Team Game Of Analytics - Ayushi Choudhary, Ritumbhra Sagar, Shilpa Chotwani, Sonal Agarwal

In [1]:
```
#import modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dtm
from datetime import date
import numpy as np

import sklearn as sk
import sklearn.tree as tree
from IPython.display import Image
import pydotplus

import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
```

In [2]:
```
# Loading Dataset
data = pd.read_csv("CSVDurData1_Final.csv")
```

In [3]:
```
# Checking unique rows
filter_data=data[['Transaction_NBR','Household_ID']].drop_duplicates()
len(filter_data)
```

Out[3]:
```
173262
```

Summary

In [5]:
```
data.describe().T
```

Out[5]:

	count	mean	std	min	25%
Household_ID	173262.0	1.292614e+08	1.714242e+07	1.000035e+08	1.141345e+08
Transaction_NBR	173262.0	1.267407e+01	1.779025e+01	1.000000e+00	3.000000e+00
Transaction_Total	173262.0	2.434814e+01	2.831114e+01	1.000000e+00	8.000000e+00
Transaction_Location	173262.0	2.544546e+03	1.353074e+03	2.000000e+00	8.600000e+02
Online_Transaction	173262.0	1.471760e-02	1.204204e-01	0.000000e+00	0.000000e+00
ORIGINAL_TICKET_NBR	173262.0	2.544820e+11	1.353241e+11	2.001612e+08	8.600255e+10
Transaction_type	173262.0	1.335267e+00	7.651849e-01	1.000000e+00	1.000000e+00
PRODUCT_ID	132099.0	7.171141e+05	1.238596e+05	5.326560e+05	5.810970e+05
Sub_Category_NBR	173262.0	3.164813e+02	1.953734e+02	0.000000e+00	1.720000e+02
Quantity	173262.0	8.219113e-01	1.380541e+00	-1.100000e+01	1.000000e+00
UNIT_PRICE	173262.0	1.089130e+02	2.954237e+02	-6.899980e+03	9.990000e+00
EXTENDED_PRICE	173262.0	1.092319e+02	2.955023e+02	-6.899980e+03	9.990000e+00
Return_Location_If_Any	173262.0	2.519373e+03	1.351602e+03	2.000000e+00	8.590000e+02
Age_HH	152291.0	4.825652e+01	1.434196e+01	1.800000e+01	3.800000e+01
Income	152807.0	5.920383e+00	2.296218e+00	1.000000e+00	4.000000e+00
MALE_CHID_AGE_0_2	173262.0	5.604229e-03	7.465155e-02	0.000000e+00	0.000000e+00
MALE_CHID_AGE_3_5	173262.0	2.114716e-02	1.438752e-01	0.000000e+00	0.000000e+00
MALE_CHID_AGE_6_10	173262.0	3.922961e-02	1.941413e-01	0.000000e+00	0.000000e+00
MALE_CHID_AGE_11_15	173262.0	5.751405e-02	2.328229e-01	0.000000e+00	0.000000e+00
MALE_CHID_AGE_16_17	173262.0	5.320844e-02	2.244495e-01	0.000000e+00	0.000000e+00
FEMALE_CHID_AGE_0_2	173262.0	6.210248e-03	7.856027e-02	0.000000e+00	0.000000e+00
FEMALE_CHID_AGE_3_5	173262.0	1.562951e-02	1.240376e-01	0.000000e+00	0.000000e+00
FEMALE_CHID_AGE_6_10	173262.0	3.133982e-02	1.742349e-01	0.000000e+00	0.000000e+00
FEMALE_CHID_AGE_11_15	173262.0	5.294294e-02	2.239202e-01	0.000000e+00	0.000000e+00
FEMALE_CHID_AGE_16_17	173262.0	5.108448e-02	2.201707e-01	0.000000e+00	0.000000e+00
UNKNOWN_CHID_AGE_0_2	173262.0	3.232099e-02	1.768517e-01	0.000000e+00	0.000000e+00
UNKNOWN_CHID_AGE_3_5	173262.0	3.699022e-02	1.887383e-01	0.000000e+00	0.000000e+00
UNKNOWN_CHID_AGE_6_10	173262.0	3.740001e-02	1.897405e-01	0.000000e+00	0.000000e+00
UNKNOWN_CHID_AGE_11_15	173262.0	2.983920e-02	1.701440e-01	0.000000e+00	0.000000e+00
UNKNOWN_CHID_AGE_16_17	173262.0	1.036003e-02	1.012559e-01	0.000000e+00	0.000000e+00

CLEANING DATA

Dropping Columns that are not required

In [6]:
```
#Creating a column Total Children that counts the number of children in
 a household
data['TotalChildren']=data.MALE_CHID_AGE_0_2+data.MALE_CHID_AGE_3_5+data
.MALE_CHID_AGE_6_10+data.MALE_CHID_AGE_11_15+\
                 data.MALE_CHID_AGE_16_17+data.FEMALE_CHID_AGE_0_2+data.
FEMALE_CHID_AGE_3_5+data.FEMALE_CHID_AGE_6_10+\
                 data.FEMALE_CHID_AGE_11_15+data.FEMALE_CHID_AGE_16_17+d
ata.UNKNOWN_CHID_AGE_0_2+\
                 data.UNKNOWN_CHID_AGE_3_5+data.UNKNOWN_CHID_AGE_6_10+da
ta.UNKNOWN_CHID_AGE_11_15+data.UNKNOWN_CHID_AGE_16_17
```

Removing or Replacing Nan values

In [7]:
```
data.isna().sum().T
```

Out[7]:
```
Household_ID                         0
Transaction_NBR                      0
Transaction_Total                    0
Transaction_Date                     0
Transaction_Location                 0
Online_Transaction                   0
ORIGINAL_TICKET_NBR                  0
Transaction_type                     0
PRODUCT_ID                       41163
Category_Description               517
Sub_Category_NBR                     0
Sub_Category_Description          3847
Transaction_Type_Description        36
Quantity                             0
UNIT_PRICE                           0
EXTENDED_PRICE                       0
RETURN_IND                       13411
Return_Location_If_Any               0
Age_HH                           20971
CHILDERN_PRESENCE                91777
Income                           20455
GENDERHH                             0
Gender_Individual                14575
MALE_CHID_AGE_0_2                    0
MALE_CHID_AGE_3_5                    0
MALE_CHID_AGE_6_10                   0
MALE_CHID_AGE_11_15                  0
MALE_CHID_AGE_16_17                  0
FEMALE_CHID_AGE_0_2                  0
FEMALE_CHID_AGE_3_5                  0
FEMALE_CHID_AGE_6_10                 0
FEMALE_CHID_AGE_11_15                0
FEMALE_CHID_AGE_16_17                0
UNKNOWN_CHID_AGE_0_2                 0
UNKNOWN_CHID_AGE_3_5                 0
UNKNOWN_CHID_AGE_6_10                0
UNKNOWN_CHID_AGE_11_15               0
UNKNOWN_CHID_AGE_16_17               0
TotalChildren                        0
dtype: int64
```

In [8]:
```
data2=data.copy()
```

In [9]:
```
#Removing Category Description with Nan values since only 517 rows
data3=data2[~(data2.Category_Description.isna())]
```

In [10]:
```
#Replacing AgeHH with mean where there are Nan
data3.loc[data3.Age_HH.isna(),'Age_HH']=data3.Age_HH.mean()
```

In [11]:
```
#Replacing IncomeTRANSACTION_TYPE DESCRIPTION  with mean where there are
Nan
data3.loc[data3.Income.isna(),'Income']=data3.Income.mean()
```

Formatting Variables

In [12]:
```
# Extracting date from timestamp
data3['Formatted_Date']=data3['Transaction_Date'].str.replace(':00:00:0
0','')
```

In [13]:
```
data3['Final_Date']=pd.to_datetime(data3['Formatted_Date'])
```

In [14]:
```
#Creating Month Variable
data3['Month']=data3['Final_Date'].astype(np.datetime64).dt.month
```

In [15]:
```
#Creating Year Variable
data3['Year']=data3['Final_Date'].astype(np.datetime64).dt.year
```

Q1 - Predicting whether a customer will return the product ?

In [16]:
```
data4=data3[(data3.GENDERHH=='M')|(data3.GENDERHH=='F')]
```

In [17]:
```
#Dropping the Id, Description Variables and all Child Dummies(combined i
nto TotalChildren)
data5=data4.drop(columns=['PRODUCT_ID','Sub_Category_NBR','Sub_Category_
Description',\
                          'CHILDERN_PRESENCE','Gender_Individual','Transa
ction_Type_Description',\
                          'Household_ID','Transaction_NBR','Transaction_T
otal','Transaction_Location',\
                          'ORIGINAL_TICKET_NBR','Transaction_Type_Descrip
tion','Quantity',\
                          'EXTENDED_PRICE','Return_Location_If_Any','MALE
_CHID_AGE_0_2', 'MALE_CHID_AGE_3_5',\
                          'MALE_CHID_AGE_6_10','MALE_CHID_AGE_11_15', 'MA
LE_CHID_AGE_16_17','FEMALE_CHID_AGE_0_2',\
                          'FEMALE_CHID_AGE_3_5', 'FEMALE_CHID_AGE_6_10',
'FEMALE_CHID_AGE_11_15',\
                          'FEMALE_CHID_AGE_16_17', 'UNKNOWN_CHID_AGE_0_2'
,'UNKNOWN_CHID_AGE_3_5',\
                          'UNKNOWN_CHID_AGE_6_10', 'UNKNOWN_CHID_AGE_11_1
5','UNKNOWN_CHID_AGE_16_17'])
```

In [18]:
```
data5.describe()
```

Out[18]:

	Online_Transaction	Transaction_type	UNIT_PRICE	Age_HH	Income	Total
count	145032.000000	145032.000000	145032.000000	145032.000000	145032.000000	14503:
mean	0.013680	1.335547	108.908803	48.629028	5.970029	(
std	0.116158	0.766242	292.713003	13.547555	2.149895	(
min	0.000000	1.000000	-6899.980000	18.000000	1.000000	(
25%	0.000000	1.000000	10.000000	40.000000	5.000000	(
50%	0.000000	1.000000	39.990000	48.261417	6.000000	(
75%	0.000000	1.000000	129.990000	56.000000	7.000000	·
max	1.000000	6.000000	6999.990000	99.000000	9.000000	(

In [19]:
```
dfr=data5.copy()
```

In [20]:
```
dfr.groupby('Transaction_type').size()
```

Out[20]:
```
Transaction_type
1    116916
2     12126
3     12704
4      2040
5      1219
6        27
dtype: int64
```

In [21]:
```
#Considering only Transaction_Type=1 since it has maximum transactions
dfr1=dfr[(dfr.Transaction_type==1)]
```

Exploring the Returns Data

In [22]:
```
#Removing not null return indicators since we need to know if product wa
s returned or not
dfr2=dfr1[~dfr1.RETURN_IND.isna()]
```

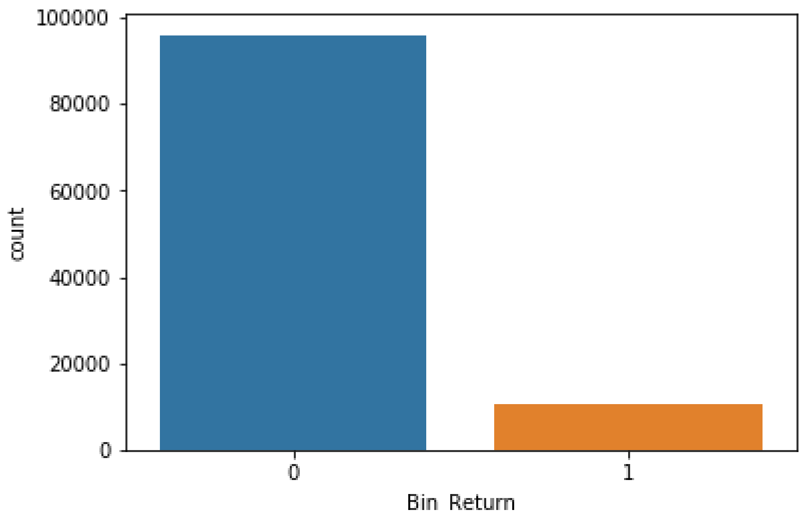
In [23]:
```
#Creating Binary Return Variable
dfr2['Bin_Return']=(dfr2.RETURN_IND=='Y')*1
```

In [24]:
```
dfr2.groupby('Bin_Return').size()
```

Out[24]:
```
Bin_Return
0    95902
1    10729
dtype: int64
```

In [25]:
```
sns.countplot(x='Bin_Return',data=dfr2)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20406fd0>



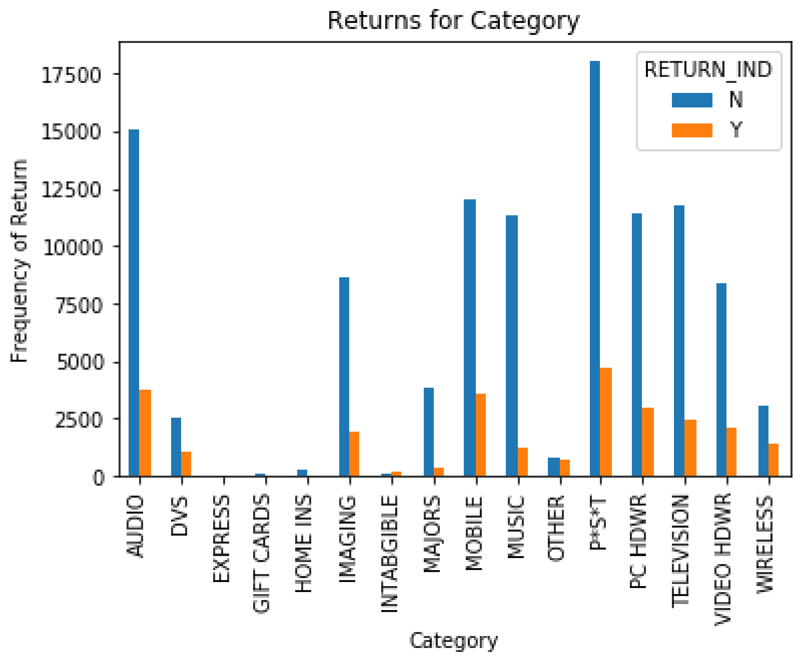
In [26]:
```
dfr2.groupby('Bin_Return').mean().T
```

Out[26]:

Bin_Return	0	1
Online_Transaction	0.012252	0.007177
Transaction_type	1.000000	1.000000
UNIT_PRICE	156.927153	172.406600
Age_HH	48.612543	48.465753
Income	6.003069	5.998745
TotalChildren	0.504421	0.519340
Month	6.128277	6.498649
Year	2001.681821	2001.495946

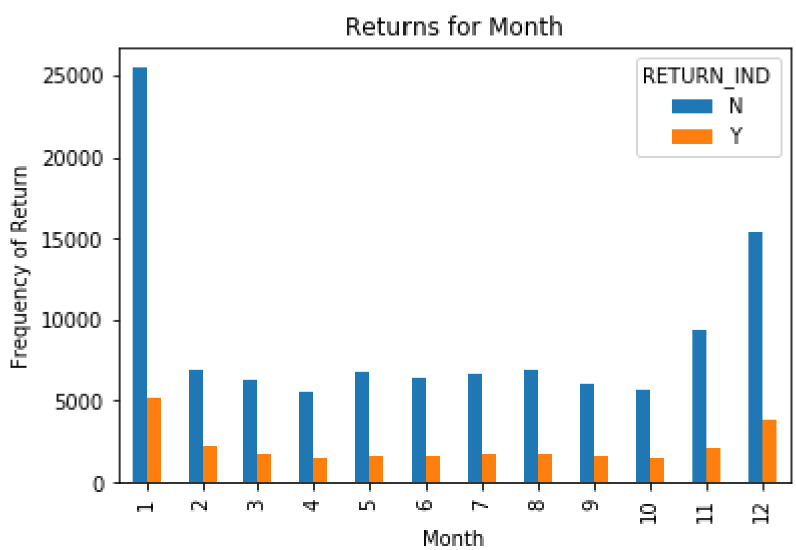
In [27]:
```
pd.crosstab(data4.Category_Description,data4.RETURN_IND).plot(kind='bar'
)
plt.xlabel('Category')
plt.ylabel('Frequency of Return')
plt.title('Returns for Category')
```

Out[27]: Text(0.5, 1.0, 'Returns for Category')



In [28]:
```
pd.crosstab(data4.Month,data4.RETURN_IND).plot(kind='bar')
plt.xlabel('Month')
plt.ylabel('Frequency of Return')
plt.title('Returns for Month')
```

Out[28]: Text(0.5, 1.0, 'Returns for Month')



In [29]:
```
dfr2.describe().T
```

Out[29]:

	count	mean	std	min	25%	50%	75%	
Online_Transaction	106631.0	0.011741	0.107720	0.00	0.00	0.000000	0.00	
Transaction_type	106631.0	1.000000	0.000000	1.00	1.00	1.000000	1.00	
UNIT_PRICE	106631.0	158.484664	296.066554	-599.99	19.99	59.990000	169.99	6
Age_HH	106631.0	48.597773	13.419048	18.00	40.00	48.261417	56.00	
Income	106631.0	6.002634	2.140006	1.00	5.00	6.000000	7.00	
TotalChildren	106631.0	0.505922	0.876111	0.00	0.00	0.000000	1.00	
Month	106631.0	6.165543	4.099826	1.00	2.00	6.000000	10.00	
Year	106631.0	2001.663119	1.524222	1998.00	2001.00	2002.000000	2003.00	2
Bin_Return	106631.0	0.100618	0.300824	0.00	0.00	0.000000	0.00	

Logistic Regression for Q1 - Predicting whether a customer will return the product ?

In [30]:
```
dfr3=pd.get_dummies(dfr2,columns=['Category_Description','GENDERHH',\
                                  'Month','Year'],drop_first=True)
```

In [31]:
```
dfr3.columns
```

Out[31]:
```
Index(['Transaction_Date', 'Online_Transaction', 'Transaction_type',
       'UNIT_PRICE', 'RETURN_IND', 'Age_HH', 'Income', 'TotalChildren',
       'Formatted_Date', 'Final_Date', 'Bin_Return',
       'Category_Description_DVS', 'Category_Description_EXPRESS',
       'Category_Description_GIFT CARDS', 'Category_Description_HOME IN
S',
       'Category_Description_IMAGING', 'Category_Description_INTABGIBL
E',
       'Category_Description_MAJORS', 'Category_Description_MOBILE',
       'Category_Description_MUSIC', 'Category_Description_OTHER',
       'Category_Description_P*S*T', 'Category_Description_PC HDWR',
       'Category_Description_TELEVISION', 'Category_Description_VIDEO H
DWR',
       'Category_Description_WIRELESS', 'GENDERHH_M', 'Month_2', 'Month
_3',
       'Month_4', 'Month_5', 'Month_6', 'Month_7', 'Month_8', 'Month_
9',
       'Month_10', 'Month_11', 'Month_12', 'Year_1999', 'Year_2000',
       'Year_2001', 'Year_2002', 'Year_2003', 'Year_2004'],
      dtype='object')
```

In [32]:
```
data4.groupby('Category_Description').size()
```

Out[32]:
```
Category_Description
AUDIO           19025
DVS              3703
EXPRESS             1
GIFT CARDS         88
HOME INS          334
IMAGING         10689
INTABGIBLE        851
MAJORS           4314
MOBILE          19386
MUSIC           12807
OTHER            5736
P*S*T           22817
PC HDWR         14664
TELEVISION      14603
VIDEO HDWR      10516
WIRELESS         5498
dtype: int64
```

In [33]:
```
X=dfr3.drop(columns=['RETURN_IND','Bin_Return','Transaction_type',\
                     'Category_Description_EXPRESS','Category_Descriptio
n_GIFT CARDS','Transaction_Date',\
                     'Formatted_Date', 'Final_Date'])
```

In [34]:
```
Y=dfr3.Bin_Return
```

In [35]:
```
over = SMOTE(random_state=0)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=0)
columns = X_train.columns
over_X,over_Y=over.fit_sample(X_train, Y_train)
over_X = pd.DataFrame(data=over_X,columns=columns )
over_Y= pd.DataFrame(data=over_Y,columns=['y'])
print("total rows",len(over_X))
print("returns",len(over_Y[over_Y['y']==0]))
print("non returns",len(over_Y[over_Y['y']==1]))
```

```
total rows 134184
returns 67092
non returns 67092
```

In [36]:
```
X=over_X
Y=over_Y['y']
```

In [37]:
```
import statsmodels.api as sm
logit_model=sm.Logit(Y,X)
result=logit_model.fit(maxiter=10000)
print(result.summary2())
```

```
Optimization terminated successfully.
         Current function value: 0.668561
         Iterations 6
                                Results: Logit
=======================================================================
=========
Model:                    Logit                Pseudo R-squared:     0.
035
Dependent Variable:       y                    AIC:                 17
9492.3286
Date:                     2019-03-16 13:06     BIC:                 17
9845.3794
No. Observations:         134184               Log-Likelihood:      -8
9710.
Df Model:                 35                   LL-Null:             -9
3009.
Df Residuals:             134148               LLR p-value:          0.
0000
Converged:                1.0000               Scale:                1.
0000
No. Iterations:           6.0000
-----------------------------------------------------------------------
---------
                                Coef.  Std.Err.    z     P>|z|   [0.02
5  0.975]
-----------------------------------------------------------------------
---------
Online_Transaction             -0.5166   0.0636  -8.1287 0.0000 -0.641
2 -0.3920
UNIT_PRICE                      0.0003   0.0000  11.2715 0.0000  0.000
2  0.0003
Age_HH                         -0.0023   0.0004  -5.9002 0.0000 -0.003
1 -0.0016
Income                          0.0079   0.0027   2.9146 0.0036  0.002
6  0.0131
TotalChildren                  -0.0440   0.0069  -6.3562 0.0000 -0.057
6 -0.0305
Category_Description_DVS        0.6296   0.0384  16.4113 0.0000  0.554
4  0.7048
Category_Description_HOME INS  -1.2197   0.1373  -8.8831 0.0000 -1.488
8 -0.9506
Category_Description_IMAGING   -0.2300   0.0263  -8.7497 0.0000 -0.281
5 -0.1785
Category_Description_INTABGIBLE 2.4980   0.1642  15.2156 0.0000  2.176
3  2.8198
Category_Description_MAJORS    -1.3965   0.0458 -30.4621 0.0000 -1.486
3 -1.3066
Category_Description_MOBILE     0.0492   0.0230   2.1405 0.0323  0.004
1  0.0942
Category_Description_MUSIC     -1.0040   0.0275 -36.4512 0.0000 -1.058
0 -0.9500
Category_Description_OTHER      0.8217   0.0564  14.5642 0.0000  0.711
1  0.9323
Category_Description_P*S*T     -0.0257   0.0204  -1.2645 0.2060 -0.065
6  0.0142
Category_Description_PC HDWR   -0.1505   0.0255  -5.8961 0.0000 -0.200
6 -0.1005
```

```
Category_Description_TELEVISION -0.3741   0.0248 -15.0851 0.0000 -0.422
7 -0.3255
Category_Description_VIDEO HDWR -0.1962   0.0262  -7.4961 0.0000 -0.247
5 -0.1449
Category_Description_WIRELESS    0.4876   0.0351  13.9043 0.0000  0.418
8  0.5563
GENDERHH_M                       0.0251   0.0127   1.9838 0.0473  0.000
3  0.0499
Month_2                          0.4175   0.0268  15.5982 0.0000  0.365
0  0.4699
Month_3                          0.1954   0.0284   6.8779 0.0000  0.139
7  0.2511
Month_4                          0.1820   0.0304   5.9859 0.0000  0.122
4  0.2416
Month_5                          0.1886   0.0280   6.7389 0.0000  0.133
7  0.2434
Month_6                          0.2107   0.0286   7.3603 0.0000  0.154
6  0.2667
Month_7                          0.2523   0.0280   9.0108 0.0000  0.197
4  0.3071
Month_8                          0.0945   0.0278   3.3933 0.0007  0.039
9  0.1491
Month_9                          0.2747   0.0289   9.5147 0.0000  0.218
1  0.3313
Month_10                         0.2267   0.0296   7.6622 0.0000  0.168
7  0.2846
Month_11                         0.3882   0.0243  15.9920 0.0000  0.340
7  0.4358
Month_12                         0.4153   0.0200  20.7427 0.0000  0.376
0  0.4545
Year_1999                       -0.0337   0.0344  -0.9792 0.3275 -0.101
1  0.0337
Year_2000                        0.1277   0.0329   3.8818 0.0001  0.063
2  0.1921
Year_2001                        0.0229   0.0293   0.7829 0.4337 -0.034
4  0.0803
Year_2002                       -0.0060   0.0291  -0.2059 0.8369 -0.063
1  0.0511
Year_2003                       -0.0663   0.0319  -2.0769 0.0378 -0.128
9 -0.0037
Year_2004                       -0.4710   0.0343 -13.7262 0.0000 -0.538
3 -0.4038
=======================================================================
=========
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,Y_train)
Y_pred = logmodel.predict(X_test)
```

In [39]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,Y_pred)
```

Out[39]:
```
array([[28810,     0],
       [ 3171,     9]])
```

In [40]:
```
# Accuracy:
1 - (Y_pred - Y_test ).abs().mean()
```
Out[40]:
```
0.9008752735229759
```

In [41]:
```
import sklearn
sklearn.metrics.precision_score(Y_test,Y_pred)
```
Out[41]:
```
1.0
```

In [42]:
```
sklearn.metrics.recall_score(Y_test,Y_pred)
```
Out[42]:
```
0.0028301886792452830
```

In [43]:
```
y_proba = logmodel.predict_proba(X_test)[:,1]
```

In [44]:
```
sklearn.metrics.roc_auc_score(Y_test,y_proba)
```
Out[44]:
```
0.5999577474627739
```

In [45]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test,Y_pred))
```
```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95     28810
           1       1.00      0.00      0.01      3180

   micro avg       0.90      0.90      0.90     31990
   macro avg       0.95      0.50      0.48     31990
weighted avg       0.91      0.90      0.85     31990
```

Random Forest Classifier Q1 - Identify the factors that influence high returns

In [46]:
```
from sklearn.ensemble import RandomForestClassifier
cl = RandomForestClassifier(random_state=2)
cl.fit(X,Y)
```
Out[46]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=2, verbose=0, warm_start=Fals
e)
```

In [47]:
```
cl.score(X_test, Y_test)
```
Out[47]:
```
0.8913723038449516
```

In [48]:
```
import pandas as pd
imp_features = pd.DataFrame(cl.feature_importances_,index = X_train.colu
mns,\
                                   columns=['importance']).sort_values(
'importance',ascending=False)
```

In [49]:
```
imp_features.reset_index()
```

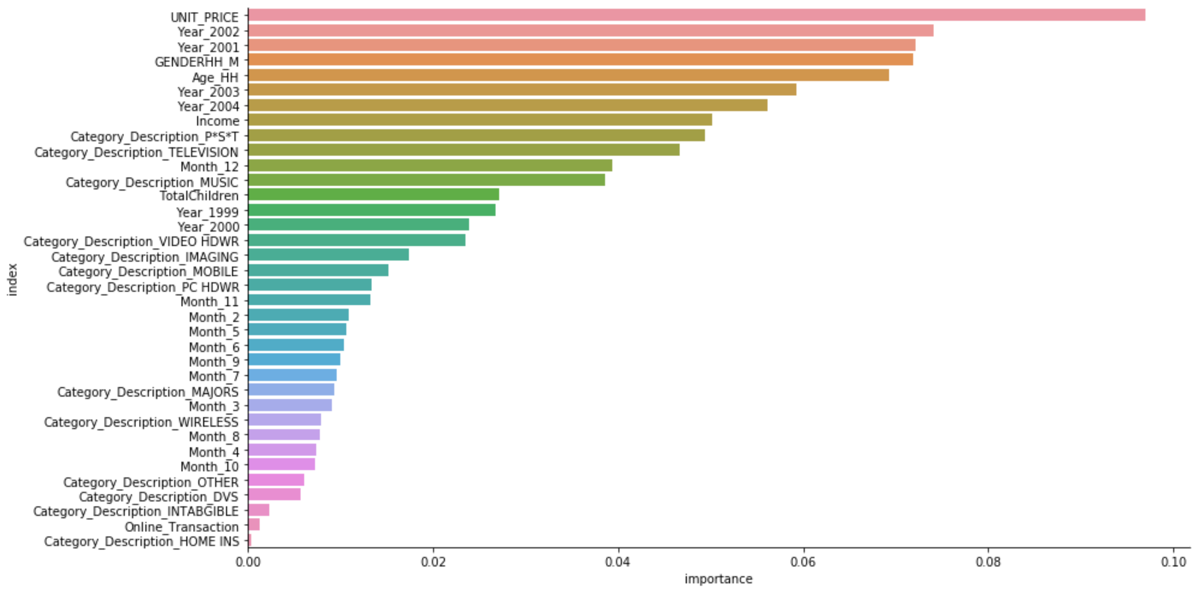
Out[49]:

	index	importance
0	UNIT_PRICE	0.096950
1	Year_2002	0.074035
2	Year_2001	0.072107
3	GENDERHH_M	0.071940
4	Age_HH	0.069239
5	Year_2003	0.059270
6	Year_2004	0.056117
7	Income	0.050157
8	Category_Description_P*S*T	0.049372
9	Category_Description_TELEVISION	0.046639
10	Month_12	0.039334
11	Category_Description_MUSIC	0.038575
12	TotalChildren	0.027195
13	Year_1999	0.026757
14	Year_2000	0.023926
15	Category_Description_VIDEO HDWR	0.023447
16	Category_Description_IMAGING	0.017400
17	Category_Description_MOBILE	0.015215
18	Category_Description_PC HDWR	0.013324
19	Month_11	0.013278
20	Month_2	0.010843
21	Month_5	0.010685
22	Month_6	0.010345
23	Month_9	0.010009
24	Month_7	0.009599
25	Category_Description_MAJORS	0.009282
26	Month_3	0.009012
27	Category_Description_WIRELESS	0.007859
28	Month_8	0.007745
29	Month_4	0.007345
30	Month_10	0.007201
31	Category_Description_OTHER	0.006061
32	Category_Description_DVS	0.005707
33	Category_Description_INTABGIBLE	0.002350

	index	importance
34	Online_Transaction	0.001303
35	Category_Description_HOME INS	0.000374

In [50]:
```
feat=imp_features.reset_index()
```

In [51]:
```
ft=sns.catplot(data=feat,x='importance',y='index',aspect=2,kind='bar',he
ight=7)
```



predict on the test set

In [52]:
```
y_pred = cl.predict(X_test)
```

collect scores

Confusion matrix

In [53]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,y_pred)
```

Out[53]:
```
array([[28047,   763],
       [ 2712,   468]])
```

Accuracy

In [54]:
```
1 - (y_pred - Y_test).abs().mean()
```
Out[54]: 0.8913723038449516

Precision

In [55]:
```
import sklearn
sklearn.metrics.precision_score(Y_test,y_pred)
```
Out[55]: 0.380178716490658

Recall

In [56]:
```
sklearn.metrics.recall_score(Y_test,y_pred)
```
Out[56]: 0.1471698113207547

AUC score

In [57]:
```
y_proba = cl.predict_proba(X_test)[:,1]
```
In [58]:
```
sklearn.metrics.roc_auc_score(Y_test,y_proba)
```
Out[58]: 0.6392544299127443

In [59]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test,y_pred))
```
```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94     28810
           1       0.38      0.15      0.21      3180

   micro avg       0.89      0.89      0.89     31990
   macro avg       0.65      0.56      0.58     31990
weighted avg       0.86      0.89      0.87     31990
```

Cross-validation

In [60]:
```
cl
```

Out[60]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=2, verbose=0, warm_start=Fals
e)
```

In [61]:
```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10,random_state=0,shuffle=True)
sklearn.model_selection.cross_val_score(cl,X,Y,cv=kf,scoring='roc_auc').
mean()
```

Out[61]:
```
0.961616392000173
```

In [62]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

clfs = [DecisionTreeClassifier(), sk.ensemble.RandomForestClassifier(n_j
obs=-1),
        sk.linear_model.LogisticRegression(n_jobs=-1),sk.tree.DecisionTr
eeClassifier()]
```

Let's find the best one in terms of average AUC

In [63]:
```
#Finding the best Classifier between Decision Tree, Logistic and Random
 Forest Classifier
maxAUC = -1
bestCL = ''
for cl in clfs:
    kf = KFold(n_splits=10,random_state=2,shuffle=True)
    auc = sklearn.model_selection.cross_val_score(cl,X,Y,cv=kf,scoring=
'roc_auc').mean()
    if auc > maxAUC:
        bestCl = cl
        maxAUC = auc
print (str(bestCl) + ': ' +str(maxAUC))
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=-
1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False): 0.9618537960135136
```

Q2: Identify the Key Customer Segments using Behavioral Segmentation

Cleaning Data

In [64]:
```
# Filtered purchase dataset only for selected columns
data_pur_fltr=data3[['Household_ID','Transaction_NBR','Final_Date','ORIG
INAL_TICKET_NBR','EXTENDED_PRICE','Quantity']]
```

In [65]:
```
# Creating a copy for analysis
df = data_pur_fltr.copy()
```

In [66]:
```
# Setting a reference date for analysis
Ref_Date = dtm.datetime(2004,11,30)
print(Ref_Date)
```

```
2004-11-30 00:00:00
```

RFM Analysis

RFM (Recency, Frequency, Monetary) analysis is a behavioral segmentation technique to divide customers based on their past transaction history.

On the basis of how recently, how often and how much did they buy, it helps divide customers into various categories to identify the most profitable customers.

In [67]:
```
# Calculating
# - Recency: No. of days passed since the last purchase
# - Frequency: No. of purchases made in the period of relationship with
 the brand
# - Monetary: Total purchase amount for a household in the period of rel
ationship

RFM = df.groupby(['Household_ID']).agg({'Final_Date': lambda date: (Ref_
Date - date.max()).days,
                                        'ORIGINAL_TICKET_NBR': lambda nu
m: num.nunique(),
                                        'EXTENDED_PRICE': lambda price:
price.sum(),
                                        'Quantity': lambda quant: quant.
sum()})
```

In [68]:
```
# Change the name of columns
RFM.columns=['Recency','Frequency', 'Monetary','Quantity']
```

In [69]:
```
RFM.head()
```

Out[69]:

	Recency	Frequency	Monetary	Quantity
Household_ID				
100003544	543	1.0	99.97	1
100012312	1421	1.0	29.98	2
100016237	1408	1.0	89.99	1
100022945	1060	3.0	628.91	9
100022976	715	2.0	849.98	2

Removing Negative Monetary Values

In [70]:
```
RFM_OLD=RFM.copy()
```

In [71]:
```
RFM=RFM[RFM.Monetary>0]
```

RFM Score: Based on the values of factors - Recency, Frequency, and Monetary, we calculated scores namely Recency_Score, Frequency_Score & Monetary_Score, for each household. The final RFM score will help us identify households that are our most profitable customers. Customers having lowest recency and highest monetary and frequency values are considered as top customers.

- Recency: Customers with the **lowest** recency are assigned a score 1.
- Frequency: Customers with the **highest** frequency are assigned a score of 1.
- Monetary: Customers with the **highest** monetary value are assigned a score of 1.

Generating Quantile Scores (Normalization)

In [72]:
```
RFM['Recency_Score'] = pd.qcut(RFM['Recency'], 4, ['1','2','3','4'], duplicates='drop')
```

In [73]:
```
RFM['Frequency_Score'] = pd.qcut(RFM['Frequency'].rank(method='first'), 4, ['4','3','2','1'], duplicates='drop')
```

In [74]:
```
RFM['Monetary_Score'] = pd.qcut(RFM['Monetary'], 4, ['4','3','2','1'], duplicates='drop')
```

In [75]:
```
RFM.head(20)
```

Out[75]:

	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	Mone
Household_ID							
100003544	543	1.0	99.97	1	2	4	
100012312	1421	1.0	29.98	2	4	4	
100016237	1408	1.0	89.99	1	4	4	
100022945	1060	3.0	628.91	9	3	1	
100022976	715	2.0	849.98	2	2	2	
100024091	1053	1.0	3002.97	6	3	4	
100024909	903	3.0	3030.78	22	2	1	
100025614	1942	1.0	114.98	2	4	4	
100025901	224	10.0	5504.71	17	1	1	
100026342	394	1.0	2208.83	18	1	4	
100031891	1338	2.0	119.98	2	3	2	
100033164	486	1.0	480.94	2	2	4	
100033277	1972	2.0	618.90	6	4	2	
100033717	1079	1.0	69.99	1	3	4	
100035849	39	1.0	2544.86	12	1	4	
100037757	1340	4.0	263.94	8	3	1	
100039371	563	2.0	2134.93	6	2	2	
100040028	1425	1.0	264.96	2	4	4	
100041095	1591	1.0	257.95	6	4	4	
100041276	1424	1.0	1378.98	4	4	4	

In [76]:
```
# RFM Final Score
RFM['RFM_Class'] = RFM.Recency_Score.astype(str)+ RFM.Frequency_Score.as
type(str) + RFM.Monetary_Score.astype(str)
```

In [77]:
```
RFM.head(10)
```

Out[77]:

	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	Mone
Household_ID							
100003544	543	1.0	99.97	1	2	4	
100012312	1421	1.0	29.98	2	4	4	
100016237	1408	1.0	89.99	1	4	4	
100022945	1060	3.0	628.91	9	3	1	
100022976	715	2.0	849.98	2	2	2	
100024091	1053	1.0	3002.97	6	3	4	
100024909	903	3.0	3030.78	22	2	1	
100025614	1942	1.0	114.98	2	4	4	
100025901	224	10.0	5504.71	17	1	1	
100026342	394	1.0	2208.83	18	1	4	

Clustering (K-Means)

In [78]:
```
RFM['Recency_Score'] = pd.to_numeric(RFM['Recency_Score'])
RFM['Frequency_Score'] = pd.to_numeric(RFM['Frequency_Score'])
RFM['Monetary_Score'] = pd.to_numeric(RFM['Monetary_Score'])
```

In [79]:
```
RFM_Cluster = RFM[['Recency_Score', 'Frequency_Score', 'Monetary_Score'
]]
```

Finding the optimal K

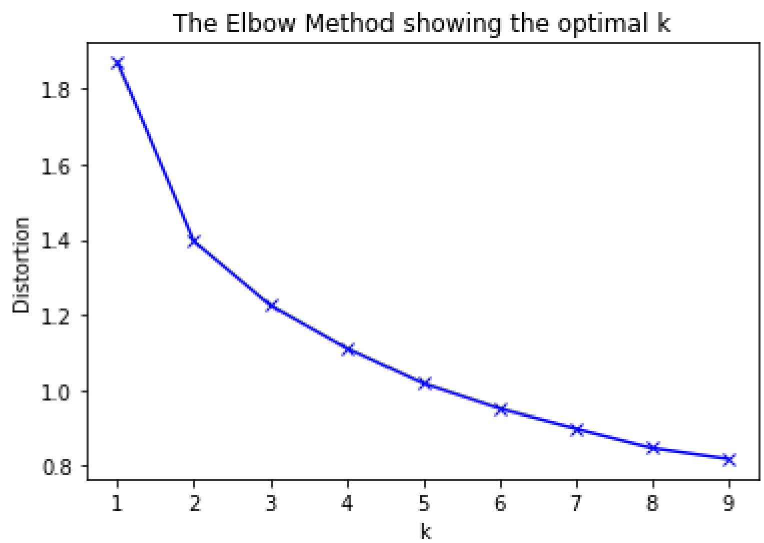
Elbow Plot

In [80]:
```
from scipy.spatial.distance import cdist
```

In [81]:
```
from sklearn.cluster import KMeans
```

In [82]:
```
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(RFM_Cluster)
    kmeanModel.fit(RFM_Cluster)
    distortions.append(sum(np.min(cdist(RFM_Cluster, kmeanModel.cluster_centers_, 'euclidean'), axis=1)) / RFM_Cluster.shape[0])
```

In [83]:
```
# Plot the elbow
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```



When K increases, the centroids are closer to the clusters centroids.

The improvements will decline, at some point rapidly, creating the elbow shape.

That point is the optimal value for K. In the image above, K=6.

Silhouette score

In [84]:
```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.cluster import Birch
from sklearn.cluster import AgglomerativeClustering

bestSil = -1
for k in range(2,10):
    clus = [KMeans(n_clusters=k,n_jobs=-1), Birch(n_clusters=k),
            AgglomerativeClustering(n_clusters=k)]
    for cl in clus:
        res = cl.fit(RFM_Cluster)
        sil = metrics.silhouette_score(RFM_Cluster, res.labels_)
        print (str(cl)[:10] + ' with k =' +str(k) + ": " + str(round(sil
,4)))
        if (sil > bestSil):
            bestSil = sil
            bestCl = cl
```

```
KMeans(alg with k =2: 0.3702
Birch(bran with k =2: 0.3037
Agglomerat with k =2: 0.292
KMeans(alg with k =3: 0.3504
Birch(bran with k =3: 0.2613
Agglomerat with k =3: 0.2846
KMeans(alg with k =4: 0.3547
Birch(bran with k =4: 0.2421
Agglomerat with k =4: 0.2864
KMeans(alg with k =5: 0.3663
Birch(bran with k =5: 0.2742
Agglomerat with k =5: 0.3472
KMeans(alg with k =6: 0.3748
Birch(bran with k =6: 0.2773
Agglomerat with k =6: 0.3395
KMeans(alg with k =7: 0.3643
Birch(bran with k =7: 0.3095
Agglomerat with k =7: 0.3554
KMeans(alg with k =8: 0.3704
Birch(bran with k =8: 0.3044
Agglomerat with k =8: 0.3727
KMeans(alg with k =9: 0.3787
Birch(bran with k =9: 0.3225
Agglomerat with k =9: 0.3767
```

Using k-means to find 5 Clusters

In [85]:
```
from sklearn.cluster import KMeans
#RFM_Cluster.sample(random_state=0,n=4)
clu = KMeans(n_clusters=5,random_state=0)
clu.fit(RFM_Cluster)
RFM_Cluster = RFM_Cluster.copy()
RFM['Cluster'] = clu.labels_
```

In [86]:
```
RFM.groupby('Cluster').mean()
```

Out[86]:

Cluster	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	Mon
0	498.319797	1.048541	654.839775	6.528871	1.509201	3.406409	
1	1132.506550	1.790393	417.314312	4.157205	2.939956	1.608352	
2	1410.375196	1.007825	171.578697	2.078834	3.581182	3.346831	
3	395.816759	3.350645	1956.067011	14.567587	1.414365	1.401105	
4	1258.895197	1.036125	1517.477829	6.497420	3.213974	3.313617	

Analysis

As we can see from the clusters,

- In cluster 0, customers have bought recently, purchased frequently and also have highest monetary value.

These are the **Loyal high value customers**.

- In cluster 3, customers have bought recently, purchased only once but have high monetary value. These are

Infrequent high value customers.

- In cluster 4, customers havent shopped since a long time, but made high value purchases in the past. These

are **churned customers.**

- In clusters 1 and 2, low value, infrequent customers who made a transaction long time back.
- Clusters of interest are:
- **0-Loyal High Value**
- **3-Infrequent High Value**
- **4-Churned High Value**

Using k-means to find 6 Clusters

In [87]:
```
from sklearn.cluster import KMeans
#RFM_Cluster.sample(random_state=0,n=4)
clu = KMeans(n_clusters=6,random_state=0)
clu.fit(RFM_Cluster)
RFM_Cluster = RFM_Cluster.copy()
RFM['Cluster'] = clu.labels_## Analysis
```

In [88]:
```
RFM.groupby('Cluster').mean()
```

Out[88]:

Cluster	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	Mon
0	460.132339	1.000000	1689.133292	11.666506	1.490375	3.503850	
1	1391.714131	1.000000	179.075199	2.124376	3.535272	3.494465	
2	592.420583	1.269515	226.069546	3.809709	1.662136	2.801553	
3	426.988182	3.429273	1811.898622	13.734364	1.473091	1.293455	
4	1381.114690	1.044412	1326.717672	5.388482	3.492435	3.272328	
5	1362.087353	1.586050	428.012060	3.744133	3.484355	1.721317	

Analysis: Based on Recency_Score, Frequency_Score, Monetary_Score

As we can see from the clusters,
- In cluster 3, customers have bought recently, purchased frequently and also have highest monetary value. These are the **loyal high value customers.**
- In cluster 0, customers havent shopped since a long time, but made frequent high value purchases in the past. These are **Infrequent High Value**.
- In cluster 4, customers havent shopped since a long time, but made high value purchases in the past. These are **Churned high value customers.**
- In cluster 2, customers have bought recently and a few times but have low monetary value. These are **Frequent low value customers.**
- In clusters 1 and 5, low value and either infrequent or havent made transaction since long time.
- Clusters of interest are:
- **3-Loyal High Value**
- **0-Infrequent High Value**
- **4-Churned High Value**
- **2-Infrequent Low Value**

Demographic Means of Clusters

In [89]:
```
datademo=data3.copy()
```

In [90]:
```
datademo.GENDERHH.value_counts()
```

Out[90]:
```
M    96659
F    48373
U    27713
Name: GENDERHH, dtype: int64
```

In [91]:
```
datademo['Bin_GENDERHH']= ((datademo.GENDERHH == 'M') | (datademo.GENDER
HH == 'U'))*1
```

In [92]:
```
datademo.Bin_GENDERHH.value_counts()
```

Out[92]:
```
1    124372
0     48373
Name: Bin_GENDERHH, dtype: int64
```

In [93]:
```
Demo = datademo.groupby(['Household_ID']).agg({'Online_Transaction': 'su
m',
                                          'Age_HH':'max',
                                          'Income': 'max',
                                          'Bin_GENDERHH': 'max',
                                          'TotalChildren': 'max',
                                          'EXTENDED_PRICE':'sum' })
```

In [94]:
```
Demo_Final=Demo[Demo.EXTENDED_PRICE>0]
```

In [95]:
```
Demo_Final.head()
```

Out[95]:

	Online_Transaction	Age_HH	Income	Bin_GENDERHH	TotalChildren	EXTENDED_
Household_ID						
100003544	0	28.000000	6.00000	1	0	
100012312	0	24.000000	1.00000	1	0	
100016237	0	48.261417	5.91973	0	0	
100022945	0	44.000000	5.00000	1	1	
100022976	0	54.000000	7.00000	1	0	

In [96]:
```
RFM.head()
```

Out[96]:

	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	Mone
Household_ID							
100003544	543	1.0	99.97	1	2	4	
100012312	1421	1.0	29.98	2	4	4	
100016237	1408	1.0	89.99	1	4	4	
100022945	1060	3.0	628.91	9	3	1	
100022976	715	2.0	849.98	2	2	2	

In [97]:
```
merged =Demo_Final.merge(RFM, on='Household_ID')
```

In [98]:
```
merged.head().T
```

Out[98]:

Household_ID	100003544	100012312	100016237	100022945	100022976
Online_Transaction	0	0	0	0	0
Age_HH	28	24	48.2614	44	54
Income	6	1	5.91973	5	7
Bin_GENDERHH	1	1	0	1	1
TotalChildren	0	0	0	1	0
EXTENDED_PRICE	99.97	29.98	89.99	628.91	849.98
Recency	543	1421	1408	1060	715
Frequency	1	1	1	3	2
Monetary	99.97	29.98	89.99	628.91	849.98
Quantity	1	2	1	9	2
Recency_Score	2	4	4	3	2
Frequency_Score	4	4	4	1	2
Monetary_Score	4	4	4	2	2
RFM_Class	244	444	444	312	222
Cluster	2	1	1	5	3

In [99]:
```
merged.groupby('Cluster').size()
```

Out[99]:
```
Cluster
0    2078
1    4607
2    2575
3    5500
4    2049
5    3068
dtype: int64
```

Analysis: Based on Recency_Score, Frequency_Score, Monetary_Score

- Clusters of interest are:
- **3-Loyal High Value** have high income and highest online transactions
- **0-Infrequent High Value** have high income and do not shop online and have comparatively more children
- **4-Churned High Value** have slighltly higher income and lowest online transactions
- **2-Infrequent Low Value** have moderate income and also have children and also shop online

In [100]:
```
merged.Online_Transaction.unique()
```

Out[100]:
```
array([  0,   4,   2,  10,   3,   1,   5,   8,   6,   7,  29,   9,  12,
        34,  43,  15,  20, 107,  11,  58,  19,  21,  17,  56,  25])
```

In [101]:
```
highvalue=merged[(merged.Cluster==3) | (merged.Cluster == 0) | (merged.C
luster == 4)]
```

In [102]:
```
def num2clus(desc):
    if desc==3:
        return 'Loyal High Value'
    elif desc==0:
        return 'Infrequent High Value'
    elif desc==4:
        return 'Churned High Value'
```

In [103]:
```
highvalue['Cluster_Type'] = highvalue['Cluster'].apply(num2clus)  # Appl
y is fast on Series but v slow on Data Frames
```

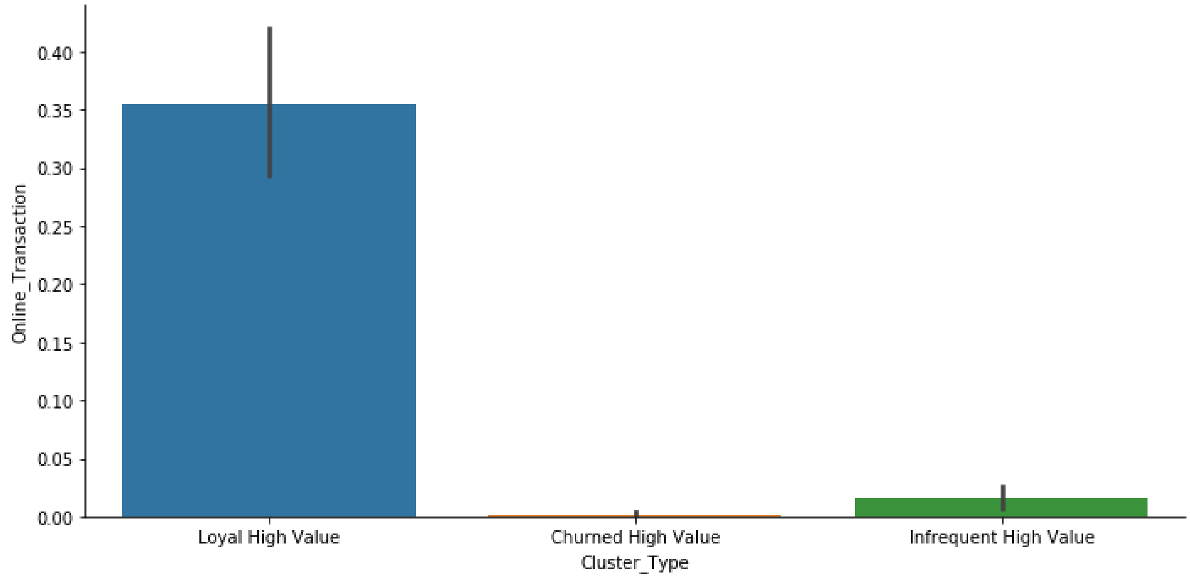
In [104]:
```
highvalue.head()
```

Out[104]:

	Online_Transaction	Age_HH	Income	Bin_GENDERHH	TotalChildren	EXTENDED_P
Household_ID						
100022976	0	54.0	7.0	1	0	8
100024091	0	44.0	5.0	1	0	30
100024909	0	56.0	7.0	1	0	30
100025901	0	72.0	7.0	1	0	55
100026342	0	48.0	9.0	0	3	22

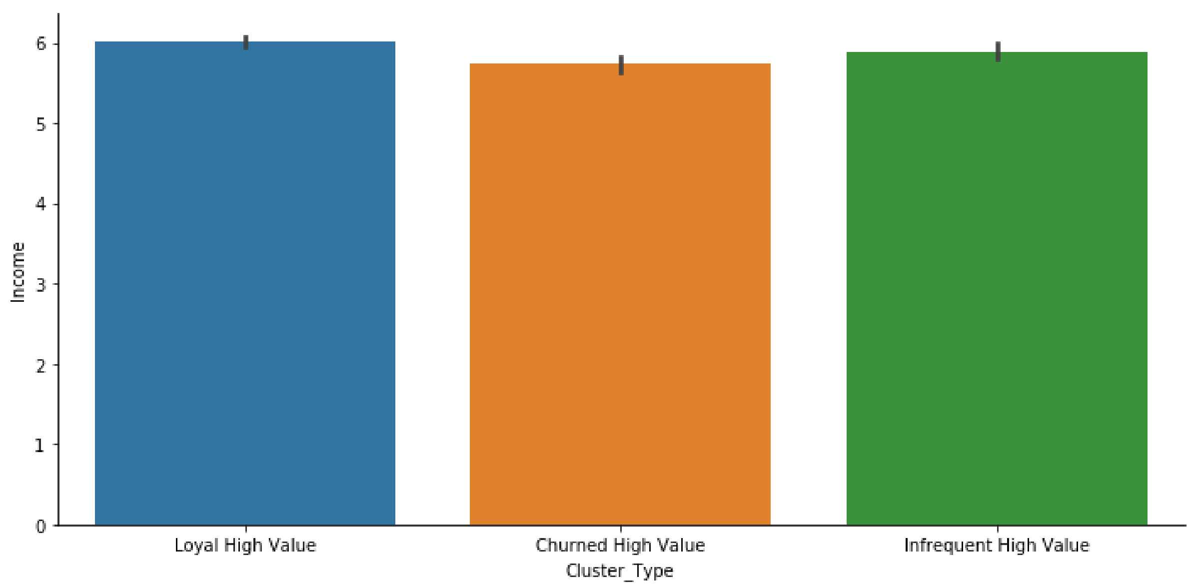
In [105]:
```
import seaborn as sns
sns.catplot(x='Cluster_Type',y='Online_Transaction',data=highvalue,kind=
'bar',aspect = 2)
```

Out[105]: <seaborn.axisgrid.FacetGrid at 0x1c43d13c18>



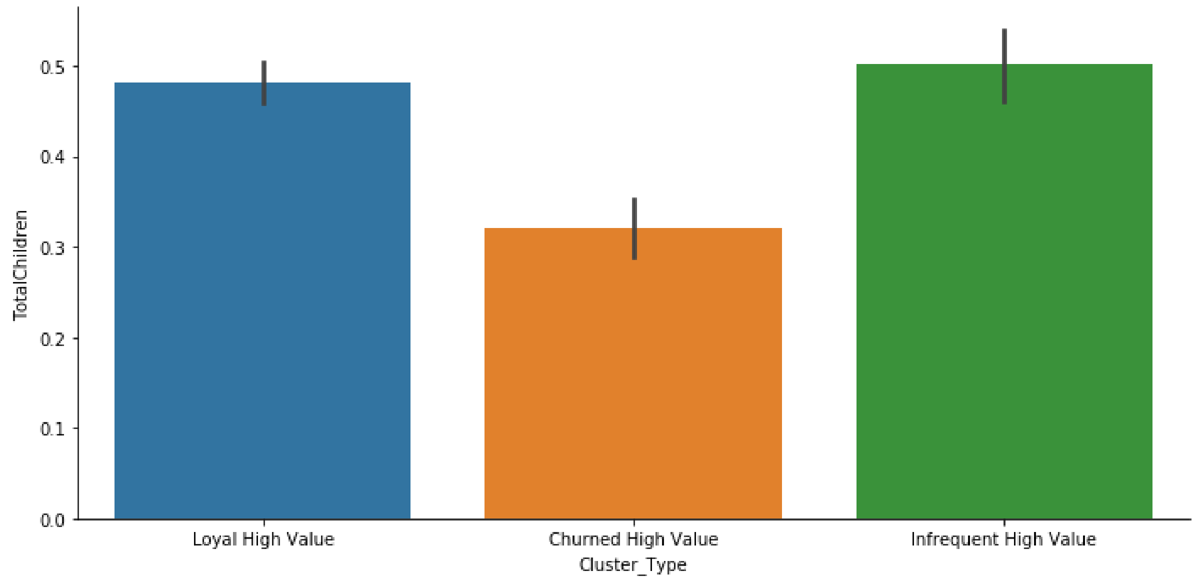
In [106]:
```
import seaborn as sns
sns.catplot(x='Cluster_Type',y='Income',data=highvalue,kind='bar',aspect
= 2)
```

Out[106]: <seaborn.axisgrid.FacetGrid at 0x1a2a2f4d68>



In [107]:
```
import seaborn as sns
sns.catplot(x='Cluster_Type',y='TotalChildren',data=highvalue,kind='bar'
,aspect = 2)
```

Out[107]: <seaborn.axisgrid.FacetGrid at 0x1a357cc7b8>



CUSTOMER LIFETIME VALUE

Customer Lifetime Value is a monetary value that represents the amount of revenue that a customer will give the company over the period of the relationship.

CLV = ((Average Order Value x Purchase Frequency)/Churn Rate) x Profit margin

Average Order Value(AOV): The Average Order value is the ratio of your total revenue and the total number of orders. AOV represents the mean amount of revenue that the customer spends on an order.

Average Order Value = Total Revenue / Total Number of Orders

In [108]:
```
RFM['Avg_Order_Value']=RFM['Monetary']/RFM['Frequency']
RFM.head()
```

Out[108]:

	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	Mone
Household_ID							
100003544	543	1.0	99.97	1	2	4	
100012312	1421	1.0	29.98	2	4	4	
100016237	1408	1.0	89.99	1	4	4	
100022945	1060	3.0	628.91	9	3	1	
100022976	715	2.0	849.98	2	2	2	

Purchase Frequency(PF): Purchase Frequency is the ratio of the total number of orders upon the total number of customer. It represents the average number of orders placed by each customer.

Purchase Frequency = Total Number of Orders / Total Number of Customers

In [109]:
```
Purchase_Frequency=sum(RFM['Frequency'])/RFM.shape[0]
Purchase_Frequency
```

Out[109]: 1.8021331186798812

Repeat Rate: Repeat rate can be defined as the ratio of the number of customers with more than one order to the number of unique customers.

Repeat Rate = Number of Customers with more than one order/ Number of Unique Customers

In [110]:
```
# Repeat Rate
Repeat_Rate=RFM[RFM.Frequency > 1].shape[0]/RFM.shape[0]
Repeat_Rate
```

Out[110]: 0.32177894048397643

Churn Rate: Churn Rate is the percentage of customers who have not ordered again.

Churn Rate = 1-Repeat Rate

In [111]:
```
# Churn Rate
Churn_Rate=1-Repeat_Rate
Churn_Rate
```

Out[111]: 0.6782210595160236

Let's assume the business is earning approximately 5% profit on the total sale.

In [112]:
```
# Profit Margin
RFM['Profit_Margin']=RFM['Monetary']*0.05
```

In [113]:
```
RFM.head()
```

Out[113]:

Household_ID	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	Mone
100003544	543	1.0	99.97	1	2	4	
100012312	1421	1.0	29.98	2	4	4	
100016237	1408	1.0	89.99	1	4	4	
100022945	1060	3.0	628.91	9	3	1	
100022976	715	2.0	849.98	2	2	2	

Customer Lifetime: Customer Lifetime is the period of time that the customer has been continuously ordering.

Customer Lifetime = 1/Churn Rate

Customer Value = Average Order Value * Purchase Frequency

In [114]:
```
# Customer Lifetime Value
RFM['CLV']=(RFM['Avg_Order_Value']*Purchase_Frequency)/Churn_Rate*RFM['P
rofit_Margin']
```

In [115]:
```
df1 = RFM.groupby('Cluster').mean().reset_index()
df1
```

Out[115]:

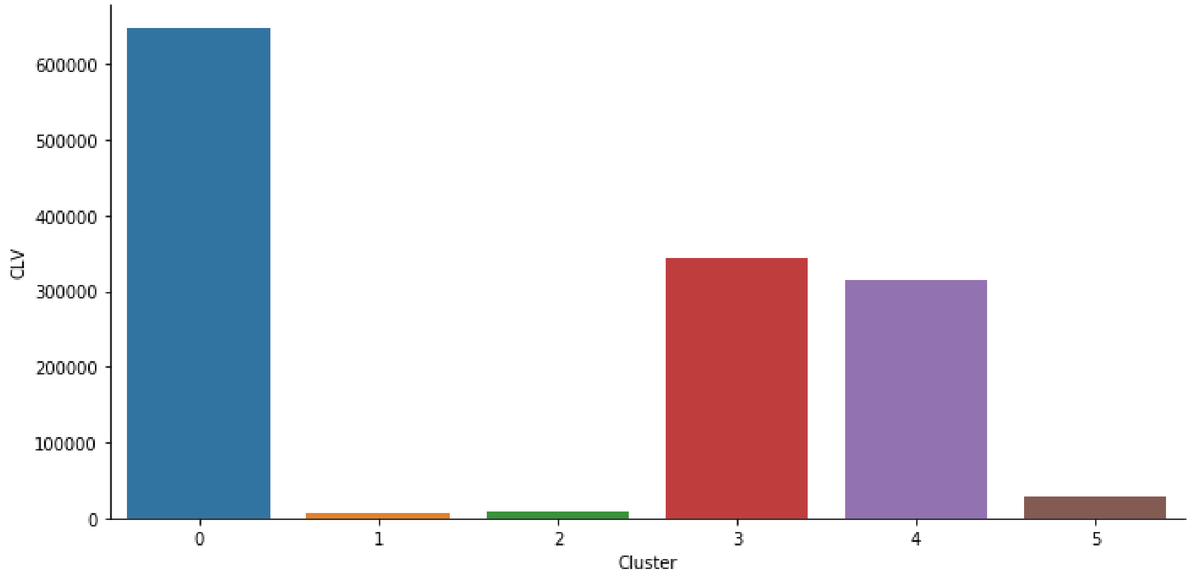
	Cluster	Recency	Frequency	Monetary	Quantity	Recency_Score	Frequency_Score	M
0	0	460.132339	1.000000	1689.133292	11.666506	1.490375	3.503850	
1	1	1391.714131	1.000000	179.075199	2.124376	3.535272	3.494465	
2	2	592.420583	1.269515	226.069546	3.809709	1.662136	2.801553	
3	3	426.988182	3.429273	1811.898622	13.734364	1.473091	1.293455	
4	4	1381.114690	1.044412	1326.717672	5.388482	3.492435	3.272328	
5	5	1362.087353	1.586050	428.012060	3.744133	3.484355	1.721317	

Plotting CLV vs Customer Segments

In [116]:
```
import seaborn as sns

sns.catplot(x='Cluster',y='CLV',data=df1,kind='bar',aspect = 2)
```

Out[116]: <seaborn.axisgrid.FacetGrid at 0x1a35833a90>



Analysis

As we can see from the graph, Cluster 0 and 3 have the highest CLV which are also the high value clusters in RFM.