799	0.0	0.0	267.750000	756.540000	8591.31

In [143]:
```
# From the above anaysis we can easily predict the Credit Card Segmentation.
```