In [4]:
```
# for numerical computing
import numpy as np
# for dataframes
import pandas as pd
# for easier visualization
import seaborn as sns
# for visualization and to display plots
from matplotlib import pyplot as plt
%matplotlib inline
# import color maps
from matplotlib.colors import ListedColormap
# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
from math import sqrt
# to split train and test set
from sklearn.model_selection import train_test_split
# to perform hyperparameter tuning
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import Ridge # Linear Regression + L2 regularizat
from sklearn.linear_model import Lasso # Linear Regression + L1 regularizat
from sklearn.svm import SVR # Support Vector Regressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
# Evaluation Metrics
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score as rs
from sklearn.metrics import mean_absolute_error as mae
#import xgboost
import os
mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-7.2.0-posix-seh-rt_v5-rev0\\mi
os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']
# to save the final model on disk
from sklearn.externals import joblib
```

# Loading Black Friday Data

In [5]:
```
df = pd.read_csv('BlackFriday 2.csv')
```

In [6]:
```
df.shape
```

Out[6]: (537577, 12)

In [7]:
```
df.columns
```

Out[7]:
```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Category_1',
       'Product_Category_2', 'Product_Category_3', 'Purchase'],
      dtype='object')
```

In [8]:
```
df.head()
```

Out[8]:

| | User_ID | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Mar |
|---|---|---|---|---|---|---|---|---|
| **0** | 1000001 | P00069042 | F | 0-17 | 10 | A | 2 | |
| **1** | 1000001 | P00248942 | F | 0-17 | 10 | A | 2 | |
| **2** | 1000001 | P00087842 | F | 0-17 | 10 | A | 2 | |
| **3** | 1000001 | P00085442 | F | 0-17 | 10 | A | 2 | |
| **4** | 1000002 | P00285442 | M | 55+ | 16 | C | 4+ | |

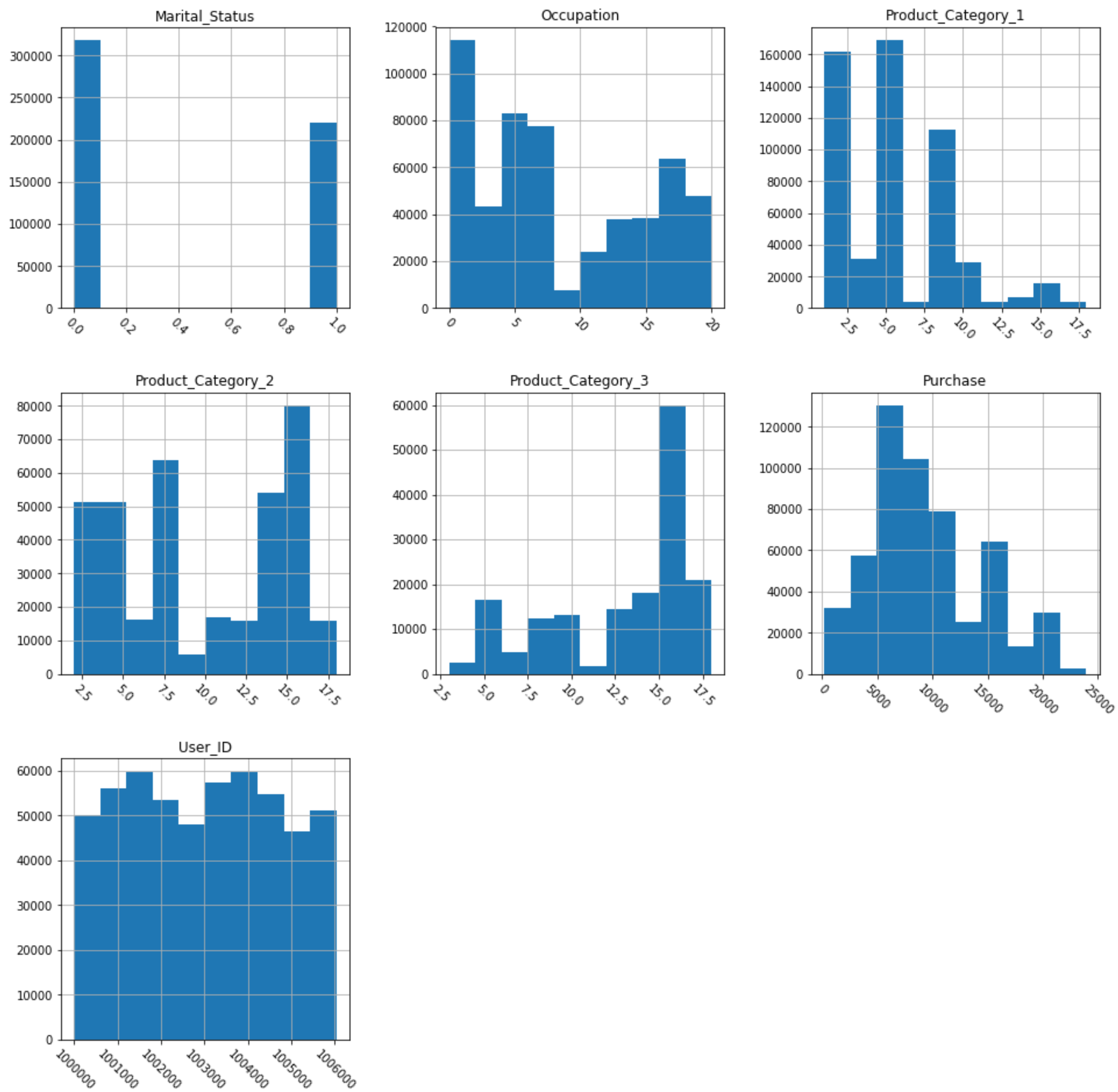
# Filtering the categorical data

In [9]:
```
df.dtypes[df.dtypes == 'object']
```

Out[9]:
```
Product_ID                    object
Gender                        object
Age                           object
City_Category                 object
Stay_In_Current_City_Years    object
dtype: object
```

# Distribution of numerical data

In [10]:
```
df.hist(figsize=(16,16), xrot=-45)
plt.show()
```



In [11]:
```
df.describe()
```

Out[11]:

| | User_ID | Occupation | Marital_Status | Product_Category_1 | Product_Category_2 | Produ |
|---|---|---|---|---|---|---|
| **count** | 5.375770e+05 | 537577.00000 | 537577.000000 | 537577.000000 | 370591.000000 | |
| **mean** | 1.002992e+06 | 8.08271 | 0.408797 | 5.295546 | 9.842144 | |
| **std** | 1.714393e+03 | 6.52412 | 0.491612 | 3.750701 | 5.087259 | |
| **min** | 1.000001e+06 | 0.00000 | 0.000000 | 1.000000 | 2.000000 | |
| **25%** | 1.001495e+06 | 2.00000 | 0.000000 | 1.000000 | 5.000000 | |
| **50%** | 1.003031e+06 | 7.00000 | 0.000000 | 5.000000 | 9.000000 | |
| **75%** | 1.004417e+06 | 14.00000 | 1.000000 | 8.000000 | 15.000000 | |
| **max** | 1.006040e+06 | 20.00000 | 1.000000 | 18.000000 | 18.000000 | |

# Distribution of categorical data

In [12]:
```
df.describe(include=['object'])
```

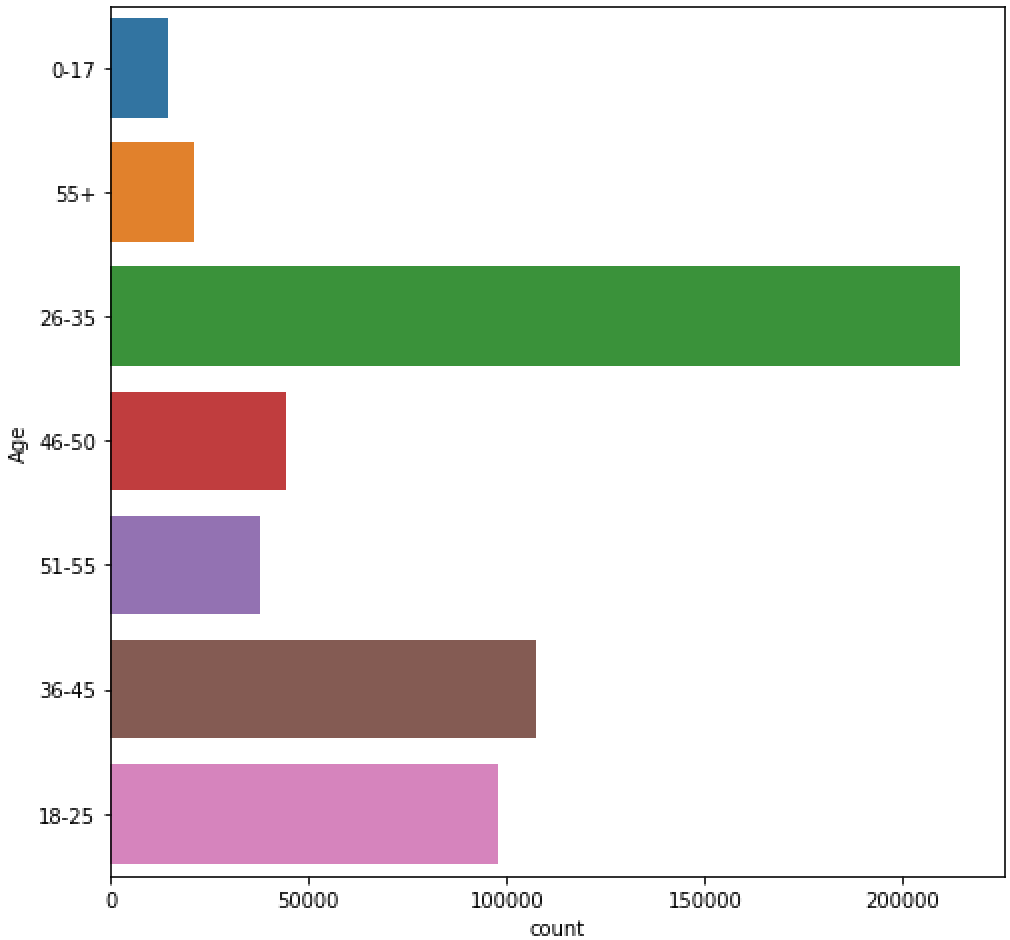
Out[12]:

| | Product_ID | Gender | Age | City_Category | Stay_In_Current_City_Years |
|---|---|---|---|---|---|
| **count** | 537577 | 537577 | 537577 | 537577 | 537577 |
| **unique** | 3623 | 2 | 7 | 3 | 5 |
| **top** | P00265242 | M | 26-35 | B | 1 |
| **freq** | 1858 | 405380 | 214690 | 226493 | 189192 |

# Bar plots for categorical data

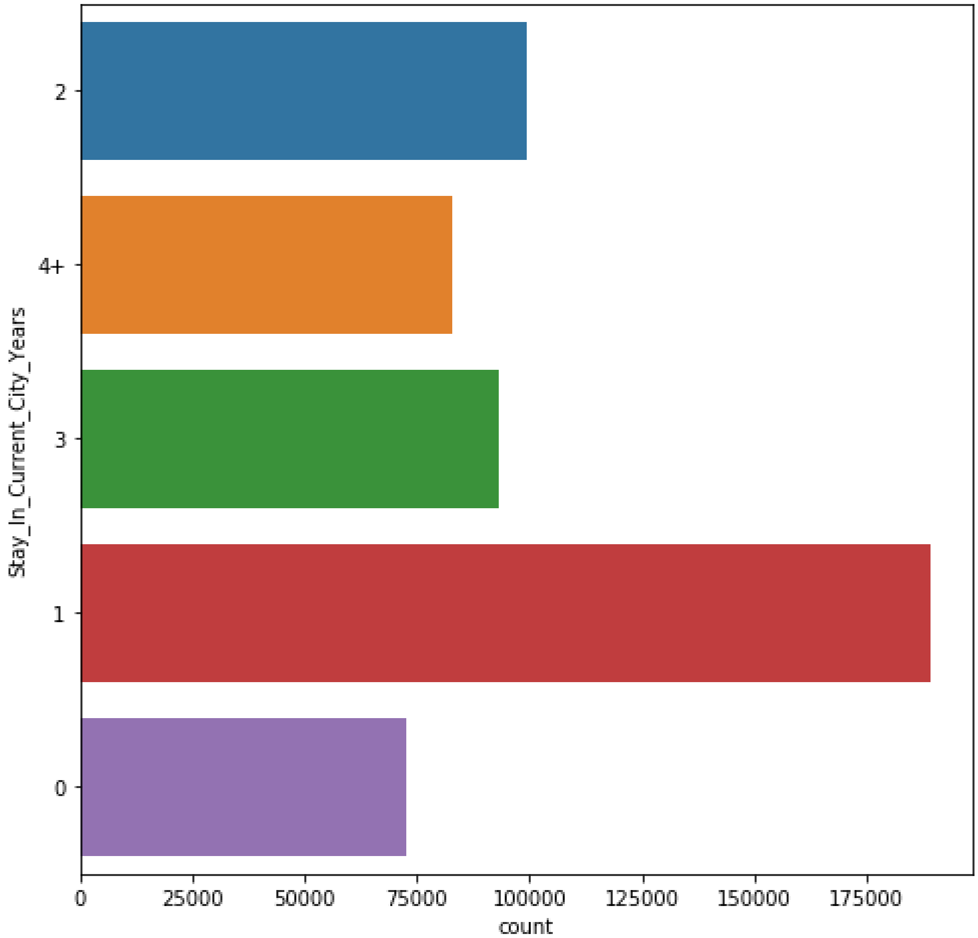
In [13]:
```
plt.figure(figsize = (8,8))
sns.countplot(y='Age',data = df)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x25033c3f780>



In [14]:
```
plt.figure(figsize = (8,8))
sns.countplot(y='Stay_In_Current_City_Years',data = df)
```

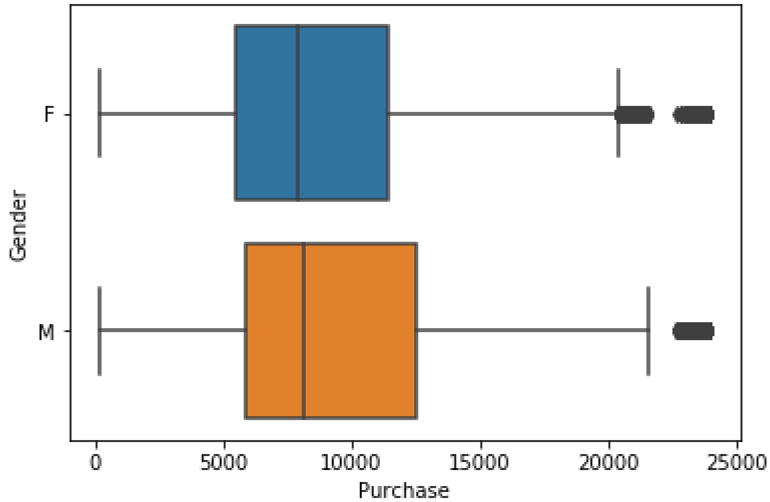
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x25034204748>




# Segmentations

In [15]:
```
sns.boxplot(y='Gender', x='Purchase', data = df)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x25033f462e8>




# Comparing two genders across other features

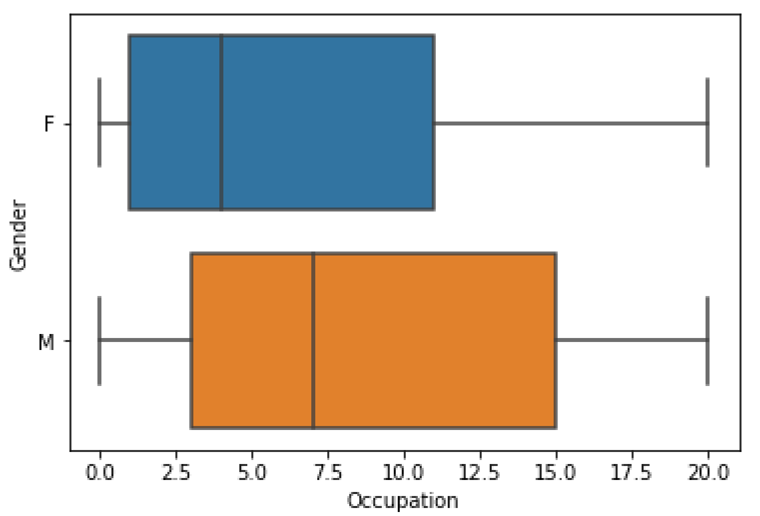
In [16]:
```
df.groupby('Gender').mean()
```

Out[16]:

| | User_ID | Occupation | Marital_Status | Product_Category_1 | Product_Category_2 | Produ |
|---|---|---|---|---|---|---|
| **Gender** | | | | | | |
| **F** | 1.003088e+06 | 6.742672 | 0.417733 | 5.595445 | 10.007969 | |
| **M** | 1.002961e+06 | 8.519705 | 0.405883 | 5.197748 | 9.789072 | |

In [17]:
```
sns.boxplot(y='Gender', x='Occupation', data = df)
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x25033cb1d68>



In [18]:
```
df.groupby('Gender').agg([np.mean,np.std])
```

Out[18]:

| | User_ID | | Occupation | | Marital_Status | | Product_Category_1 | | F |
|---|---|---|---|---|---|---|---|---|---|
| | mean | std | mean | std | mean | std | mean | std | r |
| **Gender** | | | | | | | | | |
| **F** | 1.003088e+06 | 1774.236455 | 6.742672 | 6.242116 | 0.417733 | 0.493188 | 5.595445 | 3.476495 | 1 |
| **M** | 1.002961e+06 | 1693.251916 | 8.519705 | 6.554518 | 0.405883 | 0.491063 | 5.197748 | 3.830816 | |

In [19]:
```
plt.figure(figsize=(20,20))
```

Out[19]: <Figure size 1440x1440 with 0 Axes>

<Figure size 1440x1440 with 0 Axes>

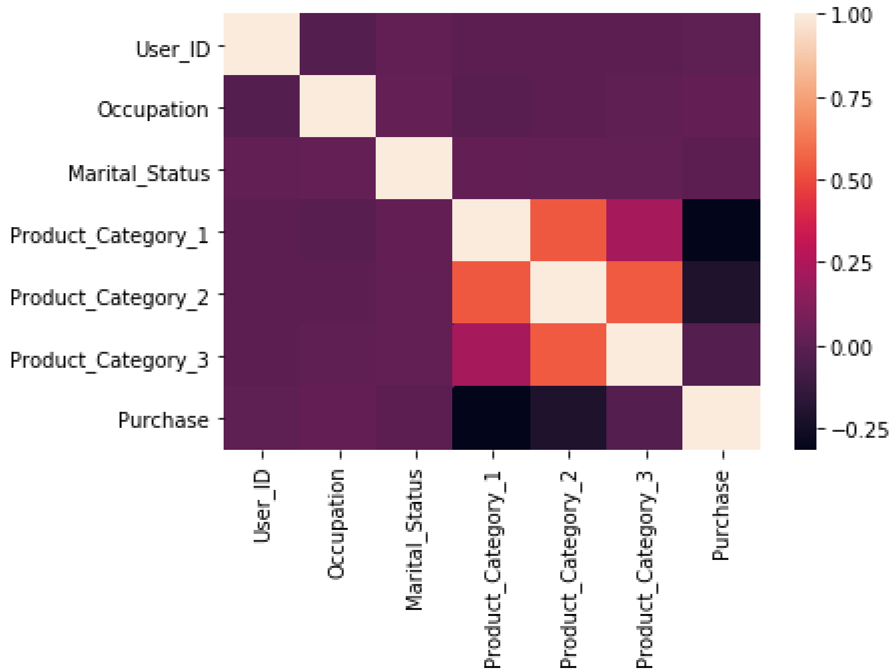
In [20]:
```
df.corr()
```

Out[20]:

| | User_ID | Occupation | Marital_Status | Product_Category_1 | Product_Category_ |
|---|---|---|---|---|---|
| **User_ID** | 1.000000 | -0.023024 | 0.018732 | 0.003687 | 0.00147 |
| **Occupation** | -0.023024 | 1.000000 | 0.024691 | -0.008114 | -0.00003 |
| **Marital_Status** | 0.018732 | 0.024691 | 1.000000 | 0.020546 | 0.01511 |
| **Product_Category_1** | 0.003687 | -0.008114 | 0.020546 | 1.000000 | 0.54042 |
| **Product_Category_2** | 0.001471 | -0.000031 | 0.015116 | 0.540423 | 1.00000 |
| **Product_Category_3** | 0.004045 | 0.013452 | 0.019452 | 0.229490 | 0.54354 |
| **Purchase** | 0.005389 | 0.021104 | 0.000129 | -0.314125 | -0.20997 |

In [21]:
```
sns.heatmap(df.corr())
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x2503433ecf8>

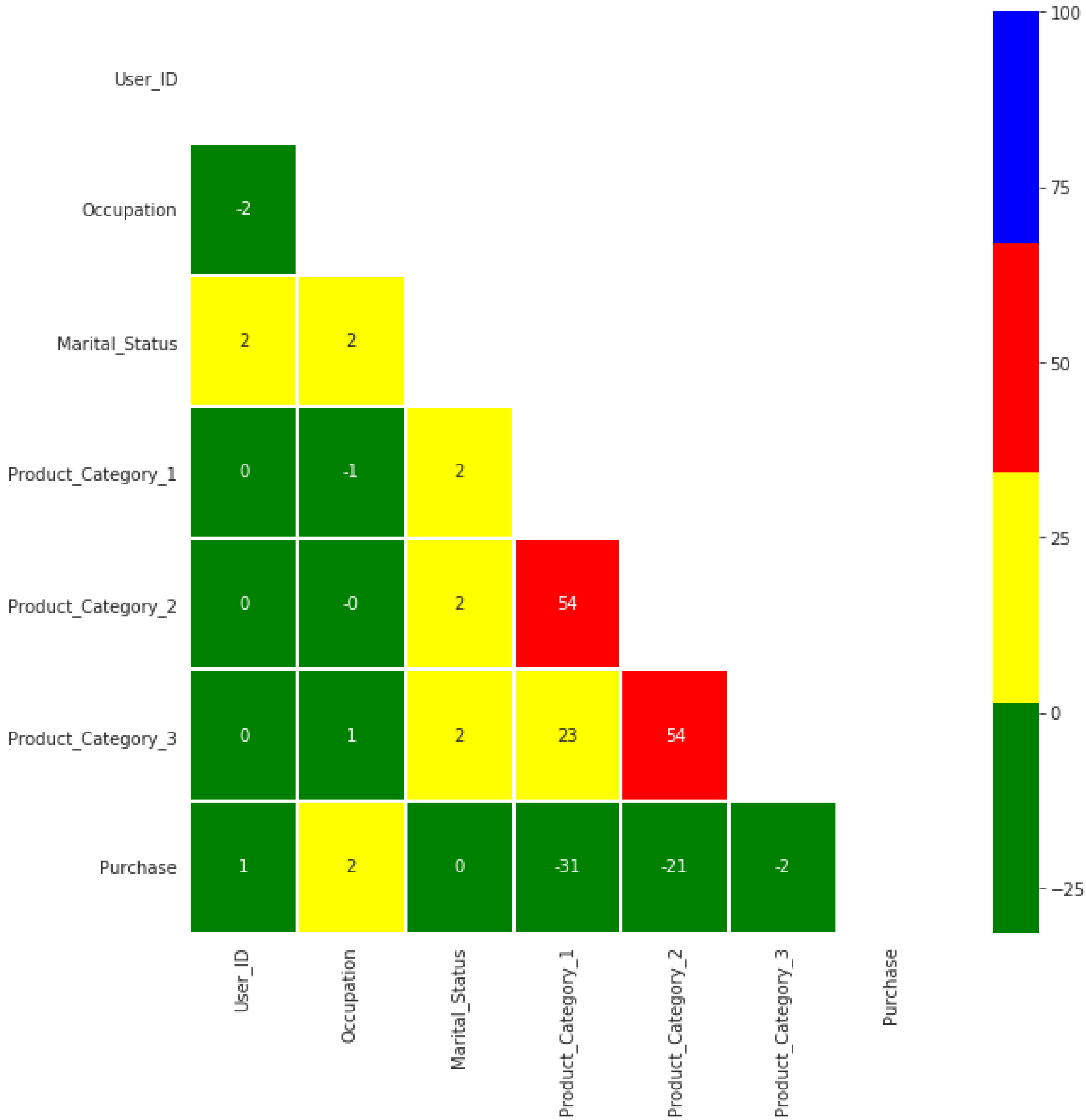



In [22]:
```
corr = df.corr()
print (corr['Purchase'].sort_values(ascending=False)[:4], '\n')
print ('----------------------')
print (corr['Purchase'].sort_values(ascending=False)[-3:]) `
```

```
Purchase          1.000000
Occupation        0.021104
User_ID           0.005389
Marital_Status    0.000129
Name: Purchase, dtype: float64

----------------------
Product_Category_3   -0.022257
Product_Category_2   -0.209973
Product_Category_1   -0.314125
Name: Purchase, dtype: float64
```

In [23]:
```
from matplotlib.colors import ListedColormap
mask = np.zeros_like(df.corr())
mask[np.triu_indices_from(mask)]= True
plt.figure(figsize = (10,10))
with sns.axes_style("white"):
    ax = sns.heatmap(df.corr()*100,mask =mask, fmt = '.0f',
                     annot = True, lw=1,cmap =ListedColormap(["green","yellow","
```




# Data Cleaning

# Drop any duplicate

In [24]:
```
df = df.drop_duplicates()
df.shape
```

Out[24]:
```
(537577, 12)
```

In [25]:
```
df.City_Category.unique()
```

Out[25]:
```
array(['A', 'C', 'B'], dtype=object)
```

In [26]:
```
df.Stay_In_Current_City_Years.unique()
```

Out[26]:
```
array(['2', '4+', '3', '1', '0'], dtype=object)
```

In [27]:
```
df.Product_Category_2.unique()
```

Out[27]:
```
array([nan,  6., 14.,  2.,  8., 15., 16., 11.,  5.,  3.,  4., 12.,  9.,
       10., 17., 13.,  7., 18.])
```

In [28]:
```
df.Product_Category_2.fillna(9, inplace = True)
```

In [29]:
```
df.Product_Category_2.unique()
```

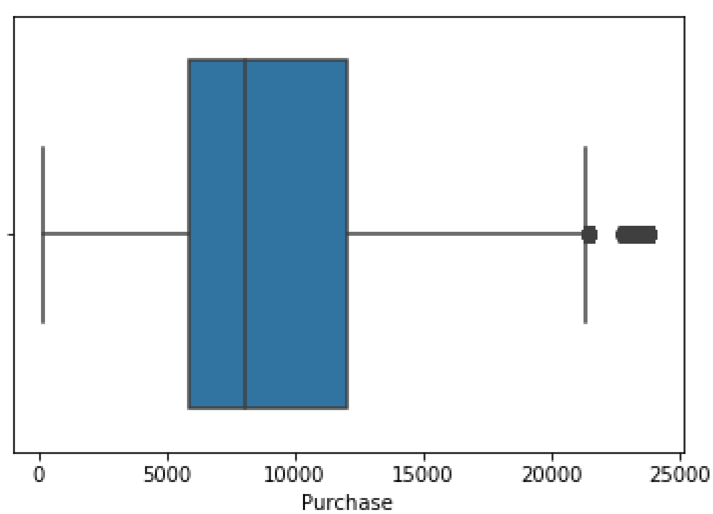
Out[29]:
```
array([ 9.,  6., 14.,  2.,  8., 15., 16., 11.,  5.,  3.,  4., 12., 10.,
       17., 13.,  7., 18.])
```

# Removing Outliers

In [30]:
```
# Outliers can cause problems with certain types of models.
# Boxplots are a nice way to detect outliers
# Let's start with a box plot of your target variable, since that's what you're c
```

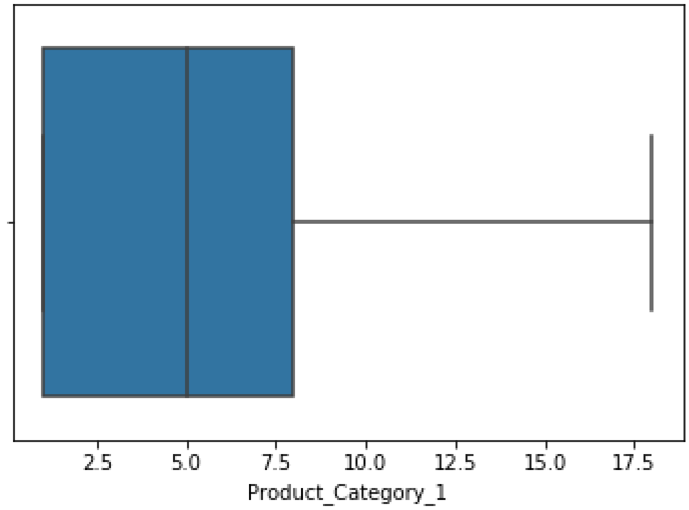
In [31]:
```
sns.boxplot(df.Purchase)
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x25033e990b8>
```



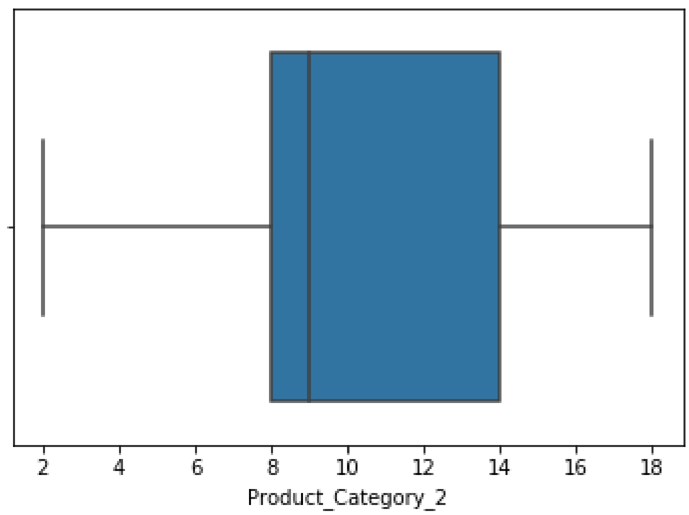
In [32]:
```
sns.boxplot(df.Product_Category_1)
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x25033df9f98>



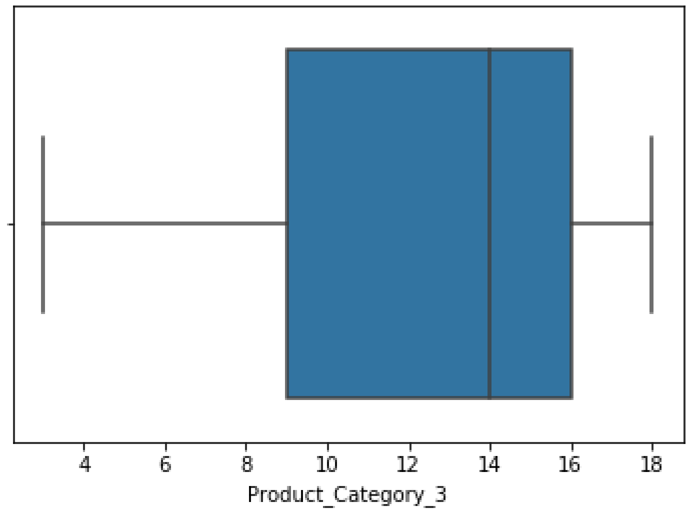

In [33]:
```
sns.boxplot(df.Product_Category_2)
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x25034414cf8>



In [34]:
```
sns.boxplot(df.Product_Category_3)
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2503447f358>
```




# Label missing categorical data

In [35]:
```
# Display number of missing values by categorical feature
df.select_dtypes(include=['object']).isnull().sum()
```

Out[35]:
```
Product_ID                    0
Gender                        0
Age                           0
City_Category                 0
Stay_In_Current_City_Years    0
dtype: int64
```

# Flag and Fill missing numerical data

In [36]:
```
# Display number of missing values by numeric feature
df.select_dtypes(exclude=['object']).isnull().sum()
```

Out[36]:
```
User_ID                    0
Occupation                 0
Marital_Status             0
Product_Category_1         0
Product_Category_2         0
Product_Category_3    373299
Purchase                   0
dtype: int64
```

In [37]:
```
df['Product_Category_3'] = df['Product_Category_3'].fillna(df['Product_Category_
df.select_dtypes(exclude=['object']).isnull().sum()
```

Out[37]:
```
User_ID               0
Occupation            0
Marital_Status        0
Product_Category_1    0
Product_Category_2    0
Product_Category_3    0
Purchase              0
dtype: int64
```

In [39]:
```
# Save cleaned dataframe to new file
#"C:\Users\Shakena Ford\Desktop\cleaneddf.csv"
df.to_csv(r'C:\Users\Shakena Ford\Desktop\cleaned.csv', index=False)
```

# Encode Dummy Variables

In [40]:
```
# Machine learning algorithms cannot directly handle categorical features. Speci
# Therefore, we need to create dummy variables for our categorical features.
# Dummy variables are a set of binary (0 or 1) features that each represent a si
# Create a new dataframe with dummy variables for for our categorical features.
```

In [41]:
```
df = pd.get_dummies(df, columns=['Gender', 'Age', 'City_Category'])
```

In [42]:
```
# Note: There are many ways to perform one-hot encoding,
# you can also use LabelEncoder and OneHotEncoder classes in SKLEARN or use the
```

In [43]:
```
df.head()
```

Out[43]:

| | User_ID | Product_ID | Occupation | Stay_In_Current_City_Years | Marital_Status | Product_Category_ |
|---|---|---|---|---|---|---|
| **0** | 1000001 | P00069042 | 10 | 2 | 0 | |
| **1** | 1000001 | P00248942 | 10 | 2 | 0 | |
| **2** | 1000001 | P00087842 | 10 | 2 | 0 | 1 |
| **3** | 1000001 | P00085442 | 10 | 2 | 0 | 1 |
| **4** | 1000002 | P00285442 | 16 | 4+ | 0 | |

5 rows × 21 columns

In [45]:
```
# Save cleaned dataframe to new file
#"C:\Users\Shakena Ford\Desktop\cleaneddf.csv"
df.to_csv(r'C:\Users\Shakena Ford\Desktop\analytical.csv', index=False)
```

# Machine Learning

# Data Preparation

In [48]:
```
df = pd.read_csv("analytical.csv")
```

In [49]:
```
y = df.Purchase
x = df.drop('Purchase', axis = 1)
```

In [50]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_
```

In [51]:
```
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(430061, 20) (107516, 20) (430061,) (107516,)
```

# Data Standardization

In [*]:
```

```

In [*]:
```
train_mean = x_train.mean()
train_std = x_train.std()
```

In [*]:
```

x_train = (x_train - train_mean) / train_std
```

In [*]:
```

x_train.describe()
```

In [*]:
```

x_test = (x_test - train_mean) / train_std
```

In [*]:
```

x_test.describe()
```

# Baseline Model

In [*]:

In [*]:

```python

y_train_pred = np.ones(y_train.shape[0])*y_train.mean()
```

In [*]:

```python
## Predict Test results
y_pred = np.ones(y_test.shape[0])*y_train.mean()
from sklearn.metrics import r2_score
```

In [*]:

```python
print("Train Results for Baseline Model:")
print("*******************************")
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pred)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
print("Mean Absolute Error: ", mae(y_train.values, y_train_pred))
```

In [ ]:

In [ ]: