In [1]:
```
# 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 regul
arization
from sklearn.linear_model import Lasso  # Linear Regression + L1 regul
arization
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\\mingw64\\bin'
```

```
os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']
from xgboost import XGBRegressor
from xgboost import plot_importance  # to plot feature importance

# to save the final model on disk
from sklearn.externals import joblib
```

In [2]:
```
np.set_printoptions(precision=2, suppress=True) #for printing floating point numbers upto  precision 2
```

In [3]:
```
df = pd.read_csv('BlackFriday.csv')
```

In [4]:
```
df.shape
```

Out[4]:
```
(537577, 12)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City
_Category',
       'Stay_In_Current_City_Years', 'Marital_Status', 'Product_Cate
gory_1',
       'Product_Category_2', 'Product_Category_3', 'Purchase'],
      dtype='object')
```

Some feaures are numeric and some are categorical

Filtering the categorical features:

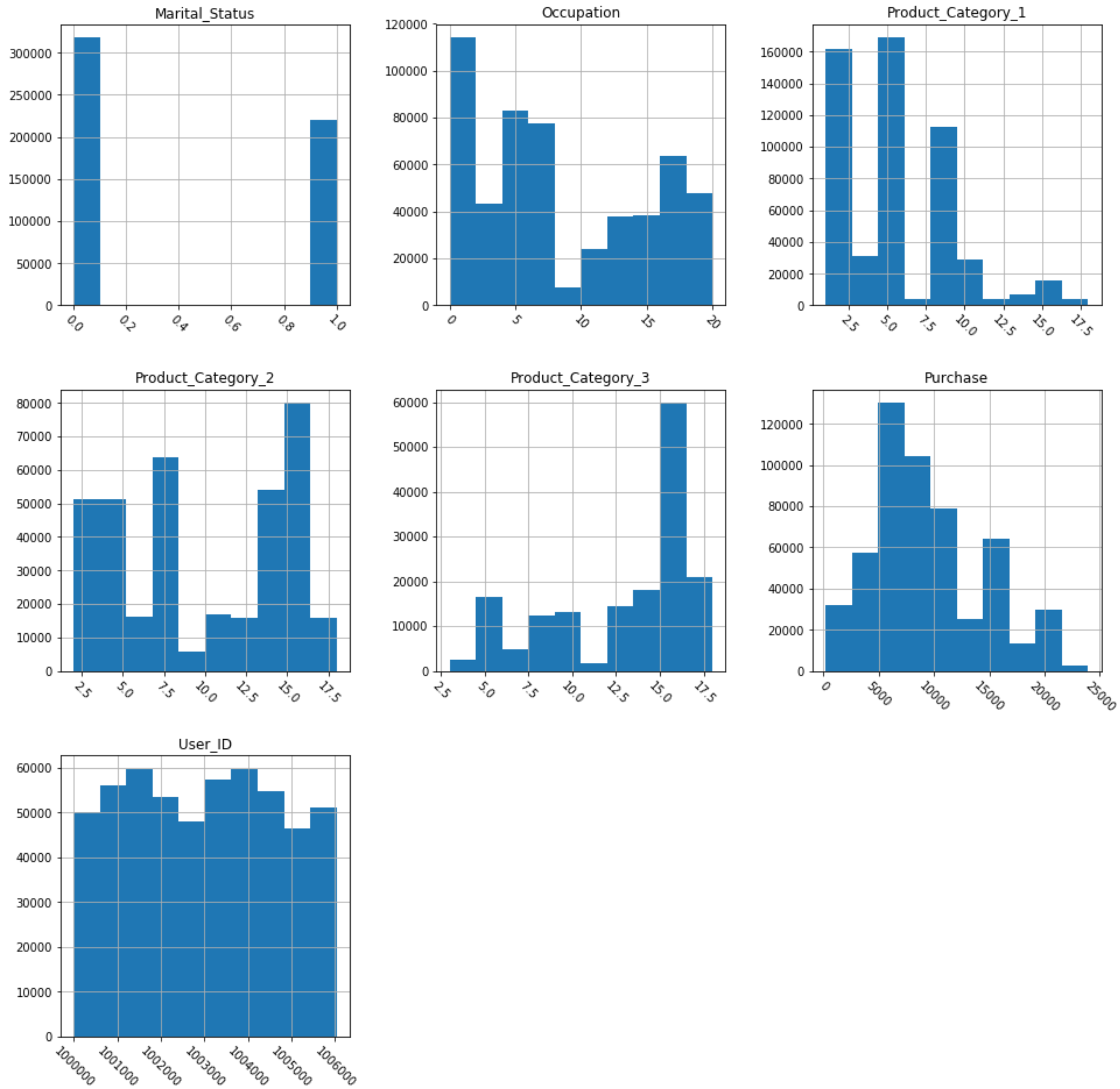
In [6]:
```
df.dtypes[df.dtypes=='object']
```

Out[6]:
```
Product_ID                    object
Gender                        object
Age                           object
City_Category                 object
Stay_In_Current_City_Years    object
dtype: object
```

Distributions of numeric features

In [7]:
```
# Plot histogram grid
df.hist(figsize=(16,16), xrot=-45) ## Display the labels rotated by 45 degress

# Clear the text "residue"
plt.show()
```



In [8]:
```
# Consider the histogram of Marital_Status:
# More than 300,000 people are unmarried and while less than 250,000 are married.

# Consider the histogram of occupation:
# It can be observed that more than 100,000 people have been unemployed compared
```

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

Out[9]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	P
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	

In [10]:
```
# The Marital_Status and Occupation coulmn have some missing values al
so it has a minimum value of 0.0
# For the column: 'Occupation', it can be observed that the max value
is 20.
```

In [11]:
```
df.describe(include=['object'])
```

Out[11]:

	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

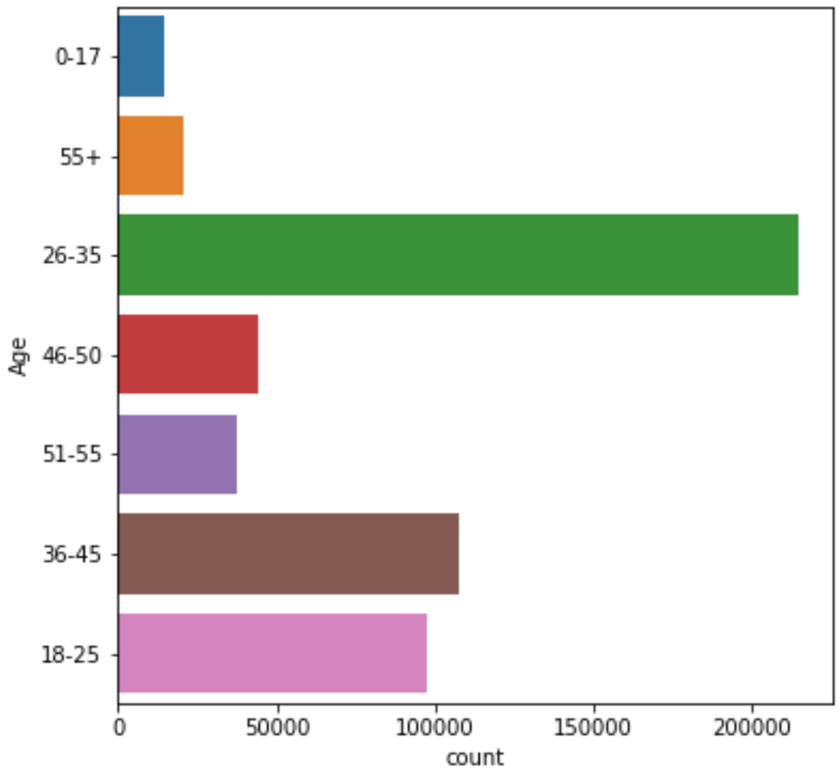
In [12]:
```
# Observation

# There are 5 unique classes for the category Stay_In_Current_City_Yea
rs
# Also, the most frequent time people have stayed in a city is 1 years
which
# has occured for 189192 times

# It can also be observed that 'B' is the most frequent element for Ci
ty_Category
# which has occured 226493 times
```

In [13]:
```
plt.figure(figsize=(6,6))
sns.countplot(y='Age', data=df)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1a17a58080>



In [14]:
```
# Observation

#Here, it can be seen that the age group of people from (26-35)
#years is most frequent

# People aged between (0-17)years occur less frequently according to t
he bar plot
```

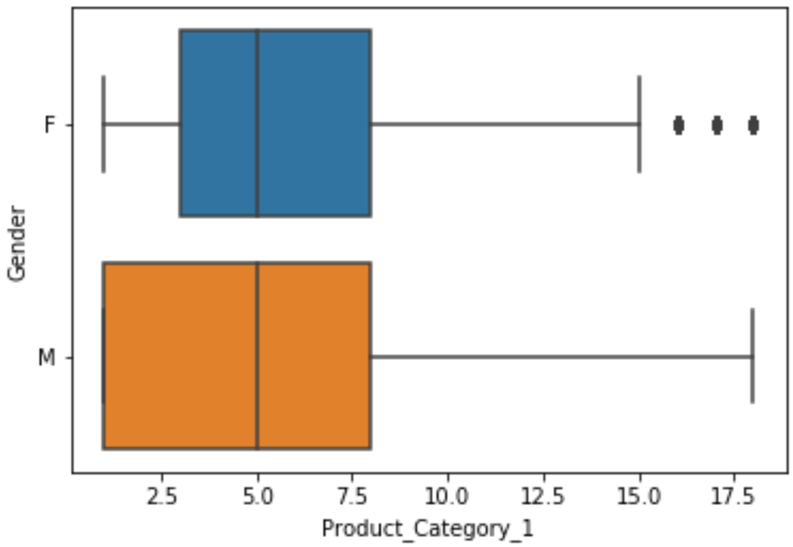
Segmentations:

In [15]:
```
#Segmentations are powerful ways to cut the data to observe the relati
onship between categorical features and numeric features.

#Segmenting the target variable by key categorical features.
```

In [16]:
```
sns.boxplot(y='Gender', x='Product_Category_1', data=df)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1a16d6cc18>



In [17]:
```
# It can be observed that more male customers were interested in Produ
ct_Category_1
```

In [18]:
```
df.groupby('Gender').mean()
```

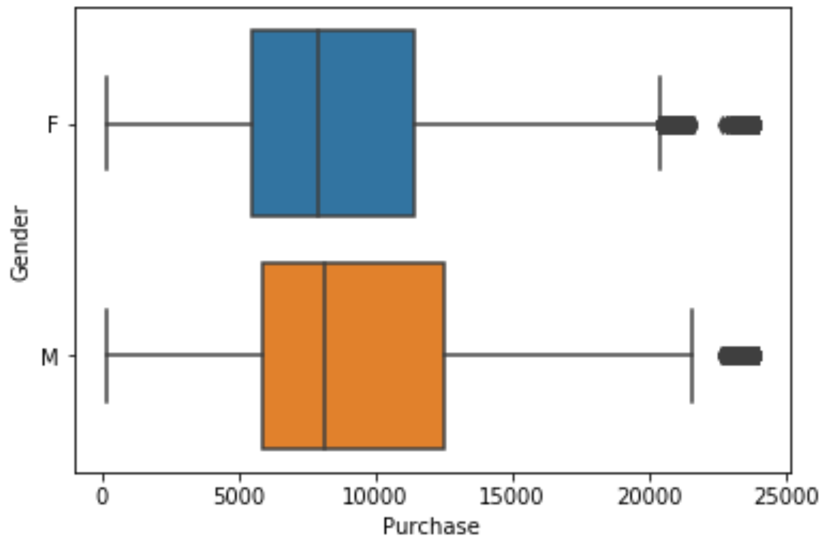
Out[18]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	P
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 [19]:
```
# Both male and female customers are highly interested in Product_Cate
gory_3
# Higher number of male customers have an occupation
# More number of females are married compared to men
# Higher purchases are made by males then females
```

In [20]:
```
sns.boxplot(y='Gender', x='Purchase', data=df)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1a17a63da0>



Observations:

In [21]:
```
# Although, the purchases made in terms of males in females
# can be visalized to be the same. In comparison, the purchases
# made by males is more than that of females
```

Segment by Gender and display the means and standard deviations within each class

In [22]:
```
df.groupby('Gender').agg([np.mean,np.std])
```

Out[22]:

	User_ID		Occupation		Marital_Status		Product_Category_	
	mean	std	mean	std	mean	std	mean	std
Gender								
F	1.003088e+06	1774.236455	6.742672	6.242116	0.417733	0.493188	5.595445	3.47649
M	1.002961e+06	1693.251916	8.519705	6.554518	0.405883	0.491063	5.197748	3.83081

In [23]:
```
# Correlation is a number between -1 and 1 that represents how closely
#related two separate features are.
#Positive number indicates that as one feature increases, the other in
creases
#whereas negative number indicates that as one increases, the other de
creases.
#Correlations near -1 or 1 indicate a strong relationship.
#Those closer to 0 indicate a weak relationship.
#0 indicates no relationship.
```

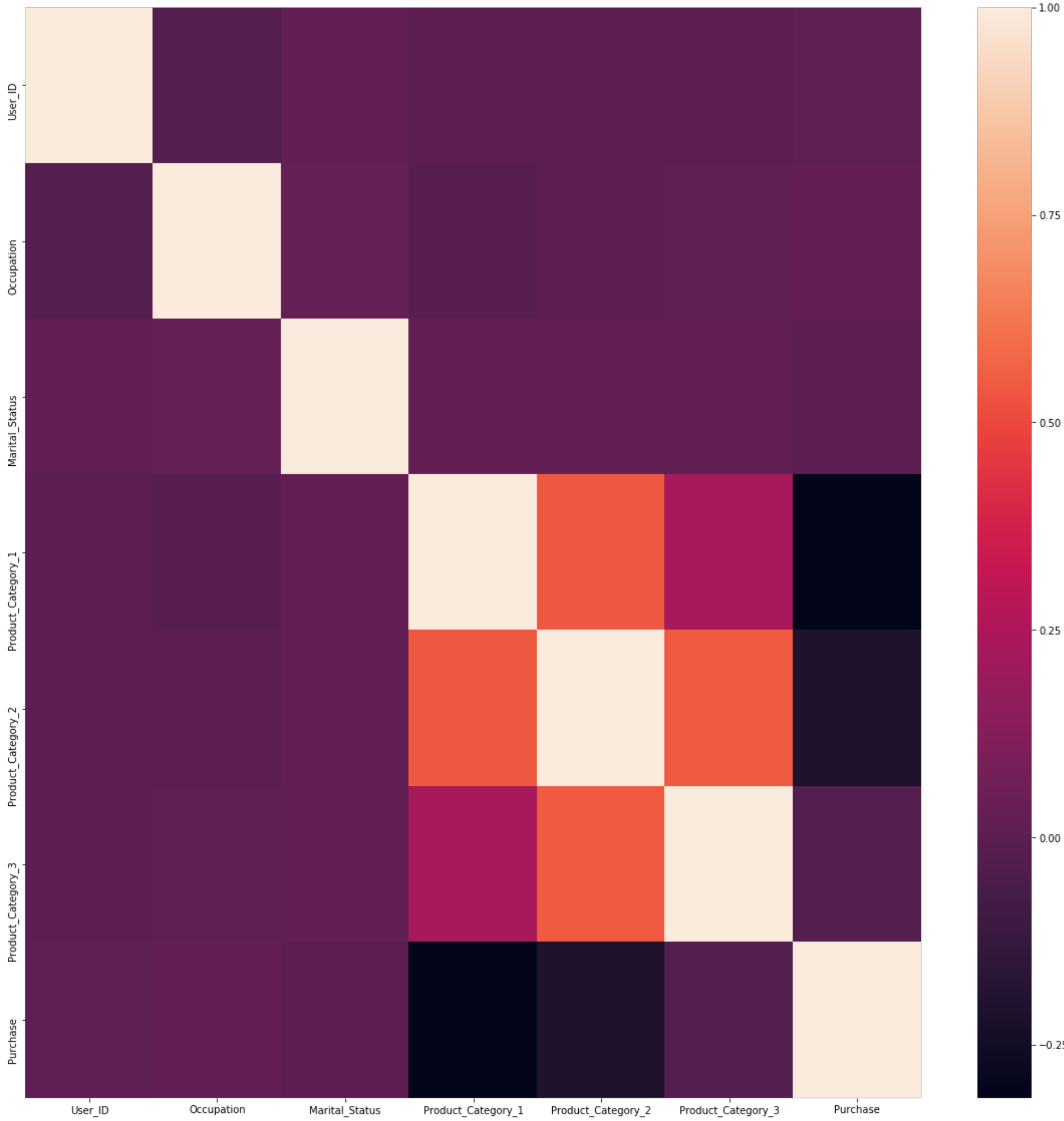
In [24]:
```
df.corr()
```

Out[24]:

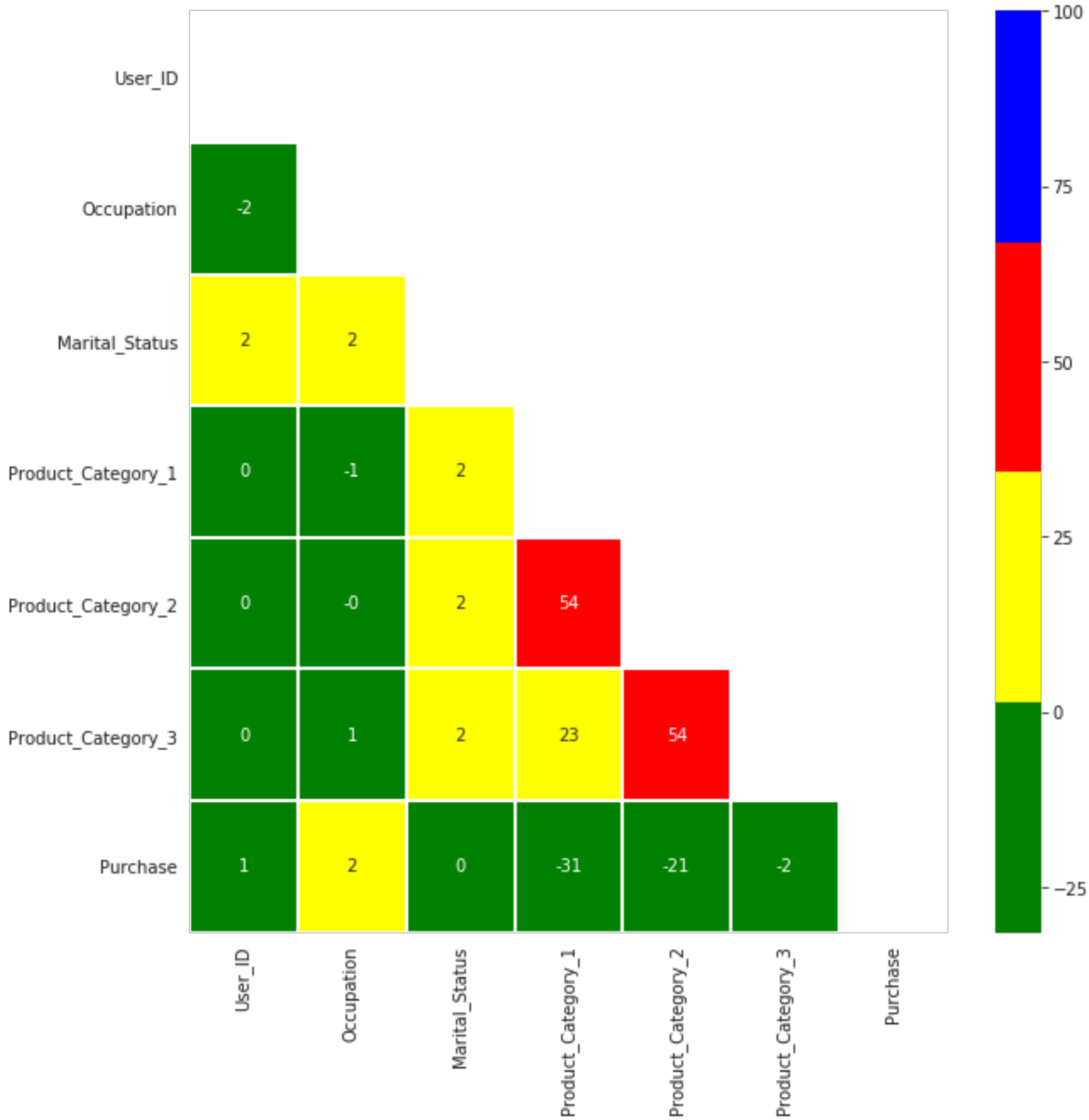
	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Categ
User_ID	1.000000	-0.023024	0.018732	0.003687	0.0
Occupation	-0.023024	1.000000	0.024691	-0.008114	-0.0
Marital_Status	0.018732	0.024691	1.000000	0.020546	0.0
Product_Category_1	0.003687	-0.008114	0.020546	1.000000	0.5
Product_Category_2	0.001471	-0.000031	0.015116	0.540423	1.0
Product_Category_3	0.004045	0.013452	0.019452	0.229490	0.5
Purchase	0.005389	0.021104	0.000129	-0.314125	-0.2

In [25]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr())
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1a174c4a20>


User_ID
Occupation
Marital_Status
Product_Category_1
Product_Category_2
Product_Category_3
Purchase
1.00
0.75
0.50
0.25
0.00
−0.25

In [26]:
```
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', 'red','blue']))
```



Data Cleaning

In [27]:
```
# De-duplication (Dropping duplicate data)
```

In [28]:
```
df = df.drop_duplicates()
print( df.shape )
```

```
(537577, 12)
```

In [29]:
```
# Since we do not get a different number than before, it looks
# like we did not have any duplicates.
```

Fix structural errors

In [30]:
```
# The Product_Category_2 and 3 feature has some nan values, to handle
them:
```

In [31]:
```
df.Product_Category_2.unique()
```

Out[31]:
```
array([nan,  6., 14.,  2.,  8., 15., 16., 11.,  5.,  3.,  4., 12.,
9.,
       10., 17., 13.,  7., 18.])
```

In [32]:
```
df.Product_Category_3.unique()
```

Out[32]:
```
array([nan, 14., 17.,  5.,  4., 16., 15.,  8.,  9., 13.,  6., 12.,
3.,
       18., 11., 10.])
```

In [33]:
```
df.Product_Category_2.fillna(0, inplace=True)
df.Product_Category_3.fillna(0, inplace=True)
df.Product_Category_2.unique()
```

Out[33]:
```
array([ 0.,  6., 14.,  2.,  8., 15., 16., 11.,  5.,  3.,  4., 12.,
9.,
       10., 17., 13.,  7., 18.])
```

In [34]:
```
df.Product_Category_3.unique()
```

Out[34]:
```
array([ 0., 14., 17.,  5.,  4., 16., 15.,  8.,  9., 13.,  6., 12.,
3.,
       18., 11., 10.])
```

Removing Outliers

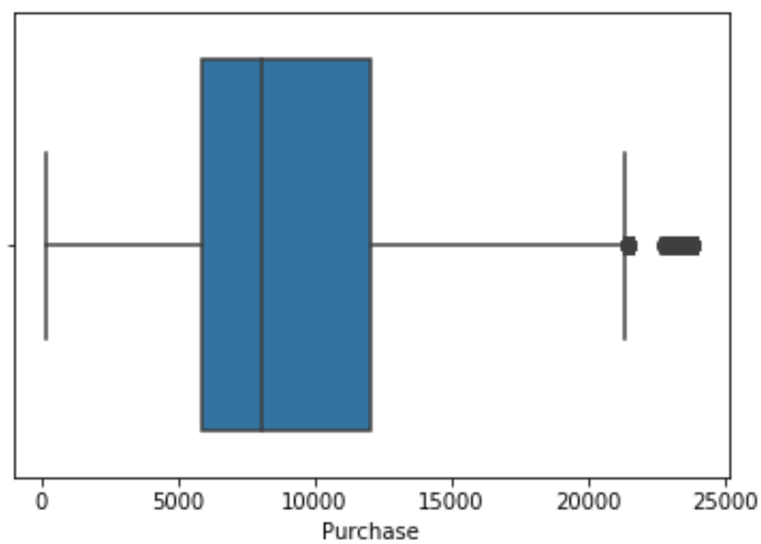
Outliers can cause problems with certain types of models.

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 actually trying to predict

In [35]:
```
sns.boxplot(df.Purchase)
```

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1742fda0>



Interpretation

In [36]:
```
# The two vertical bars on the ends are the min and max values. All pr
operties sold for between \$200,000 and \$800,000.
# The box in the middle is the interquartile range (25th percentile to
75th percentile).
# Half of all observations fall in that box.
# Finally, the vertical bar in the middle of the box is the median.
```

In [37]:
```
df.Purchase.sort_values(ascending=False).head()
```

Out[37]:
```
87440     23961
93016     23961
370891    23961
503697    23960
321782    23960
Name: Purchase, dtype: int64
```

Label missing categorical data

In [38]:
```
# You cannot simply ignore missing values in your dataset.
# You must handle them in some way for the very practical reason that Scikit-Learn algorithms
# do not accept missing values.
```

In [39]:
```
# Display number of missing values by categorical feature
df.select_dtypes(include=['object']).isnull().sum()
```

Out[39]:
```
Product_ID                    0
Gender                        0
Age                           0
City_Category                 0
Stay_In_Current_City_Years    0
dtype: int64
```

In [40]:
```
# There are no missing values in this dataset
```

Flag and fill missing numeric data

In [41]:
```
# Display number of missing values by numeric feature
df.select_dtypes(exclude=['object']).isnull().sum()
```

Out[41]:
```
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 [42]:
```
# There are no numerical features with missing values in the dataset
```

Saving the recently cleaned data set to a new file

In [43]:
```
df.to_csv(r'/Users/tenzinpelchok/Desktop\CleanFile.csv', index=False)
```

Feature Engineering

Encode dummy variables (One Hot Encoding)

In [44]:
```
# Machine learning algorithms cannot directly handle categorical featu
res. Specifically, they cannot handle text values.
# Therefore, we need to create dummy variables for our categorical fea
tures.
# Dummy variables are a set of binary (0 or 1) features that each repr
esent a single class from a categorical feature.
```

In [45]:
```
# Create a new dataframe with dummy variables for for our categorical
features.
df = pd.get_dummies(df, columns=['Gender', 'Stay_In_Current_City_Years
'])
```

In [46]:
```
df.head()
```

Out[46]:

	User_ID	Product_ID	Age	Occupation	City_Category	Marital_Status	Product_Category_1
0	1000001	P00069042	0-17	10	A	0	3
1	1000001	P00248942	0-17	10	A	0	1
2	1000001	P00087842	0-17	10	A	0	12
3	1000001	P00085442	0-17	10	A	0	12
4	1000002	P00285442	55+	16	C	0	8

Remove unused or redundant features

In [47]:
```
# I am removing the features: User_ID, Occupation, Marital_Status
```

In [48]:
```
df=df.drop(['Age','City_Category','Marital_Status','Product_ID', 'User
_ID', 'Product_Category_3'], axis=1)
```

In [51]:
```
df.to_csv(r'C:\Users\tenzinpelchok\Desktop\analytical.csv', index=None
)
```

In [52]:
```
df.shape
```

Out[52]:
```
(537577, 11)
```

Machine Learning Models

In [55]:
```
# Data Preparation
df = pd.read_csv("analytical.csv")
```

In [56]:
```
#In this step, the data is separated into two different parts
# Target Variable(y) and input feature(x) at a 80-20 ratio.
#80%:target variable and 20%:input variable
```

In [57]:
```
#Creating separate objects for target and input:
y= df.Purchase
X=df.drop('Purchase', axis=1)
```

In [58]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
2, random_state=1234)
```

In [59]:
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(430061, 10) (107516, 10) (430061,) (107516,)
```

In [60]:
```
# Data Standardization
```

In [61]:
```
#In this step, we make all the means of the features to zero
#and the standard deviation to 1.
```

In [62]:
```
train_mean = X_train.mean()
train_std = X_train.std()
```

In [63]:
```
X_train = (X_train - train_mean) / train_std
```

In [64]:
```
X_train.describe()
```

Out[64]:

	Occupation	Product_Category_1	Product_Category_2	Gender_F	Gender_M
count	4.300610e+05	4.300610e+05	4.300610e+05	4.300610e+05	4.300610e+05
mean	-1.303425e-15	-2.232773e-16	8.255428e-17	1.978527e-15	-1.978527e-15
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.239073e+00	-1.146040e+00	-1.094008e+00	-5.717209e-01	-1.749101e+00
25%	-9.323963e-01	-1.146040e+00	-1.094008e+00	-5.717209e-01	5.717209e-01
50%	-1.657057e-01	-7.924827e-02	-2.892124e-01	-5.717209e-01	5.717209e-01
75%	9.076610e-01	7.208454e-01	1.159419e+00	-5.717209e-01	5.717209e-01
max	1.827690e+00	3.387824e+00	1.803255e+00	1.749101e+00	5.717209e-01

In [65]:
```
X_test = (X_test - train_mean) / train_std
```

In [66]:
```
X_test.describe()
```

Out[66]:

	Occupation	Product_Category_1	Product_Category_2	Gender_F	Gender_M
count	107516.000000	107516.000000	107516.000000	107516.000000	107516.000000
mean	0.001575	-0.002133	-0.009578	-0.005007	0.005007
std	1.001983	1.001522	0.999048	0.997039	0.997039
min	-1.239073	-1.146040	-1.094008	-0.571721	-1.749101
25%	-0.932396	-1.146040	-1.094008	-0.571721	0.571721
50%	-0.165706	-0.079248	-0.289212	-0.571721	0.571721
75%	0.907661	0.720845	1.159419	-0.571721	0.571721
max	1.827690	3.387824	1.803255	1.749101	0.571721

Model 1 - Baseline Model

In [67]:
```
# The average of the train labels are taken as output
# for every test data point
```

In [68]:
```
## Predict Train results
y_train_pred = np.ones(y_train.shape[0])*y_train.mean()
```

In [69]:
```
## Predict Test results
y_pred = np.ones(y_test.shape[0])*y_train.mean()
from sklearn.metrics import r2_score
```

In [70]:
```
print("Train Results for Baseline Model:")
print("*******************************")
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pr
ed)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
print("Mean Absolute Error: ", mae(y_train.values, y_train_pred))
```

```
Train Results for Baseline Model:
*******************************
Root mean squared error:  4981.515912062438
R-squared:  0.0
Mean Absolute Error:  4047.5660267444778
```

In [71]:
```
print("Results for Baseline Model:")
print("*******************************")
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
print("Mean Absolute Error: ", mae(y_test, y_pred))
```

```
Results for Baseline Model:
*******************************
Root mean squared error:  4979.023398336429
R-squared:  -6.53743990053357e-08
Mean Absolute Error:  4047.0879520090007
```

Model-2 Ridge Regression

In [72]:
```
tuned_params = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
10000, 100000]}
model = GridSearchCV(Ridge(), tuned_params, scoring = 'neg_mean_absolu
te_error', cv=10, n_jobs=-1)
model.fit(X_train, y_train)
```

Out[72]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, m
ax_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1
000, 10000, 100000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn
',
       scoring='neg_mean_absolute_error', verbose=0)
```

In [73]:
```
model.best_estimator_
```

Out[73]:
```
Ridge(alpha=0.0001, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [74]:
```
## Predict Train results
y_train_pred = model.predict(X_train)
```

In [75]:
```
## Predict Test results
y_pred = model.predict(X_test)
```

In [76]:
```
print("Train Results for Ridge Regression:")
print("*******************************")
print("Root mean squared error: ", sqrt(mse(y_train.values, y_train_pr
ed)))
print("R-squared: ", r2_score(y_train.values, y_train_pred))
print("Mean Absolute Error: ", mae(y_train.values, y_train_pred))
```

```
Train Results for Ridge Regression:
*******************************
Root mean squared error:  4721.927282620077
R-squared:  0.10150524633350189
Mean Absolute Error:  3631.6689728071633
```

In [77]:
```
print("Test Results for Ridge Regression:")
print("*******************************")
print("Root mean squared error: ", sqrt(mse(y_test, y_pred)))
print("R-squared: ", r2_score(y_test, y_pred))
print("Mean Absolute Error: ", mae(y_test, y_pred))
```

```
Test Results for Ridge Regression:
*******************************
Root mean squared error:  4717.695389577035
R-squared:  0.10221677543778518
Mean Absolute Error:  3629.6535458550857
```

Feature Importance

In [78]:
```
## Building the model again with the best hyperparameters
model = Ridge(alpha=100)
model.fit(X_train, y_train)
```

Out[78]:
```
Ridge(alpha=100, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [79]:
```
indices = np.argsort(-abs(model.coef_))
print("The features in order of importance are:")
print(50*'-')
for feature in X.columns[indices]:
    print(feature)
```

```
The features in order of importance are:
--------------------------------------------------
Product_Category_1
Product_Category_2
Gender_M
Gender_F
Occupation
Stay_In_Current_City_Years_0
Stay_In_Current_City_Years_2
Stay_In_Current_City_Years_4+
Stay_In_Current_City_Years_1
Stay_In_Current_City_Years_3
```