Predicting Customer Spending on Black Friday

Data cleaning and feature extraction

Import libraries and methods.

In [125]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import statsmodels.api as sm
from sklearn import linear_model, metrics
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import tree
from sklearn.utils import shuffle
```

Read dataset and print features and categories.

In [4]:
```
df = pd.read_csv('C:/Users/Nolan/.PyCharmCE2018.3/config/scratches/Practice/BlackFri

for x in list(df):
    if len(df[x].unique())>21:
        print(x, df[x].unique(), str(len(df[x].unique()))+' categories')
    else:
        print(x, sorted(df[x].unique()))
```

```
User_ID [1000001 1000002 1000003 ... 1004113 1005391 1001529] 5891 categories
Product_ID ['P00069042' 'P00248942' 'P00087842' ... 'P00038842' 'P00295642'
 'P00091742'] 3623 categories
Gender ['F', 'M']
Age ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+']
Occupation [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 2
0]
City_Category ['A', 'B', 'C']
Stay_In_Current_City_Years ['0', '1', '2', '3', '4+']
Marital_Status [0, 1]
Product_Category_1 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
Product_Category_2 [nan, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0,
13.0, 14.0, 15.0, 16.0, 17.0, 18.0]
Product_Category_3 [nan, 3.0, 4.0, 5.0, 6.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.
0, 15.0, 16.0, 17.0, 18.0]
Purchase [ 8370 15200  1422 ... 14539 11120 18426] 17959 categories
```

From the above lists, it is clear that all initial features are categorical.

Check for duplicates.

In [5]:
```
df.duplicated().sum()
```
Out[5]:
```
0
```

No duplicates found.

Next, check the number of missing values.

In [6]:
```
df.isnull().sum()
```
Out[6]:
```
User_ID                            0
Product_ID                         0
Gender                             0
Age                                0
Occupation                         0
City_Category                      0
Stay_In_Current_City_Years         0
Marital_Status                     0
Product_Category_1                 0
Product_Category_2            166986
Product_Category_3            373299
Purchase                           0
dtype: int64
```

The only missing values are in product category 2 and 3, for products that belong in less than 3 categories. The fact that Product_Category_2 does not have the level 1 and Product_Category_3 does not have level 2 suggests the categories are placed in numerical order. This means the order has no meaning, and the important information is simply the categories the product belongs to.

Thus we can dummy code all 3 features and add them together.

In [7]:
```
prodcatdummy1 = pd.get_dummies(df.Product_Category_1).astype(int)
prodcatdummy2 = pd.get_dummies(df.Product_Category_2.fillna(0).astype(int))
prodcatdummy3 = pd.get_dummies(df.Product_Category_3.fillna(0).astype(int))

prodcatdummy = prodcatdummy1

for x in list(prodcatdummy):
    if x in list(prodcatdummy2):
        prodcatdummy[x] += prodcatdummy2[x]
    if x in list(prodcatdummy3):
        prodcatdummy[x] += prodcatdummy3[x]

prodcatdummy=prodcatdummy.rename({x:'PC'+str(x) for x in list(prodcatdummy)},axis=1)

prodcatdummy.head(10)
```

Out[7]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

We can combine the three columns into a single column representing the unique combination of categories.

In [8]:
```
df['Product_Categories']=df['Product_Category_1'].astype(str).str.cat(
    df[['Product_Category_2','Product_Category_3']].astype(str))

combinationdummies =pd.get_dummies(df['Product_Categories'])

print(len(df.Product_Categories.unique()))
```

```
235
```

There are 235 categories in this new feature. To prevent overfitting, this feature might not be used.

Now dummy code the remaining variables.

In [9]:

```
df['Male']=np.where(df['Gender']=='M',1,0)
agedummies = pd.get_dummies(df['Age'])
occupationdummies =pd.get_dummies(df['Occupation'])
citydummies = pd.get_dummies(df['City_Category'])
staydummies =pd.get_dummies(df['Stay_In_Current_City_Years'])
df['Constant']=1
```

Next we can create variables for the popularity of products (count) and different products each customer bought.

In [10]:

```
df['Product_Count']=df['Product_ID'].map(df.groupby('Product_ID')['Purchase'].count(
df['User_Count']=df['User_ID'].map(df.groupby('User_ID')['Purchase'].count())

df[['User_ID','Product_ID','User_Count','Product_Count']].head(10)
```

Out[10]:

	User_ID	Product_ID	User_Count	Product_Count
0	1000001	P00069042	34	221
1	1000001	P00248942	34	570
2	1000001	P00087842	34	99
3	1000001	P00085442	34	334
4	1000002	P00285442	76	200
5	1000003	P00193542	29	606
6	1000004	P00184942	13	1424
7	1000004	P00346142	13	586
8	1000004	P0097242	13	896
9	1000005	P00274942	106	782

Next, we can create a new response of the number purchased. The number purchased may serve as an intermediate response that can be modeled separately if necessary.

In [11]:
```
pricedict = {}
prices = df.groupby('Product_ID')['Purchase'].min()
for x in df.Product_ID.unique():
    pricedict.update({x:prices.loc[x]})
df['Price']=df['Product_ID'].map(pricedict)
df['Number']=round(df['Purchase']/df['Price']).astype(int)
df['User_Sum']=df['User_ID'].map(df.groupby('User_ID')['Purchase'].sum())
df[['Product_ID','Purchase','Price','Number','User_Sum']].head(10)
```

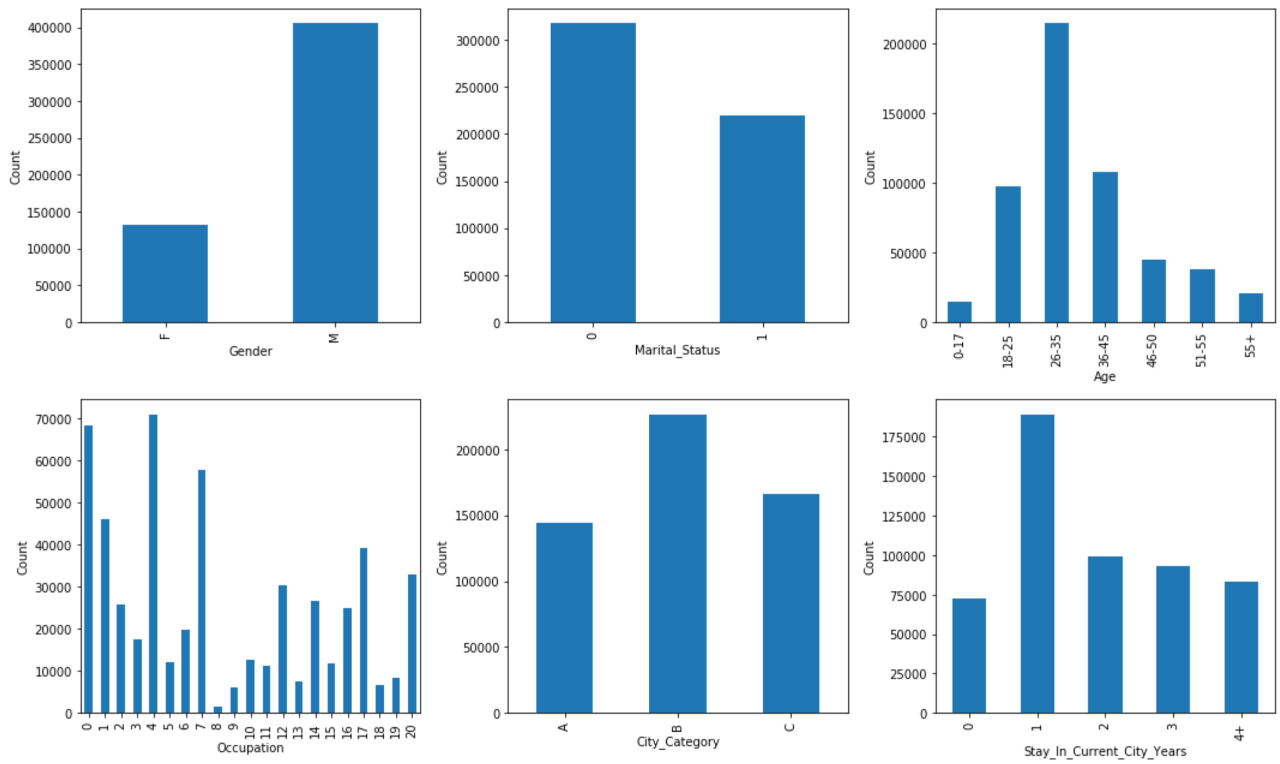
Out[11]:

	Product_ID	Purchase	Price	Number	User_Sum
0	P00069042	8370	2648	3	333481
1	P00248942	15200	3880	4	333481
2	P00087842	1422	346	4	333481
3	P00085442	1057	365	3	333481
4	P00285442	7969	3920	2	810353
5	P00193542	15227	3828	4	341635
6	P00184942	19215	3809	5	205987
7	P00346142	15854	3847	4	205987
8	P0097242	15686	3936	4	205987
9	P00274942	7871	1940	4	821001

Exploratory Data Visualization

The counts of each category are plotted below.

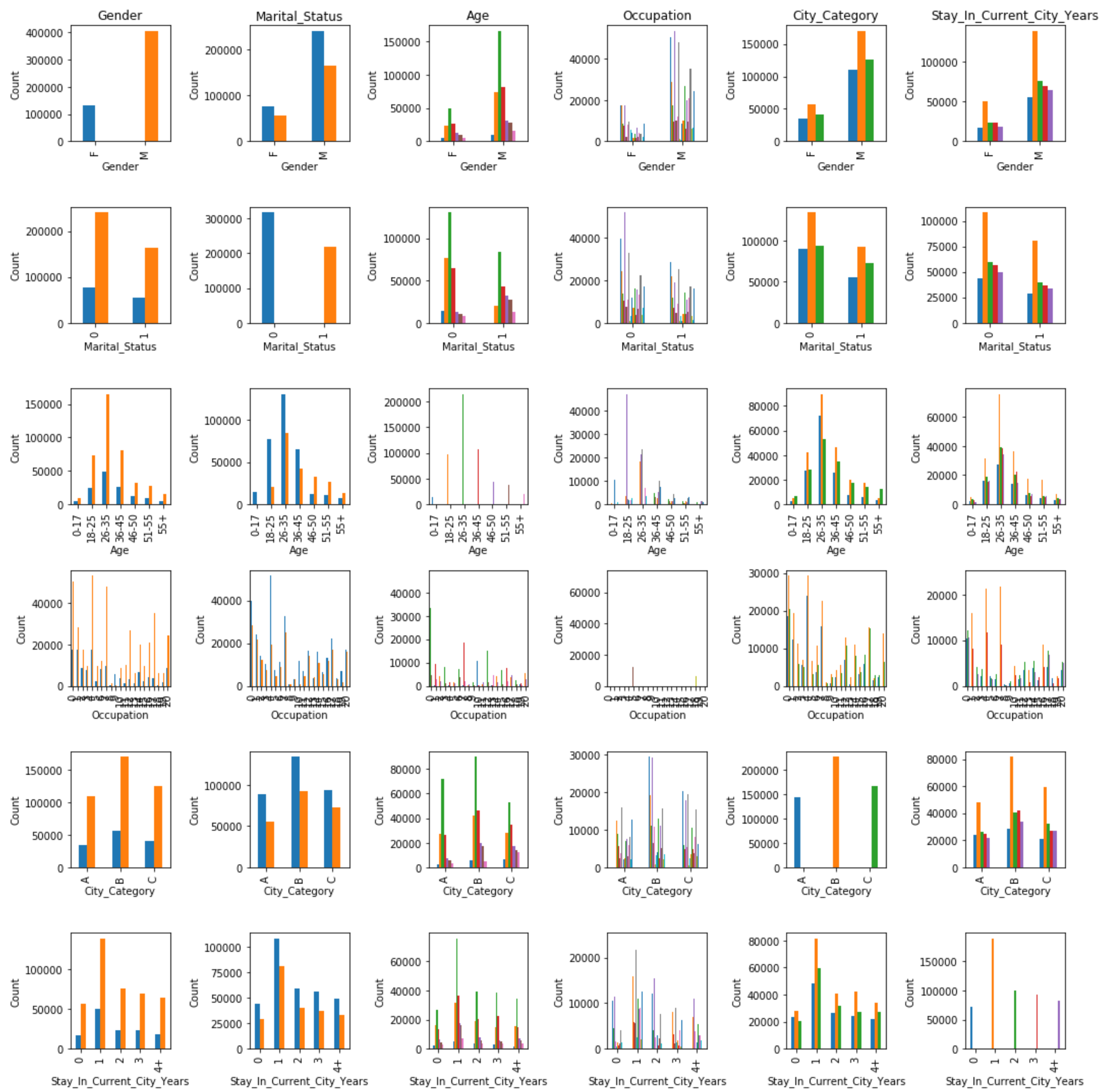
```
In [12]: userplotlist=['Gender','Marital_Status','Age','Occupation','City_Category','Stay_In_
         fig0, ax0 = plt.subplots(2,3)
         for i in range(len(userplotlist)):
             df.groupby(userplotlist[i])['Purchase'].count().plot(kind='bar',ax=ax0[i//3,i%3]
             ax0[i//3,i%3].set_ylabel('Count')
         fig0.set_size_inches(15,9)
         fig0.tight_layout()
```



There is reasonable variability in each of the categorical variables in the customer population. There is no basis for excluding any feature yet.

Now perform a quick visual check for independence/balance of variables. For each level of a variable the distribution of other variable should be similar.

In [13]:
```
fig3, ax3 = plt.subplots(6,6)
for i in range(len(userplotlist)):
    for j in range(len(userplotlist)):
        df.groupby([userplotlist[i],userplotlist[j]])['Purchase'].count().unstack().
        ax3[i,j].set_ylabel('Count')
        if i==0:
            ax3[i,j].set_title(userplotlist[j])
fig3.set_size_inches(15,15)
fig3.tight_layout()
```

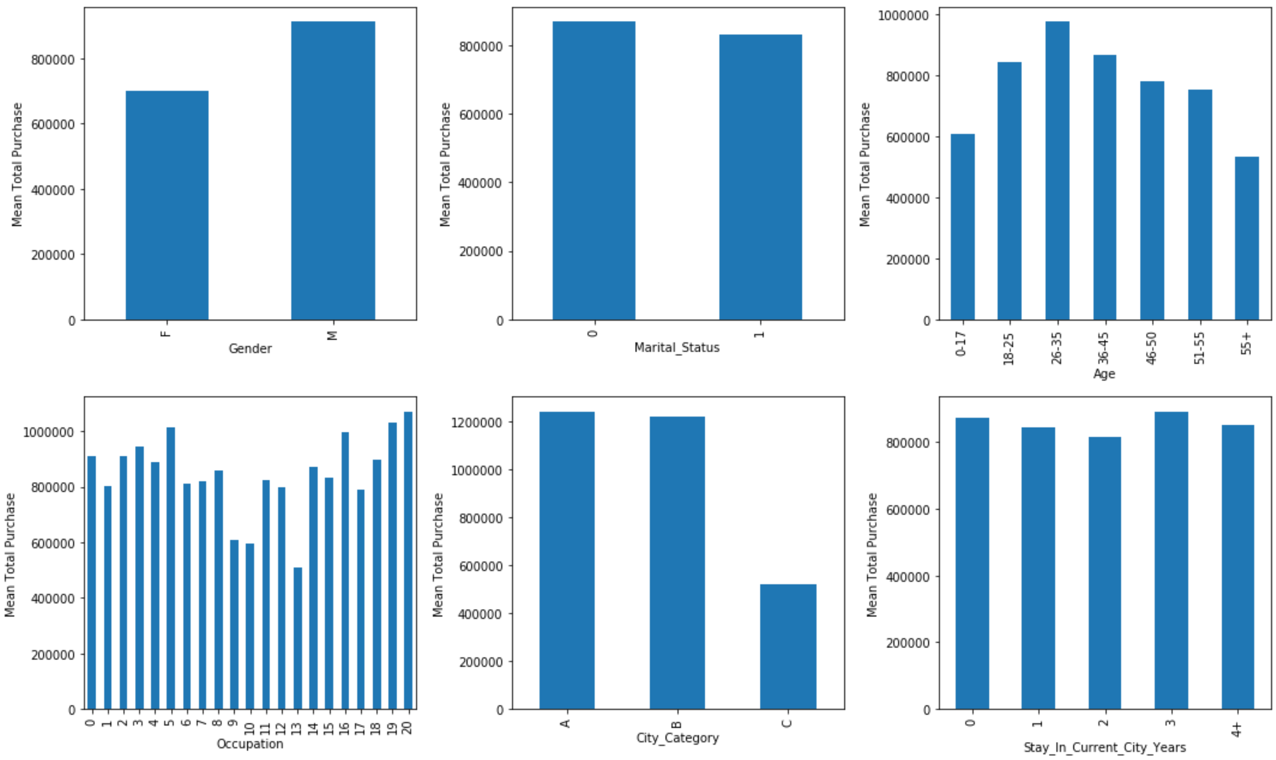


Age and marital status are related (older people tend to be married). Occupation and city category and stay in current city appear related as well. However, the correlations are not so strong that independence is lost (most levels are represented to some degree in each level of the other variable). Therefore there is no basis to combine any variables yet.

The next plot shows the mean total purchase for each categorical variable.

In [14]:

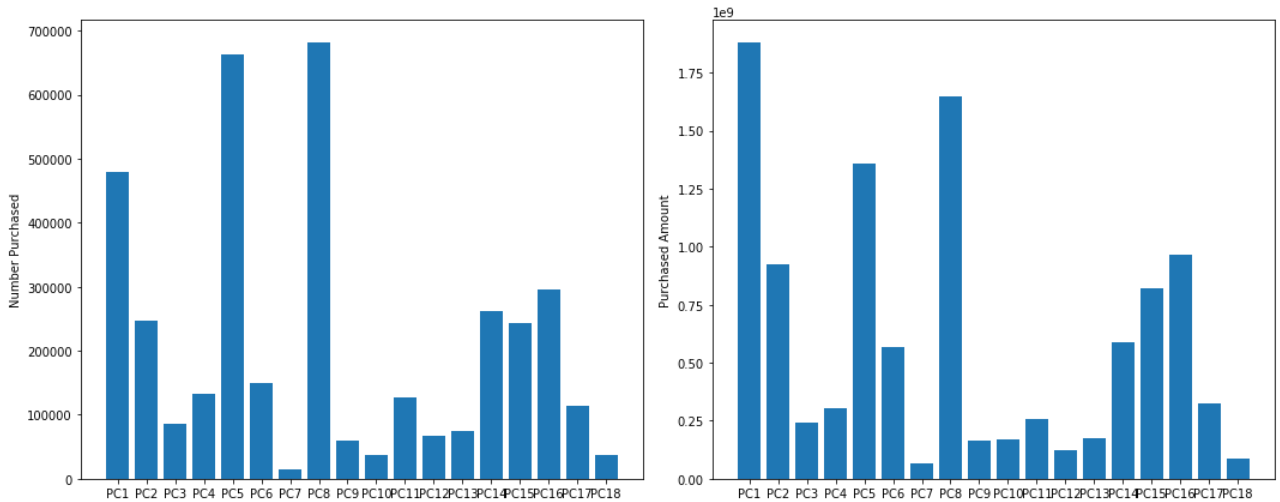
```
fig1, ax1 = plt.subplots(2,3)
sumdf=df[['User_ID','Gender','Marital_Status','Age','Occupation','City_Category','St
for i in range(len(userplotlist)):
    sumdf.groupby(userplotlist[i])['User_Sum'].mean().plot(kind='bar',ax=ax1[i//3,i%
    ax1[i//3,i%3].set_ylabel('Mean Total Purchase')
fig1.set_size_inches(15,9)
fig1.tight_layout()
```



City categories A and B are similar, and marital status and stay in current city appear to have little effect on average spending. However, this may differ when the product categories are considered.

The next plot shows the number and total purchase amount for each category of product.

In [15]:
```
productdf = pd.concat([df[['Product_ID','Number','Purchase']],prodcatdummy],axis=1)
categorynumbertotals = []
categorypurchasetotals = []
for x in list(prodcatdummy):
    categorynumbertotals.append(np.dot(productdf[x],productdf['Number']))
    categorypurchasetotals.append(np.dot(productdf[x],productdf['Purchase']))
fig2, ax2 = plt.subplots(1,2)
ax2[0].bar(list(prodcatdummy),categorynumbertotals)
ax2[0].set_ylabel('Number Purchased')
ax2[1].bar(list(prodcatdummy),categorypurchasetotals)
ax2[1].set_ylabel('Purchased Amount')
fig2.set_size_inches(15,6)
fig2.tight_layout()
```



The number and amount purchased varies greatly by product category.

Feature Selection for Least Squares Regression

First scale the continuous variables.

In [16]:
```
continuous = ['Price','User_Count','Product_Count']
for x in continuous:
    df[x]/=df[x].std()*2
```

Then create the centered design matrix and response vector.

In [17]:
```
X=(pd.concat([df[['Constant','Male','Marital_Status','Price','User_Count','Product_C
occupationdummies[list(occupationdummies)[1:]],citydummies[list(citydummies)[1:]],st
prodcatdummy],axis=1)-0.5)*2
y=df['Purchase']
```

We will first use a linear model to obtain p values.

In [157]:

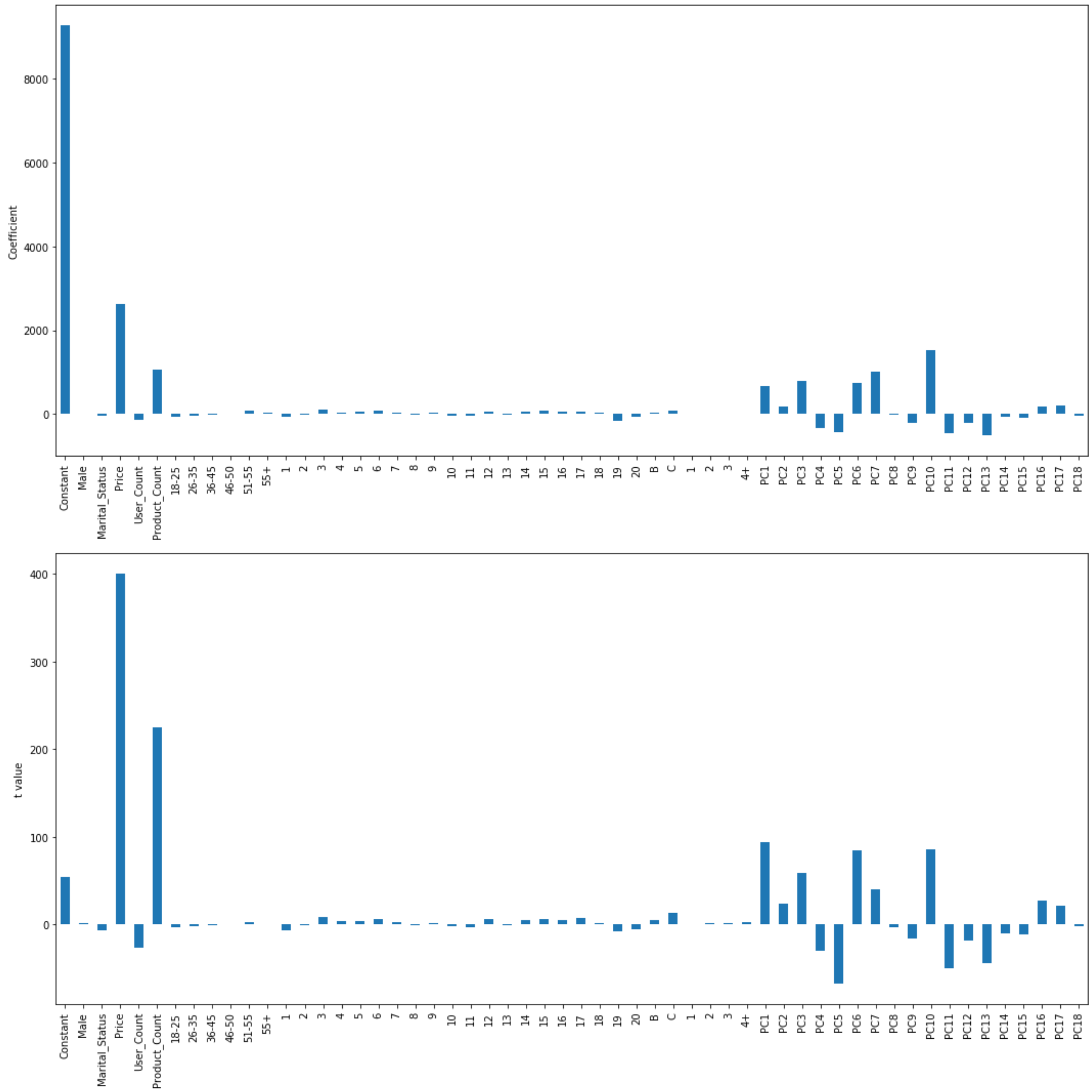
```
model = sm.OLS(y,X)
results=model.fit()
resultssummary = results.summary()
resultssummaryhtml = resultssummary.tables[1].as_html()
resultssummarydf = pd.read_html(resultssummaryhtml,header=0,index_col=0)[0]
print(resultssummary)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Purchase   R-squared:                       0.612
Model:                            OLS   Adj. R-squared:                  0.612
Method:                 Least Squares   F-statistic:                 1.543e+04
Date:                Thu, 11 Apr 2019   Prob (F-statistic):               0.00
Time:                        19:39:44   Log-Likelihood:            -5.0847e+06
No. Observations:              537577   AIC:                         1.017e+07
Df Residuals:                  537521   BIC:                         1.017e+07
Df Model:                          55
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Constant        9273.4215    169.163     54.819      0.000    8941.867    9604.976
Male               8.8494      5.188      1.706      0.088      -1.320      19.018
Marital_Status   -27.6085      4.603     -5.998      0.000     -36.629     -18.588
Price           2619.7751      6.534    400.956      0.000    2606.969    2632.581
User_Count      -134.2451      5.198    -25.825      0.000    -144.434    -124.057
Product_Count   1065.0613      4.736    224.909      0.000    1055.780    1074.343
18-25            -70.2463     21.311     -3.296      0.001    -112.014     -28.478
26-35            -37.8884     21.250     -1.783      0.075     -79.537       3.760
36-45             -3.5074     21.555     -0.163      0.871     -45.754      38.739
46-50             -1.4072     22.514     -0.063      0.950     -45.533      42.719
51-55             74.0105     22.768      3.251      0.001      29.387     118.634
55+               22.7481     24.116      0.943      0.346     -24.518      70.015
1                -60.4847      9.452     -6.399      0.000     -79.010     -41.960
2                -12.2785     11.378     -1.079      0.281     -34.579      10.022
3                119.0292     13.251      8.983      0.000      93.057     145.001
4                 35.9358      9.039      3.975      0.000      18.219      53.653
5                 54.6786     15.416      3.547      0.000      24.464      84.893
6                 82.2209     12.656      6.497      0.000      57.416     107.026
7                 24.9372      8.875      2.810      0.005       7.543      42.332
8                -21.3737     40.340     -0.530      0.596    -100.440      57.692
9                 36.1389     21.034      1.718      0.086      -5.087      77.365
10               -35.4290     22.836     -1.551      0.121     -80.188       9.330
11               -39.2844     15.810     -2.485      0.013     -70.272      -8.297
12                62.4662     10.802      5.783      0.000      41.295      83.638
13               -20.3484     20.342     -1.000      0.317     -60.219      19.522
14                64.0199     11.232      5.700      0.000      42.005      86.035
15                92.5123     15.519      5.961      0.000      62.096     122.929
16                59.4179     11.636      5.106      0.000      36.611      82.225
17                70.0608      9.950      7.041      0.000      50.559      89.563
18                39.4676     20.201      1.954      0.051      -0.125      79.060
19              -149.9148     18.107     -8.279      0.000    -185.404    -114.426
20               -54.7030     10.487     -5.216      0.000     -75.258     -34.148
B                 26.4198      5.424      4.871      0.000      15.789      37.051
C                 91.3146      6.664     13.703      0.000      78.254     104.375
1                  1.5024      6.809      0.221      0.825     -11.842      14.847
2                 11.8786      7.606      1.562      0.118      -3.030      26.787
```

```
3                  10.7198      7.724      1.388      0.165      -4.420      25.859
4+                 19.4464      7.917      2.456      0.014       3.929      34.963
PC1               659.3219      6.984     94.400      0.000     645.633     673.011
PC2               181.5946      7.502     24.208      0.000     166.892     196.297
PC3               801.2992     13.495     59.378      0.000     774.850     827.749
PC4              -320.1572     10.745    -29.795      0.000    -341.218    -299.097
PC5              -418.2902      6.209    -67.363      0.000    -430.461    -406.120
PC6               738.4822      8.704     84.843      0.000     721.422     755.542
PC7              1014.5667     25.407     39.932      0.000     964.769    1064.364
PC8               -20.8859      6.116     -3.415      0.001     -32.872      -8.900
PC9              -204.7667     12.834    -15.955      0.000    -229.922    -179.612
PC10             1518.9864     17.743     85.609      0.000    1484.210    1553.763
PC11             -452.6535      9.166    -49.382      0.000    -470.619    -434.688
PC12             -217.4070     12.321    -17.646      0.000    -241.555    -193.259
PC13             -513.4249     11.568    -44.382      0.000    -536.099    -490.751
PC14              -66.4772      6.484    -10.252      0.000     -79.186     -53.768
PC15              -77.9154      7.061    -11.034      0.000     -91.755     -64.076
PC16              187.3284      6.875     27.247      0.000     173.853     200.803
PC17              204.8190      9.707     21.099      0.000     185.793     223.845
PC18              -35.1863     15.724     -2.238      0.025     -66.005      -4.368
==============================================================================
Omnibus:                    26758.015   Durbin-Watson:                   1.771
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            54499.899
Skew:                          -0.356   Prob(JB):                         0.00
Kurtosis:                       4.388   Cond. No.                         246.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
ecified.
```

In [162]:
```
fig4, ax4 = plt.subplots(2,1)
resultssummarydf['coef'].plot(kind='bar',ax=ax4[0])
resultssummarydf['t'].plot(kind='bar',ax=ax4[1])
ax4[0].set_ylabel('Coefficient')
ax4[1].set_ylabel('t value')
fig4.set_size_inches(15,15)
fig4.tight_layout()
```



It is clear that the purchase amount depends the most on the price, popularity, and category of the product. But the product category effect may differ among different customers. This would be captured by interaction effects.

Because there are so many possible interactions, we will invoke the heirarchy of effects principle and assume significant interactions involve only significant main effects. So first we will check for interactions between significant features. Due to the large number of interactions, this will be done in several stages. The levels are first grouped by categorical feature.

In [19]:
```
sigage = ['18-25','51-55']
sigocc = [1,3,4,5,6,7,11,12,14,15,16,17,18,20]
sigcity = ['B','C']
sigpc = []
for i in range(1,19):
    sigpc.append('PC'+str(i))
sigother = ['Marital_Status','Price','User_Count','Product_Count']
sig = sigother+sigage+sigocc+sigcity+sigpc
```

First interactions between the all features and features in groups with the least number of categories is evaluated.

In [20]:
```
X1 = X.copy()
for i in range(len(sig)):
    for j in range(i+1,len(sig)):
        a=sig[i]
        b=sig[j]
        if not (a in sigage and b in sigage) and not (a in sigocc and b in sigocc)\
            and not (a in sigcity and b in sigcity) and not (a in sigpc and b in sig
            and not (a in sigocc and b in sigpc) and not (a in sigpc and b in sigocc
                X1[str(a)+'x'+str(b)]=X1[a]*X1[b]

y=df['Purchase']

X1.head(10)
```

Out[20]:

	Constant	Male	Marital_Status	Price	User_Count	Product_Count	18-25	26-35	36-45	46-50	...	CxP
0	1.0	-1.0	-1.0	1.120588	-0.804753	-0.381538	-1.0	-1.0	-1.0	-1.0	...	
1	1.0	-1.0	-1.0	2.107206	-0.804753	0.595130	-1.0	-1.0	-1.0	-1.0	...	
2	1.0	-1.0	-1.0	-0.722914	-0.804753	-0.722951	-1.0	-1.0	-1.0	-1.0	...	
3	1.0	-1.0	-1.0	-0.707698	-0.804753	-0.065310	-1.0	-1.0	-1.0	-1.0	...	
4	1.0	1.0	-1.0	2.139239	-0.563565	-0.440305	-1.0	-1.0	-1.0	-1.0	...	-
5	1.0	1.0	-1.0	2.065563	-0.833466	0.695875	-1.0	1.0	-1.0	-1.0	...	
6	1.0	1.0	1.0	2.050347	-0.925347	2.985025	-1.0	-1.0	-1.0	1.0	...	
7	1.0	1.0	1.0	2.080778	-0.925347	0.639905	-1.0	-1.0	-1.0	1.0	...	
8	1.0	1.0	1.0	2.152052	-0.925347	1.507432	-1.0	-1.0	-1.0	1.0	...	
9	1.0	1.0	1.0	0.553603	-0.391288	1.188406	-1.0	1.0	-1.0	-1.0	...	

10 rows × 338 columns

In [21]:
```
model = sm.OLS(y,X1)
results=model.fit()
summary = results.summary()
summaryhtml = summary.tables[1].as_html()
summarydf = pd.read_html(summaryhtml,header=0,index_col=0)[0]
print(summarydf)
```

```
                      coef  std err       t  P>|t|     [0.025      0.975]
Constant        14500.0000  288.544  50.257  0.000  13900.000   15100.000
Male               15.9504    4.965   3.213  0.001      6.220      25.681
Marital_Status    570.9279   90.616   6.301  0.000    393.324     748.532
Price            1038.1643   96.735  10.732  0.000    848.567    1227.762
User_Count       -144.7470  111.262  -1.301  0.193   -362.816      73.322
Product_Count   -4440.0222  109.931 -40.389  0.000  -4655.483   -4224.561
18-25              -95.1387  134.266  -0.709  0.479   -358.296     168.018
26-35              -32.6309   20.663  -1.579  0.114    -73.129       7.867
36-45               18.2178   20.934   0.870  0.384    -22.813      59.248
46-50               19.3909   21.856   0.887  0.375    -23.446      62.228
51-55              831.5974  182.705   4.552  0.000    473.501    1189.694
55+                 56.9920   23.315   2.444  0.015     11.296     102.688
1                 -138.3672   22.624  -6.116  0.000   -182.710     -94.025
2                  -19.7758   10.779  -1.835  0.067    -40.902       1.351
3                   60.5897   33.291   1.820  0.069     -4.659     125.839
4                 -120.1853   49.988  -2.404  0.016   -218.160     -22.211
5                  144.5690   48.458   2.983  0.003     49.594     239.544
6                 -210.3857   31.308  -6.720  0.000   -271.749    -149.022
7                   12.0219   24.174   0.497  0.619    -35.358      59.402
8                    6.6448   38.353   0.173  0.862    -68.526      81.816
9                   32.5327   19.941   1.631  0.103     -6.551      71.617
10                 -49.0725   21.833  -2.248  0.025    -91.865      -6.280
11                  -8.5162   40.560  -0.210  0.834    -88.012      70.980
12                 102.3980   28.353   3.612  0.000     46.827     157.969
13                  -8.0480   19.712  -0.408  0.683    -46.683      30.587
14                  12.4916   31.544   0.396  0.692    -49.333      74.317
15                 489.8461   47.292  10.358  0.000    397.156     582.537
16                 -71.4333   27.140  -2.632  0.008   -124.627     -18.240
17                 125.2610   24.886   5.033  0.000     76.486     174.036
18                   5.6574   46.037   0.123  0.902    -84.574      95.888
...                    ...      ...     ...    ...        ...         ...
BxPC7              -91.4643   29.073  -3.146  0.002   -148.447     -34.482
BxPC8               -3.5424    7.266  -0.488  0.626    -17.784      10.699
BxPC9               15.6290   15.609   1.001  0.317    -14.965      46.223
BxPC10             -51.0543   21.401  -2.386  0.017    -92.999      -9.109
BxPC11              -4.1884   11.064  -0.379  0.705    -25.873      17.496
BxPC12             -24.5459   14.535  -1.689  0.091    -53.033       3.942
BxPC13              -1.9330   13.842  -0.140  0.889    -29.063      25.197
BxPC14               2.2896    7.624   0.300  0.764    -12.653      17.232
BxPC15             -17.4096    8.564  -2.033  0.042    -34.195      -0.624
BxPC16               9.2164    8.235   1.119  0.263     -6.925      25.357
BxPC17              -9.9917   11.924  -0.838  0.402    -33.362      13.379
BxPC18              10.3153   19.260   0.536  0.592    -27.433      48.064
CxPC1               77.5597   10.205   7.600  0.000     57.558      97.561
CxPC2               35.6175   10.920   3.262  0.001     14.215      57.020
CxPC3              -19.3201   19.543  -0.989  0.323    -57.623      18.983
CxPC4               45.3462   15.910   2.850  0.004     14.163      76.529
CxPC5               -6.6441    9.022  -0.736  0.461    -24.326      11.038
CxPC6               -0.6157   12.591  -0.049  0.961    -25.294      24.062
```

```
CxPC7           -112.9169   36.521  -3.092  0.002   -184.497    -41.336
CxPC8             -8.0195    8.876  -0.903  0.366    -25.417      9.378
CxPC9             10.9962   18.650   0.590  0.555    -25.558     47.550
CxPC10           -60.1545   25.127  -2.394  0.017   -109.404    -10.905
CxPC11             3.3816   13.696   0.247  0.805    -23.463     30.226
CxPC12           -20.7018   17.939  -1.154  0.249    -55.862     14.459
CxPC13           -10.1790   16.746  -0.608  0.543    -43.001     22.642
CxPC14            27.2937    9.314   2.930  0.003      9.039     45.548
CxPC15            11.0644   10.180   1.087  0.277     -8.887     31.016
CxPC16            40.7283   10.026   4.062  0.000     21.078     60.379
CxPC17             9.9203   14.109   0.703  0.482    -17.733     37.574
CxPC18            11.8925   23.598   0.504  0.614    -34.359     58.144

[338 rows x 6 columns]
```

Cells with t value below 5 are screened out. (Some significant interactions are screened out, but this high t value was chosen to filter out more interactions to make the regression analysis more efficient).

In [22]:
```python
significant = summarydf[((summarydf['t']>=5)|(summarydf['t']<=-5))&(summarydf.index.
print(significant)
print(len(significant))
```

```
Index(['Marital_StatusxPrice', 'Marital_Statusx7', 'Marital_Statusx12',
       'Marital_Statusx14', 'Marital_Statusx20', 'PricexProduct_Count',
       'Pricex18-25', 'Pricex1', 'Pricex7', 'Pricex12', 'Pricex14', 'Pricex17',
       'PricexPC1', 'PricexPC2', 'PricexPC3', 'PricexPC4', 'PricexPC5',
       'PricexPC6', 'PricexPC7', 'PricexPC8', 'PricexPC9', 'PricexPC10',
       'PricexPC11', 'PricexPC12', 'PricexPC13', 'PricexPC14', 'PricexPC16',
       'PricexPC17', 'PricexPC18', 'User_CountxProduct_Count', 'User_Countx5',
       'User_Countx6', 'User_Countx11', 'User_Countx16', 'User_CountxB',
       'User_CountxPC1', 'User_CountxPC4', 'User_CountxPC5', 'User_CountxPC10',
       'User_CountxPC17', 'Product_Countx18-25', 'Product_Countx51-55',
       'Product_Countx6', 'Product_CountxPC1', 'Product_CountxPC2',
       'Product_CountxPC3', 'Product_CountxPC4', 'Product_CountxPC5',
       'Product_CountxPC6', 'Product_CountxPC7', 'Product_CountxPC9',
       'Product_CountxPC10', 'Product_CountxPC11', 'Product_CountxPC12',
       'Product_CountxPC13', 'Product_CountxPC14', 'Product_CountxPC15',
       'Product_CountxPC16', 'Product_CountxPC17', 'Product_CountxPC18',
       '18-25x1', '18-25x6', '18-25x15', '18-25xB', '18-25xPC5', '51-55x15',
       '51-55x17', '51-55xB', '51-55xPC2', '51-55xPC10', '5xB', '5xC', '7xB',
       '11xB', '11xC', '12xB', '20xB', 'CxPC1'],
      dtype='object')
78
```

Separately, the interactions between the two features with the most categories (occupation and product type) are evaluated.

In [23]:
```
X2 = X.copy()
for i in range(len(sigocc)):
    for j in range(len(sigpc)):
        a=sigocc[i]
        b=sigpc[j]
        X2[str(a)+'x'+str(b)]=X2[a]*X2[b]

y=df['Purchase']

X2.head(10)
```

Out[23]:

	Constant	Male	Marital_Status	Price	User_Count	Product_Count	18-25	26-35	36-45	46-50	...	20xI
0	1.0	-1.0	-1.0	1.120588	-0.804753	-0.381538	-1.0	-1.0	-1.0	-1.0	...	
1	1.0	-1.0	-1.0	2.107206	-0.804753	0.595130	-1.0	-1.0	-1.0	-1.0	...	
2	1.0	-1.0	-1.0	-0.722914	-0.804753	-0.722951	-1.0	-1.0	-1.0	-1.0	...	
3	1.0	-1.0	-1.0	-0.707698	-0.804753	-0.065310	-1.0	-1.0	-1.0	-1.0	...	
4	1.0	1.0	-1.0	2.139239	-0.563565	-0.440305	-1.0	-1.0	-1.0	-1.0	...	
5	1.0	1.0	-1.0	2.065563	-0.833466	0.695875	-1.0	1.0	-1.0	-1.0	...	
6	1.0	1.0	1.0	2.050347	-0.925347	2.985025	-1.0	-1.0	-1.0	1.0	...	
7	1.0	1.0	1.0	2.080778	-0.925347	0.639905	-1.0	-1.0	-1.0	1.0	...	
8	1.0	1.0	1.0	2.152052	-0.925347	1.507432	-1.0	-1.0	-1.0	1.0	...	
9	1.0	1.0	1.0	0.553603	-0.391288	1.188406	-1.0	1.0	-1.0	-1.0	...	

10 rows × 308 columns

In [24]:
```
model = sm.OLS(y,X2)
results2=model.fit()
summary2 = results2.summary()
summaryhtml2 = summary2.tables[1].as_html()
summarydf2 = pd.read_html(summaryhtml2,header=0,index_col=0)[0]
print(summarydf2)
```

```
                    coef  std err        t  P>|t|    [0.025     0.975]
Constant       9141.3253  883.657   10.345  0.000  7409.386  10900.000
Male             11.8108    5.195    2.273  0.023     1.629     21.993
Marital_Status  -28.5738    4.605   -6.205  0.000   -37.599    -19.549
Price          2620.2654    6.536  400.891  0.000  2607.455   2633.076
User_Count     -134.0123    5.201  -25.768  0.000  -144.206   -123.819
Product_Count  1066.4779    4.737  225.152  0.000  1057.194   1075.762
18-25           -70.9498   21.314   -3.329  0.001  -112.725    -29.174
26-35           -38.3284   21.253   -1.803  0.071   -79.984      3.327
36-45            -3.8939   21.559   -0.181  0.857   -46.149     38.361
46-50            -1.8568   22.519   -0.082  0.934   -45.993     42.279
51-55            73.7717   22.772    3.240  0.001    29.139    118.404
55+              24.2404   24.120    1.005  0.315   -23.033     71.514
1               207.4069  108.551    1.911  0.056    -5.350    420.164
2               -14.4740   11.379   -1.272  0.203   -36.777      7.829
3               142.8304  160.498    0.890  0.374  -171.741    457.402
4               207.2518   96.030    2.158  0.031    19.035    395.468
5               227.5009  212.757    1.069  0.285  -189.497    644.499
6                60.2347  152.532    0.395  0.693  -238.722    359.192
7              -111.7035  102.173   -1.093  0.274  -311.960     88.552
8               -19.5794   40.321   -0.486  0.627   -98.608     59.449
9                33.1228   21.070    1.572  0.116    -8.173     74.418
10              -34.3473   22.841   -1.504  0.133   -79.115     10.420
11             -103.4703  193.578   -0.535  0.593  -482.877    275.936
12             -150.3344  130.712   -1.150  0.250  -406.527    105.858
13              -24.9027   20.372   -1.222  0.222   -64.831     15.026
14              125.7281  138.299    0.909  0.363  -145.333    396.790
15                1.3935  193.228    0.007  0.994  -377.328    380.115
16                0.0389  139.624    0.000  1.000  -273.620    273.698
17              -11.1744  120.315   -0.093  0.926  -246.988    224.639
18             -441.1250  263.026   -1.677  0.094  -956.647     74.397
...                  ...      ...      ...    ...       ...        ...
18xPC7         -531.5277  123.662   -4.298  0.000  -773.901   -289.154
18xPC8          -25.7327   28.277   -0.910  0.363   -81.156     29.690
18xPC9          -50.0799   61.558   -0.814  0.416  -170.732     70.572
18xPC10        -105.0670   81.622   -1.287  0.198  -265.044     54.910
18xPC11          81.4256   34.510    2.359  0.018    13.786    149.065
18xPC12         -87.1403   60.983   -1.429  0.153  -206.665     32.384
18xPC13          31.5646   53.320    0.592  0.554   -72.941    136.070
18xPC14           6.3118   33.559    0.188  0.851   -59.463     72.087
18xPC15         -26.1513   31.959   -0.818  0.413   -88.790     36.488
18xPC16         -18.8785   30.299   -0.623  0.533   -78.263     40.506
18xPC17          72.4218   46.902    1.544  0.123   -19.504    164.347
18xPC18          43.7074   66.960    0.653  0.514   -87.533    174.948
20xPC1          -64.6001   14.340   -4.505  0.000   -92.706    -36.494
20xPC2          -20.0387   17.246   -1.162  0.245   -53.841     13.764
20xPC3           65.4210   29.516    2.216  0.027     7.570    123.272
20xPC4            7.9462   23.946    0.332  0.740   -38.988     54.880
20xPC5           10.6699   13.169    0.810  0.418   -15.142     36.482
20xPC6           22.6396   18.698    1.211  0.226   -14.007     59.287
```

```
20xPC7              64.9110   46.708    1.390  0.165   -26.636    156.458
20xPC8               3.0866   13.390    0.231  0.818   -23.157     29.330
20xPC9              39.1513   30.720    1.274  0.203   -21.059     99.361
20xPC10             30.5503   34.677    0.881  0.378   -37.416     98.516
20xPC11             76.9431   20.681    3.720  0.000    36.409    117.477
20xPC12             61.6434   26.540    2.323  0.020     9.625    113.661
20xPC13             30.7553   25.219    1.220  0.223   -18.673     80.184
20xPC14             -6.1659   14.431   -0.427  0.669   -34.451     22.119
20xPC15             67.1351   16.509    4.067  0.000    34.778     99.492
20xPC16            -37.4510   15.631   -2.396  0.017   -68.086     -6.816
20xPC17             22.8954   22.128    1.035  0.301   -20.475     66.266
20xPC18             45.8951   36.622    1.253  0.210   -25.884    117.674

[308 rows x 6 columns]
```

Type *Markdown* and LaTeX: α^2

In [25]:
```
significant2 = summarydf2[((summarydf2['t']>=5)|(summarydf2['t']<=-5))&(summarydf2.i
print(significant2)
print(len(significant2))
```

```
Index(['5xPC1', '7xPC1', '17xPC1'], dtype='object')
3
```

Finally, the 235 different combinations of product category are evaluated.

```
In [26]: X3 = pd.concat([X,combinationdummies],axis=1)
         y=df['Purchase']
         model = sm.OLS(y,X3)
         results3=model.fit()
         summary3 = results3.summary()
         summaryhtml3 = summary3.tables[1].as_html()
         summarydf3 = pd.read_html(summaryhtml3,header=0,index_col=0)[0]
         print(summarydf3)
```

```
                     coef  std err        t  P>|t|    [0.025    0.975]
Constant        7472.3126  210.803   35.447  0.000  7059.145  7885.480
Male              -5.4838    4.741   -1.157  0.247   -14.775     3.808
Marital_Status   -24.2188    4.199   -5.767  0.000   -32.449   -15.988
Price            531.6964    9.386   56.647  0.000   513.300   550.093
User_Count      -125.3046    4.747  -26.395  0.000  -134.609  -116.000
Product_Count    937.3614    4.848  193.337  0.000   927.859   946.864
18-25            -78.0671   19.444   -4.015  0.000  -116.177   -39.957
26-35            -38.3684   19.388   -1.979  0.048   -76.368    -0.369
36-45             22.5996   19.667    1.149  0.251   -15.947    61.146
46-50             40.8842   20.543    1.990  0.047     0.620    81.148
51-55            119.9773   20.776    5.775  0.000    79.256   160.698
55+               70.0986   22.008    3.185  0.001    26.964   113.234
1                -61.5016    8.624   -7.131  0.000   -78.405   -44.598
2                 -8.9872   10.381   -0.866  0.387   -29.334    11.359
3                114.6363   12.090    9.482  0.000    90.941   138.332
4                 34.3615    8.247    4.166  0.000    18.197    50.526
5                 43.2242   14.065    3.073  0.002    15.658    70.791
6                 79.8843   11.547    6.918  0.000    57.252   102.516
7                 20.6677    8.098    2.552  0.011     4.797    36.539
8                -52.0197   36.806   -1.413  0.158  -124.158    20.119
9                 31.5228   19.191    1.643  0.100    -6.092    69.137
10               -20.5449   20.837   -0.986  0.324   -61.385    20.295
11               -30.4601   14.426   -2.112  0.035   -58.734    -2.186
12                55.9217    9.856    5.674  0.000    36.604    75.239
13                 4.4517   18.560    0.240  0.810   -31.925    40.828
14                55.3432   10.249    5.400  0.000    35.256    75.430
15                92.9658   14.159    6.566  0.000    65.214   120.717
16                57.1254   10.617    5.381  0.000    36.317    77.934
17                58.6287    9.079    6.458  0.000    40.834    76.423
18                27.4042   18.431    1.487  0.137    -8.720    63.528
...                   ...      ...      ...    ...       ...       ...
712.0nan        1926.1185  222.527    8.656  0.000  1489.973  2362.264
717.0nan        2440.1088  239.827   10.174  0.000  1970.054  2910.163
78.0nan         2348.8389  258.121    9.100  0.000  1842.929  2854.748
7nannan         1785.6055  115.397   15.474  0.000  1559.431  2011.780
810.016.0      -5345.7231  394.498  -13.551  0.000 -6118.926 -4572.520
810.0nan       -5506.1343  273.660  -20.120  0.000 -6042.500 -4969.769
811.016.0       1629.2569  418.883    3.890  0.000   808.259  2450.255
811.0nan        -836.4709  164.792   -5.076  0.000 -1159.458  -513.484
812.017.0        697.5453  178.428    3.909  0.000   347.833  1047.258
812.0nan         -22.0083   93.220   -0.236  0.813  -204.716   160.699
813.014.0        343.5918  169.216    2.030  0.042    11.933   675.251
813.015.0        -82.5245  109.036   -0.757  0.449  -296.231   131.182
813.016.0       -528.4923  113.050   -4.675  0.000  -750.067  -306.917
813.0nan         252.2708   77.527    3.254  0.001   100.320   404.221
814.015.0         65.2626  445.190    0.147  0.883  -807.295   937.820
814.016.0      -1418.6685  122.780  -11.555  0.000 -1659.314 -1178.024
```

```
814.017.0      -1735.5492    71.272  -24.351  0.000 -1875.240 -1595.859
814.018.0      -1376.9480   604.200   -2.279  0.023 -2561.160  -192.736
814.0nan       -2135.8032    56.658  -37.696  0.000 -2246.851 -2024.756
815.016.0      -1471.0223   116.836  -12.590  0.000 -1700.018 -1242.027
815.0nan       -2051.4243    64.806  -31.655  0.000 -2178.441 -1924.408
816.017.0      -1247.3347   130.318   -9.571  0.000 -1502.753  -991.916
816.0nan       -2169.5633    66.101  -32.822  0.000 -2299.119 -2040.007
817.0nan       -1827.9441    58.011  -31.511  0.000 -1941.643 -1714.245
818.0nan        -384.1842   140.493   -2.735  0.006  -659.546  -108.822
89.014.0         107.1941   216.922    0.494  0.621  -317.966   532.354
89.0nan        -1051.0156   276.056   -3.807  0.000 -1592.076  -509.955
8nannan        -2369.9094    66.023  -35.895  0.000 -2499.313 -2240.506
915.0nan        5819.2891   222.791   26.120  0.000  5382.626  6255.952
9nannan         2974.3192  2620.021    1.135  0.256 -2160.839  8109.477

[291 rows x 6 columns]
```

In [29]:
```
significant3 = summarydf3[((summarydf3['t']>=5)|(summarydf3['t']<=-5))&((summarydf3.
print(sorted(significant3))
print(len(significant3))
```

```
['1013.016.0', '1013.0nan', '1014.016.0', '1015.016.0', '1015.0nan', '1016.0nan',
'10nannan', '111.015.0', '111.016.0', '111.0nan', '1113.016.0', '1114.0nan', '1115.
016.0', '1115.0nan', '1116.0nan', '113.014.0', '113.016.0', '114.017.0', '115.016.
0', '115.018.0', '115.0nan', '117.0nan', '118.0nan', '11nannan', '12.013.0', '12.01
4.0', '12.015.0', '12.016.0', '12.018.0', '12.03.0', '12.04.0', '12.05.0', '12.06.
0', '12.08.0', '12.09.0', '12.0nan', '1214.017.0', '1214.0nan', '1217.0nan', '12nan
nan', '13.04.0', '1314.016.0', '1315.016.0', '1315.0nan', '1316.0nan', '13nannan',
'14.0nan', '1416.0nan', '1417.0nan', '1418.0nan', '14nannan', '15.012.0', '15.017.
0', '15.018.0', '15.08.0', '1516.017.0', '1516.0nan', '1517.0nan', '15nannan', '16.
013.0', '16.015.0', '16.016.0', '16.0nan', '16nannan', '18.013.0', '18.017.0', '18.
09.0', '18.0nan', '18nannan', '1nannan', '214.0nan', '215.016.0', '215.0nan', '217.
0nan', '218.0nan', '23.010.0', '23.015.0', '24.012.0', '24.014.0', '24.015.0', '24.
05.0', '24.08.0', '24.09.0', '24.0nan', '25.012.0', '25.015.0', '25.08.0', '25.0na
n', '26.015.0', '28.014.0', '28.016.0', '28.018.0', '29.014.0', '29.015.0', '29.0na
n', '34.012.0', '34.05.0', '34.08.0', '34.09.0', '34.0nan', '35.016.0', '412.0nan',
'415.0nan', '45.015.0', '45.08.0', '45.0nan', '48.09.0', '48.0nan', '49.015.0', '4n
annan', '511.012.0', '512.014.0', '512.0nan', '513.014.0', '513.016.0', '514.016.
0', '514.017.0', '514.0nan', '515.018.0', '515.0nan', '516.0nan', '517.0nan', '518.
0nan', '56.011.0', '56.013.0', '56.016.0', '56.08.0', '56.09.0', '56.0nan', '57.0na
n', '58.012.0', '58.018.0', '58.0nan', '59.014.0', '59.0nan', '5nannan', '611.013.
0', '611.016.0', '611.0nan', '613.0nan', '616.0nan', '68.013.0', '68.014.0', '68.01
5.0', '68.016.0', '68.0nan', '6nannan', '712.0nan', '717.0nan', '78.0nan', '7nanna
n', '810.016.0', '810.0nan', '811.0nan', '814.016.0', '814.017.0', '814.0nan', '81
5.016.0', '815.0nan', '816.017.0', '816.0nan', '817.0nan', '8nannan', '915.0nan']
164
```

The significant (t>=5) interactions from each stage of evaluation are redefined below for convenience. All these interactions are evaluated together, and insignificant interactions are eliminated.

```
In [30]: significant = ['Marital_StatusxPrice', 'Marital_Statusx7', 'Marital_Statusx12',
                'Marital_Statusx14', 'Marital_Statusx20', 'PricexProduct_Count',
                'Pricex18-25', 'Pricex1', 'Pricex7', 'Pricex12', 'Pricex14', 'Pricex17',
                'PricexPC1', 'PricexPC2', 'PricexPC3', 'PricexPC4', 'PricexPC5',
                'PricexPC6', 'PricexPC7', 'PricexPC8', 'PricexPC9', 'PricexPC10',
                'PricexPC11', 'PricexPC12', 'PricexPC13', 'PricexPC14', 'PricexPC16',
                'PricexPC17', 'PricexPC18', 'User_CountxProduct_Count', 'User_Countx5',
                'User_Countx6', 'User_Countx11', 'User_Countx16', 'User_CountxB',
                'User_CountxPC1', 'User_CountxPC4', 'User_CountxPC5', 'User_CountxPC10',
                'User_CountxPC17', 'Product_Countx18-25', 'Product_Countx51-55',
                'Product_Countx6', 'Product_CountxPC1', 'Product_CountxPC2',
                'Product_CountxPC3', 'Product_CountxPC4', 'Product_CountxPC5',
                'Product_CountxPC6', 'Product_CountxPC7', 'Product_CountxPC9',
                'Product_CountxPC10', 'Product_CountxPC11', 'Product_CountxPC12',
                'Product_CountxPC13', 'Product_CountxPC14', 'Product_CountxPC15',
                'Product_CountxPC16', 'Product_CountxPC17', 'Product_CountxPC18',
                '18-25x1', '18-25x6', '18-25x15', '18-25xB', '18-25xPC5', '51-55x15',
                '51-55x17', '51-55xB', '51-55xPC2', '51-55xPC10', '5xB', '5xC', '7xB',
                '11xB', '11xC', '12xB', '20xB', 'CxPC1']
         significant2 = ['5xPC1', '7xPC1', '17xPC1']
         significant3 = ['1013.016.0', '1013.0nan', '1014.016.0', '1015.016.0', '1015.0nan',
                         '111.015.0', '111.016.0', '111.0nan', '1113.016.0', '1114.0nan', '11
                         '1116.0nan', '113.014.0', '113.016.0', '114.017.0', '115.016.0', '11
                         '118.0nan', '11nannan', '12.013.0', '12.014.0', '12.015.0', '12.016.
                         '12.05.0', '12.06.0', '12.08.0', '12.09.0', '12.0nan', '1214.017.0',
                         '13.04.0', '1314.016.0', '1315.016.0', '1315.0nan', '1316.0nan', '13
                         '1417.0nan', '1418.0nan', '14nannan', '15.012.0', '15.017.0', '15.01
                         '1516.0nan', '1517.0nan', '15nannan', '16.013.0', '16.015.0', '16.01
                         '18.013.0', '18.017.0', '18.09.0', '18.0nan', '18nannan', '1nannan',
                         '217.0nan', '218.0nan', '23.010.0', '23.015.0', '24.012.0', '24.014.
                         '24.09.0', '24.0nan', '25.012.0', '25.015.0', '25.08.0', '25.0nan',
                         '28.018.0', '29.014.0', '29.015.0', '29.0nan', '34.012.0', '34.05.0'
                         '35.016.0', '412.0nan', '415.0nan', '45.015.0', '45.08.0', '45.0nan'
                         '4nannan', '511.012.0', '512.014.0', '512.0nan', '513.014.0', '513.0
                         '514.0nan', '515.018.0', '515.0nan', '516.0nan', '517.0nan', '518.0n
                         '56.08.0', '56.09.0', '56.0nan', '57.0nan', '58.012.0', '58.018.0',
                         '5nannan', '611.013.0', '611.016.0', '611.0nan', '613.0nan', '616.0n
                         '68.016.0', '68.0nan', '6nannan', '712.0nan', '717.0nan', '78.0nan',
                         '811.0nan', '814.016.0', '814.017.0', '814.0nan', '815.016.0', '815.
                         '817.0nan', '8nannan', '915.0nan']


         X4 = pd.concat([X,X1[significant],X2[significant2],X3[significant3]],axis=1)
         y=df['Purchase']
```

In [31]:

```
model = sm.OLS(y,X4)
results4=model.fit()
summary4 = results4.summary()
summaryhtml4 = summary4.tables[1].as_html()
summarydf4 = pd.read_html(summaryhtml4,header=0,index_col=0)[0]
print(summary4)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Purchase   R-squared:                       0.686
Model:                            OLS   Adj. R-squared:                  0.686
Method:                 Least Squares   F-statistic:                     3926.
Date:                Thu, 11 Apr 2019   Prob (F-statistic):               0.00
Time:                        12:23:11   Log-Likelihood:            -5.0280e+06
No. Observations:              537577   AIC:                         1.006e+07
Df Residuals:                  537277   BIC:                         1.006e+07
Df Model:                         299
Covariance Type:            nonrobust
================================================================================
===========
                              coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
Constant                 7294.1192    359.336     20.299      0.000    6589.831
7998.407
```

In [39]:
```
significant4 = summarydf4[((summarydf4['t']>=5)|(summarydf4['t']<=-5))&((summarydf4.
print(sorted(significant4))
print(len(significant4))
```

```
['1013.016.0', '1013.0nan', '1014.016.0', '1015.016.0', '1015.0nan', '1016.0nan',
'10nannan', '111.015.0', '111.016.0', '1114.0nan', '1115.016.0', '1115.0nan', '111
6.0nan', '113.014.0', '113.016.0', '114.017.0', '115.018.0', '115.0nan', '117.0na
n', '118.0nan', '11nannan', '11xB', '11xC', '12.013.0', '12.014.0', '12.015.0', '1
2.016.0', '12.018.0', '12.03.0', '12.04.0', '12.06.0', '12.08.0', '12.09.0', '12.0n
an', '1214.017.0', '1214.0nan', '1217.0nan', '12nannan', '12xB', '13.04.0', '1314.0
16.0', '1315.016.0', '1315.0nan', '1316.0nan', '13nannan', '1416.0nan', '1417.0na
n', '1418.0nan', '14nannan', '15.012.0', '15.017.0', '15.018.0', '15.08.0', '1516.0
17.0', '1516.0nan', '1517.0nan', '15nannan', '16.013.0', '16.015.0', '16.016.0', '1
6.0nan', '16nannan', '17xPC1', '18-25x1', '18-25x15', '18-25x6', '18-25xB', '18.01
3.0', '18.017.0', '18.09.0', '18nannan', '20xB', '214.0nan', '215.016.0', '217.0na
n', '23.010.0', '23.015.0', '24.012.0', '24.014.0', '24.015.0', '24.05.0', '24.08.
0', '24.09.0', '24.0nan', '25.015.0', '25.08.0', '28.014.0', '28.016.0', '29.015.
0', '29.0nan', '34.012.0', '34.05.0', '34.08.0', '34.09.0', '34.0nan', '35.016.0',
'415.0nan', '45.015.0', '45.08.0', '45.0nan', '48.09.0', '48.0nan', '49.015.0', '4n
annan', '51-55x15', '51-55x17', '51-55xB', '51-55xPC2', '511.012.0', '512.014.0',
'512.0nan', '513.014.0', '513.016.0', '514.0nan', '515.018.0', '515.0nan', '516.0na
n', '517.0nan', '518.0nan', '56.011.0', '56.013.0', '56.016.0', '56.08.0', '56.09.
0', '56.0nan', '57.0nan', '58.012.0', '58.018.0', '58.0nan', '5nannan', '5xB', '5x
C', '5xPC1', '611.013.0', '611.016.0', '611.0nan', '613.0nan', '616.0nan', '68.013.
0', '68.014.0', '68.015.0', '68.016.0', '68.0nan', '6nannan', '712.0nan', '717.0na
n', '78.0nan', '7nannan', '7xB', '7xPC1', '810.016.0', '810.0nan', '811.0nan', '81
4.016.0', '814.017.0', '814.0nan', '815.016.0', '815.0nan', '816.017.0', '816.0na
n', '817.0nan', '8nannan', '915.0nan', 'CxPC1', 'Marital_Statusx12', 'Marital_Statu
sx14', 'Marital_Statusx20', 'Marital_Statusx7', 'Pricex1', 'PricexPC1', 'PricexPC1
0', 'PricexPC14', 'PricexPC16', 'PricexPC17', 'PricexPC2', 'PricexPC9', 'PricexProd
uct_Count', 'Product_Countx18-25', 'Product_Countx51-55', 'Product_Countx6', 'Produ
ct_CountxPC10', 'Product_CountxPC11', 'Product_CountxPC12', 'Product_CountxPC14',
'Product_CountxPC17', 'Product_CountxPC18', 'Product_CountxPC2', 'Product_CountxPC
3', 'Product_CountxPC4', 'Product_CountxPC5', 'Product_CountxPC6', 'User_Countx11',
'User_Countx16', 'User_Countx5', 'User_Countx6', 'User_CountxB', 'User_CountxPC1',
'User_CountxPC17', 'User_CountxPC4', 'User_CountxPC5', 'User_CountxProduct_Count']
201
```

The resulting model is shown below.

In [40]:
```
X5 = pd.concat([X,X4[significant4]],axis=1)
y=df['Purchase']
model = sm.OLS(y,X5)
results5=model.fit()
summary5 = results5.summary()
summaryhtml5 = summary5.tables[1].as_html()
summarydf5 = pd.read_html(summaryhtml5,header=0,index_col=0)[0]
print(summary5)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Purchase   R-squared:                       0.686
Model:                            OLS   Adj. R-squared:                  0.685
Method:                 Least Squares   F-statistic:                     4596.
Date:                Thu, 11 Apr 2019   Prob (F-statistic):               0.00
Time:                        13:25:16   Log-Likelihood:            -5.0283e+06
No. Observations:              537577   AIC:                         1.006e+07
Df Residuals:                  537321   BIC:                         1.006e+07
Df Model:                         255
Covariance Type:            nonrobust
=================================================================================
===========
                              coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------------
-----------
Constant                 7945.8332    210.575     37.734      0.000    7533.112
8358.554
```

Now interactions with similar coefficients can be grouped.

In [92]:
```
inter = summarydf5.loc['13nannan':,'coef']
names=inter.index.to_series(name='Name')
inter = pd.concat([inter,names],axis=1)
inter=inter.sort_values('coef')
inter.reset_index(drop=True,inplace=True)

combined = [[0]]
group = []
for i in range(1,len(inter)):
    if min(inter.loc[i-1,'coef'],inter.loc[i,'coef'])/max(inter.loc[i-1,'coef'],inte
        group.append(i)
    else:
        combined.append(group)
        group = [i]
combined.append([len(inter)-1])

X6=X.copy()
for a in combined:
    name = ''
    for b in a:
        name += inter.loc[b,'Name']
    X6[name]=0
    for b in a:
        X6[name]+=X5[inter.loc[b,'Name']]

model = sm.OLS(y,X6)
results6=model.fit()
summary6 = results6.summary()
summaryhtml6 = summary6.tables[1].as_html()
summarydf6 = pd.read_html(summaryhtml6,header=0,index_col=0)[0]
print(summary6)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Purchase   R-squared:                       0.646
Model:                            OLS   Adj. R-squared:                  0.646
Method:                 Least Squares   F-statistic:                 1.128e+04
Date:                Thu, 11 Apr 2019   Prob (F-statistic):               0.00
Time:                        14:54:10   Log-Likelihood:            -5.0601e+06
No. Observations:              537577   AIC:                         1.012e+07
Df Residuals:                  537489   BIC:                         1.012e+07
Df Model:                          87
Covariance Type:            nonrobust
=================================================================================
=================================================================================
=================================================================================
=================================================================================
=================================================================================
========================

coef    std err          t      P>|t|      [0.025      0.975]
```

Next, we check for quadratic and cubic effects on continuous variables price, user count, and product count.

In [94]:
```
X7 = X6.copy()
X7['PriceSq']=X7['Price']*2
X7['UserCountSq']=X7['User_Count']*2
X7['ProdCountSq']=X7['Product_Count']*2
X7['PriceCub']=X7['Price']*3
X7['UserCountCub']=X7['User_Count']*3
X7['ProdCountCub']=X7['Product_Count']*3

model = sm.OLS(y,X7)
results7=model.fit()
summary7 = results7.summary()
summaryhtml7 = summary7.tables[1].as_html()
summarydf7 = pd.read_html(summaryhtml7,header=0,index_col=0)[0]
print(summary7)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Purchase   R-squared:                       0.646
Model:                            OLS   Adj. R-squared:                  0.646
Method:                 Least Squares   F-statistic:                 1.128e+04
Date:                Thu, 11 Apr 2019   Prob (F-statistic):               0.00
Time:                        15:00:36   Log-Likelihood:            -5.0601e+06
No. Observations:              537577   AIC:                         1.012e+07
Df Residuals:                  537489   BIC:                         1.012e+07
Df Model:                          87
Covariance Type:            nonrobust
================================================================================
================================================================================
================================================================================
================================================================================
================================================================================
========================

coef    std err          t      P>|t|      [0.025      0.975]
```

All squared and cubed terms are significant.

The squared and cubed terms are also evaluated for the full model (without combining interactions).

In [95]:

```
X8 = X5.copy()
X8['PriceSq']=X8['Price']*2
X8['UserCountSq']=X8['User_Count']*2
X8['ProdCountSq']=X8['Product_Count']*2
X8['PriceCub']=X8['Price']*3
X8['UserCountCub']=X8['User_Count']*3
X8['ProdCountCub']=X8['Product_Count']*3

model = sm.OLS(y,X8)
results8=model.fit()
summary8 = results8.summary()
summaryhtml8 = summary8.tables[1].as_html()
summarydf8 = pd.read_html(summaryhtml8,header=0,index_col=0)[0]
print(summary8)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Purchase   R-squared:                       0.686
Model:                            OLS   Adj. R-squared:                  0.685
Method:                 Least Squares   F-statistic:                     4596.
Date:                Thu, 11 Apr 2019   Prob (F-statistic):               0.00
Time:                        15:06:28   Log-Likelihood:            -5.0283e+06
No. Observations:              537577   AIC:                         1.006e+07
Df Residuals:                  537321   BIC:                         1.006e+07
Df Model:                         255
Covariance Type:            nonrobust
=================================================================================
===========
                             coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------------
-----------
Constant                7945.8332    210.575     37.734      0.000    7533.112
8358.554
```

The square and cube of the user count is not significant.

In [110]:

```
X8=X8.drop(['UserCountSq','UserCountCub'],axis=1)
```

Now we have two potential feature sets for linear modeling, X7 and X8. Both these models have better R-squared value and AIC than the model with no interactions. The model with more interactions has a better R-squared and AIC than the one with interactions combined.

Next we will cross validate each of these models (regularized and unregularized).

In [173]:
```
def evaluatemodel(model,matrix,importance=False):
    data = pd.concat([matrix,y],axis=1)
    data = shuffle(data)
    X = data.iloc[:,:-1]
    Y = data.iloc[:,-1]
    Model = model
    Model.fit(X,Y)
    Ypredict = Model.predict(X)
    print('R-squared: '+str(metrics.r2_score(Y,Ypredict)))
    print('RMSE: '+str(np.sqrt(metrics.mean_squared_error(Y,Ypredict))))
    CVscore = cross_val_score(Model,X,Y,cv=5,scoring='neg_mean_squared_error')
    print('CV Mean RMSE: '+str(np.mean(np.sqrt(-1*CVscore))))
    if importance:
        fig5, ax5 = plt.subplots()
        ax5.bar([str(a) for a in list(matrix)],Model.feature_importances_)
        ax5.set_title('Feature Importance')
        ax5.set_xticklabels([str(a) for a in list(matrix)],rotation=90)
        fig5.set_size_inches(15,8)


evaluatemodel(linear_model.LinearRegression(),X8)
```

```
R-squared: 0.6856332167786194
RMSE: 2792.7778613349474
CV Mean RMSE: 2794.5883661837847
```

In [112]:
```
evaluatemodel(linear_model.LinearRegression(),X7)
```

```
R-squared: 0.646180835824119
RMSE: 2962.843714140378
CV Mean RMSE: 2963.5695803064104
```

We see that the reduction in features by combining features with similar coefficients does not noticeably help during cross validation. The full model's CV score is similar to the score obtained using the entire data set, indicating the model is not overfit, even with over 200 features.

Next, we evaluate the performance of ridge and lasso regression on X8. Since the cross validation score is already good, these steps would only serve to simplify the model, but may have little impact on the model variance.

In [113]:
```
evaluatemodel(linear_model.Ridge(alpha=0.5),X8)
```

```
R-squared: 0.6856337969010652
RMSE: 2792.7752844821166
CV Mean RMSE: 2794.42075404524
```

In [115]:
```
evaluatemodel(linear_model.Lasso(alpha=0.5,tol=0.1),X8)
```

```
R-squared: 0.6830708596105055
RMSE: 2804.1365223217813
CV Mean RMSE: 2805.8791806238214
```

The ridge regression model gives slightly better results than the unregularized model, surprisingly. Different alpha values are evaluated to check if this has any effect.

In [121]:
```
evaluatemodel(linear_model.Ridge(alpha=0.1),X8)
```

```
R-squared: 0.6856339255159195
RMSE: 2792.7747131859346
CV Mean RMSE: 2794.4221427450466
```

In [117]:
```
evaluatemodel(linear_model.Ridge(alpha=0.25),X8)
```

```
R-squared: 0.6856338972265197
RMSE: 2792.774838845038
CV Mean RMSE: 2794.643214747238
```

In [118]:
```
evaluatemodel(linear_model.Ridge(alpha=0.75),X8)
```

```
R-squared: 0.685633631088085
RMSE: 2792.7760210090196
CV Mean RMSE: 2794.4423847080147
```

The best model in terms of RMSE is ridge regression with alpha=0.1. This also has a negligibly lower cross validation RMSE than the unregularized model.

Looking at the regression model statistics from the statsmodels summaries, it is clear that there is no significant serial correlation. The skew value indicates residuals are fairly symmetric, but slightly skewed left. However, the residual distribution has extremely heavy tails, as indicated by the kurtosis. This indicates that the residuals are not due to gaussian random error. This combined with the overall poor fit suggests there are unknown variables that have large effects on customer purchase. If such variables were binary categorical variables, it could explain the heavy tails.

Decision Tree

Due to the large number of significant interactions observed in linear regression, a decision tree may be a good model, because it can account for any degree of interaction as well as non-linearity.

First evaluate using the default settings.

In [124]:
```
evaluatemodel(tree.DecisionTreeRegressor(),X)
```

```
R-squared: 0.9998134481838908
RMSE: 68.0327158438232
CV Mean RMSE: 3746.8512714664917
```

Clearly there is overfitting. GridSearchCV will be used to tune the hyperparameters.

In [129]:
```
grid = {'min_impurity_decrease':[0.1,0.01,0.001,0.0001,0.00001]}
reg = GridSearchCV(tree.DecisionTreeRegressor(),grid,cv=5)
reg.fit(X,y)
print(reg.best_params_)
```

```
{'min_impurity_decrease': 0.001}
```

In [134]:
```
grid = {'max_depth':[9,10,11,12,13]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_impurity_decrease=0.001),grid,cv=5
reg.fit(X,y)
print(reg.best_params_)
```

```
{'max_depth': 11}
```

In [139]:
```
grid = {'min_samples_split':[2,5,10,20,50],'min_samples_leaf':[1,2,5,10,25]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_impurity_decrease=0.001,max_depth=
reg.fit(X,y)
print(reg.best_params_)
```

```
{'min_samples_leaf': 25, 'min_samples_split': 10}
```

In [140]:
```
grid = {'min_samples_leaf':[25,50,75,100]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_impurity_decrease=0.001,max_depth=
reg.fit(X,y)
print(reg.best_params_)
```

```
{'min_samples_leaf': 75}
```

In [141]:
```
evaluatemodel(tree.DecisionTreeRegressor(min_impurity_decrease=0.001,max_depth=11,mi
```

```
R-squared: 0.7053264391071729
RMSE: 2703.8877185178103
CV Mean RMSE: 2727.385630592458
```

More fine tuning ...

In [142]:
```
grid = {'min_impurity_decrease':[0.0005,0.001,0.002,0.005],'max_depth':[10,11,12]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_samples_split=10,min_samples_leaf=
reg.fit(X,y)
print(reg.best_params_)
```

```
{'max_depth': 12, 'min_impurity_decrease': 0.0005}
```

In [143]:
```
grid = {'min_samples_split':[7,10,15],'min_samples_leaf':[65,75,85]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_impurity_decrease=0.0005,max_depth
reg.fit(X,y)
print(reg.best_params_)
```

```
{'min_samples_leaf': 85, 'min_samples_split': 10}
```

Since minimum samples per leaf is much higher than minimum samples to split, minimum samples to split will not have any effect.

In [144]:
```
evaluatemodel(tree.DecisionTreeRegressor(min_impurity_decrease=0.0005,max_depth=12,m
```

```
R-squared: 0.7090098026990052
RMSE: 2686.9355354772692
CV Mean RMSE: 2716.73401195658
```

In [145]:
```
grid = {'min_impurity_decrease':[0.0004,0.0005,0.0006],'max_depth':[11,12,13]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_samples_split=10,min_samples_leaf=
reg.fit(X,y)
print(reg.best_params_)
```

```
{'max_depth': 13, 'min_impurity_decrease': 0.0004}
```

In [147]:
```
grid = {'min_impurity_decrease':[0.0001,0.0002,0.0004],'max_depth':[14,15,16]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_samples_split=10,min_samples_leaf=
reg.fit(X,y)
print(reg.best_params_)
```

```
{'max_depth': 16, 'min_impurity_decrease': 0.0001}
```

In [150]:
```
grid = {'min_impurity_decrease':[0.00004,0.00006,0.00008,0.0001],'max_depth':[16,17,
reg = GridSearchCV(tree.DecisionTreeRegressor(min_samples_split=10,min_samples_leaf=
reg.fit(X,y)
print(reg.best_params_)
```

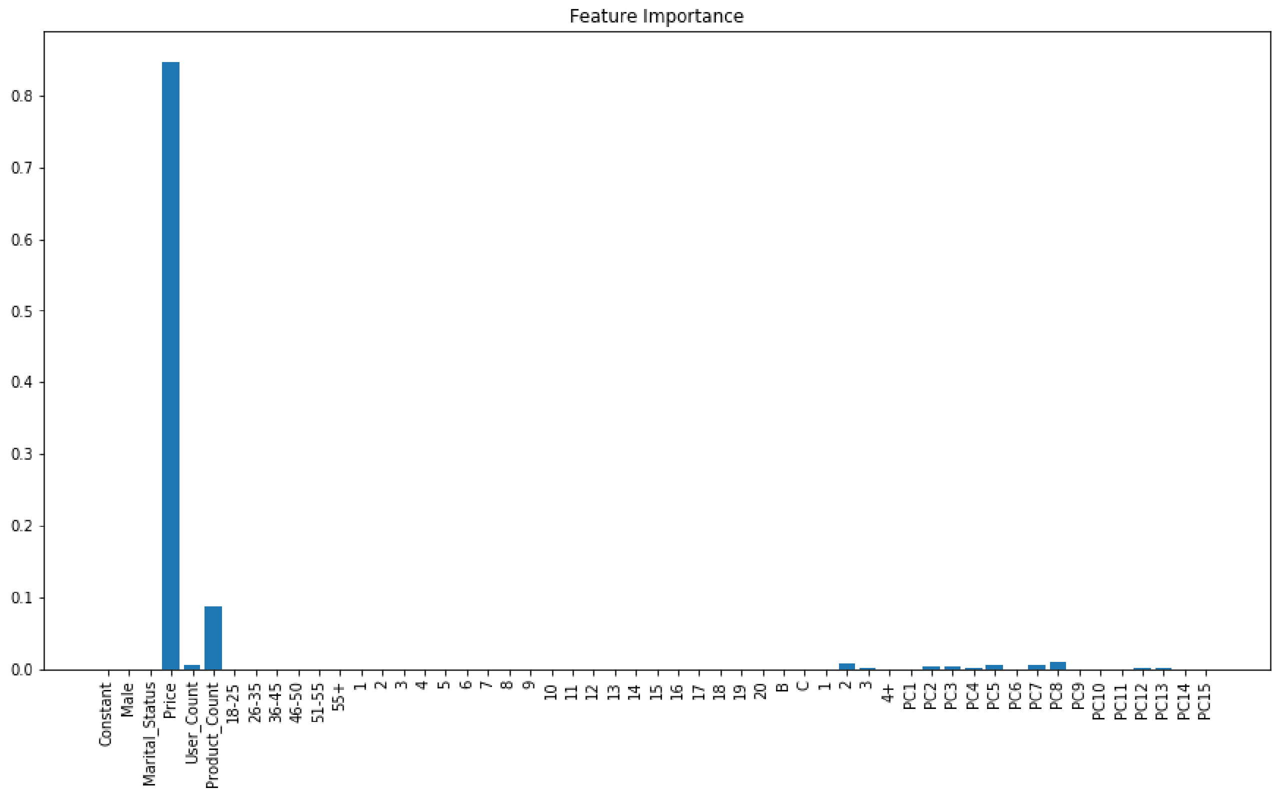
```
{'max_depth': 19, 'min_impurity_decrease': 6e-05}
```

In [152]:
```
grid = {'min_impurity_decrease':[0.00003,0.00004,0.00005],'max_depth':[21,22,23]}
reg = GridSearchCV(tree.DecisionTreeRegressor(min_samples_split=10,min_samples_leaf=
reg.fit(X,y)
print(reg.best_params_)
```

```
{'max_depth': 21, 'min_impurity_decrease': 4e-05}
```

In [174]:
```
evaluatemodel(tree.DecisionTreeRegressor(min_impurity_decrease=0.00004,max_depth=21,
```

```
R-squared: 0.7210230848251868
RMSE: 2630.887038842364
CV Mean RMSE: 2701.243691940237
```



This optimized model performs better than linear regression both in terms of RMSE and cross validation RMSE. This is likely because it can capture any degree of interaction between any number of features as well as nonlinear behavior.

Similar to the regression model, the price and popularity of the product is the most important, followed by product category. In the decision tree mode, the length of time in the same city has a slight effect also.

Overall the characteristics of the product has more impact on the amount a customer spends on a product than the characteristics of the customer. This indicates most purchases made on Black Friday are items that most people would want to buy. The characteristics of the customer helps refine the model, but accounts for only a small portion of the sum of squares. The rest of the variability is presumably from characteristics of the product or customers that are not extractable from the data set.