In [79]:
```
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report

pd.set_option('display.max_columns',None)
import warnings
warnings.filterwarnings("ignore")
```

In [8]:
```
df = pd.read_csv('/Users/mac/Desktop/DataScience/Pojects_ds/Market Segmentation/Customer-I
```

In [9]:
```
df.head()
```

Out[9]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCI
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	

In [10]:
```
df.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 [11]:
```
df.isnull().sum()
```

Out[11]:
```
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 [12]:
```
df.shape
```

Out[12]:
```
(8950, 18)
```

In [21]:
```
#FILLING MISSING VALUE IN DATASET
df['MINIMUM_PAYMENTS'] = df['MINIMUM_PAYMENTS'].fillna(df['MINIMUM_PAYMENTS'].median())
df['CREDIT_LIMIT'] = df['CREDIT_LIMIT'].fillna(df['MINIMUM_PAYMENTS'].median())
```

In [23]:
```
df.isnull().sum()
```

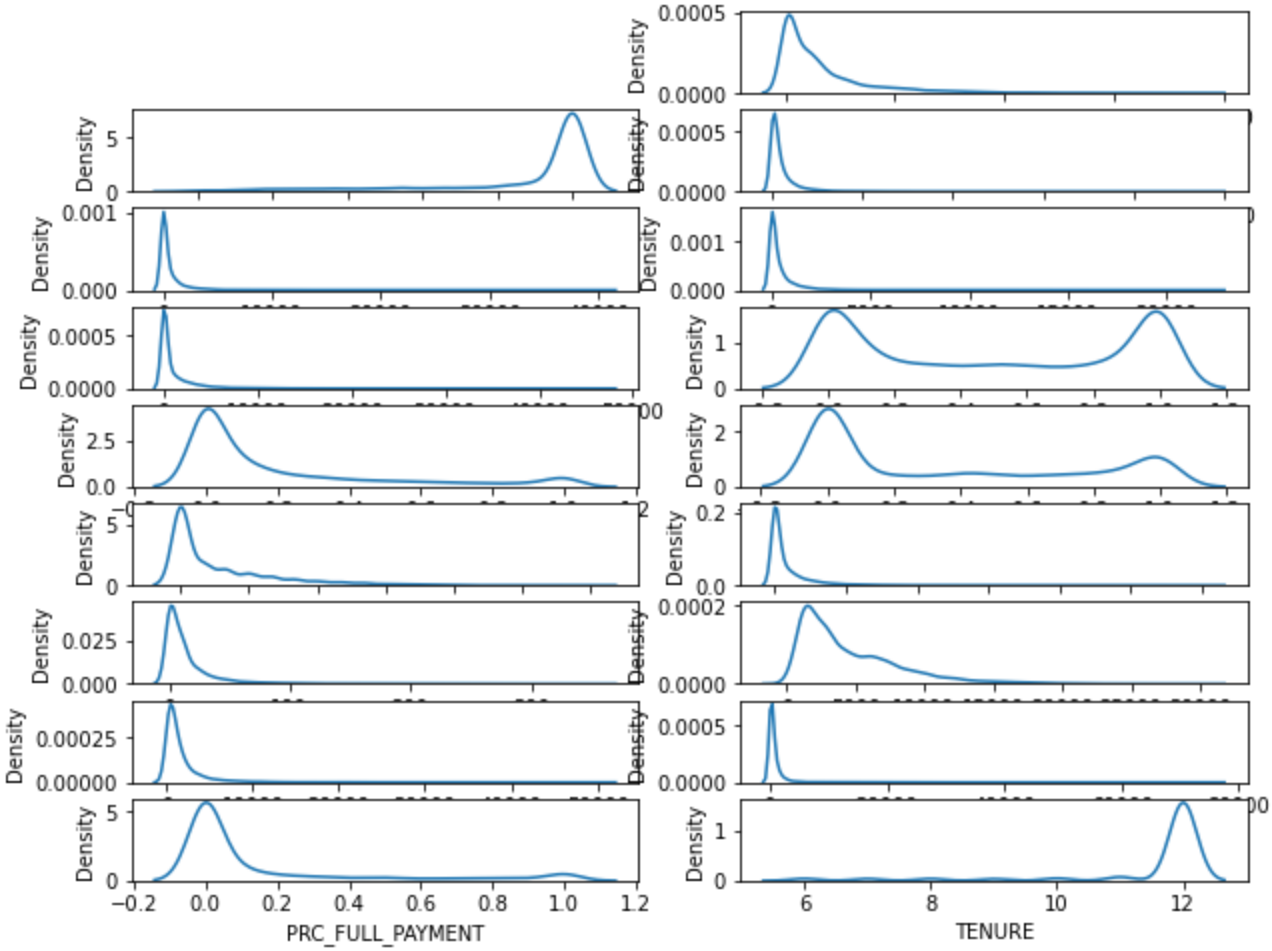
Out[23]:
```
CUST_ID                           0
BALANCE                           0
BALANCE_FREQUENCY                 0
PURCHASES                         0
ONEOFF_PURCHASES                  0
INSTALLMENTS_PURCHASES            0
CASH_ADVANCE                      0
PURCHASES_FREQUENCY               0
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
```

In [25]:
```
#checking if we have duplicate values or not
df.duplicated().sum()
```

Out[25]:
```
0
```

In [26]:
```
plt.figure(figsize=(10,8))
for i, col in enumerate(df.columns):
    if df[col].dtype != 'object':
        ax = plt.subplot(9, 2, i+1)
```

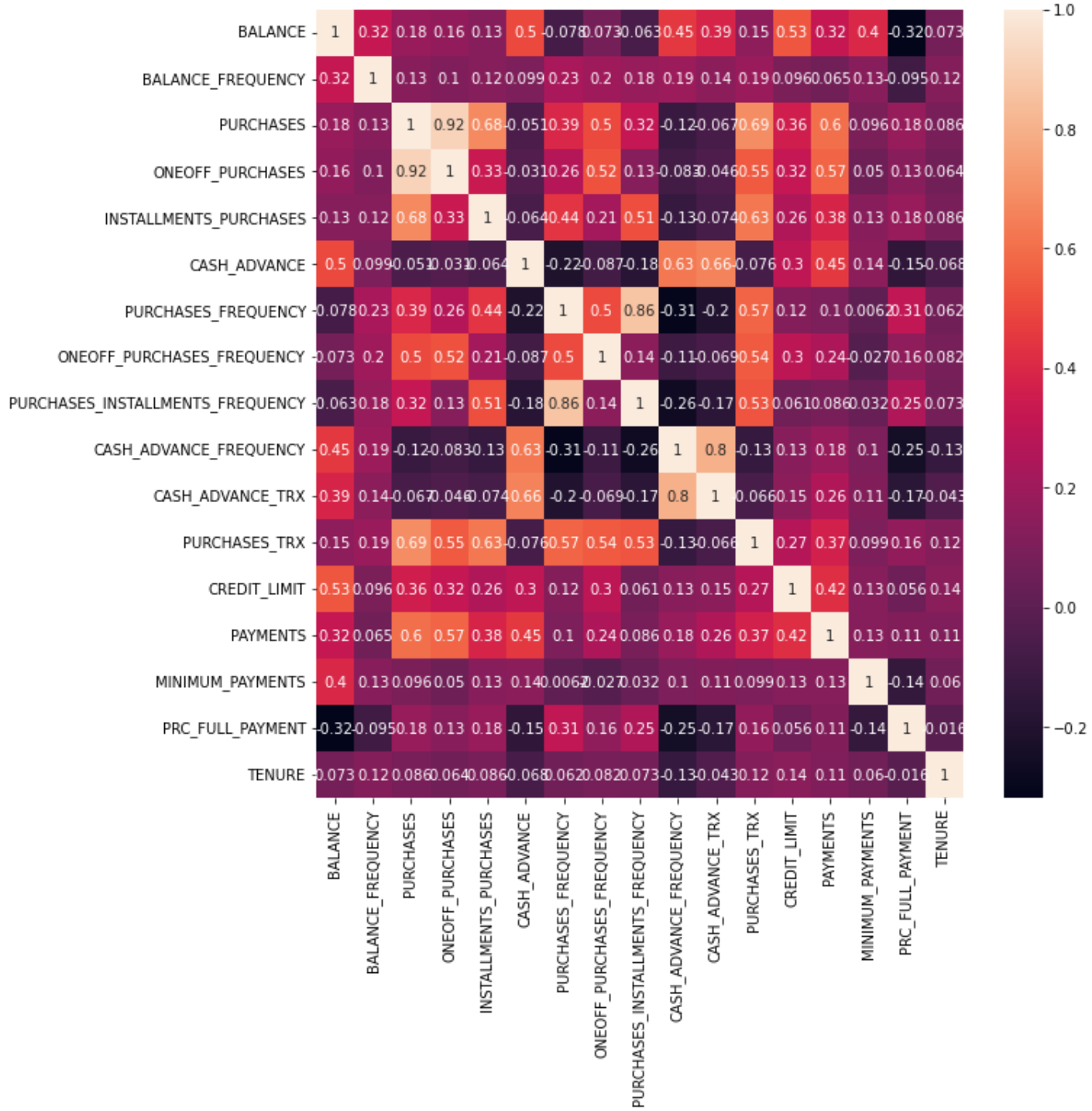
```
        sns.kdeplot(df[col], ax=ax)
        plt.xlabel(col)

plt.show()
```



- Inital judgement is that all the features are not normally distributed.

In [27]:

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



In [33]:
```
df.drop('CUST_ID',axis=1,inplace=True)
```

Feature Scaling

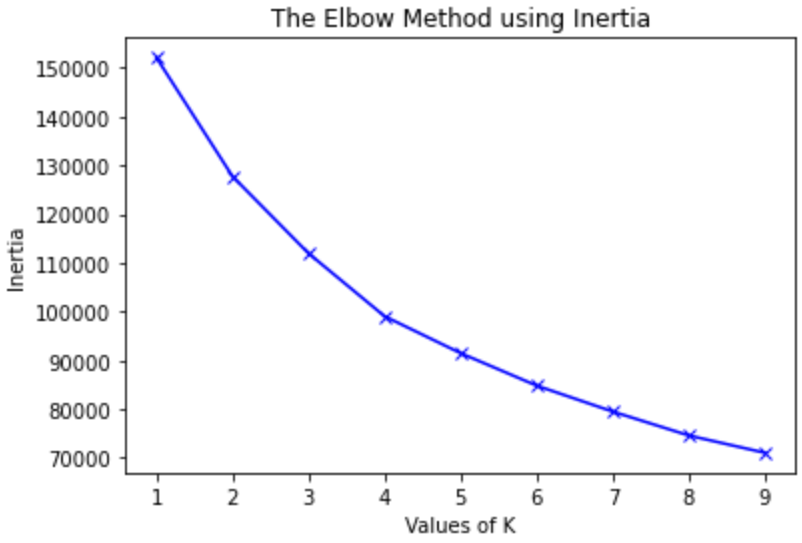
- It means that bringing the values at the same level.

In [34]:
```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
```

Hyperparameter tuning

In [43]:
```
#finding optimal value of k
```

```
inertia = []
range_value = range(1,10)
for i in range_value:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit_predict(pd.DataFrame(scaled_df))
    inertia.append(kmeans.inertia_)
plt.plot(range_value,inertia,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```



- According to the graph, we should take the value of K as 4.

In [45]:
```
k_means_model = KMeans(4)
k_means_pred = k_means_model.fit_predict(scaled_df)
```

In [51]:
```
#creating a new col called clusters in the data
df['Cluster'] = k_means_pred
```

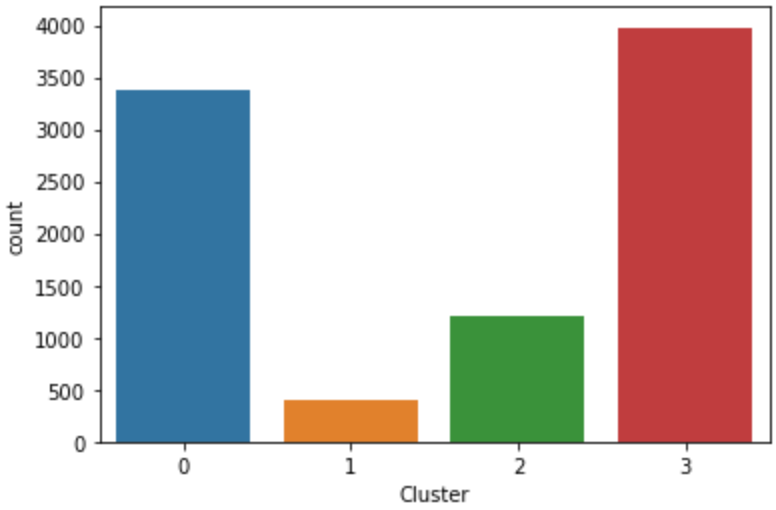
In [54]:
```
df.head(10)
```

Out[54]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CA
0	40.900749	0.818182	95.40	0.00	95.40	
1	3202.467416	0.909091	0.00	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	0.00	
3	1666.670542	0.636364	1499.00	1499.00	0.00	
4	817.714335	1.000000	16.00	16.00	0.00	
5	1809.828751	1.000000	1333.28	0.00	1333.28	
6	627.260806	1.000000	7091.01	6402.63	688.38	
7	1823.652743	1.000000	436.20	0.00	436.20	
8	1014.926473	1.000000	861.49	661.49	200.00	
9	152.225975	0.545455	1281.60	1281.60	0.00	

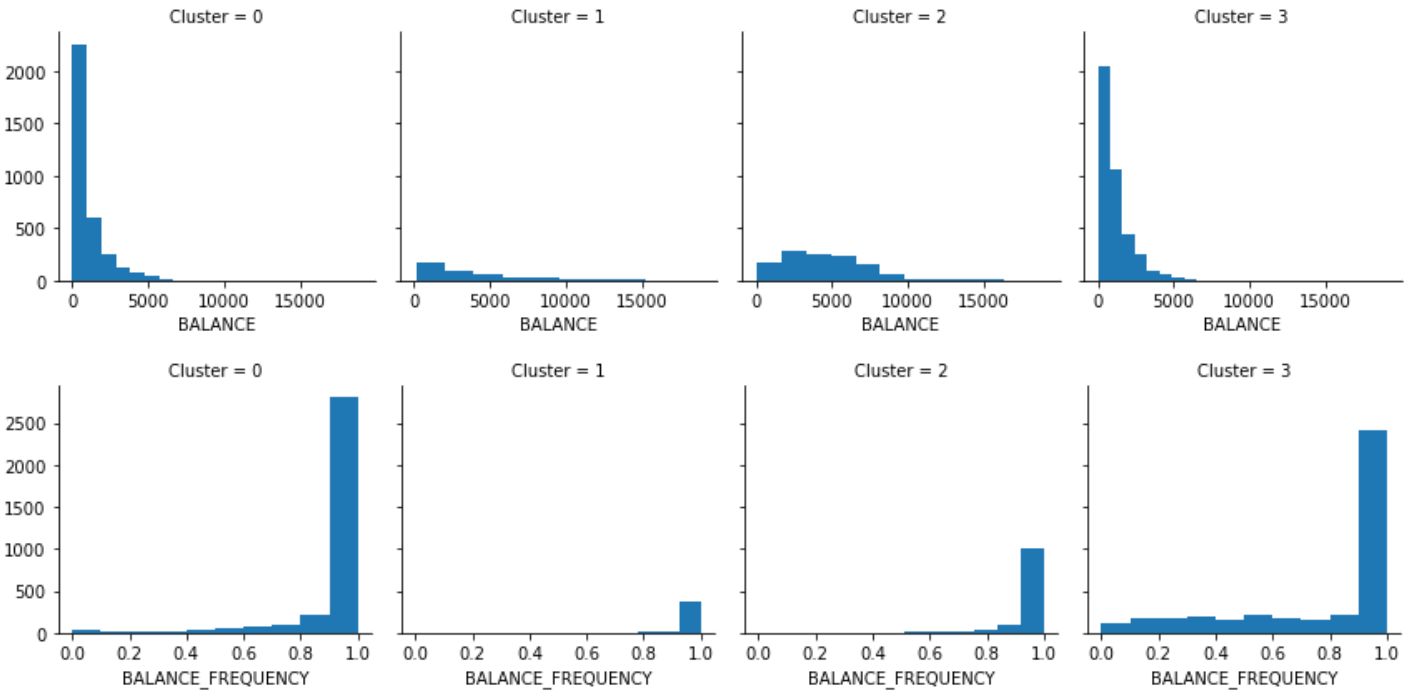
In [57]:
```
df0 = df[df["Cluster"]==0]
df1 = df[df["Cluster"]==1]
df2 = df[df["Cluster"]==2]
df3 = df[df["Cluster"]==3]
```

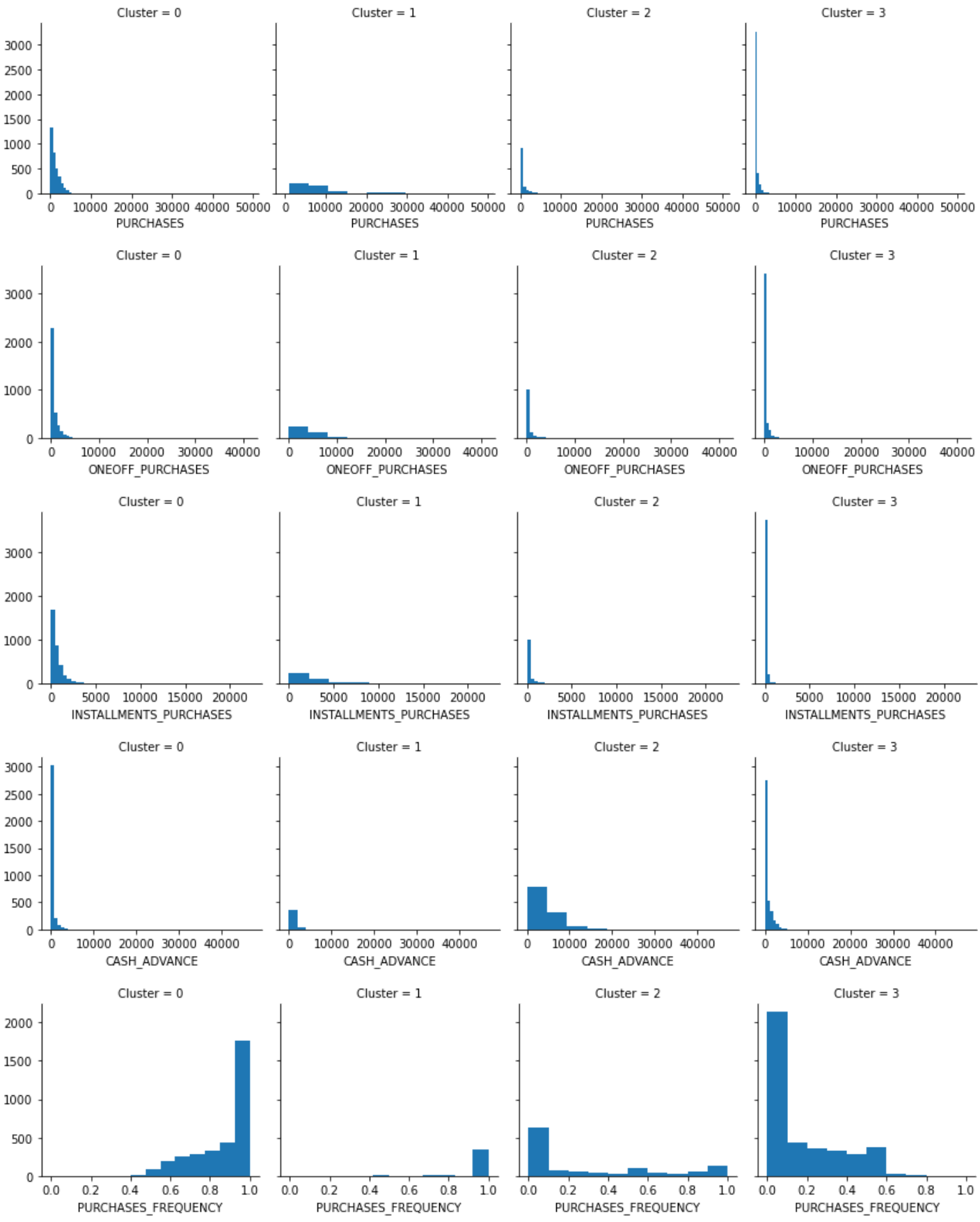
In [60]:
```
#cluster distribution
sns.countplot(x='Cluster',data = df);
```

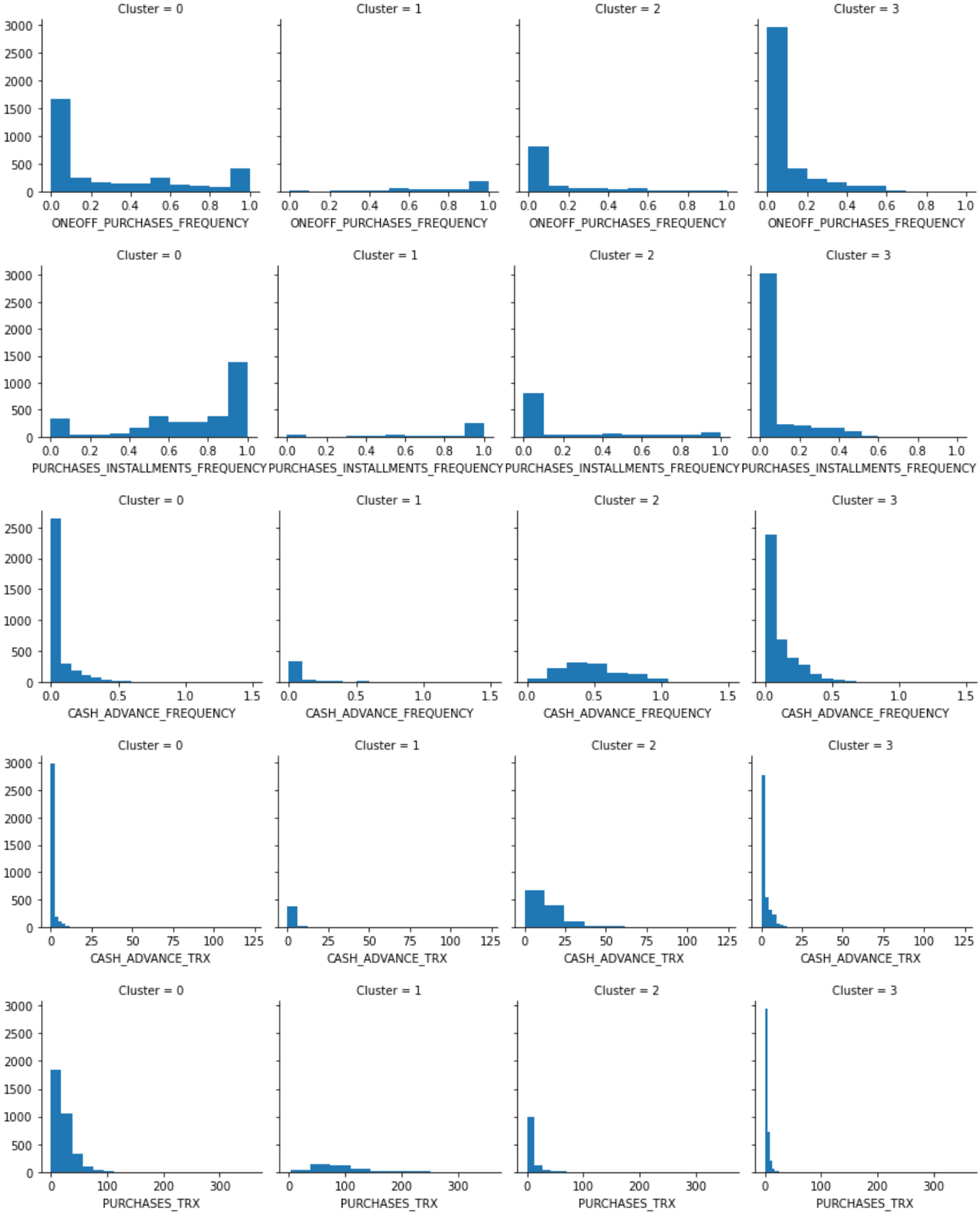


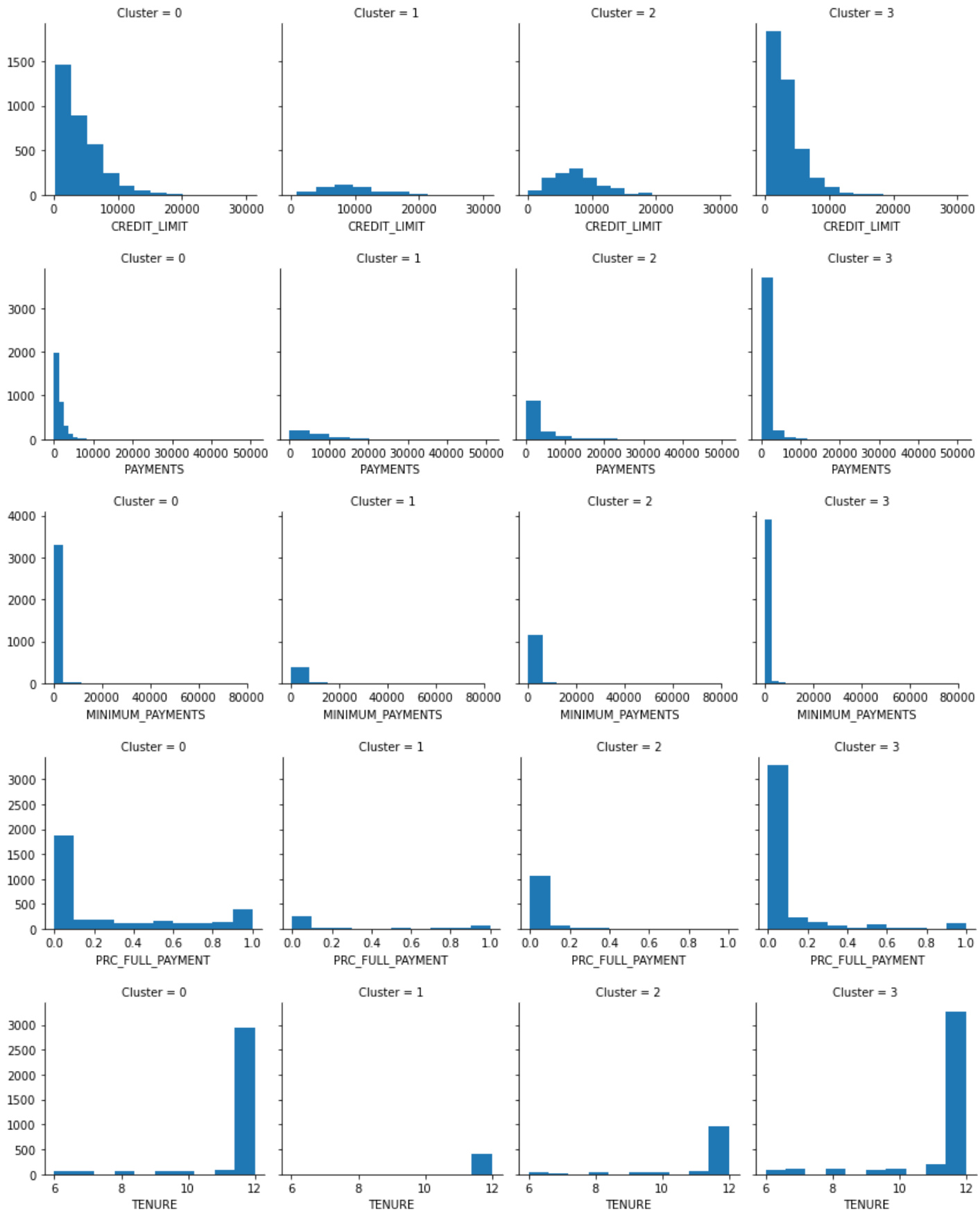
- Cluster 0 and 3 are in higher number.

In [61]:
```
for c in df.drop('Cluster',axis=1):
    grid = sns.FacetGrid(df,col='Cluster')
    grid = grid.map(plt.hist,c)
plt.show()
```




Cluster = 0
Cluster = 1
Cluster = 2
Cluster = 3
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY


Cluster = 0
Cluster = 1
Cluster = 2
Cluster = 3
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX



- characteristics of cluster

In [63]:
```
import joblib
joblib.dump(k_means_model, "kmeans_model.pkl")
```

Out[63]:
```
['kmeans_model.pkl']
```

In [66]:
```
df.to_csv("Clustered_Customer_Data1.csv")
```

In [69]:
```
#Split Dataset
X = df.drop(['Cluster'],axis=1)
y= df[['Cluster']]
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.3)
```

In [70]:
```
#Decision_Tree
model= DecisionTreeClassifier(criterion="entropy")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

In [80]:
```
#Confusion_Matrix
print(metrics.confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 992   10   13   24]
 [  16  114    2    2]
 [  16    1  339   23]
 [  39    0   32 1062]]
              precision    recall  f1-score   support

           0       0.93      0.95      0.94      1039
           1       0.91      0.85      0.88       134
           2       0.88      0.89      0.89       379
           3       0.96      0.94      0.95      1133

    accuracy                           0.93      2685
   macro avg       0.92      0.91      0.91      2685
weighted avg       0.93      0.93      0.93      2685
```

In [81]:
```
import pickle
filename = 'final_model_dt.sav'
pickle.dump(model, open(filename, 'wb'))

# some time later...

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result,'% Acuuracy')
```

```
0.933705772811918 % Acuuracy
```

In [ ]:
```

```