# Applied Data Analytics II - Course Project


### Diallo Youssouf


### Date completed: 05/04/2019

## Research questions

### I will be studying customers behaviors in a Black Friday sales

***1- Predict the amount purchased by a client using Regression Analysis,***

***2- Define which categories of Age tend to buy a lot using Classfication***

***3- 3- Determine which city is more productive***

***4- Run Cluster analysis for segmenting customers based on their spending***

***We argue that by answering these questions, it will help the retail store to understand their businesses and make data driven decision.***

# 1. Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from pandas.plotting import scatter_matrix
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from matplotlib import rcParams
rcParams['figure.dpi']= 90
sn.set_style("whitegrid")
sn.set_context("poster")
```

# 2. Loading dataset

***The dataset that I am using is from Kaggle and it can be found at this***

***link [https://www.kaggle.com/mehdidag/black-friday/version/1](https://www.kaggle.com/mehdidag/black-friday/version/1) (https://www.kaggle.com/mehdidag/black-friday/version/1)***

***once you clik on the link, you will need to go under download button to download***

In [2]:
```
## load the csv file into a pandas Data Frame
blackFriday =pd.read_csv('Project_Diallo_Youssouf_Dataset.csv')
```

In [3]:
```
## check the number of rows and colummns of the data
blackFriday.shape
```

Out[3]:
```
(537577, 12)
```

In [4]:
```
## check the data types of each features in the data set
blackFriday.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                       537577 non-null int64
Product_ID                    537577 non-null object
Gender                        537577 non-null object
Age                           537577 non-null object
Occupation                    537577 non-null int64
City_Category                 537577 non-null object
Stay_In_Current_City_Years    537577 non-null object
Marital_Status                537577 non-null int64
Product_Category_1            537577 non-null int64
Product_Category_2            370591 non-null float64
Product_Category_3            164278 non-null float64
Purchase                      537577 non-null int64
dtypes: float64(2), int64(5), object(5)
memory usage: 49.2+ MB
```

# 3. Data set Description

**The dataset contains 537,577 observations about black Friday sales a for a retail company.**

**It contains 12 columns with numerical and categorical data types**

# 4. Columns or Attributes

***USER_ID : Define the User id, which will help identify a user who made a purchase from the store***

***Product_ID : Help identify a prodcut in the store***

***Gender: Define the user gender which will be either Male (M) or Female(F)***

***Age : Define the age of user, which is given in a range of values.***

***Ocupation : Define the ID occupation of each customer***

***City_Category : Define the category of each city (A, B, and C)***

***Stay_In_Current_City_Years : Define how many years, a customer resides in that city***

***Marital_Status : Define the customer's status, which is either married coded by 1 or not married coded by 0***

***The retail company is selling three categories of products***

***Product_Category_1 : Define the number of items purchased within the Product_Category_1***

***Product_Category_2 : Define the number of items purchased within the Product_Category_2***

***Product_Category_3 : Define the number of items purchased within the Product_Category_3***

***Purchase : Define the amount spent by each customer to purchase the three prodcuts. it is in US dollars***

# 5. Data Exploration

In [6]:
```
## This print the first 5 elements
blackFriday.head()
```

Out[6]:

| | User_ID | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Mari |
|---|---|---|---|---|---|---|---|---|
| **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+ | |

In [7]:
```
## This print the last 5 elements
blackFriday.tail()
```

Out[7]:

| | User_ID | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years |
|---|---|---|---|---|---|---|---|
| **537572** | 1004737 | P00193542 | M | 36-45 | 16 | C | 1 |
| **537573** | 1004737 | P00111142 | M | 36-45 | 16 | C | 1 |
| **537574** | 1004737 | P00345942 | M | 36-45 | 16 | C | 1 |
| **537575** | 1004737 | P00285842 | M | 36-45 | 16 | C | 1 |
| **537576** | 1004737 | P00118242 | M | 36-45 | 16 | C | 1 |

In [8]:
```
## let check if we have null
blackFriday.isnull().values.any()
```

Out[8]: True

**yes we have null values in the dataset, but I will handle it later**

# To better understand our data let's find some statistics

## what was the most profitable product on Black Friday ?

In [8]:
```
## let's check products that are sold the most and just pick the top 5
blackFriday.groupby(["Product_ID"]).sum().sort_values("Purchase", ascending=False)
```

Out[8]:

| | User_ID | Occupation | Marital_Status | Product_Category_1 | Product_Category_2 | Proc |
|---|---|---|---|---|---|---|
| **Product_ID** | | | | | | |
| **P00025442** | 1590774903 | 13112 | 635 | 1586 | 3172.0 | |
| **P00110742** | 1595784075 | 12933 | 637 | 1591 | 3182.0 | |
| **P00255842** | 1358323490 | 11196 | 526 | 21664 | 0.0 | |
| **P00184942** | 1428158481 | 11680 | 589 | 1424 | 11392.0 | |
| **P00059442** | 1388200080 | 11456 | 567 | 8304 | 11072.0 | |

**We can see from the result above the product with ID (P00025442) was the most sold with a total purchase of $ 27,532426**

# what was the most profitable City on Black Friday ?

In [9]:
```
group = blackFriday.groupby(["City_Category"]).sum()
total_price = group["Purchase"].groupby(level=0, group_keys=False)
total_price.nlargest(5)
```

Out[9]:
```
City_Category
A    1295668797
B    2083431612
C    1638567969
Name: Purchase, dtype: int64
```

**from the result above the city category B was the most profitable on Black Friday**

**the purchase amount was $ 2,083 431 612 something above 2 Billion dollars**

# Determine which group of people buy a lot

In [10]:
```
## this count the number in each category
blackFriday.Gender.value_counts()
```

Out[10]:
```
M    405380
F    132197
Name: Gender, dtype: int64
```

**The result above shows that the retail company recorded 405380 transactions from Males**

**which represent 75.40 % and 132197 transactions which also represent 24.6 % from Females**

In [11]:
```
## this line of code group by Marital status
blackFriday.groupby(["Marital_Status"]).sum().sort_values("Purchase", ascending=Fa
```

Out[11]:

| | User_ID | Occupation | Product_Category_1 | Product_Category_2 | Product_Categ |
|---|---|---|---|---|---|
| **Marital_Status** | | | | | |
| **0** | 318759372904 | 2526251 | 1662649 | 2154478.0 | 1237 |
| **1** | 220425975246 | 1818828 | 1184115 | 1492932.0 | 843 |

In [12]:
```
## we check the number of people who are married or unmarried
blackFriday.Marital_Status.value_counts()
```

Out[12]:
```
0    317817
1    219760
Name: Marital_Status, dtype: int64
```

## This result shows that 59 % of the customers are married and 41 % are unmarried

In [80]:
```
## let group the data set by Purchase amount
#blackFriday.groupby('Purchase').sum()
```

#### This result above shows us that the minimum amount purchase was 185 dollars and the maximum was 23,961 dollars

## Let's run some descriptive statistics

In [9]:
```
## this line of code compute the descriptive statistics
blackFriday.describe()
```

Out[9]:

| | 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 | |

## This result gives us a brief description of mean, standard deviation, max, min and etc of the data set

In [10]:
```
## Let's calculate the total purchase and group by Gender and Marital_Status
blackFriday.groupby(["Gender","Marital_Status"]).sum().sort_values("Purchase", asc
```

Out[10]:

| Gender | Marital_Status | User_ID | Occupation | Product_Category_1 | Product_Category_2 | Produ |
|---|---|---|---|---|---|---|
| M | 0 | 241555332289 | 2042528 | 1235727 | 1630874.0 | |
| | 1 | 165024843194 | 1411190 | 871336 | 1117320.0 | |
| F | 0 | 77204040615 | 483723 | 426922 | 523604.0 | |
| | 1 | 55401132052 | 407638 | 312779 | 375612.0 | |

In [11]:
```
## compute the number of product category 1 that was sold
blackFriday.Product_Category_1.sum()
```

Out[11]: 2846764

In [12]:
```
## compute the number of product category 2 that was sold
blackFriday.Product_Category_2.sum()
```

Out[12]: 3647410.0

In [13]:
```
## compute the number of product category 3 that was sold
blackFriday.Product_Category_3.sum()
```

Out[13]: 2081376.0

**The conclusion that we can draw from this is that product category 3 was the most sold**

**3647410 items sold**

# 6. Preprocessing

In [14]:
```
## let's do some data cleaning
blackFriday.shape
```

Out[14]: (537577, 12)

In [4]:
```
## check data types
blackFriday.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                       537577 non-null int64
Product_ID                    537577 non-null object
Gender                        537577 non-null object
Age                           537577 non-null object
Occupation                    537577 non-null int64
City_Category                 537577 non-null object
Stay_In_Current_City_Years    537577 non-null object
Marital_Status                537577 non-null int64
Product_Category_1            537577 non-null int64
Product_Category_2            370591 non-null float64
Product_Category_3            164278 non-null float64
Purchase                      537577 non-null int64
dtypes: float64(2), int64(5), object(5)
memory usage: 49.2+ MB
```

#### *From the result above, we can see that Prodcut_Category_2 and Product_Category_3 have NaN values*

#### *Instead of deteting the attributes Product_Category_2 and Product_Category_3, we will replace the NaN values by zero*

In [3]:
```
## this line of code replaces the NaN values with zero
blackFriday=blackFriday.fillna(0)
```

In [4]:
```
## let's check again if we still have NaN
blackFriday.isnull().values.any()
```

Out[4]: False

#### we can see that we do not have any more missing

## Let's do some data types transformations on Product_Category_2 and Product_Category_3

In [5]:
```
## we replace the data types by int, we want to keep everyting in int instead of f
blackFriday['Product_Category_2'] = blackFriday.Product_Category_2.astype(np.int64
blackFriday['Product_Category_3'] = blackFriday.Product_Category_3.astype(np.int64
```

In [6]:
```
## we check again to see if there is change on the data types
blackFriday.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                       537577 non-null int64
Product_ID                    537577 non-null object
Gender                        537577 non-null object
Age                           537577 non-null object
Occupation                    537577 non-null int64
City_Category                 537577 non-null object
Stay_In_Current_City_Years    537577 non-null object
Marital_Status                537577 non-null int64
Product_Category_1            537577 non-null int64
Product_Category_2            537577 non-null int64
Product_Category_3            537577 non-null int64
Purchase                      537577 non-null int64
dtypes: int64(7), object(5)
memory usage: 49.2+ MB
```

**we need also to transform the type of Stay_In_Current_City_Years from object to int**

In [7]:
```
## we need to replace the + sign with empty string and change the type to int64
blackFriday['Stay_In_Current_City_Years'] = blackFriday.Stay_In_Current_City_Years
```

# Let's group the amount purchase by category Age

In [8]:
```
blackFriday.groupby(["Age"]).sum().sort_values("Purchase", ascending=False)
```

Out[8]:

| Age | User_ID | Occupation | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | P |
|---|---|---|---|---|---|---|
| 26-35 | 215350137743 | 1696554 | 406963 | 84166 | 1120056 | |
| 36-45 | 107824726995 | 951060 | 203189 | 42507 | 579163 | |
| 18-25 | 97904093780 | 657774 | 177953 | 20641 | 488498 | |
| 46-50 | 44666364150 | 379645 | 78462 | 32194 | 250663 | |
| 51-55 | 37728982395 | 331396 | 66964 | 26979 | 212529 | |
| 55+ | 20964683314 | 199372 | 39865 | 13273 | 123256 | |
| 0-17 | 14746359773 | 129278 | 26206 | 0 | 72599 | |

**This result shows that people with Age 26-35 made a lot purchase on Black Friday followed by 36-45**

**But, people with age range between 0-17 made the least purchase**

In [9]:
```
## let's see the result of the datatype
blackFriday.info()
```

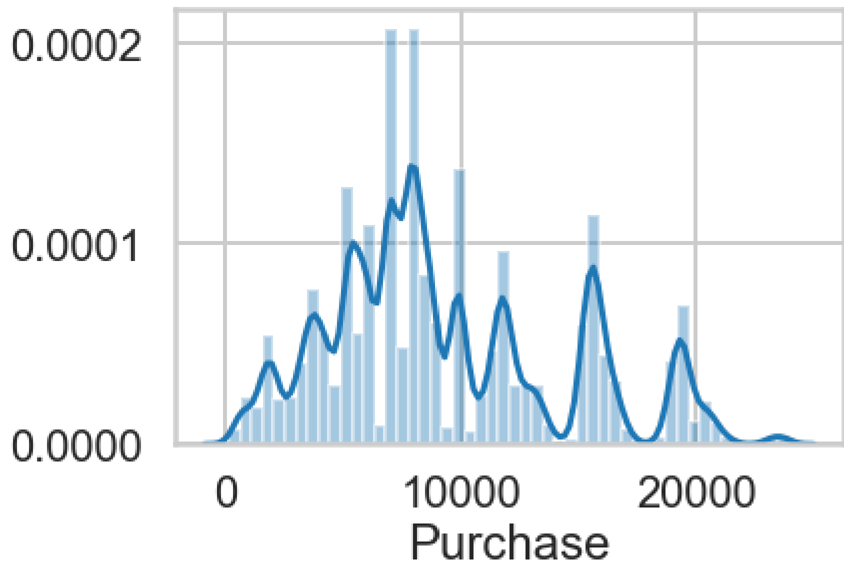
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                       537577 non-null int64
Product_ID                    537577 non-null object
Gender                        537577 non-null object
Age                           537577 non-null object
Occupation                    537577 non-null int64
City_Category                 537577 non-null object
Stay_In_Current_City_Years    537577 non-null int64
Marital_Status                537577 non-null int64
Product_Category_1            537577 non-null int64
Product_Category_2            537577 non-null int64
Product_Category_3            537577 non-null int64
Purchase                      537577 non-null int64
dtypes: int64(8), object(4)
memory usage: 49.2+ MB
```

## Now our data is clean and ready for building model

# 7. Visualization

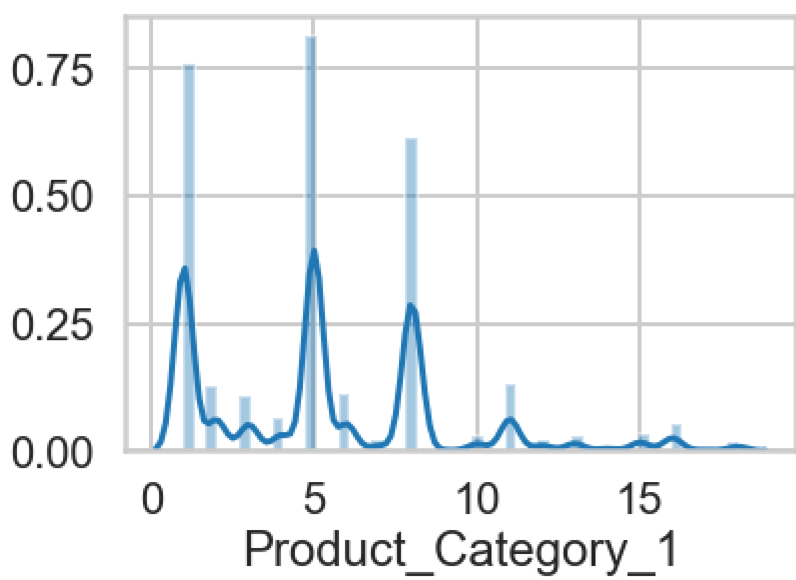
In [28]:
```
## check the distribution of purchase
sn.distplot(blackFriday.Purchase)
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x2a9042443c8>



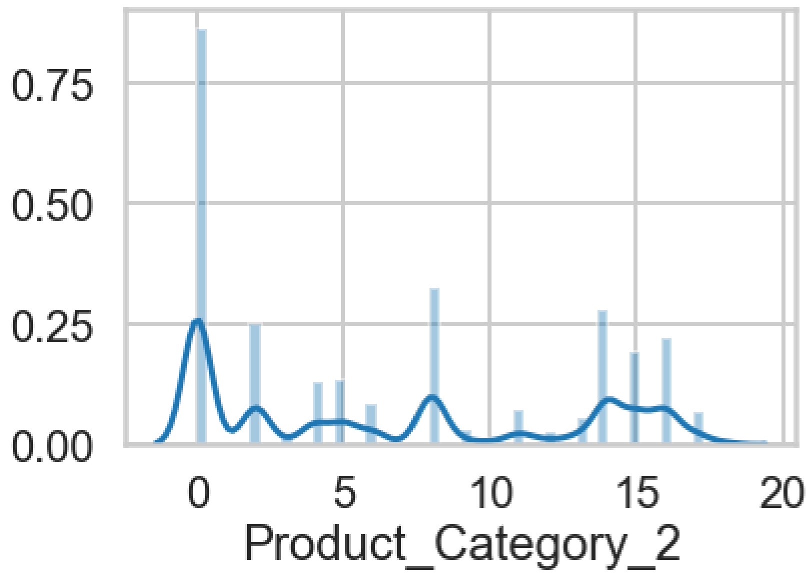

In [29]:
```
## check the distribution of Product Category 1
sn.distplot(blackFriday.Product_Category_1)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x2a904232048>



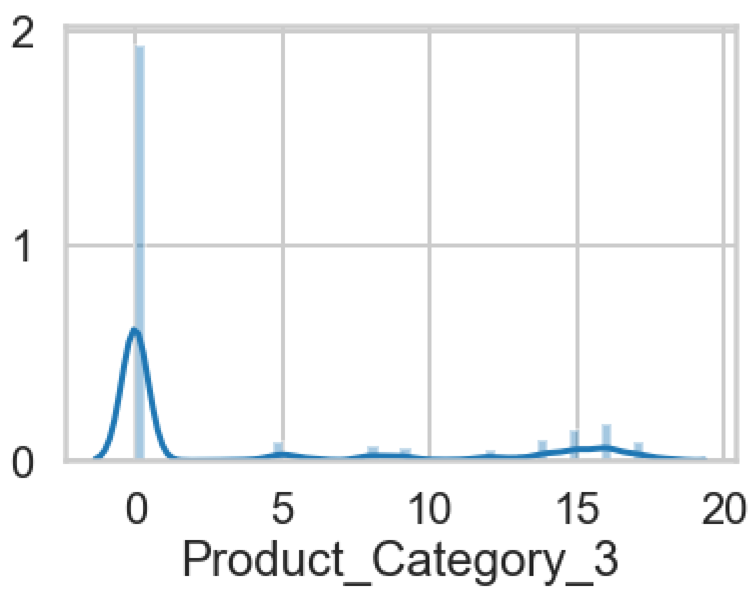
In [30]:
```
## check the distribution of Product category 2
sn.distplot(blackFriday.Product_Category_2)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x2a903b96c18>



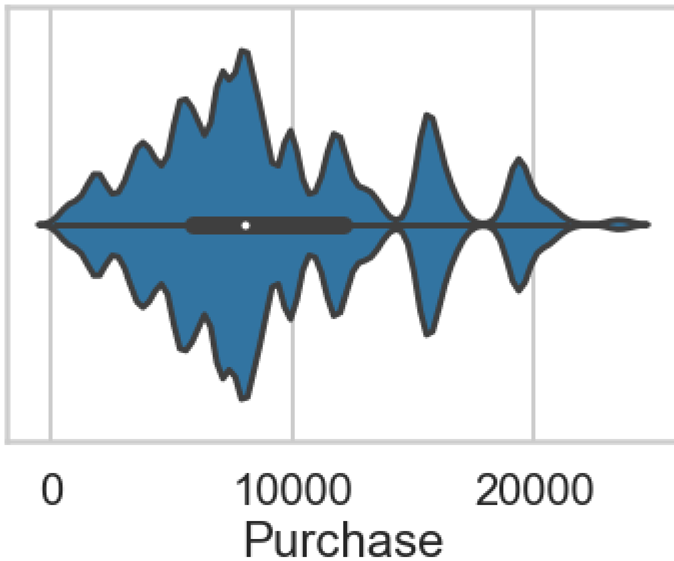

In [31]:
```
## check the distribution of Product category 3
sn.distplot(blackFriday.Product_Category_3)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1f41ebc5668>



In [32]:
```
## let find the distribution and the density
sn.violinplot(x = "Purchase", data=blackFriday)
```

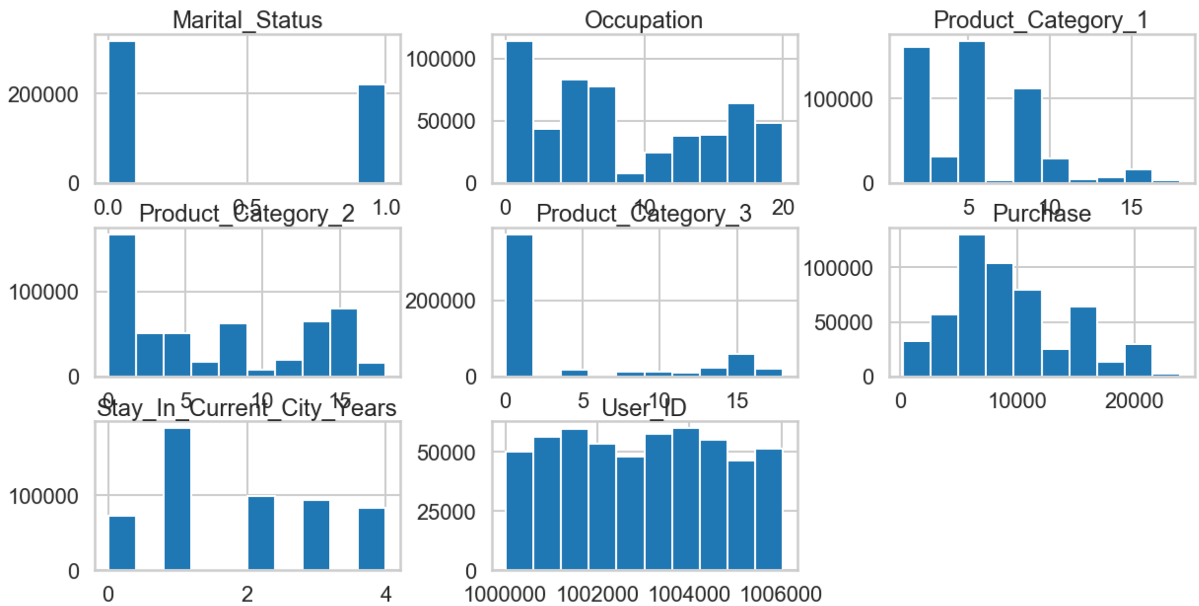
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1f41ec65e80>




### **The violinplot is used to visualize the distribution of the data and its probability density**

In [33]:
```
## check the hostogram of the data set for the numerical values
blackFriday.hist(figsize=(20,10))
```

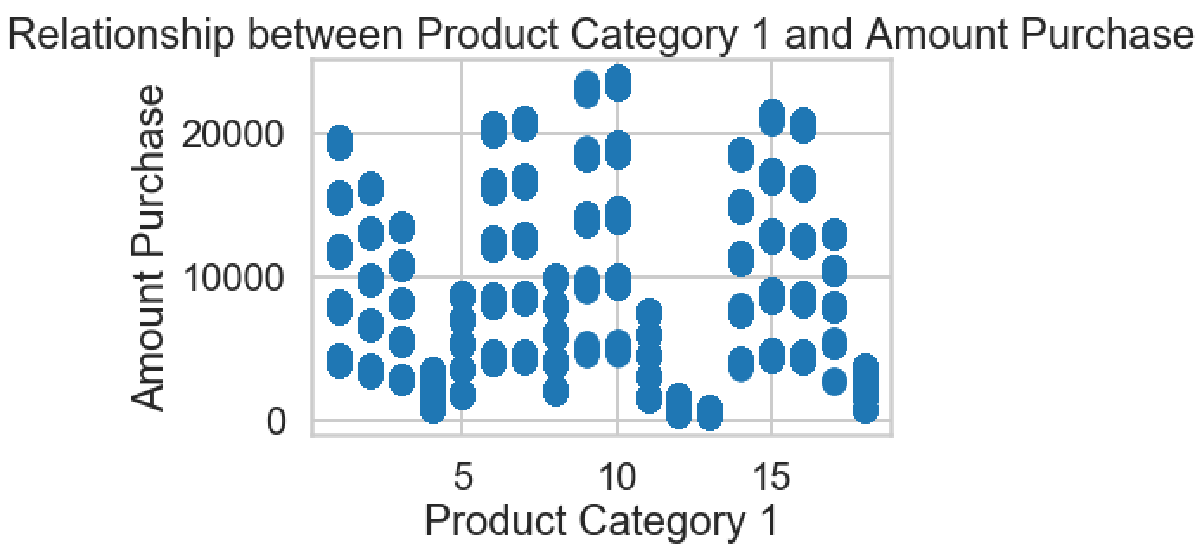
Out[33]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001F41F0DFCC0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001F41ED025C0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001F41ED28E80>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001F41ED58828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001F41ED881D0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001F41EDABB38>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001F41EDDB4E0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001F41EE01E80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001F41EE01EB8>]],
      dtype=object)
```




# the result above gives us more visibility of distributions

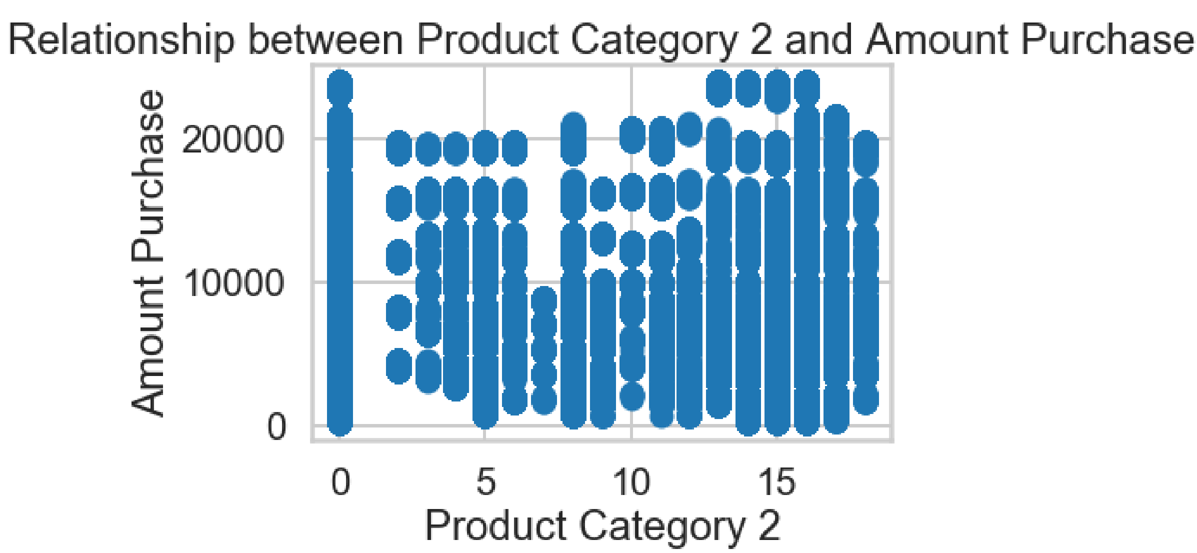
In [35]:
```
plt.scatter(blackFriday.Product_Category_1, blackFriday.Purchase)
plt.xlabel("Product Category 1")
plt.ylabel("Amount Purchase")
plt.title("Relationship between Product Category 1 and Amount Purchase")
```

Out[35]: Text(0.5, 1.0, 'Relationship between Product Category 1 and Amount Purchase')



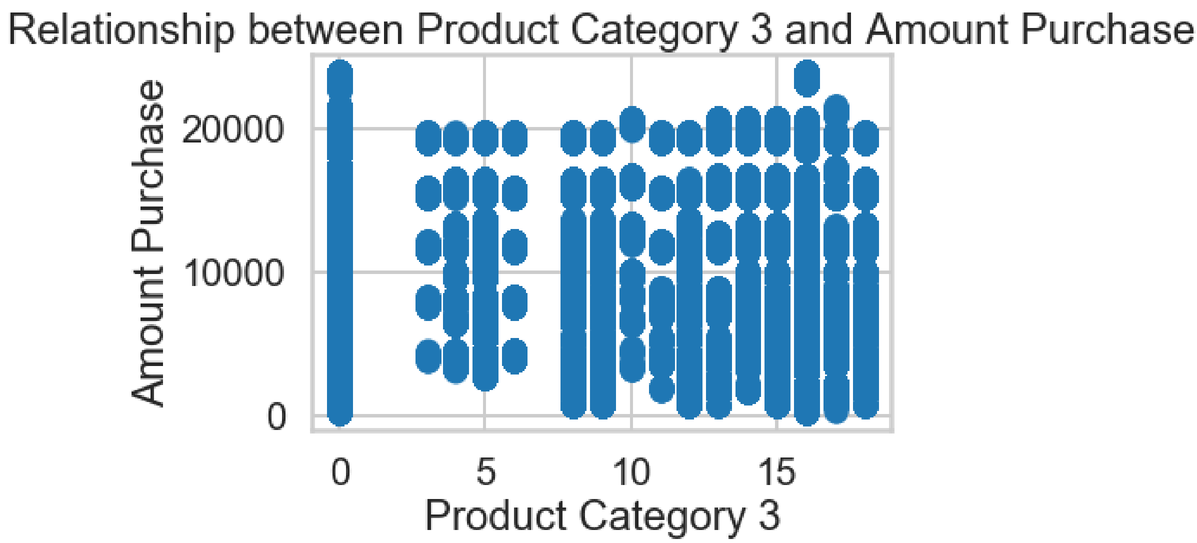

In [36]:
```
plt.scatter(blackFriday.Product_Category_2, blackFriday.Purchase)
plt.xlabel("Product Category 2")
plt.ylabel("Amount Purchase")
plt.title("Relationship between Product Category 2 and Amount Purchase")
```

Out[36]: Text(0.5, 1.0, 'Relationship between Product Category 2 and Amount Purchase')



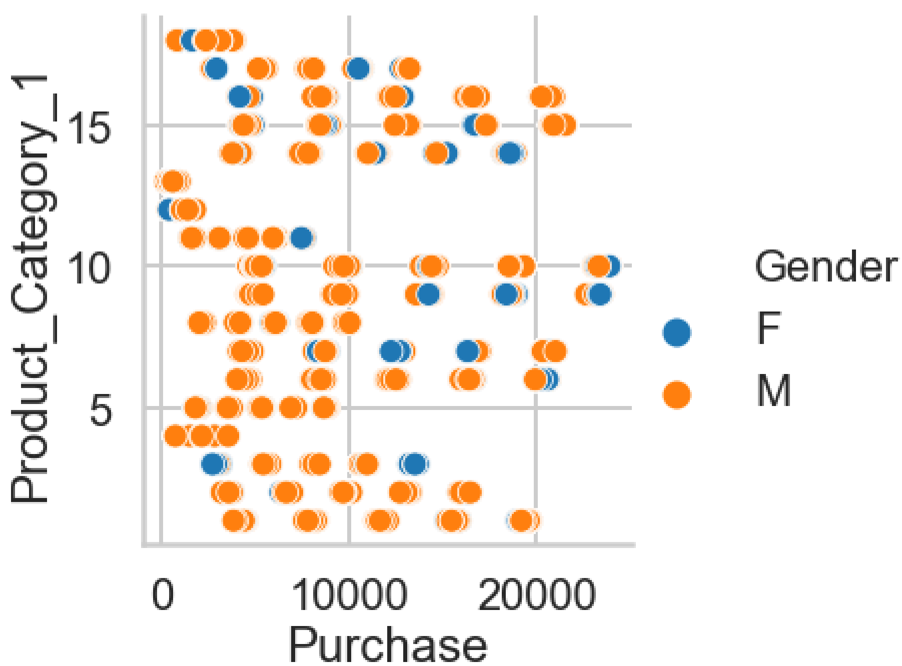
In [37]:
```
plt.scatter(blackFriday.Product_Category_3, blackFriday.Purchase)
plt.xlabel("Product Category 3")
plt.ylabel("Amount Purchase")
plt.title("Relationship between Product Category 3 and Amount Purchase")
```

Out[37]: Text(0.5, 1.0, 'Relationship between Product Category 3 and Amount Purchase')



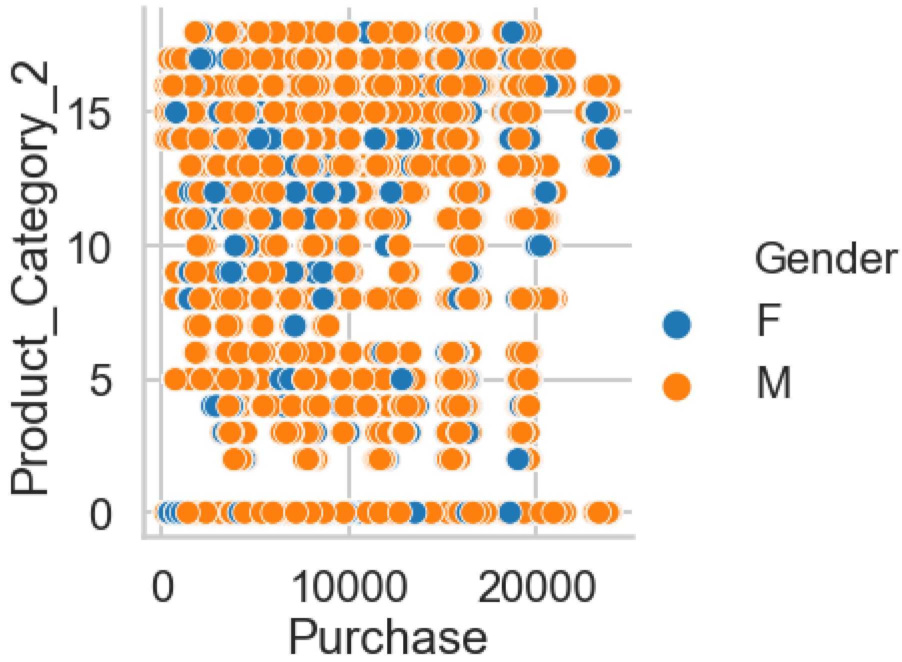

In [34]:
```
## check the relationship between Purchase and Product category based on gender
sn.relplot(x="Purchase", y="Product_Category_1", hue="Gender", data=blackFriday)
```

Out[34]: <seaborn.axisgrid.FacetGrid at 0x1f41efa09b0>



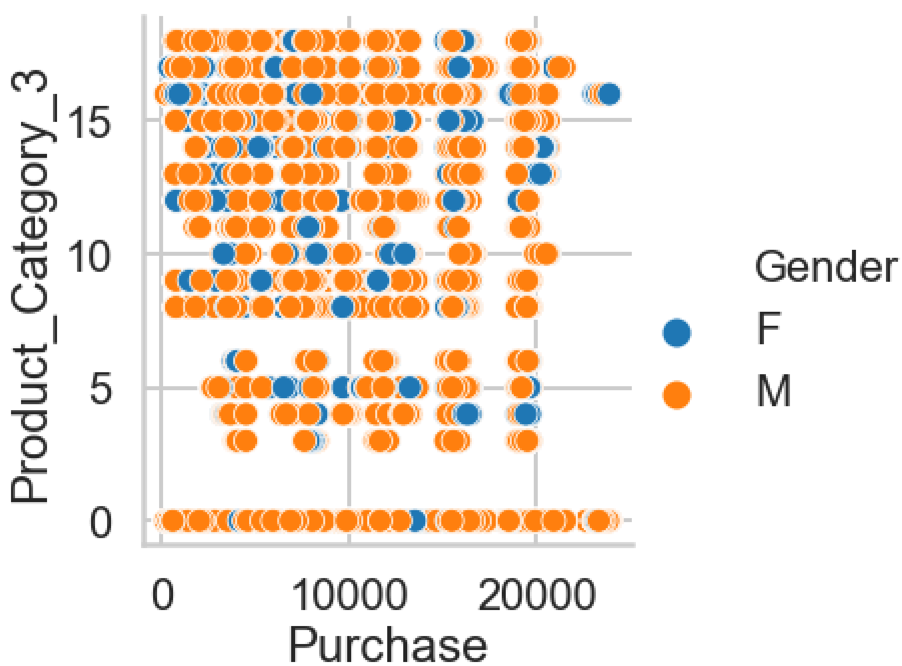
In [38]:
```python
## check the relationship between Purchase and Product category 2 based on gender
sn.relplot(x="Purchase", y="Product_Category_2", hue="Gender", data=blackFriday)
```

Out[38]: <seaborn.axisgrid.FacetGrid at 0x1f41f174d68>



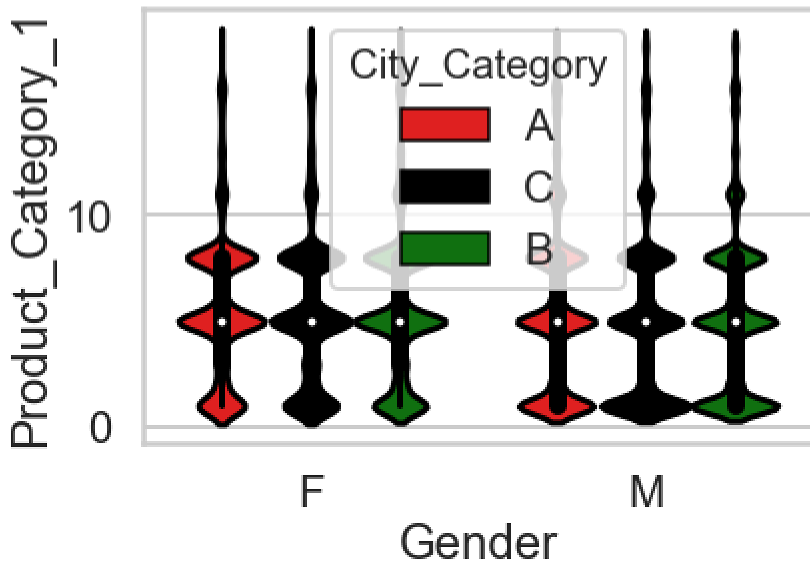

In [39]:
```python
## check the relationship between Purchase and Product category 3 based on gender
sn.relplot(x="Purchase", y="Product_Category_3", hue="Gender", data=blackFriday)
```

Out[39]: <seaborn.axisgrid.FacetGrid at 0x1f41f1f5a90>



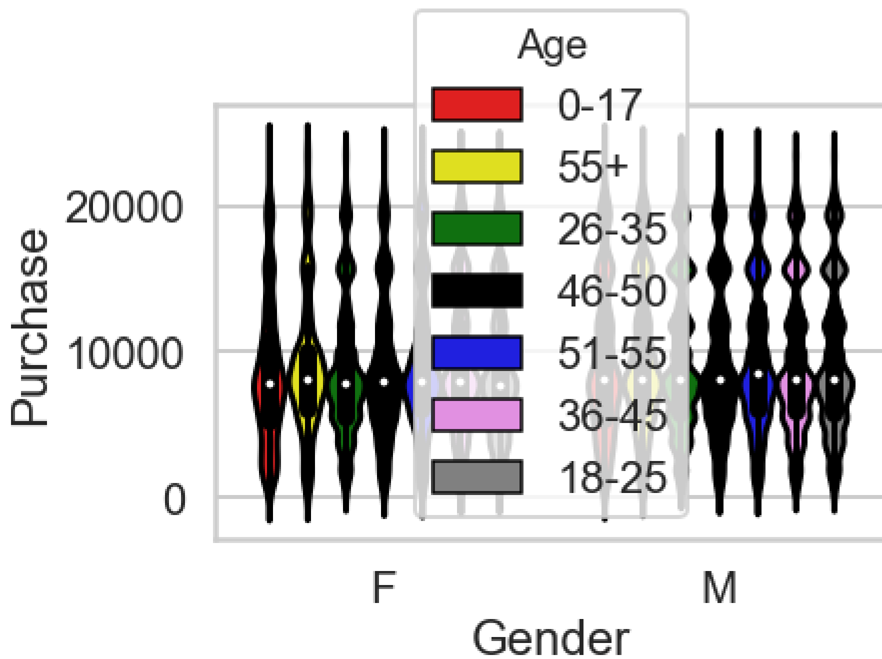
In [40]:
```
sn.violinplot(x="Gender", y="Product_Category_1",hue='City_Category', data=blackFr
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x1f41f2586d8>



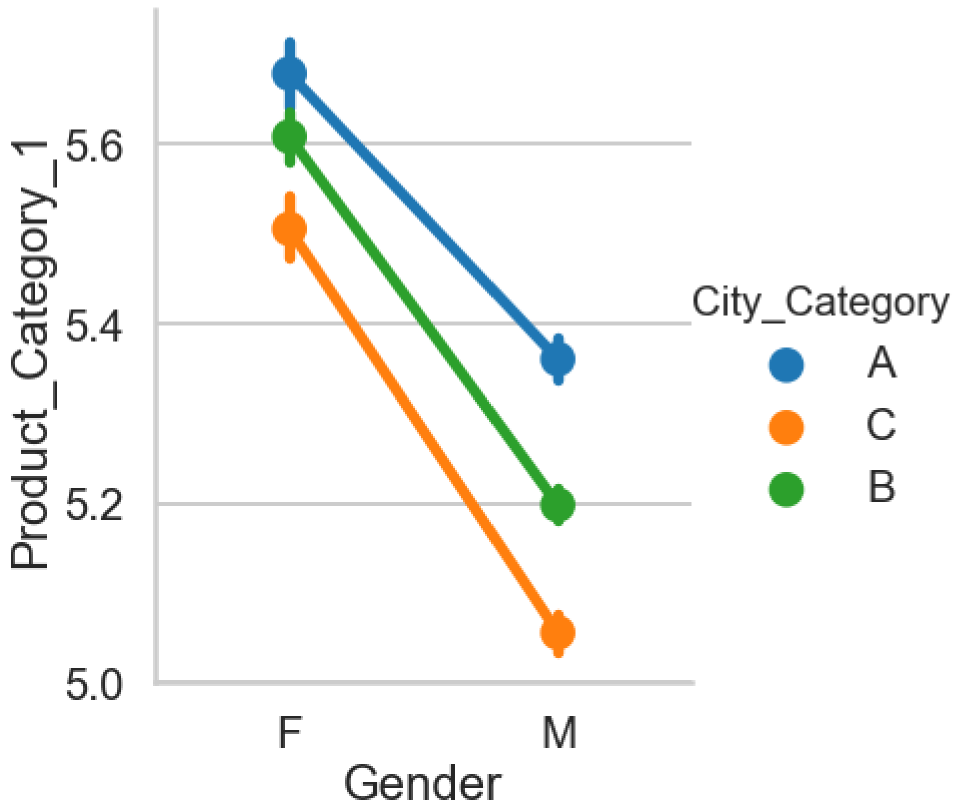

In [41]:
```
sn.violinplot(x="Gender", y="Purchase",hue='Age', data=blackFriday,
              palette=["red", "yellow","green","black","blue","violet","gray"],fiz
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1f41f2e1dd8>



In [42]:
```
## check the relationship between numerical and categorical variables

sn.catplot(x="Gender", y="Product_Category_1",hue="City_Category", data=blackFrida
```




In [12]:
```
#let's verify the data types
blackFriday.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                       537577 non-null int64
Product_ID                    537577 non-null object
Gender                        537577 non-null object
Age                           537577 non-null object
Occupation                    537577 non-null int64
City_Category                 537577 non-null object
Stay_In_Current_City_Years    537577 non-null int64
Marital_Status                537577 non-null int64
Product_Category_1            537577 non-null int64
Product_Category_2            537577 non-null int64
Product_Category_3            537577 non-null int64
Purchase                      537577 non-null int64
dtypes: int64(8), object(4)
memory usage: 49.2+ MB
```

# I will be working with the entire dataset 537577 records

# I will reserve 10 % for validation

In [11]:
```
## This line of code takes 10 % for validation
blackFridayValidation=blackFriday.sample(frac=0.1, random_state=1)
```

In [12]:
```
## this is the validation data set
blackFridayValidation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53758 entries, 94689 to 449041
Data columns (total 12 columns):
User_ID                       53758 non-null int64
Product_ID                    53758 non-null object
Gender                        53758 non-null object
Age                           53758 non-null object
Occupation                    53758 non-null int64
City_Category                 53758 non-null object
Stay_In_Current_City_Years    53758 non-null int64
Marital_Status                53758 non-null int64
Product_Category_1            53758 non-null int64
Product_Category_2            53758 non-null int64
Product_Category_3            53758 non-null int64
Purchase                      53758 non-null int64
dtypes: int64(8), object(4)
memory usage: 5.3+ MB
```

## Our validation data has 53,758 records

# Let's copy the data set for Regression, Classification and Clustering

In [14]:
```
blackFridayRegression=blackFriday          ### use for the regression
blackFridayClassification=blackFriday      ### use for classification
blackFridayClustering=blackFriday          ### use for clustering
```

In [14]:
```
# this check for duplicated
blackFriday.duplicated(keep=False).value_counts()
```

Out[14]:
```
False    537577
dtype: int64
```

In [ ]:
```
sn.pairplot(blackFriday, kind="reg");
```

In [ ]:
```
corr = blackFiday.corr()
sn.heatmap(corr)
```

# this result shows us that we do not have duplicate values in our data set

# Algorithm #1 Linear Regression

## Predict the purchase amount

#### *Y = Purchase (called target data in python, and referred to as the dependent variable or response variable)*

#### *X = independent variables, or explanatory variables (Occupation, Marital_Status,Stay_In_Current_City_Years,Product_category1,2 and 3*

#### *we will use to fit a linear regression model and predict amount Purchase. We will use the r squared method to estimate the accuracy.*

In [31]:
```
## we check again to see the data
blackFridayRegression.head()
```

Out[31]:

| | User_ID | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Mari |
|---|---|---|---|---|---|---|---|---|
| **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 | |

### Fitting Linear Regression using sklearn

In [29]:
```
## we import the libraries for building the linear model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [30]:
```
## creates a list of independent variables
predictors =['Occupation','Stay_In_Current_City_Years','Marital_Status',
             'Product_Category_1','Product_Category_2','Product_Category_3']
```

### we create our target and independent variables

In [32]:
```
X=blackFridayRegression[predictors]  ## this hold the independent variable
Y=blackFridayRegression.Purchase  ## hold the target variable
```

# Splitting the Data

In [33]:
```
# we need to split the data into training set and test set
# we do 70 % training and 30 % test
X_train, X_test, y_train, y_test=train_test_split(X,Y,test_size=0.3,random_state=1
```

# Building Regression model

In [34]:
```
lm = LinearRegression()
lm
```

Out[34]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

# Fit the Regression model

In [35]:
```
lm.fit(X_train,y_train)
```

Out[35]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

# Make prediction using testing set

In [36]:
```
model_yPred=lm.predict(X_test)
```

In [37]:
```
# The coefficients
print('Coefficients: \n', lm.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, model_yPred))
# Explained variance score: 1 is perfect prediction
print('R squared : %.2f' % r2_score(y_test, model_yPred))
```

```
Coefficients: 
 [  13.46204589    6.46363812   36.6513731  -319.25980883    9.23250636
  151.0691684 ]
Mean squared error: 21561218.90
R squared : 0.13
```

**The result above shows the mean squared error and the coefficient**

**the mean squared error is very high which means the model did not perform well**

In [38]:
```
lm.score(X_test, y_test)
```

Out[38]: 0.1291209162422322

**the model is very terrible only 13 % of purchase amount can be explained by the chosen independent variables**

## Make validation using validation set

In [57]:
```
valid=blackFridayValidation.Purchase
Xvalidation=blackFridayValidation[predictors]
```

In [61]:
```
X_train, X_valid, y_train, y_valid=train_test_split(Xvalidation,valid,test_size=0.
```

In [62]:
```
## we predict with the validation set
model_yPred=lm.predict(X_valid)
```

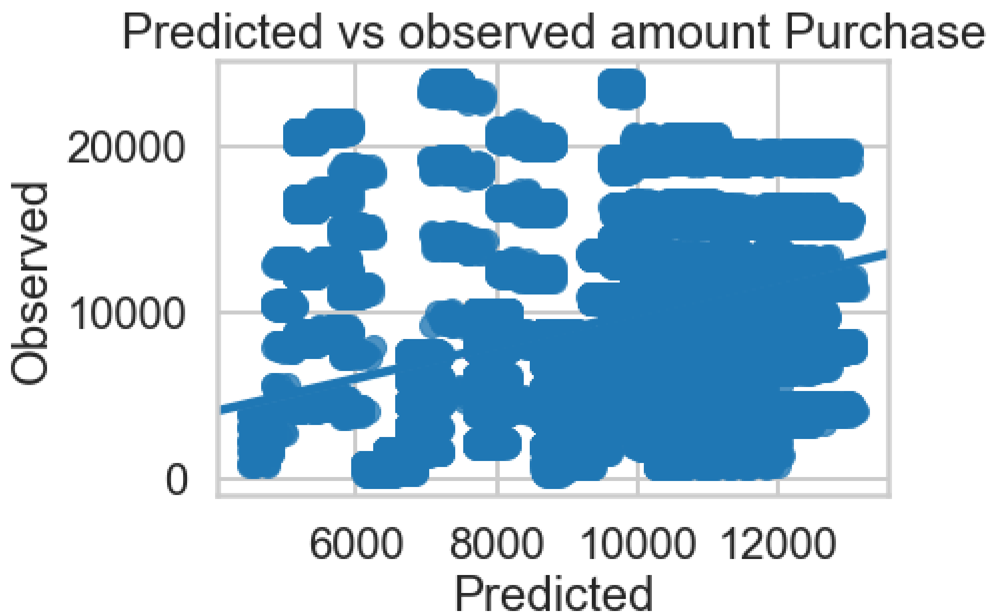
In [65]:
```
## check the score based on the validation set
lm.score(X_valid, y_valid)
```

Out[65]: 0.1333709930717688

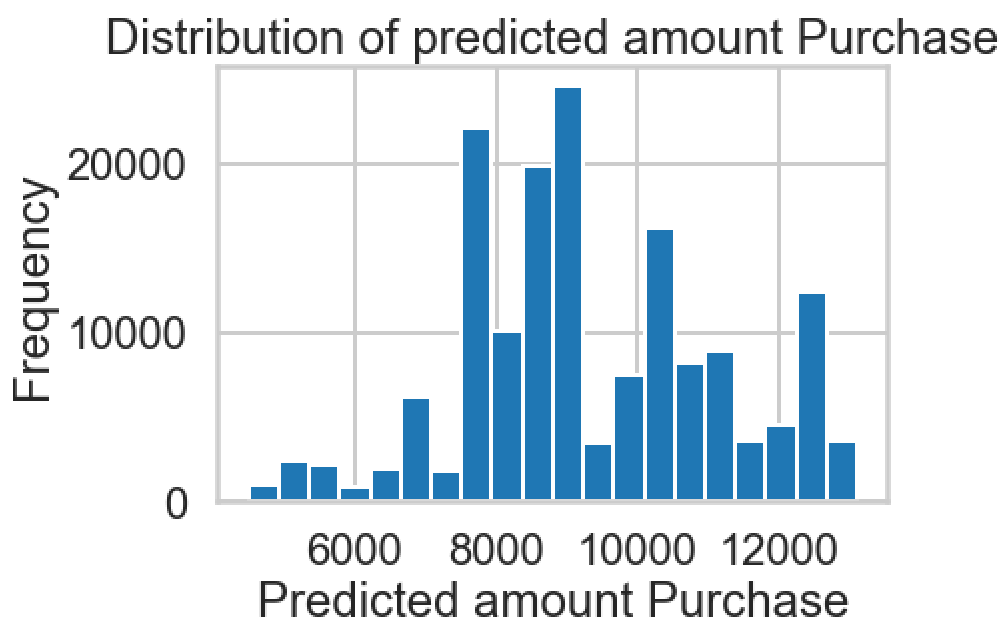
### Using the validation set, we find out that the accuracy score is slightly high

## Regression plot

In [57]:
```
# plot relationship between observed and predicted amount purchase
sn.regplot(x=model_yPred, y=y_test)
plt.xlabel('Predicted')
plt.ylabel('Observed')
plt.title(('Predicted vs observed amount Purchase'));
```

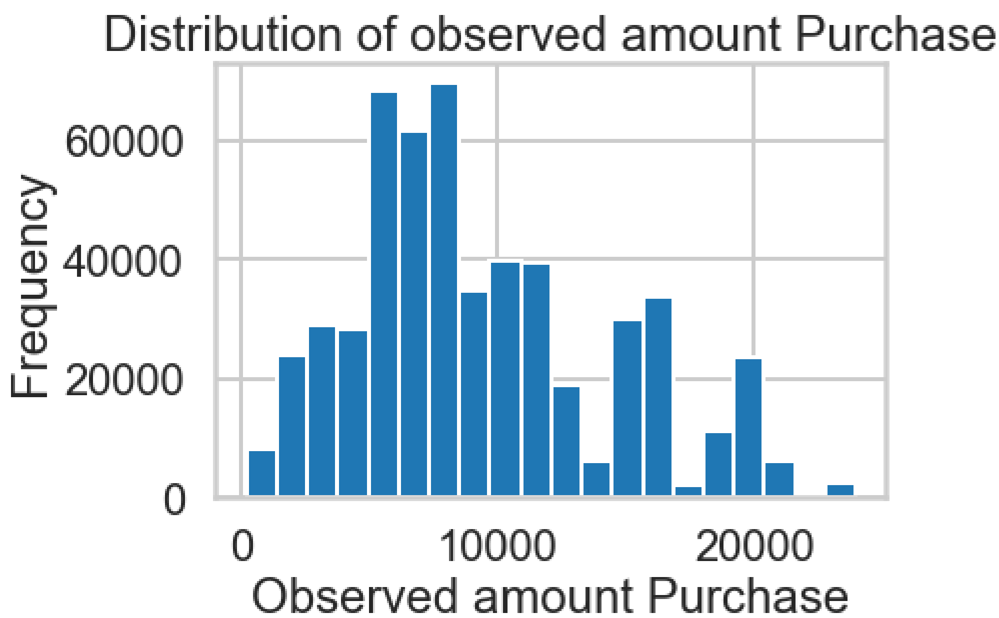



In [173]:
```
# plot histogram of predicted amount purchase
plt.hist(lm.predict(X_test), bins=20)
plt.xlabel("Predicted amount Purchase")
plt.ylabel("Frequency")
plt.title("Distribution of predicted amount Purchase");
```



In [172]:
```
# plot histogram of observed amount purchase
plt.hist(blackFridayRegression.Purchase, bins=20)
plt.xlabel("Observed amount Purchase")
plt.ylabel("Frequency")
plt.title("Distribution of observed amount Purchase")
```

Out[172]: Text(0.5, 1.0, 'Distribution of observed amount Purchase')




## Let's use another technique to see if we can improve the model

## We will use XGboost for Regression

In [32]:
```
## import the libraries for XGBoost
import xgboost
from sklearn.metrics import explained_variance_score
```

## Splitting Data into training and testing 80 % and 20 %

In [190]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y ,test_size=0.2)
```

## Building XGBoost Model

In [191]:
```
# Let's try XGboost algorithm to see if we can get better results
xgb = xgboost.XGBRegressor(n_estimators=100, learning_rate=0.08, gamma=0, subsampl
                           colsample_bytree=1, max_depth=7)
```

In [197]:
```
traindf, testdf = train_test_split(X_train, test_size = 0.2)
xgb.fit(X_train,y_train)
```

Out[197]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, importance_type='gain',
       learning_rate=0.08, max_delta_step=0, max_depth=7,
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='reg:linear', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=0.75)
```

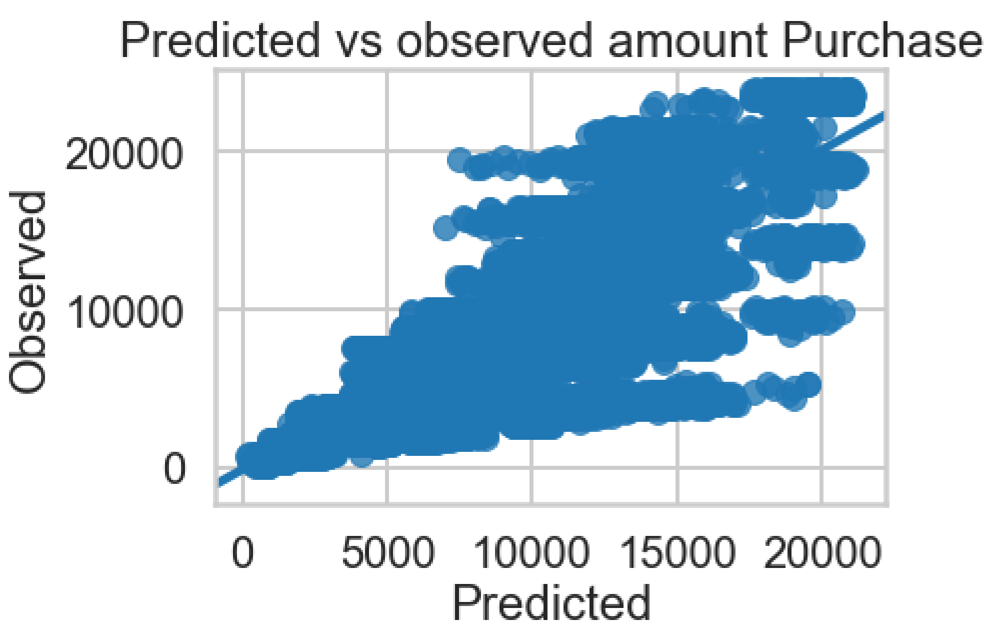
## Predict XGBoost Model

In [198]:
```
predictions = xgb.predict(X_test)
print(explained_variance_score(predictions,y_test))
```

```
0.45930676102949164
```

### We clearly see that XGboost has improved our model and the accuracy moves from 0.13 to 0.459

In [199]:
```
# plot relationship between observed and predicted amount purchase
sn.regplot(x=predictions, y=y_test)
plt.xlabel('Predicted')
plt.ylabel('Observed')
plt.title(('Predicted vs observed amount Purchase'));
```




### we see that the data points are clustered around the regression line

### this is the best we can get

# Algorithm #2 Decision Tree Classifier

### We are using classification to classify the age and the city

In [67]:
```
blackFridayClassification.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 537577 entries, 0 to 537576
Data columns (total 12 columns):
User_ID                       537577 non-null int64
Product_ID                    537577 non-null object
Gender                        537577 non-null object
Age                           537577 non-null object
Occupation                    537577 non-null int64
City_Category                 537577 non-null object
Stay_In_Current_City_Years    537577 non-null int64
Marital_Status                537577 non-null int64
Product_Category_1            537577 non-null int64
Product_Category_2            537577 non-null int64
Product_Category_3            537577 non-null int64
Purchase                      537577 non-null int64
dtypes: int64(8), object(4)
memory usage: 49.2+ MB
```

# We are going to classify the age group

In [68]:
```
target=blackFridayClassification.Age ## this the target
features =['Occupation','Stay_In_Current_City_Years','Marital_Status','Purchase',
             'Product_Category_1','Product_Category_2','Product_Category_3']
```

In [69]:
```
X=blackFridayClassification[features]
```

## Splitting the Data

In [70]:
```
# we need to split the data into training set and test set
# we do 70 % training and 30 % test
X_train, X_test, y_train, y_test=train_test_split(X,target,test_size=0.3,random_st
```

## Building Decision Tree Model

In [74]:
```
## import libraries for building a decision tree classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

In [75]:
```
# we create an object decision tree
decisionTree =DecisionTreeClassifier()
```

In [76]:
```
# We train the decision tree classifier
decisionTree.fit(X_train,y_train)
```

Out[76]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, 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, presort=False, random_state=None,
            splitter='best')
```

### Make prediction using testing set

In [77]:
```
# we predict the response for the test
y_predict=decisionTree.predict(X_test)
```

# Evaluation the decision tree classifier

In [78]:
```
# Accuracy is calculted by using the test score by comparing the test values and p
print("Accuracy",metrics.accuracy_score(y_test,y_predict))
```

```
Accuracy 0.416837183923013
```

### we find that the accuracy of the decision tree classifier is 41.68.2 %

# vusialization decision trees libraries

In [45]:
```
## this is the libraries for plotting
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
```

# Optimizing Decision tree classifier

In [87]:
```
# let see if we can optimize the decison tree by specifying the criterion and max
decisionOptimize =DecisionTreeClassifier(criterion="entropy",max_depth=8)
```

In [88]:
```
## we train the model
decisionOptimize.fit(X_train,y_train)
```

Out[88]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=8,
            max_features=None, 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, presort=False, random_state=None,
            splitter='best')
```

In [89]:
```
# predict the response for the dataset
y_preditOpt=decisionOptimize.predict(X_test)
```

In [90]:
```
# Measure de accuracy
print("Accurcy :",metrics.accuracy_score(y_test,y_preditOpt))
```

```
Accurcy : 0.5068392921363643
```

### the accuracy of the model is now 50.6 % , a bit of improvement

# Visualizing the tree

In [ ]:
```
dot_data = StringIO()
export_graphviz(decisionTree, out_file=dot_data,
         filled=True, rounded=True,
        special_characters=True,feature_names = features,class_names=['0-17','18-2
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('age.png')
Image(graph.create_png())
```

## Now, we are going to classify the city

In [91]:
```
mytarget=blackFridayClassification.City_Category ## this the target
features =['Occupation','Stay_In_Current_City_Years','Marital_Status','Purchase',
             'Product_Category_1','Product_Category_2','Product_Category_3']
```

In [92]:
```
X=blackFridayClassification[features]
```

In [93]:
```
blackFridayClassification.head()
```

Out[93]:

| | User_ID | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Mari |
|---|---|---|---|---|---|---|---|---|
| 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 | |

## Splitting Data

In [94]:
```
# we need to split the data into training set and test set
# we do 70 % training and 30 % test
X_train, X_test, y_train, y_test=train_test_split(X,mytarget,test_size=0.3,random_
```

In [95]:
```
# we create an object decision tree
decisionTree =DecisionTreeClassifier()
```

In [96]:
```
# We train the decision tree classifier
decisionTree.fit(X_train,y_train)
```

Out[96]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, 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, presort=False, random_state=None,
            splitter='best')
```

## Make prediction using testing set

In [97]:
```
# we predict the response for the test
y_predict=decisionTree.predict(X_test)
```

## Evaluation of model

In [98]:
```
# Accuracy is calculted by using the test score by comparing the test values and p
print("Accuracy",metrics.accuracy_score(y_test,y_predict))
```

```
Accuracy 0.4234656547242581
```

### the accuracy of the model is 42.23 %

## We are going to use XGboost Classifier to see if we can improve the model

In [99]:
```
# this import the library for encoding categorical variables
from sklearn.preprocessing import LabelEncoder
```

In [100]:
```
## Because XGBoost takes only numerical values, we are going to transform city fro
number =LabelEncoder()
blackFridayClassification2=blackFridayClassification
blackFridayClassification2['City_Category']=number.fit_transform(blackFridayClass
```

In [101]:
```
## import libraries for XGBoost classifier
import xgboost as xgb
from xgboost import XGBClassifier
```

In [102]:
```
mytarget=blackFridayClassification2.City_Category ## this the target
features =['Occupation','Stay_In_Current_City_Years','Marital_Status','Purchase',
           'Product_Category_1','Product_Category_2','Product_Category_3']
X=blackFridayClassification2[features]
```

In [103]:
```
####Converting the dataset into a Dmatrix will allow us to take advantage of the p
dmatrix = xgb.DMatrix(data=X, label=mytarget)
```

In [104]:
```
## We will take 70 % for traing and 30 % for testing
x_train, x_test, y_train, y_test = train_test_split(X, mytarget, test_size=0.3, ra
```

# Building and training Model

In [105]:
```
model =XGBClassifier()
model.fit(x_train,y_train)
```

Out[105]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='multi:softprob', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

# Making prediction with XGBoost Classifier

In [106]:
```
y_pred =model.predict(x_test)
predictions =[round(value) for value in y_pred]
```

# Test the performance of XGBoost

In [107]:
```
accurary =accuracy_score(y_test,predictions)
print(accurary)
```

```
0.4488076193310763
```

### The accuracy of the model is 44.88 % the model improves slightly with XGboost

# k-fold Cross Validation using XGBoost

#### "In order to build more robust models, it is common to do a k-fold cross validation

#### where all the entries in the original training dataset are used for both training as well as validation. " Datacamp

In [108]:
```
params = {"objective":"multi:softmax",'num_class':3,'colsample_bytree': 0.3,'lear
                'max_depth': 8, 'alpha': 10}


cv_results = xgb.cv(dtrain=dmatrix, params=params, nfold=3,
                    num_boost_round=100,early_stopping_rounds=10,metrics="mloglos
```

In [110]:
```
cv_results.head()
```

Out[110]:

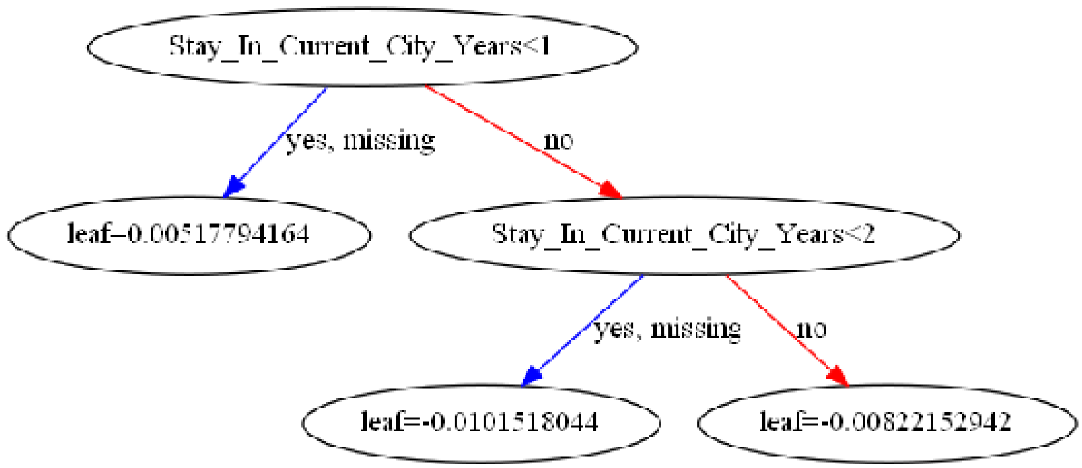
| | train-mlogloss-mean | train-mlogloss-std | test-mlogloss-mean | test-mlogloss-std |
|---|---|---|---|---|
| **0** | 1.095657 | 0.000023 | 1.095684 | 0.000008 |
| **1** | 1.091976 | 0.000058 | 1.092066 | 0.000034 |
| **2** | 1.087536 | 0.000616 | 1.087670 | 0.000570 |
| **3** | 1.084858 | 0.000849 | 1.085052 | 0.000778 |
| **4** | 1.082032 | 0.001115 | 1.082295 | 0.001123 |

#### cv_results contains train and test mlogloss for each boosting round. we take only the first 5 elements

# Visualizing XGBoost Classifier

In [98]:
```
xg_claasifier = xgb.train(params=params, dtrain=dmatrix, num_boost_round=15)
```

In [107]:
```
xgb.plot_tree(xg_claasifier,num_trees=30)
plt.show()
```




### we can conclude that XGboost classifier did improve the model

# Algorithm #3 DBSCAN

### We are going to use cluster analysis to segment customers based on their spending

#### this part is bit different fom the previous one because we do not have target

In [64]:
```
## we import the libraries require for running DBSCAN
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
```

In [43]:
```
## import library for Principal component analysis
from sklearn.decomposition import PCA
```

In [54]:
```
## we select features that we wish to use in the clustering
clusterFeatures =['Product_Category_1','Product_Category_2','Product_Category_3','
```

In [55]:
```
## we plot in X the features and in y Gender
X=blackFridayClustering[clusterFeatures]
y=blackFridayClustering.Gender
```

In [47]:
```
## we scale the data
X= StandardScaler().fit_transform(X)
```

```
c:\users\diall\appdata\local\programs\python\python37\lib\sklearn\preprocessin
g\data.py:645: DataConversionWarning: Data with input dtype int64 were all con
verted to float64 by StandardScaler.
  return self.partial_fit(X, y)
c:\users\diall\appdata\local\programs\python\python37\lib\sklearn\base.py:464:
DataConversionWarning: Data with input dtype int64 were all converted to float
64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

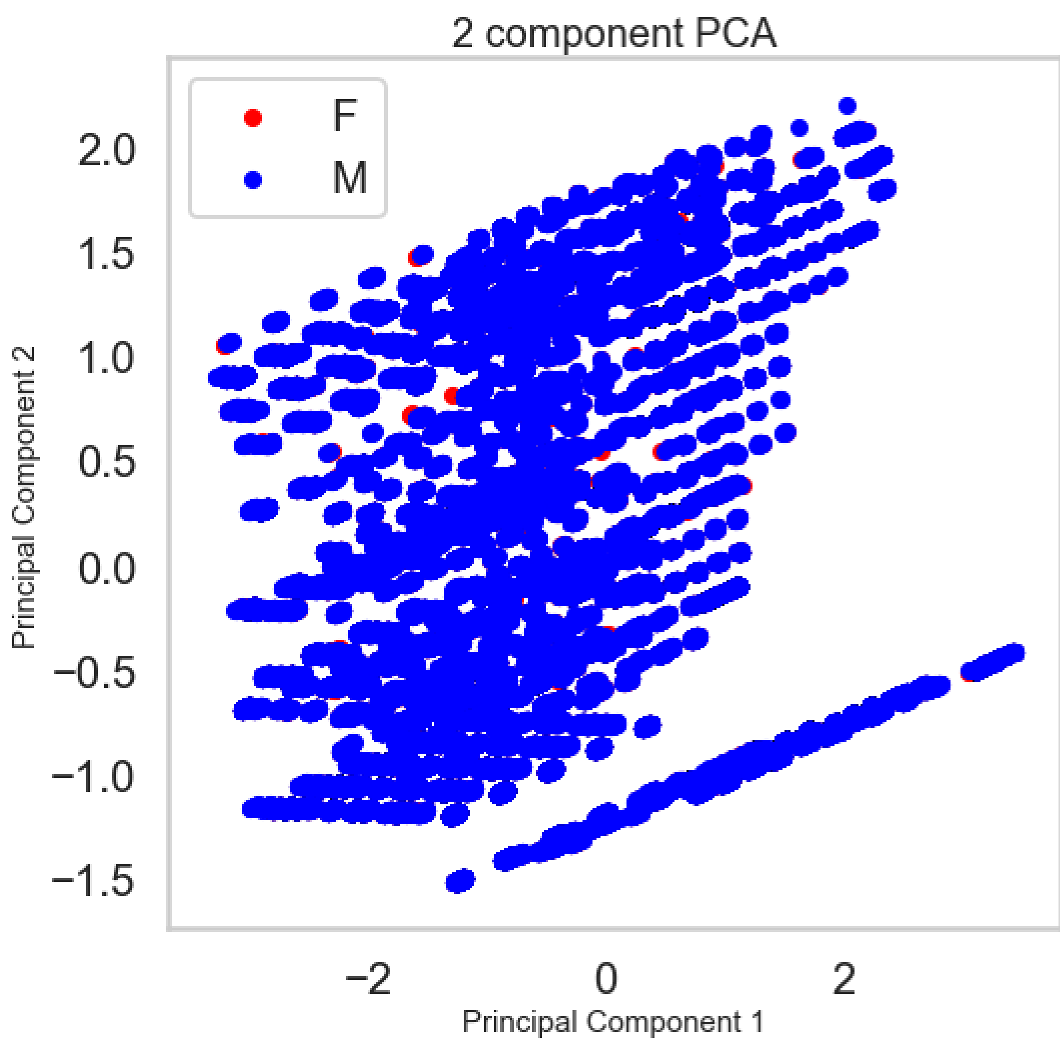
## We are going to reduce the dinmension form 4 to 2 dimensions using PCA

In [48]:
```
## create the object PCA and fit the model
pca = PCA(2)  # project from 4 to 2 dimensions
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['Principal Component 1', 'Principal Component 2'])
```

In [49]:
```
# we combine the two component with the target that we are using which gender in t
finalDf = pd.concat([principalDf, blackFridayClustering[['Gender']]], axis = 1)
```

***This section is just plotting 2 dimensional data.***

In [51]:
```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets =['F','M']
colors = ['r','b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Gender'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'Principal Component 1']
               , finalDf.loc[indicesToKeep, 'Principal Component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```




**what we see from this result is that the class are not well separeted**

# We are going to use Kmeans to see what the clustering will be

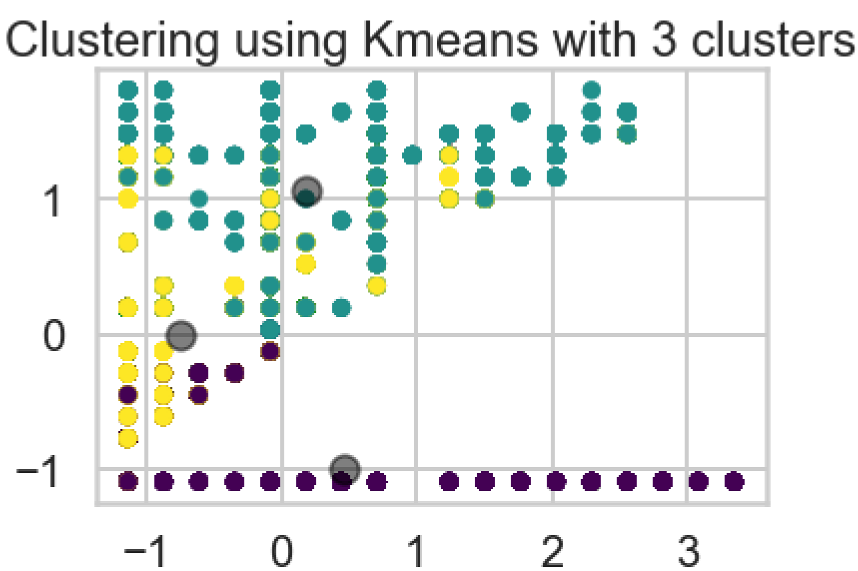
In [45]:
```
## we reduce the sample size
clustering=blackFriday.sample(frac=0.1, random_state=1)
```

In [74]:
```
## we import Kmeans libraries, create the model with 3 clusters, and fit the model
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

### We plot the result of the clustering

In [79]:
```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
plt.title('Clustering using Kmeans with 3 clusters')
```

Out[79]:
```
Text(0.5, 1.0, 'Clustering using Kmeans with 3 clusters')
```

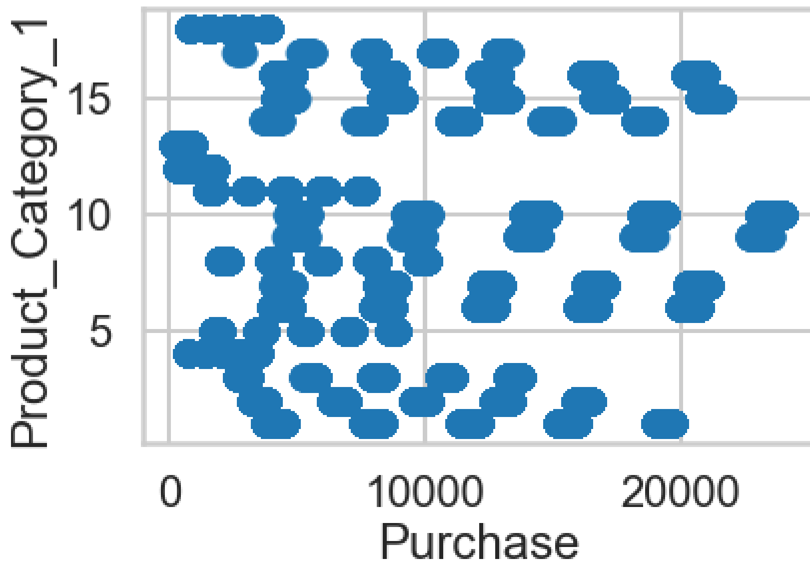



#### we clearly see that the data are not around the cluster, that means the clustering did not work well

## Let's plot the relationship

In [63]:
```
x = blackFridayClustering['Purchase']
y = blackFridayClustering['Product_Category_1']

plt.scatter(x,y)   ## we plot the scatter
plt.xlabel("Purchase")
plt.ylabel("Product_Category_1")
plt.show()
```




## We create the features for the DBSCAN model

In [65]:
```
## features selections
clusterFeatures =['Product_Category_1','Product_Category_2','Product_Category_3','
```

In [67]:
```
features=clustering[clusterFeatures]
```

In [68]:
```
## we scale the data
stscaler = StandardScaler()

X = stscaler.fit_transform(features)
```

```
c:\users\diall\appdata\local\programs\python\python37\lib\sklearn\preprocessin
g\data.py:645: DataConversionWarning: Data with input dtype int64 were all con
verted to float64 by StandardScaler.
  return self.partial_fit(X, y)
c:\users\diall\appdata\local\programs\python\python37\lib\sklearn\base.py:464:
DataConversionWarning: Data with input dtype int64 were all converted to float
64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

## Building, training the model

In [20]:
```
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
```

In [69]:
```
## this line of code we select label
labels = db.labels_
core_samples = np.zeros_like(labels, dtype = bool)
core_samples[db.core_sample_indices_] = True
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
```

In [70]:
```
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
```

In [71]:
```
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
```

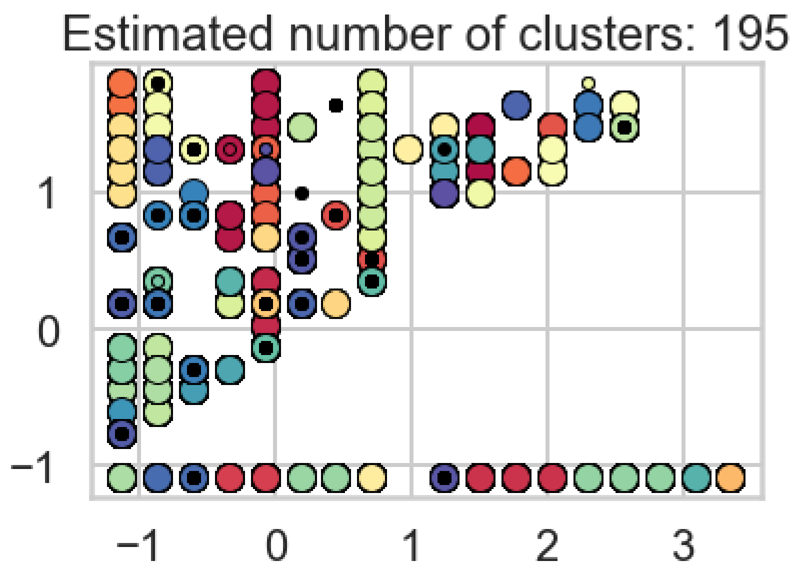
```
Estimated number of clusters: 195
Estimated number of noise points: 176
```

In [72]:
```
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy =X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)
    plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```




**we see from this result that the classes are not well separeted**

## Test of performance of DBSCAN

In [73]:
```
## the result of the accuracy
score=metrics.silhouette_score(X,labels)
print('Percentage of accuracy: %f' % score)
```

```
Percentage of accuracy: 0.428055
```

## The accuracy of the model is 42.80 %

### this is the end of my project, my goal was to use Regression, Classification and Clustering

### to see how I can help advice the retail store about their business based on data driven

# References

### [https://www.ritchieng.com/pandas-changing-datatype/ (https://www.ritchieng.com/pandas-changing-datatype/)](https://www.ritchieng.com/pandas-changing-datatype/)

### [https://seaborn.pydata.org/generated/seaborn.boxplot.html (https://seaborn.pydata.org/generated/seaborn.boxplot.html)](https://seaborn.pydata.org/generated/seaborn.boxplot.html)

### [http://jonathansoma.com/lede/foundations-2017/classes/pandas-text-part-1/classwork/ (http://jonathansoma.com/lede/foundations-2017/classes/pandas-text-part-1/classwork/)](http://jonathansoma.com/lede/foundations-2017/classes/pandas-text-part-1/classwork/)

### [https://towardsdatascience.com/analyze-the-data-through-data-visualization-using-seaborn-255e1cd3948e (https://towardsdatascience.com/analyze-the-data-through-data-visualization-using-seaborn-255e1cd3948e)](https://towardsdatascience.com/analyze-the-data-through-data-visualization-using-seaborn-255e1cd3948e)

### [https://www.kaggle.com/mburakergenc/predictions-with-xgboost-and-linear-regression (https://www.kaggle.com/mburakergenc/predictions-with-xgboost-and-linear-regression)](https://www.kaggle.com/mburakergenc/predictions-with-xgboost-and-linear-regression)

### [https://www.w3cschool.cn/doc_scikit_learn/scikit_learn-auto_examples-cluster-plot_dbscan.html?lang=en (https://www.w3cschool.cn/doc_scikit_learn/scikit_learn-auto_examples-cluster-plot_dbscan.html?lang=en)](https://www.w3cschool.cn/doc_scikit_learn/scikit_learn-auto_examples-cluster-plot_dbscan.html?lang=en)