Black friday Dataset EDA and Feature Engineering

Cleaning and preparing the data for model training

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 os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
os.chdir(r'F:\EDA-Project\Blackfridaytest&train.csv')
os.listdir()
```

Out[2]:
```
['test.csv', 'train.csv']
```

In [3]:
```
# importing the train dataset
```

In [4]:
```
df_train=pd.read_csv('train.csv')
df_train.head()
```

Out[4]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969

In [5]:
```
df_test=pd.read_csv('test.csv')
df_test.head()
```

Out[5]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3
0	1000004	P00128942	M	46-50	7	B	2	1	1	11.0	NaN
1	1000009	P00113442	M	26-35	17	C	0	0	3	5.0	NaN
2	1000010	P00288442	F	36-45	1	B	4+	1	5	14.0	NaN
3	1000010	P00145342	F	36-45	1	B	4+	1	4	9.0	NaN
4	1000011	P00053842	F	26-35	1	C	1	0	4	5.0	12.0

In [6]:
```
# Merge the two DataFrames
# we can do either merge or append
# it appends at the bottom
```

In [7]:
```
df=df_train.append(df_test)
df.head()
```

Out[7]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370.0
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200.0
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422.0
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057.0
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969.0

In [8]:
```
# Basic
```

In [9]:
```
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 [10]:
```
df.describe()
```

Out[10]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
count	7.836670e+05	783667.000000	783667.000000	783667.000000	537685.000000	237858.000000	550068.000000
mean	1.003029e+06	8.079300	0.409777	5.366196	9.844506	12.668605	9263.968713
std	1.727267e+03	6.522206	0.491793	3.878160	5.089093	4.125510	5023.065394
min	1.000001e+06	0.000000	0.000000	1.000000	2.000000	3.000000	12.000000
25%	1.001519e+06	2.000000	0.000000	1.000000	5.000000	9.000000	5823.000000
50%	1.003075e+06	7.000000	0.000000	5.000000	9.000000	14.000000	8047.000000
75%	1.004478e+06	14.000000	1.000000	8.000000	15.000000	16.000000	12054.000000
max	1.006040e+06	20.000000	1.000000	20.000000	18.000000	18.000000	23961.000000

In [11]:
```
# We can drop User_id as it is just a unique id so we can drop it off we have no use of it
```

In [12]:
```
df.drop(['User_ID'],axis=1,inplace=True)
df.head()
```

Out[12]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370.0
1	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200.0
2	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422.0
3	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057.0
4	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969.0

Handling Categorical features

In [13]:
```
# Handling Categorical feature Gender
```

In [14]:
```
df['Gender']=df['Gender'].replace(['M','F'],[1,0])
```

In [15]:
```
df.head()
```

Out[15]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	P00069042	0	0-17	10	A	2	0	3	NaN	NaN	8370.0
1	P00248942	0	0-17	10	A	2	0	1	6.0	14.0	15200.0
2	P00087842	0	0-17	10	A	2	0	12	NaN	NaN	1422.0
3	P00085442	0	0-17	10	A	2	0	12	14.0	NaN	1057.0
4	P00285442	1	55+	16	C	4+	0	8	NaN	NaN	7969.0

In [16]:
```
# Handling Categorical feature Age
```

In [17]:
```
df.Age.unique()
```

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

In [18]:
```
# when training the model it will be able to understand
# This is known as Target ordinal encoding or Target Guiding
```

In [19]:
```
#pd.get_dummies(df['Age'],drop_first=True)
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})
```

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

Out[20]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	P00069042	0	1	10	A	2	0	3	NaN	NaN	8370.0
1	P00248942	0	1	10	A	2	0	1	6.0	14.0	15200.0
2	P00087842	0	1	10	A	2	0	12	NaN	NaN	1422.0
3	P00085442	0	1	10	A	2	0	12	14.0	NaN	1057.0
4	P00285442	1	7	16	C	4+	0	8	NaN	NaN	7969.0

In [21]:
```
# Handling  Categorical feature City_Category
```

In [22]:
```
# drop_first=True
# because if we have 3 categories then 2 categories are sufficient to represent all the 3 categories
```

In [23]:
```
df_city=pd.get_dummies(df['City_Category'],drop_first=True)
df_city
```

Out[23]:

	B	C
0	0	0
1	0	0
2	0	0
3	0	0
4	0	1
...	...	...
233594	1	0
233595	1	0
233596	1	0
233597	0	1
233598	1	0

783667 rows × 2 columns

In [24]:
```
df=pd.concat([df,df_city],axis=1)
df.head()
```

Out[24]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
0	P00069042	0	1	10	A	2	0	3	NaN	NaN	8370.0	0	0
1	P00248942	0	1	10	A	2	0	1	6.0	14.0	15200.0	0	0
2	P00087842	0	1	10	A	2	0	12	NaN	NaN	1422.0	0	0
3	P00085442	0	1	10	A	2	0	12	14.0	NaN	1057.0	0	0
4	P00285442	1	7	16	C	4+	0	8	NaN	NaN	7969.0	0	1

In [25]:
```
# City_Category is not required anymore can be dropped
```

In [26]:
```
#inplace= True is used to remove City_Category perminently
```

In [27]:
```
df.drop('City_Category',axis=1,inplace=True)
```

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

Out[28]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
0	P00069042	0	1	10	2	0	3	NaN	NaN	8370.0	0	0
1	P00248942	0	1	10	2	0	1	6.0	14.0	15200.0	0	0
2	P00087842	0	1	10	2	0	12	NaN	NaN	1422.0	0	0
3	P00085442	0	1	10	2	0	12	14.0	NaN	1057.0	0	0
4	P00285442	1	7	16	4+	0	8	NaN	NaN	7969.0	0	1

Finding the Missing values and handling them

In [29]:
```
df.isnull().sum()
```

Out[29]:
```
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 [30]:
```
# Purchase has the null data because of the test data
```

In [31]:
```
## Focusing on replacing missing values
# Product_Category_2            245982
# Product_Category_3            545809
```

In [32]:
```
# For 'Product_Category_2' missing values
```

In [33]:
```
# The best way for replacing the missing values categorical or discrete variables is with the mode
# .fillna() function is used to fill nan values
```

In [34]:
```
df['Product_Category_2']=df['Product_Category_2'].fillna(df['Product_Category_2'].mode()[0])
```

In [35]:
```
df['Product_Category_2'].isnull().sum()
```

Out[35]:
```
0
```

In [36]:
```
# For 'Product_Category_3' missing values
```

In [37]:
```
df['Product_Category_3'].unique()
```

Out[37]:
```
array([nan, 14., 17.,  5.,  4., 16., 15.,  8.,  9., 13.,  6., 12.,  3.,
       18., 11., 10.])
```

In [38]:
```
df['Product_Category_3'].mode()[0]
```

Out[38]:
```
16.0
```

In [39]:
```
df['Product_Category_3']=df['Product_Category_3'].fillna(df['Product_Category_3'].mode()[0])
df['Product_Category_3'].isnull().sum()
```

Out[39]:
```
0
```

In [40]:
```
df.isnull().sum()
```

Out[40]:
```
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                 0
Product_Category_3                 0
Purchase                      233599
B                                  0
C                                  0
dtype: int64
```

In [41]:
```
df.shape
```

Out[41]:
```
(783667, 12)
```

In [42]:
```
df.head()
```

Out[42]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
0	P00069042	0	1	10	2	0	3	8.0	16.0	8370.0	0	0
1	P00248942	0	1	10	2	0	1	6.0	14.0	15200.0	0	0
2	P00087842	0	1	10	2	0	12	8.0	16.0	1422.0	0	0
3	P00085442	0	1	10	2	0	12	14.0	16.0	1057.0	0	0
4	P00285442	1	7	16	4+	0	8	8.0	16.0	7969.0	0	1

In [43]:
```
# There's 4+ in Stay_In_Current_City_Years we need to fix this
# Anything above 4 can be taken as 4
```

In [44]:
```
df['Stay_In_Current_City_Years']=df['Stay_In_Current_City_Years'].str.replace('+','')
df.head()
```

```
<ipython-input-44-82959a44b5b1>:1: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single character 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('+','')
```

Out[44]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	B	C
0	P00069042	0	1	10	2	0	3	8.0	16.0	8370.0	0	0
1	P00248942	0	1	10	2	0	1	6.0	14.0	15200.0	0	0
2	P00087842	0	1	10	2	0	12	8.0	16.0	1422.0	0	0
3	P00085442	0	1	10	2	0	12	14.0	16.0	1057.0	0	0
4	P00285442	1	7	16	4	0	8	8.0	16.0	7969.0	0	1

In [45]:
```
# converting object type of Stay_In_Current_City_Years into integers
# .astype(int)
```

In [46]:
```
df['Stay_In_Current_City_Years']=df['Stay_In_Current_City_Years'].astype(int)
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  int32  
 5   Marital_Status              783667 non-null  int64  
 6   Product_Category_1          783667 non-null  int64  
 7   Product_Category_2          783667 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), int32(1), int64(5), object(1), uint8(2)
memory usage: 64.3+ MB
```

In [47]:
```
df['B']=df['B'].astype(int)
df['C']=df['C'].astype(int)
```

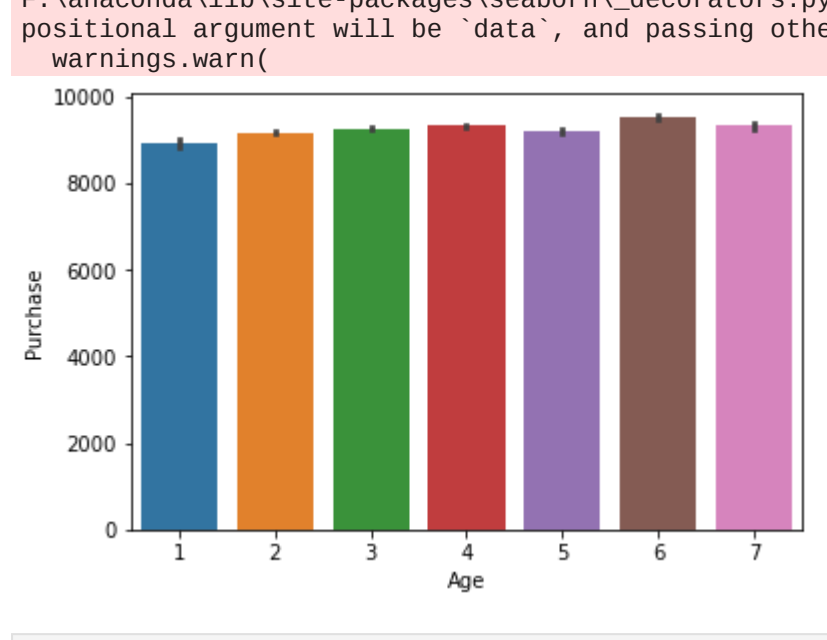
In [48]:
```
df.info()
```

```
<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  int32  
 5   Marital_Status              783667 non-null  int64  
 6   Product_Category_1          783667 non-null  int64  
 7   Product_Category_2          783667 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(3), int64(5), object(1)
memory usage: 68.8+ MB
```

Visualization

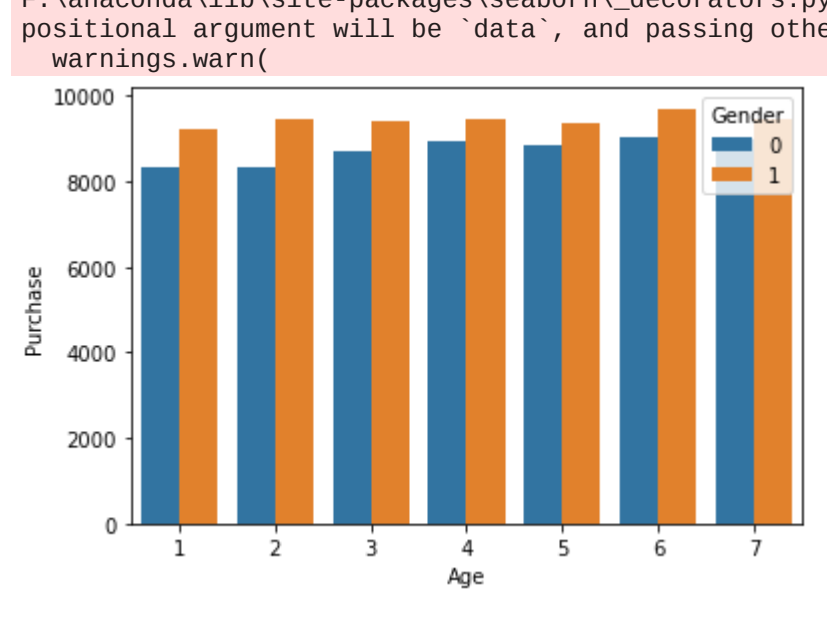
In [49]:
```
sns.barplot('Age','Purchase',data=df);
```

```
F:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [78]:
```
sns.barplot('Age','Purchase',hue='Gender',data=df,ci=None);
```

```
F:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



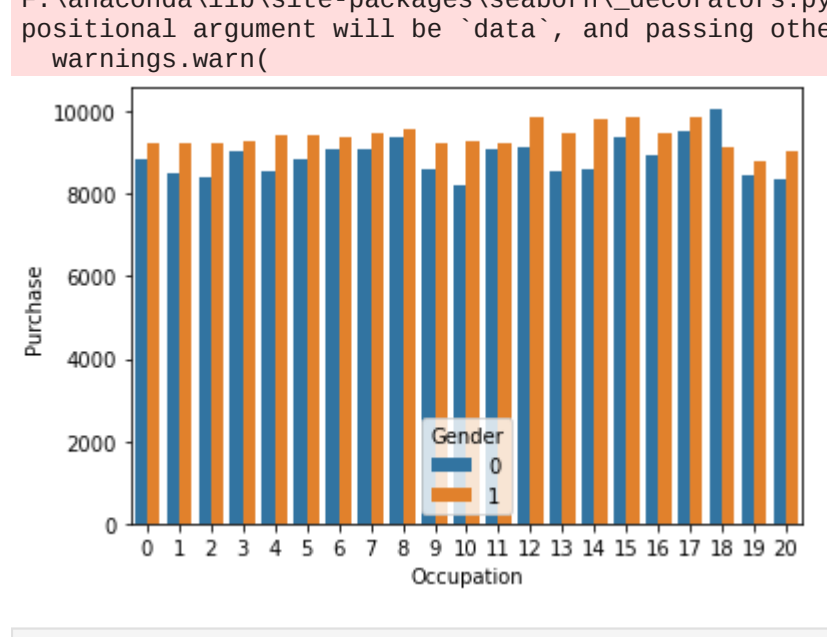
Observation

Purchasing of Men is higher than the Women

In [51]:
```
# Visualization of Purchase with respective Occupations
```

In [79]:
```
sns.barplot('Occupation','Purchase',hue='Gender',data=df,ci=None);
```

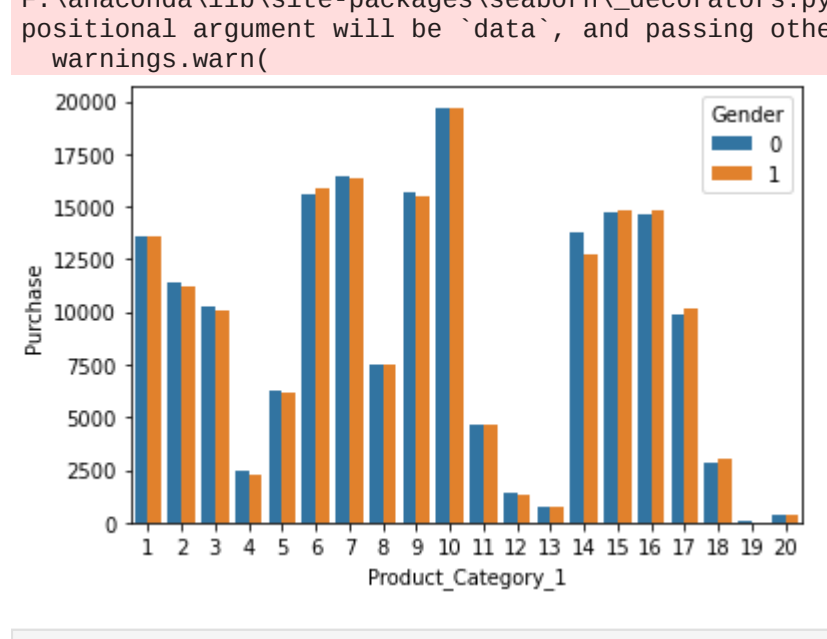
```
F:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [53]:
```
# Visualization of Purchase with respective 'Product_Category_1'
```

In [80]:
```
sns.barplot('Product_Category_1','Purchase',hue='Gender',data=df,ci=None);
```

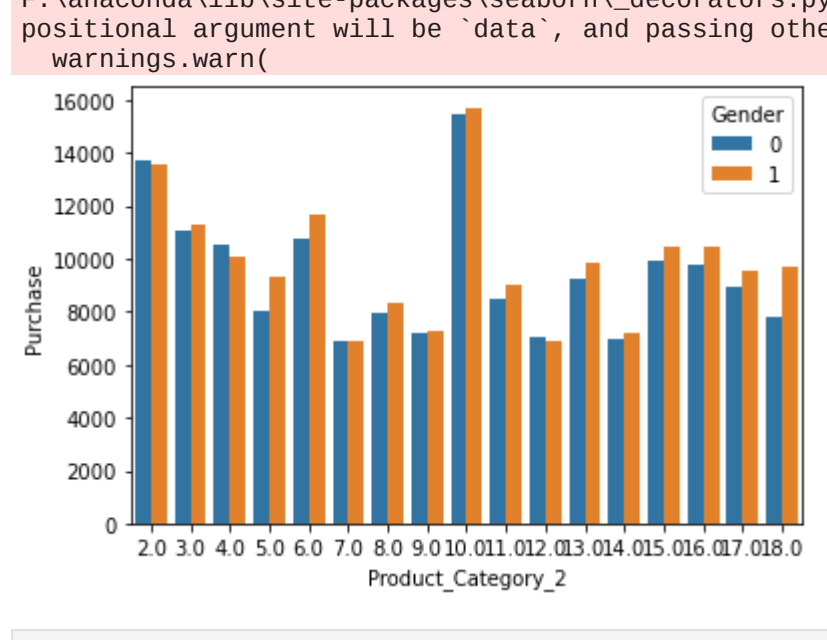
```
F:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [55]:
```
# Visualization of Purchase with respective 'Product_Category_2'
```

In [81]:
```
sns.barplot('Product_Category_2','Purchase',hue='Gender',data=df,ci=None);
```

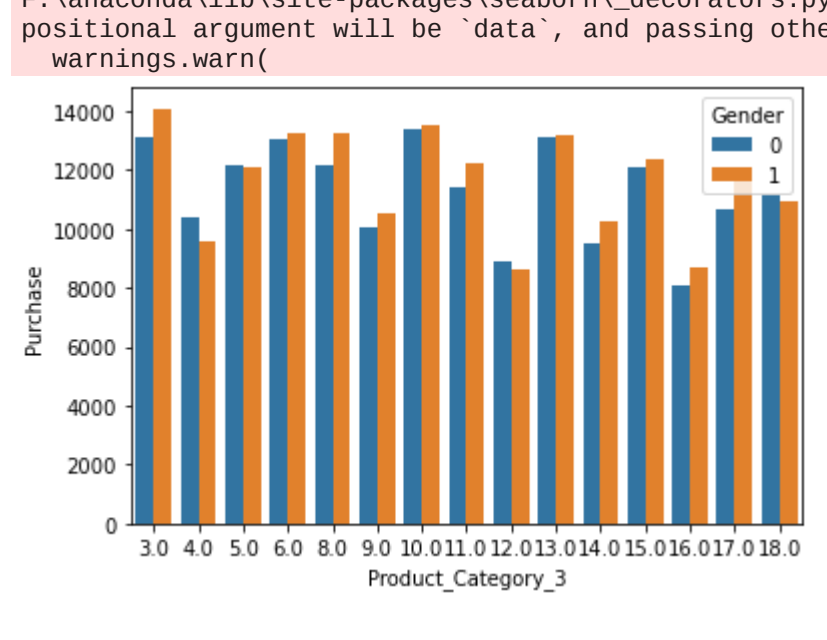
```
F:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [57]:
```
# Visualization of Purchase with respective 'Product_Category_3'
```

In [82]:
```
sns.barplot('Product_Category_3','Purchase',hue='Gender',data=df,ci=None);
```

```
F:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Feature Scaling

In [69]:
```
df_test=df[df['Purchase'].isnull()]
```

In [70]:
```
df_train=df[~df['Purchase'].isnull()]
```

In [71]:
```
X=df_train.drop('Purchase',axis=1)
```

In [72]:
```
X.head()
```

Out[72]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	B	C
0	P00069042	0	1	10	2	0	3	8.0	16.0	0	0
1	P00248942	0	1	10	2	0	1	6.0	14.0	0	0
2	P00087842	0	1	10	2	0	12	8.0	16.0	0	0
3	P00085442	0	1	10	2	0	12	14.0	16.0	0	0
4	P00285442	1	7	16	4	0	8	8.0	16.0	0	1

In [73]:
```
X.shape
```

Out[73]:
```
(550068, 11)
```

In [74]:
```
y=df_train['Purchase']
y
```

Out[74]:
```
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 [75]:
```
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 [76]:
```
X_train.drop('Product_ID',axis=1,inplace=True)
X_test.drop('Product_ID',axis=1,inplace=True)
```

```
F:\anaconda\lib\site-packages\pandas\core\frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

In [77]:
```
## feature Scaling
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

Train the Model