BLACK FRIDAY

Data Science Project

Professor Bertnart Hassani

University of Paris 1 Panthéon-Sorbonne

January 2019

Azadeh Mojarad

Student of MMMEF Track

Problem Description:

The dataset here is a sample of the transactions made in a retail store. The store wants to know better the customer purchase behaviour against different products. Specifically, here the problem is a regression problem where we are trying to predict the dependent variable (the amount of purchase) with the help of the information contained in the other variables.

Classification problem can also be settled in this dataset since several variables are categorical, and some other approaches could be "Predicting the age of the consumer" or even "Predict the category of goods bought". This dataset is also particularly convenient for clustering and maybe find different clusters of consumers within it.

Importing the basic libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
```

```
dataset_orig=pd.read_csv('C:/Users/azade/Desktop/BlackFriday.csv')
dataset_orig.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN

```
data: pd.DataFrame = pd.read_csv('C:/Users/azade/Desktop/BlackFriday.csv')
describe = data.describe()
describe.loc['#unique'] = data.nunique()
display(describe)
```

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
count	5.375770e+05	537577.00000	537577.000000	537577.000000	370591.000000	164278.000000	537577.000000
mean	1.002992e+06	8.08271	0.408797	5.295546	9.842144	12.669840	9333.859853
std	1.714393e+03	6.52412	0.491612	3.750701	5.087259	4.124341	4981.022133
min	1.000001e+06	0.00000	0.000000	1.000000	2.000000	3.000000	185.000000
25%	1.001495e+06	2.00000	0.000000	1.000000	5.000000	9.000000	5866.000000
50%	1.003031e+06	7.00000	0.000000	5.000000	9.000000	14.000000	8062.000000
75%	1.004417e+06	14.00000	1.000000	8.000000	15.000000	16.000000	12073.000000
max	1.006040e+06	20.00000	1.000000	18.000000	18.000000	18.000000	23961.000000
#unique	5.891000e+03	21.00000	2.000000	18.000000	17.000000	15.000000	17959.000000

Data Overview:

Dataset has 537577 rows (transactions) and 12 columns (features) about the black Friday in a retail store, as described below:

(It contains different kinds of variables either numerical or categorical and also missing values too).

- User_ID: Unique ID of the user. There are a total of 5891 users in the dataset.
- Product_ID: Unique ID of the product. There are a total of 3623 products in the dataset.
- Gender: indicates the gender of the person making the transaction.
- Age: indicates the age group of the person making the transaction.
- Occupation: shows the occupation of the user, already labeled with numbers 0 to 20.
- City_Category: User's living city category. Cities are categorized into 3 different categories 'A', 'B' and 'C'.
- Stay_In_Current_City_Years: Indicates how long the users has lived in this city.
- Marital_Status: is 0 if the user is not married and 1 otherwise.
- Product_Category_1 to _3: Category of the product. All 3 are already labelled with numbers.
- Purchase: Purchase amount.

Some insights about the dataset and transactions:

```
purchase_desc = data['Purchase'].describe()
purchase_desc.drop(['count', 'std'], inplace=True)
purchase_desc.loc['sum'] = data['Purchase'].sum()
purchase_desc.loc['mean_by_user'] = data['Purchase'].sum() / data['User_ID'].nunique()
display(pd.DataFrame(purchase_desc).T)
```

	mean	min	25%	50%	75%	max	sum	mean_by_user
Purchase	9333.859853	185.0	5866.0	8062.0	12073.0	23961.0	5.017668e+09	851751.549482

Mean purchase amount by transaction is 9333 and mean amount by each user is about 850,000. Values are probably not in USD.

```
null_percent = (data.isnull().sum() / len(data))*100
display(pd.DataFrame(null_percent[null_percent > 0].apply(lambda x: "{:.2f}%".format(x)),columns=['Null %']))
```

	Null %
Product_Category_2	31.06%
Product_Category_3	69.44%

Only Product_Category_2 and Product_Category_3 have null values. However Product_Category_3 is null for nearly 70% of transactions so it can't give us much information.

```python
cat_describe = data[['Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category', 'Marital_Status', 'Product_Category_1
cat_describe.loc['percent'] = 100*cat_describe.loc['freq'] / cat_describe.loc['count']
display(cat_describe)
```

	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1
count	537577	537577	537577	537577.00000	537577	537577.000000	537577.000000
unique	3623	2	7	21.00000	3	2.000000	18.000000
top	P00265242	M	26-35	4.00000	B	0.000000	5.000000
freq	1858	405380	214690	70862.00000	226493	317817.000000	148592.000000
percent	0.345625	75.4087	39.9366	13.18174	42.1322	59.120275	27.641064

A basic observation is that:

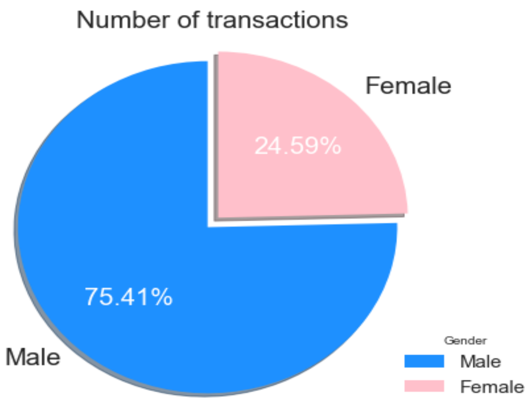
- Product `P00265242` is the most popular product.
- Most of the transactions were made by men.
- Age group with most transactions was `26-35`.

Feature Analysis:

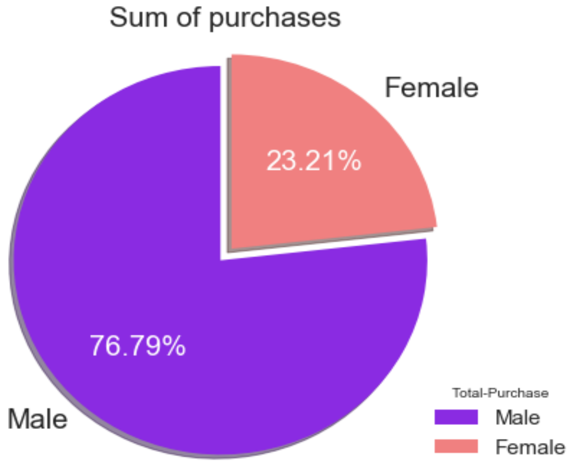
Gender:

Want to see how much men and women have purchased and how many transactions they have done.

```python
plt.figure(figsize=(9, 6))

colors = ['dodgerblue', 'pink']
labels = ['Male', 'Female']

patches, l_text, p_text = plt.pie(
    dataset_orig.Gender.value_counts(),
    labels=labels,
    colors=colors,
    explode=(0, 0.08),
    autopct='%4.2f%%',
    startangle=90,
    shadow=True)

for t in l_text + p_text:
    t.set_size(20)
for t in p_text:
    t.set_color('white')
plt.legend(fontsize=15, loc='best', title='Gender', frameon=False)
plt.axis('equal')
plt.title('Number of transactions', size=20)
plt.show()
```

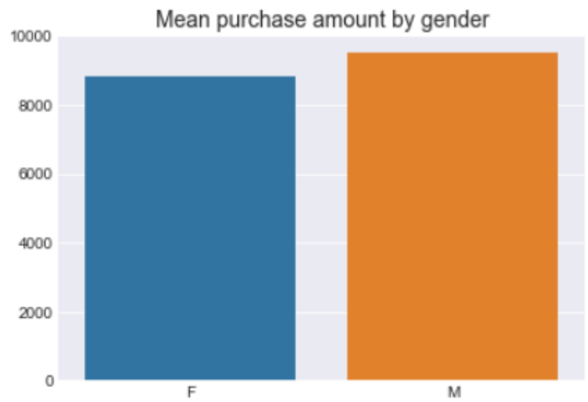


```python
plt.figure(figsize=(9, 6))

Gender_M = dataset_orig[dataset_orig.Gender == 'M'].Purchase.sum()
Gender_F = dataset_orig[dataset_orig.Gender == 'F'].Purchase.sum()
colors = ['blueviolet', 'lightcoral']
labels = ['Male', 'Female']

patches, l_text, p_text = plt.pie([Gender_M, Gender_F],
                                  labels=labels,
                                  colors=colors,
                                  explode=(0, 0.08),
                                  autopct='%4.2f%%',
                                  startangle=90,
                                  shadow=True)

for t in l_text + p_text:
    t.set_size(20)
for t in p_text:
    t.set_color('white')
plt.legend(fontsize=15, loc='best', title='Total-Purchase', frameon=False)
plt.axis('equal')
plt.title('Sum of purchases', size=20)
plt.show()
```



Men have had transactions about 3 times higher than women in black friday.

```python
gender_gb = data[['Gender', 'Purchase']].groupby('Gender', as_index=False).agg('mean')
sns.barplot(x='Gender', y='Purchase', data=gender_gb)
plt.ylabel('')
plt.xlabel('')
for spine in plt.gca().spines.values():
    spine.set_visible(False)
plt.title('Mean purchase amount by gender', size=14)
plt.show()
```

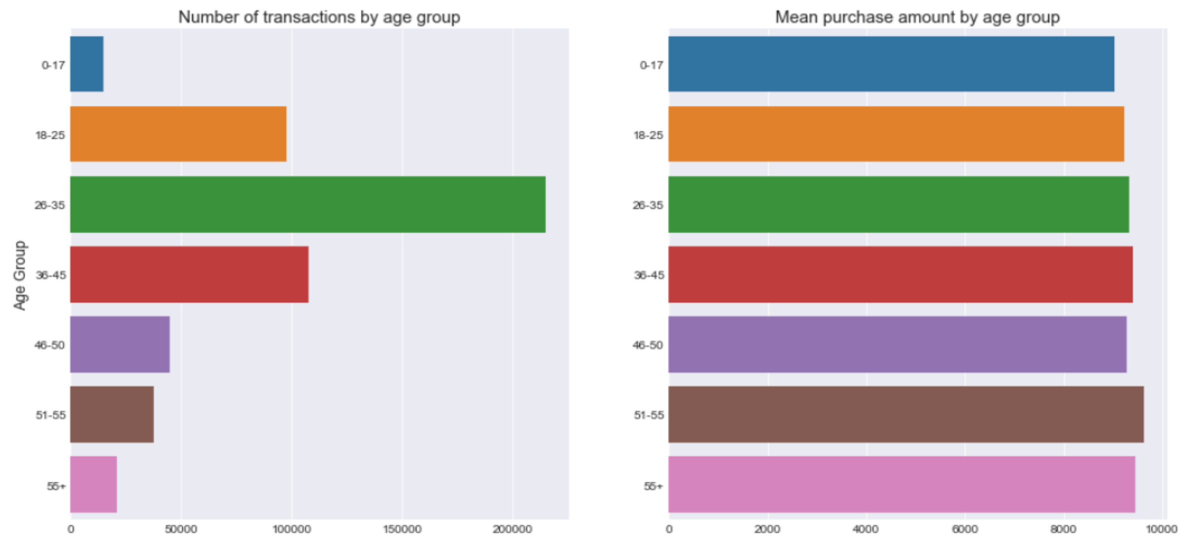


They've also had proportionally higher purchase amount.

Age:

Let's check from Age groups provided in dataset.

```
plt.figure(figsize=(16, 8))
plt.subplot(121)
sns.countplot(y='Age', data=data, order=sorted(data.Age.unique()))
plt.title('Number of transactions by age group', size=14)
plt.xlabel('')
plt.ylabel('Age Group', size=13)
plt.subplot(122)
age_gb = data[['Age', 'Purchase']].groupby('Age', as_index=False).agg('mean')
sns.barplot(y='Age', x='Purchase', data=age_gb, order=sorted(data.Age.unique()))
plt.title('Mean purchase amount by age group', size=14)
plt.xlabel('')
plt.ylabel('')
plt.show()
```



People within the ages of 26 to 35 have purchased the most (in number and amount), and as we saw about gender, people in different ages have nearly same mean purchase amount, too.

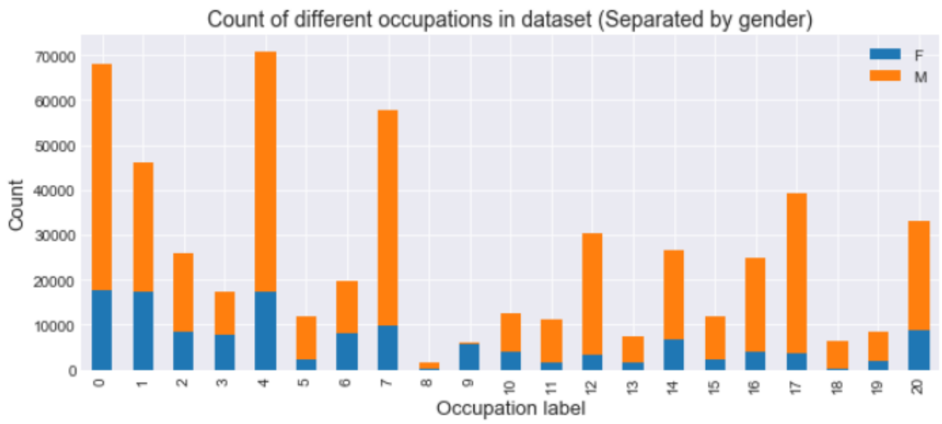
Let's check what products were most popular in each age group.

```
age_product_gb = data[['Age', 'Product_ID', 'Purchase']].groupby(['Age', 'Product_ID']).agg('count').rename(columns={'Pu
age_product_gb.sort_values('count', inplace=True, ascending=False)
ages = sorted(data.Age.unique())
result = pd.DataFrame({
    x: list(age_product_gb.loc[x].index)[:5] for x in ages
}, index=['#{}'.format(x) for x in range(1,6)])
display(result)
```

	0-17	18-25	26-35	36-45	46-50	51-55	55+
#1	P00255842	P00265242	P00265242	P00025442	P00265242	P00265242	P00265242
#2	P00145042	P00112142	P00110742	P00110742	P00046742	P00025442	P00080342
#3	P00112142	P00110742	P00112142	P00265242	P00025442	P00110742	P00051442
#4	P00242742	P00237542	P00025442	P00112142	P00051442	P00059442	P00184942
#5	P00000142	P00046742	P00058042	P00057642	P00184942	P00010742	P00025442

Occupation:

```python
men = data[data.Gender == 'M']['Occupation'].value_counts(sort=False)
women = data[data.Gender == 'F']['Occupation'].value_counts(sort=False)
pd.DataFrame({'M': men, 'F': women}, index=range(0,21)).plot.bar(stacked=True)
plt.gcf().set_size_inches(10, 4)
plt.title("Count of different occupations in dataset (Separated by gender)", size=14)
plt.legend(loc="upper right")
plt.xlabel('Occupation label', size=13)
plt.ylabel('Count', size=13)
plt.show()
```



Observation is that people occupied in job labels 0, 4 and 7 have purchased the most in black Friday.

Now checking what products people from different occupations were most interested in:

```python
import random
color_mapping = {}
def random_color(val):
    if val in color_mapping.keys():
        color = color_mapping[val]
    else:
        r = lambda: random.randint(0,255)
        color = 'rgba({}, {}, {}, 0.4)'.format(r(), r(), r())
        color_mapping[val] = color
    return 'background-color: %s' % color

occ_product_gb = data[['Occupation', 'Product_ID', 'Purchase']].groupby(['Occupation', 'Product_ID']).agg('count').renam
occ_product_gb.sort_values('count', inplace=True, ascending=False)
result = pd.DataFrame({
    x: list(occ_product_gb.loc[x].index)[:5] for x in range(21)
}, index=['#{}'.format(x) for x in range(1,6)])
display(result.style.applymap(random_color))
```

	0	1	2	3	4	5	6	7	8	9	10	11	
#1	P00265242	P00265242	P00265242	P00265242	P00265242	P00265242	P00265242	P00265242	P00112142	P00034742	P00145042	P00265242	P00057
#2	P00110742	P00220442	P00025442	P00117942	P00110742	P00114942	P00058042	P00110742	P00242742	P00117442	P00242742	P00059442	P00265
#3	P00025442	P00110742	P00058042	P00025442	P00112142	P00251242	P00110742	P00025442	P00117942	P00265242	P00112142	P00025442	P00112
#4	P00057642	P00025442	P00110842	P00110842	P00237542	P00110742	P00031042	P00112142	P00114942	P00000142	P00025442	P00110942	P00025
#5	P00112142	P00059442	P00059442	P00110742	P00025442	P00057642	P00255842	P00184942	P00127842	P00102642	P00255842	P00112142	P00237

Table above represents top 5 seller products categorized by the user occupation (same products have the same background color).

1. First thing you notice is that P00265242 is the most-purchased product for 15 out of 21 occupations and an interesting fact is that this product is not even present in top-5 products of occupations 8, 10 and 17. I wonder what this product is and what these occupations are.
2. Second interesting thing about this illustration is how similar the first 4 occupations' top-5 are.
3. Third and last interesting fact from these charts: from top 5 products of occupation 9, one of them is P00265242 and present in most of other top 5s, one of them is only present in occupation 16's list and the rest are not repeated in any other lists. Adding to account the fact that we saw from previous chart, this was the only occupation with more women than men (even though the totall number of men in dataset was higher), makes occupation 9 a unique occupation among the list.

City Category and City Stability:

Cities are categorized in 3 different categories A, B and C. We have living city of each user in the time of transaction and also we know for how long the user had been in that city, by that time.

Let's explore number of transactions in each city.

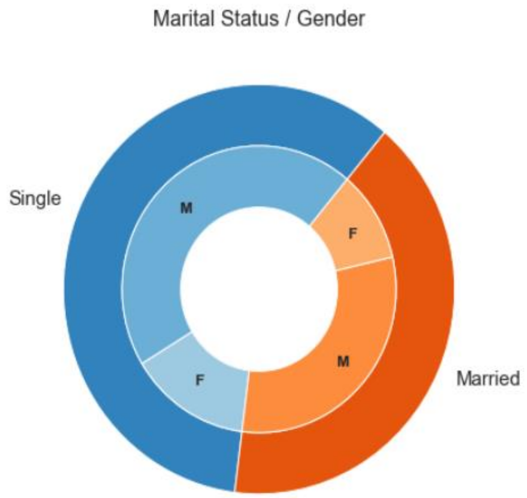
```
stay_years = [data[data.Stay_In_Current_City_Years == x]['City_Category'].value_counts(sort=False).iloc[::-1] for x in s

f, (ax1, ax2) = plt.subplots(1,2, gridspec_kw = {'width_ratios':[1, 2]}, sharey=True)

years = sorted(data.Stay_In_Current_City_Years.unique())
pd.DataFrame(stay_years, index=years).T.plot.bar(stacked=True, width=0.3, ax=ax1, rot=0, fontsize=11)
ax1.set_xlabel('City Category', size=13)
ax1.set_ylabel('# Transactions', size=14)
ax1.set_title('# Transactions by city (separated by stability)', size=14)

sns.countplot(x='Stay_In_Current_City_Years', data=data, ax=ax2, order=years)
ax2.set_title('# Transactions by stability', size=14)
ax2.set_ylabel('')
ax2.set_xlabel('Years in current city', size=13)

plt.gcf().set_size_inches(15, 6)
plt.show()
```

People living in city category of B have had most transactions to this store, following by categories C and B with relatively close values. Those who have been in their living city for 1 year had double the number of transactions than any other stay durations, and then comes the people living in their city for 2 years, 3 years, 4+ years and 0 years (<1 year). The pattern is the same within each city category as well.

Seems like people living their second year in a city tend to shop more than others.

Marital Status:

```
out_vals = data.Marital_Status.value_counts()
in_vals = np.array([data[data.Marital_Status==x]['Gender'].value_counts() for x in [0,1]]).flatten()

fig, ax = plt.subplots(figsize=(7, 7))

size = 0.3
cmap = plt.get_cmap("tab20c")
outer_colors = cmap(np.arange(2)*4)
inner_colors = cmap(np.array([1, 2, 5, 6]))

ax.pie(out_vals, radius=1, colors=outer_colors,
       wedgeprops=dict(width=size, edgecolor='w'), labels=['Single', 'Married'],
       textprops={'fontsize': 15}, startangle=50)

ax.pie(in_vals, radius=1-size, colors=inner_colors,
       wedgeprops=dict(width=size, edgecolor='w'), labels=['M', 'F', 'M', 'F'],
       labeldistance=0.75, textprops={'fontsize': 12, 'weight': 'bold'}, startangle=50)

ax.set(aspect="equal")
plt.title('Marital Status / Gender', fontsize=16)
plt.show()
```



Single people have purchased more than married people and in both categories men, following the general pattern of dataset, have purchased more than women.

Best sellers:

Which products sold the most and which categories contain most-sold products? We will only use Product_Category_1 since the other two have a lot of null values. Also, let's see which users have purchased the most.

```python
col_names = ['Product_ID', 'Product_Category_1', 'User_ID']
renames = ['Product', 'Category', 'User']
results = []
for col_name, new_name in zip(col_names, renames):
    group = data[[col_name, 'Purchase']].groupby(col_name, as_index=False).agg('count')
    result = group.sort_values('Purchase', ascending=False)[:10]
    result.index = ['#{}'.format(x) for x in range(1,11)]
    results.append(result.rename(columns={col_name: new_name}))

from IPython.display import display_html
def display_side_by_side(*args):
    html_str=''
    for df in args:
        html_str+=df.to_html()
    display_html(html_str.replace('table','table style="display:inline; padding-right: 3em !important;"'),raw=True)
display_side_by_side(*results)
```

	Product	Purchase
#1	P00265242	1858
#2	P00110742	1591
#3	P00025442	1586
#4	P00112142	1539
#5	P00057642	1430
#6	P00184942	1424
#7	P00046742	1417
#8	P00058042	1396
#9	P00145042	1384
#10	P00059442	1384

	Category	Purchase
#1	5	148592
#2	1	138353
#3	8	112132
#4	11	23960
#5	2	23499
#6	6	20164
#7	3	19849
#8	4	11567
#9	16	9697
#10	15	6203

	User	Purchase
#1	1001680	1025
#2	1004277	978
#3	1001941	898
#4	1001181	861
#5	1000889	822
#6	1003618	766
#7	1001150	752
#8	1001015	739
#9	1002909	717
#10	1001449	714

Filling the missing values:

```python
bf.Product_Category_2.fillna(value=9.84, inplace=True)

bf.Product_Category_2.head(10)
```

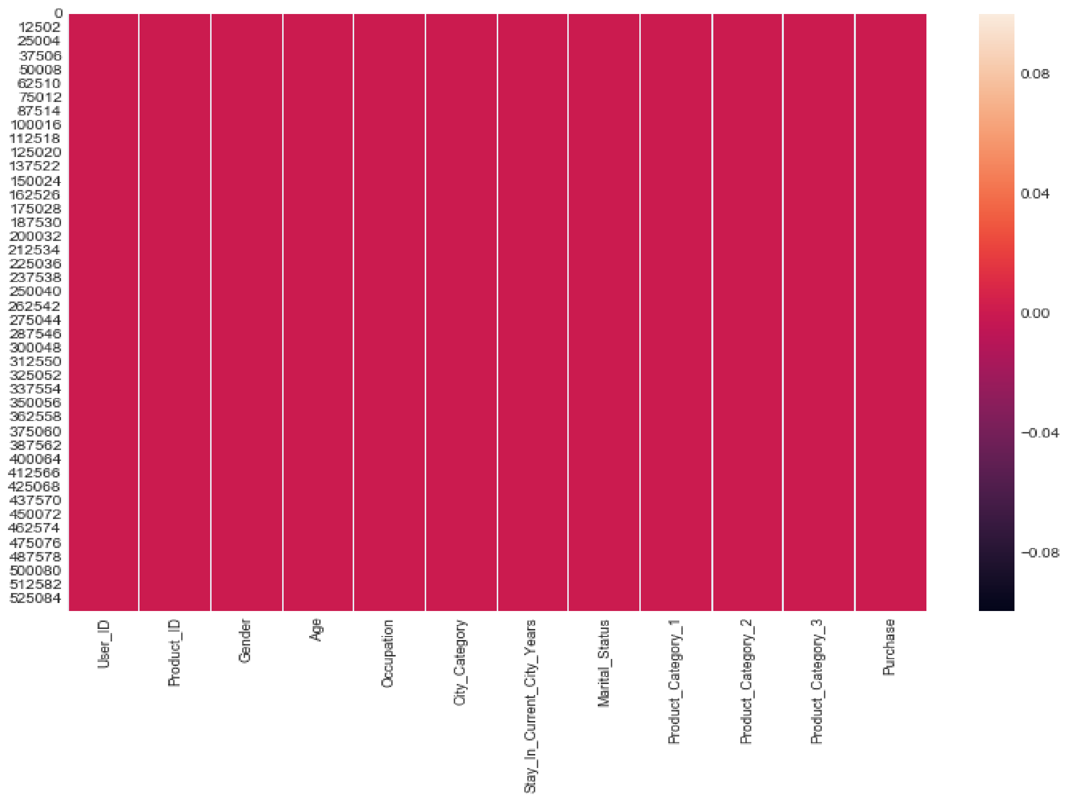
```
0     8.0
1     6.0
2     8.0
3    14.0
4    11.0
5     2.0
6     8.0
7    15.0
8    16.0
9     8.0
Name: Product_Category_2, dtype: float64
```

```python
bf.Product_Category_3.fillna(value=12.67, inplace=True)

bf.Product_Category_3.head(10)
```

```
0    15.5
1    14.0
2    15.5
3    15.5
4    15.5
5    15.5
6    17.0
7    15.5
8    15.5
9    15.5
Name: Product_Category_3, dtype: float64
```

```
plt.figure(figsize=(12,8))

sns.heatmap(bf.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d3944831d0>
```



Data is ready now, we are going to build the mode.

Splitting the data set:

```
bf = dataset_orig
```

```
X = dataset_orig.iloc[:,2:-1].values
y =dataset_orig.iloc[:, -1].values
```

```
X
```

```
array([['F', '0-17', 10, ..., 3, nan, nan],
       ['F', '0-17', 10, ..., 1, 6.0, 14.0],
       ['F', '0-17', 10, ..., 12, nan, nan],
       ...,
       ['M', '36-45', 16, ..., 8, 15.0, nan],
       ['M', '36-45', 16, ..., 5, nan, nan],
       ['M', '36-45', 16, ..., 5, 8.0, nan]], dtype=object)
```

```
y
```

```
array([ 8370, 15200,  1422, ...,  8043,  7172,  6875], dtype=int64)
```

Encoding categorical data

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 0] = labelencoder_X_1.fit_transform(X[:, 0])
labelencoder_X_2 = LabelEncoder()
X[:, 1] = labelencoder_X_2.fit_transform(X[:, 1])
labelencoder_X_3 = LabelEncoder()
X[:, 3] = labelencoder_X_3.fit_transform(X[:, 3])
labelencoder_X_4 = LabelEncoder()
X[:, 4] = labelencoder_X_4.fit_transform(X[:, 4])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:,1:]
```

```
C:\Users\azade\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:363: FutureWarning: The handling of integer d
ata will change in version 0.22. Currently, the categories are determined based on the range [0, max(values)], while in the
future they will be determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the O
neHotEncoder directly.
  warnings.warn(msg, FutureWarning)
C:\Users\azade\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:385: DeprecationWarning: The 'categorical_fea
tures' keyword is deprecated in version 0.20 and will be removed in 0.22. You can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

```
X
```

```
array([[ 0. ,  0. ,  0. , ...,  3. ,  8. , 15.5],
       [ 0. ,  0. ,  0. , ...,  1. ,  6. , 14. ],
       [ 0. ,  0. ,  0. , ..., 12. ,  8. , 15.5],
       ...,
       [ 0. ,  0. ,  1. , ...,  8. , 15. , 15.5],
       [ 0. ,  0. ,  1. , ...,  5. ,  9. , 15.5],
       [ 0. ,  0. ,  1. , ...,  5. ,  8. , 15.5]])
```

Splitting the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

Feature Scaling

```
from  sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Using some Regression models and figure out what will be the best, that is, the lowest "Mean Squared Error".

Fitting Multiple Linear Regression to the Training set

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Predicting the Test set results

```
y_pred = regressor.predict(X_test)
```

Mean Squared Error

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

```
22002098.86844547
```

Fitting Polynomial Regression to the dataset

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=3)
X_poly = poly_reg.fit_transform(X_train)
regressor = LinearRegression()
regressor.fit(X_poly,y_train)
```

Predicting the Test set results

```
y_pred = regressor.predict(poly_reg.fit_transform(X_test))
```

Mean Squared Error

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

```
17556876.898933
```

Fitting Random Forest Regression to the dataset

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 600, random_state = 0)
regressor.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=600, n_jobs=None,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
y_pred = regressor.predict(X_test)
```

Mean Squared Error

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

Conclusion

By comparing the "Mean Squared Error" between the three models, we can realize that the "Random Forest Regression" is the better model for this problem.