Black Friday Dataset Exploratory Data Analysis

Problem Statement

A retail company "ABC Private Limited" wants to understand the customer purchase behaviour (specifically, purchase amount) against various products of different categories. They have shared purchase summary of various customers for selected high volume products from last month. The data set also contains customer demographics (age, gender, marital status, city type, stay_in_current_city), product details (product_id and product category) and Total purchase_amount from last month. Now, they want to build a model to predict the purchase amount of customer against various products which will help them to create personalized offer for customers against different products.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
df_train=pd.read_csv('Black_Friday_Dataset/train.csv')
df_train.head()
```

Out[2]:

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

In [3]:
```
df_test=pd.read_csv('Black_Friday_Dataset/test.csv')
df_test.head()
```

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_
0	1000004	P00128942	M	46-50	7	B	2	1	
1	1000009	P00113442	M	26-35	17	C	0	0	
2	1000010	P00288442	F	36-45	1	B	4+	1	
3	1000010	P00145342	F	36-45	1	B	4+	1	
4	1000011	P00053842	F	26-35	1	C	1	0	

In [4]:
```
# Firstly merge both the test and train data.
df=df_train.append(df_test)
```

```
C:\Users\Soumyadipta\AppData\Local\Temp\ipykernel_16008\2602686175.py:2: FutureWarning: The fra
me.append method is deprecated and will be removed from pandas in a future version. Use pandas.
concat instead.
  df=df_train.append(df_test)
```

In [5]:
```
1 df.head()
```

Out[5]:

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

In [6]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   User_ID                     783667 non-null  int64
 1   Product_ID                  783667 non-null  object
 2   Gender                      783667 non-null  object
 3   Age                         783667 non-null  object
 4   Occupation                  783667 non-null  int64
 5   City_Category               783667 non-null  object
 6   Stay_In_Current_City_Years  783667 non-null  object
 7   Marital_Status              783667 non-null  int64
 8   Product_Category_1          783667 non-null  int64
 9   Product_Category_2          537685 non-null  float64
 10  Product_Category_3          237858 non-null  float64
 11  Purchase                    550068 non-null  float64
dtypes: float64(3), int64(4), object(5)
memory usage: 77.7+ MB
```

In [7]:
```
1 df.describe()
```

Out[7]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	
count	7.836670e+05	783667.000000	783667.000000	783667.000000	537685.000000	237858.000000	550
mean	1.003029e+06	8.079300	0.409777	5.366196	9.844506	12.668605	9
std	1.727267e+03	6.522206	0.491793	3.878160	5.089093	4.125510	5
min	1.000001e+06	0.000000	0.000000	1.000000	2.000000	3.000000	
25%	1.001519e+06	2.000000	0.000000	1.000000	5.000000	9.000000	5
50%	1.003075e+06	7.000000	0.000000	5.000000	9.000000	14.000000	8
75%	1.004478e+06	14.000000	1.000000	8.000000	15.000000	16.000000	12
max	1.006040e+06	20.000000	1.000000	20.000000	18.000000	18.000000	23

Here User_ID is just useless because it is not going to help in our analysis

In [8]:
```
1 df.drop(['User_ID'],axis=1,inplace=True)
```

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

Out[9]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_
0	P00069042	F	0-17	10	A	2	0	
1	P00248942	F	0-17	10	A	2	0	
2	P00087842	F	0-17	10	A	2	0	1
3	P00085442	F	0-17	10	A	2	0	1
4	P00285442	M	55+	16	C	4+	0	

In [10]:
```
# Lets convert the categorical features into numerical
pd.get_dummies(df['Gender']) # One way
```

Out[10]:

	F	M
0	1	0
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
233594	1	0
233595	1	0
233596	1	0
233597	1	0
233598	1	0

783667 rows × 2 columns

In [11]:
```
df['Gender']=df['Gender'].map({'F':0,'M':1})
df.head()
```

Out[11]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_
0	P00069042	0	0-17	10	A	2	0	
1	P00248942	0	0-17	10	A	2	0	
2	P00087842	0	0-17	10	A	2	0	1
3	P00085442	0	0-17	10	A	2	0	1
4	P00285442	1	55+	16	C	4+	0	

In [12]:
```
df['Age'].unique()
```

Out[12]:
```
array(['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25'],
      dtype=object)
```

In [13]:
```
pd.get_dummies(df['Age']) # This is one way but lets use some ordinal encoding by providing
# This way my map model will be able to understand the pattern (Target Guiding)
# Pattern in the sence that people within the range 18 to 50 will have more num of orders th
```

Out[13]:

	0-17	18-25	26-35	36-45	46-50	51-55	55+
0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0
3	1	0	0	0	0	0	0
4	0	0	0	0	0	0	1
...	...	...	...	...	...	...	...
233594	0	0	1	0	0	0	0
233595	0	0	1	0	0	0	0
233596	0	0	1	0	0	0	0
233597	0	0	0	0	1	0	0
233598	0	0	0	0	1	0	0

783667 rows × 7 columns

In [14]:
```
df['Age']=df['Age'].map({'0-17':1,'18-25':2,'26-35':3,'36-45':4,'46-50':5,'51-55':6,'55+':7
df.head()
```

Out[14]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_
0	P00069042	0	1	10	A	2	0	
1	P00248942	0	1	10	A	2	0	
2	P00087842	0	1	10	A	2	0	1
3	P00085442	0	1	10	A	2	0	1
4	P00285442	1	7	16	C	4+	0	

In [15]:
```
# Fixing categorical City_Category
df_city=pd.get_dummies(df['City_Category'],drop_first=True)
```

In [16]:
```
df_city.head()
```

Out[16]:

	B	C
0	0	0
1	0	0
2	0	0
3	0	0
4	0	1

In [17]:
```
df=pd.concat([df,df_city],axis=1)
df.head()
```

Out[17]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_
0	P00069042	0	1	10	A	2	0	
1	P00248942	0	1	10	A	2	0	
2	P00087842	0	1	10	A	2	0	1
3	P00085442	0	1	10	A	2	0	1
4	P00285442	1	7	16	C	4+	0	

In [18]:
```
# drop City_Category
df.drop('City_Category',axis=1,inplace=True)
```

In [19]:
```
df.head()
```

Out[19]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Cate
0	P00069042	0	1	10	2	0	3	
1	P00248942	0	1	10	2	0	1	
2	P00087842	0	1	10	2	0	12	
3	P00085442	0	1	10	2	0	12	
4	P00285442	1	7	16	4+	0	8	

In [20]:
```
# Missing Values
df.isnull().sum()
```

Out[20]:
```
Product_ID                       0
Gender                           0
Age                              0
Occupation                       0
Stay_In_Current_City_Years       0
Marital_Status                   0
Product_Category_1               0
Product_Category_2          245982
Product_Category_3          545809
Purchase                    233599
B                                0
C                                0
dtype: int64
```

In [21]:
```
# Focus on replacing missing values
df['Product_Category_2'].unique()
# This gives a discrete feature
```

Out[21]:
```
array([nan,  6., 14.,  2.,  8., 15., 16., 11.,  5.,  3.,  4., 12.,  9.,
       10., 17., 13.,  7., 18.])
```

In [22]:
```
df['Product_Category_2'].value_counts()
```

Out[22]:
```
8.0     91317
14.0    78834
2.0     70498
16.0    61687
15.0    54114
5.0     37165
4.0     36705
6.0     23575
11.0    20230
17.0    19104
13.0    15054
9.0      8177
12.0     7801
10.0     4420
3.0      4123
18.0     4027
7.0       854
Name: Product_Category_2, dtype: int64
```

In [23]:
```
# Best way is to Replace the missing values with mode
df['Product_Category_3']=df['Product_Category_3'].fillna(df['Product_Category_3'].mode()[0])
```

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

Out[24]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Cate
0	P00069042	0	1	10	2	0	3	
1	P00248942	0	1	10	2	0	1	
2	P00087842	0	1	10	2	0	12	
3	P00085442	0	1	10	2	0	12	
4	P00285442	1	7	16	4+	0	8	

In [25]:
```
df.shape
```

Out[25]:
```
(783667, 12)
```

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

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

In [27]:
```
df['Stay_In_Current_City_Years']=df['Stay_In_Current_City_Years'].str.replace('+','')
```

```
C:\Users\Soumyadipta\AppData\Local\Temp\ipykernel_16008\2063355665.py:1: FutureWarning: The def
ault value of regex will change from True to False in a future version. In addition, single cha
racter regular expressions will *not* be treated as literal strings when regex=True.
  df['Stay_In_Current_City_Years']=df['Stay_In_Current_City_Years'].str.replace('+','')
```

In [28]:
```
df.head()
```

Out[28]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Cate
0	P00069042	0	1	10	2	0	3	
1	P00248942	0	1	10	2	0	1	
2	P00087842	0	1	10	2	0	12	
3	P00085442	0	1	10	2	0	12	
4	P00285442	1	7	16	4	0	8	

In [29]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   Product_ID                  783667 non-null  object 
 1   Gender                      783667 non-null  int64  
 2   Age                         783667 non-null  int64  
 3   Occupation                  783667 non-null  int64  
 4   Stay_In_Current_City_Years  783667 non-null  object 
 5   Marital_Status              783667 non-null  int64  
 6   Product_Category_1          783667 non-null  int64  
 7   Product_Category_2          537685 non-null  float64
 8   Product_Category_3          783667 non-null  float64
 9   Purchase                    550068 non-null  float64
 10  B                           783667 non-null  uint8  
 11  C                           783667 non-null  uint8  
dtypes: float64(3), int64(5), object(2), uint8(2)
memory usage: 67.3+ MB
```

In [30]:
```
df['B']=df['B'].astype(int)
df['C']=df['C'].astype(int)
```

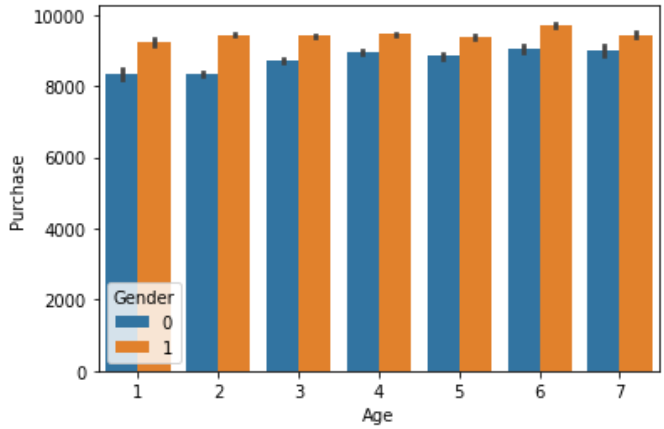
In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   Product_ID                  783667 non-null  object 
 1   Gender                      783667 non-null  int64  
 2   Age                         783667 non-null  int64  
 3   Occupation                  783667 non-null  int64  
 4   Stay_In_Current_City_Years  783667 non-null  object 
 5   Marital_Status              783667 non-null  int64  
 6   Product_Category_1          783667 non-null  int64  
 7   Product_Category_2          537685 non-null  float64
 8   Product_Category_3          783667 non-null  float64
 9   Purchase                    550068 non-null  float64
 10  B                           783667 non-null  int32  
 11  C                           783667 non-null  int32  
dtypes: float64(3), int32(2), int64(5), object(2)
memory usage: 71.7+ MB
```

In [32]:
```
##Visualisation Age vs Purchased
sns.barplot('Age','Purchase',hue='Gender',data=df)
```

```
C:\Users\Soumyadipta\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_decorat
ors.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[32]: <Axes: xlabel='Age', ylabel='Purchase'>



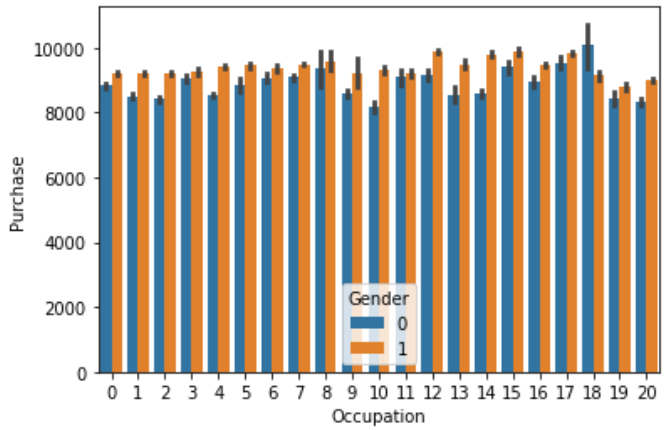
Observations:

- Purchasing of men is high then women

In [33]:
```
# Visualization of Purchase with occupation
sns.barplot('Occupation','Purchase',hue='Gender',data=df)
```

```
C:\Users\Soumyadipta\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_decorat
ors.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

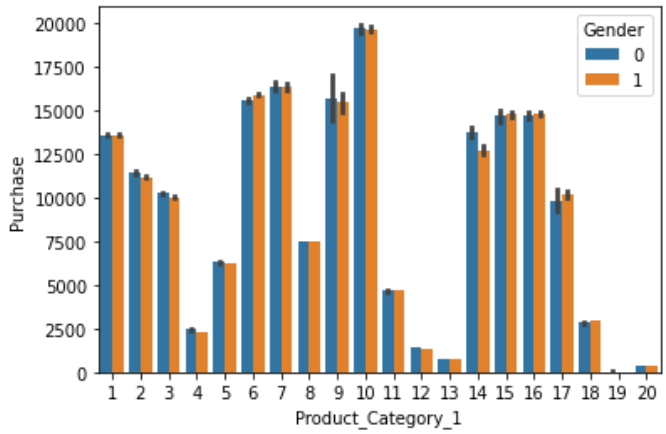
Out[33]: <Axes: xlabel='Occupation', ylabel='Purchase'>



In [34]:
```
1 sns.barplot('Product_Category_1','Purchase',hue='Gender',data=df)
```

```
C:\Users\Soumyadipta\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_decorat
ors.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

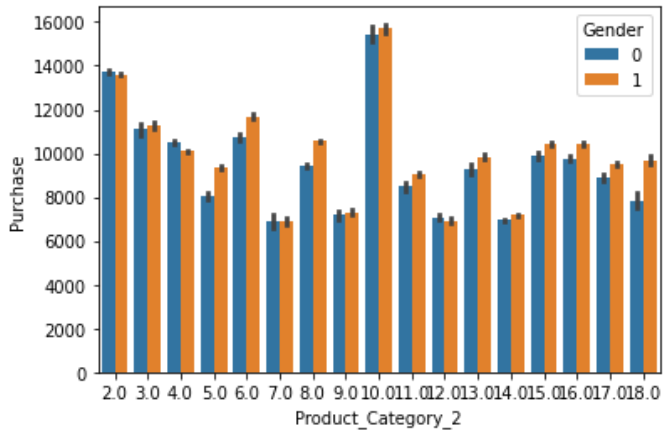
Out[34]: <Axes: xlabel='Product_Category_1', ylabel='Purchase'>



In [35]:
```
1 sns.barplot('Product_Category_2','Purchase',hue='Gender',data=df)
```

```
C:\Users\Soumyadipta\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_decorat
ors.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

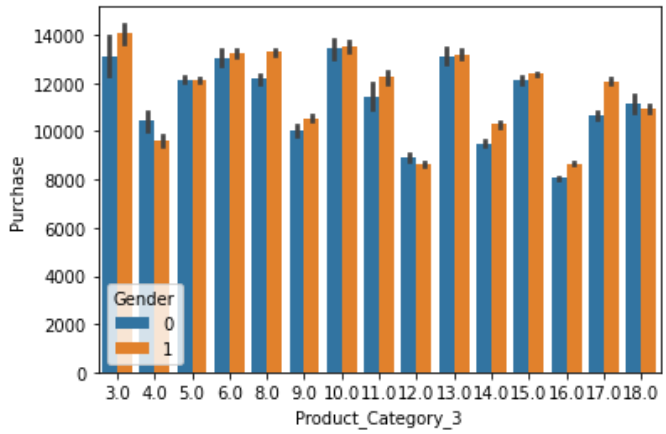
Out[35]: <Axes: xlabel='Product_Category_2', ylabel='Purchase'>



In [36]:
```
sns.barplot('Product_Category_3','Purchase',hue='Gender',data=df)
```

```
C:\Users\Soumyadipta\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\_decorat
ors.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.1
2, the only valid positional argument will be `data`, and passing other arguments without an ex
plicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[36]: <Axes: xlabel='Product_Category_3', ylabel='Purchase'>



In [37]:
```
df.head()
```

Out[37]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Cate
0	P00069042	0	1	10	2	0	3	
1	P00248942	0	1	10	2	0	1	
2	P00087842	0	1	10	2	0	12	
3	P00085442	0	1	10	2	0	12	
4	P00285442	1	7	16	4	0	8	

In [38]:
```
#Feature Scaling
df_test=df[df['Purchase'].isnull()]
```

In [39]:
```
df_train=df[~df['Purchase'].isnull()]
```

In [40]:
```
X=df_train.drop('Purchase',axis=1)
```

In [41]:
```
X.head()
```

Out[41]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Cate
0	P00069042	0	1	10	2	0	3	
1	P00248942	0	1	10	2	0	1	
2	P00087842	0	1	10	2	0	12	
3	P00085442	0	1	10	2	0	12	
4	P00285442	1	7	16	4	0	8	

In [42]:
```
X.shape
```

Out[42]: (550068, 11)

In [43]:
```
y=df_train['Purchase']
```

In [44]:
```
y.shape
```

Out[44]:
```
(550068,)
```

In [45]:
```
y
```

Out[45]:
```
0          8370.0
1         15200.0
2          1422.0
3          1057.0
4          7969.0
            ...
550063      368.0
550064      371.0
550065      137.0
550066      365.0
550067      490.0
Name: Purchase, Length: 550068, dtype: float64
```

In [46]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
     X, y, test_size=0.33, random_state=42)
```

In [47]:
```
X_train.drop('Product_ID',axis=1,inplace=True)
X_test.drop('Product_ID',axis=1,inplace=True)
```

In [48]:
```
## feature Scaling
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [49]:
```
# After this you can train your Model
```

In [ ]:
```

```