In [1]:

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

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 [2]:

```
#importing the train data
df_train = pd.read_csv('train.csv')
```

In [3]:

```
df_train.head(3)
```

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_
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	

In [4]:

```
df_train.shape
```

Out[4]:

```
(550068, 12)
```

In [5]:

```
#importing the test data
df_test = pd.read_csv('test.csv')
```

In [6]:

```
df_test.head(3)
```

Out[6]:

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

In [7]:

```
df_test.shape
```

Out[7]:

```
(233599, 11)
```

In [8]:

```
#append both train and test data
df = df_train.append(df_test)
```

```
C:\Users\vcyad\AppData\Local\Temp\ipykernel_11136\4095870650.py:2: FutureWarning: The
frame.append method is deprecated and will be removed from pandas in a future versio
n. Use pandas.concat instead.
  df = df_train.append(df_test)
```

In [9]:

```
df.head()
```

Out[9]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_
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 [10]:

```
df.shape
```

Out[10]:

```
(783667, 12)
```

Understanding the data

In [11]:

```
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 [12]:
```
df.describe()
```

Out[12]:

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

In [13]:
```
df.drop(['User_ID'], axis = 1, inplace = True)
```

In [14]:
```
df.head(3)
```

Out[14]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Pr
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	

In [15]:
```
#df['Gender'] = pd.get_dummies(df['Gender'], drop_first = 1)
```

In [16]:

```
#handling categorical feature gender using map function
df['Gender'] = df['Gender'].map({'F':0,'M':1})
df.head()
```

Out[16]:

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

In [17]:

```
#handling categorical feature age
df['Age'].unique()
```

Out[17]:

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

In [18]:

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

second technique

from sklearn import preprocessing

label_encoder object knows how to understand word labels.

label_encoder = preprocessing.LabelEncoder()

Encode labels in column 'species'

df['Age'] = label_encoder.fit_transform(df['Age'])

df['Age'].unique()

In [19]:

```
df.head()
```

Out[19]:

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

In [20]:

```
print(df['City_Category'].unique(),'\n')
print(df['City_Category'].value_counts())
```

```
['A' 'C' 'B']

B    329739
C    243684
A    210244
Name: City_Category, dtype: int64
```

In [21]:

```
pd.get_dummies(df['City_Category'])
```

Out[21]:

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

783667 rows × 3 columns

In [22]:

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

In [23]:

```
df_city.head()
```

Out[23]:

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

In [24]:

```
df = pd.concat([df,df_city], axis = 1)
df
```

Out[24]:

	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Stat
0	P00069042	0	1	10	A	2	
1	P00248942	0	1	10	A	2	
2	P00087842	0	1	10	A	2	
3	P00085442	0	1	10	A	2	
4	P00285442	1	7	16	C	4+	
...	...	...	...	...	...	...	
233594	P00118942	0	3	15	B	4+	
233595	P00254642	0	3	15	B	4+	
233596	P00031842	0	3	15	B	4+	
233597	P00124742	0	5	1	C	4+	
233598	P00316642	0	5	0	B	4+	

783667 rows × 13 columns

In [25]:

```
#drop city category feature
df.drop('City_Category', axis = 1, inplace = True)
df.head()
```

Out[25]:

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

In [26]:

```
#missing values
df.isnull().sum()
```

Out[26]:

```
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 [27]:

```
# focusing on replacing missing values
df['Product_Category_2'].unique()
```

Out[27]:

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

In [28]:

```
df['Product_Category_2'].value_counts()
```

Out[28]:

```
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 [29]:

```
df['Product_Category_2'].mode()[0]
```

Out[29]:

```
8.0
```

In [30]:

```
#replace the missing values with mode
df['Product_Category_2']=df['Product_Category_2'].fillna(df['Product_Category_2'].mode
```

In [31]:

```
df['Product_Category_2'].isnull().sum()
```

Out[31]:

```
0
```

In [32]:

```
#product_category_3 replace missing values
df['Product_Category_3'].unique()
```

Out[32]:

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

In [33]:

```
df['Product_Category_3'].value_counts()
```

Out[33]:

```
16.0    46469
15.0    39968
14.0    26283
17.0    23818
5.0     23799
8.0     17861
9.0     16532
12.0    13115
13.0     7849
6.0      6888
18.0     6621
4.0      2691
11.0     2585
10.0     2501
3.0       878
Name: Product_Category_3, dtype: int64
```

In [34]:

```
#replace missing values with mode
df['Product_Category_3']=df['Product_Category_3'].fillna(df['Product_Category_3'].mode
```

In [35]:

```
df.head()
```

Out[35]:

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

In [36]:

```
df.shape
```

Out[36]:

```
(783667, 12)
```

In [37]:

```
df['Stay_In_Current_City_Years'].unique()
```

Out[37]:

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

In [38]:

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

```
C:\Users\vcyad\AppData\Local\Temp\ipykernel_11136\2063355665.py:1: FutureWarning: The
default value of regex will change from True to False in a future version. In additio
n, 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
('+','')
```

In [39]:

```
df.head()
```

Out[39]:

	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category
0	P00069042	0	1	10	2	0	
1	P00248942	0	1	10	2	0	
2	P00087842	0	1	10	2	0	
3	P00085442	0	1	10	2	0	
4	P00285442	1	7	16	4	0	

In [40]:

```
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          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), int64(5), object(2), uint8(2)
memory usage: 67.3+ MB
```

In [41]:

```
#convert object into integers
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 [42]:

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

In [43]:

```
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
```

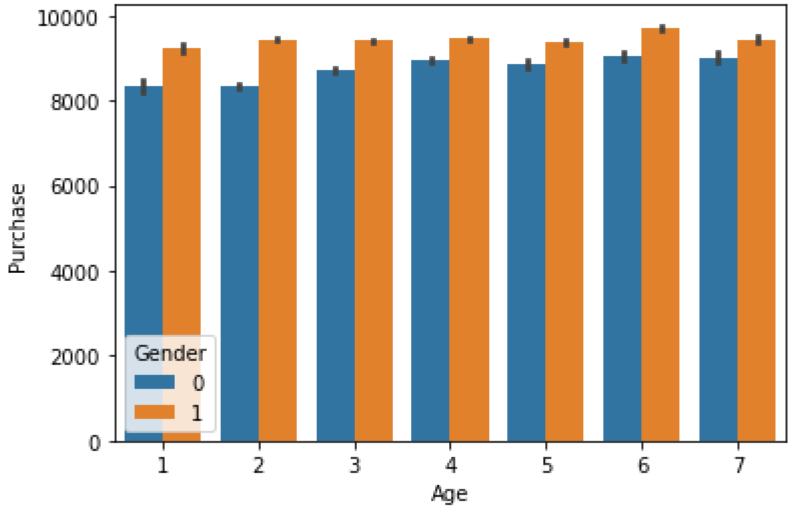
In [44]:

```
#visualizing age vs purchased
sns.barplot('Age','Purchase',hue = 'Gender', data = df)
```

```
C:\Users\vcyad\anaconda3\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 k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[44]:

```
<AxesSubplot:xlabel='Age', ylabel='Purchase'>
```



Observation: purchasing of men is high than women

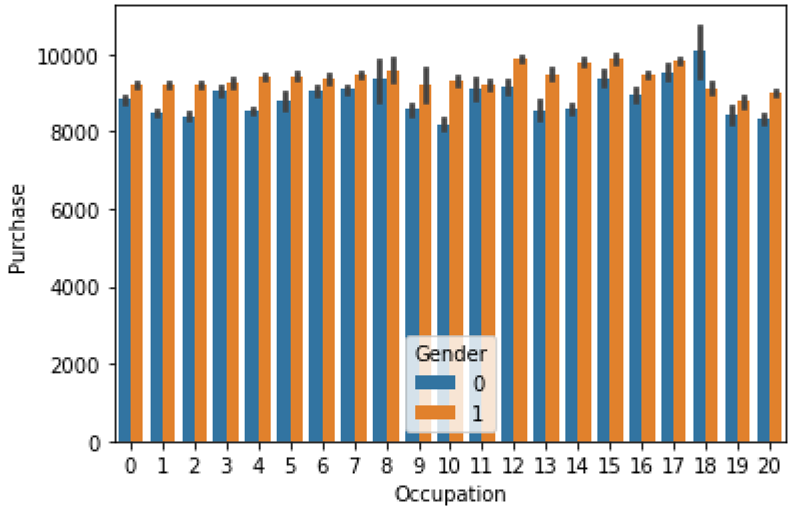
In [45]:

```
#visualizing purchase with occupation
sns.barplot('Occupation' ,'Purchase', hue = 'Gender', data = df)
```

```
C:\Users\vcyad\anaconda3\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 k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[45]:

```
<AxesSubplot:xlabel='Occupation', ylabel='Purchase'>
```



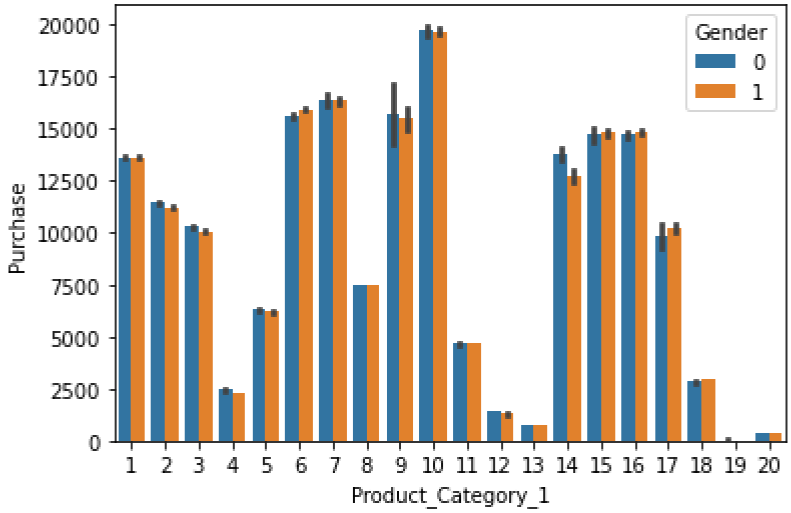
In [46]:

```
sns.barplot('Product_Category_1','Purchase',hue='Gender',data=df)
```

```
C:\Users\vcyad\anaconda3\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 k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[46]:

```
<AxesSubplot:xlabel='Product_Category_1', ylabel='Purchase'>
```



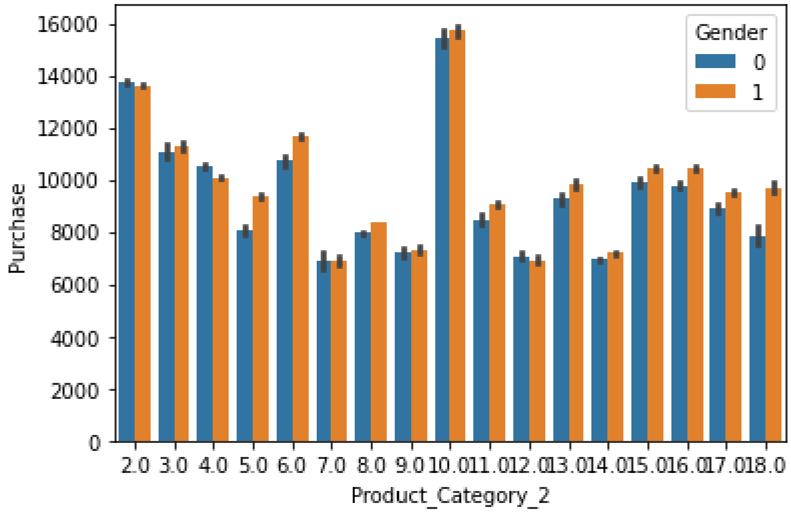
In [47]:

```
sns.barplot('Product_Category_2','Purchase',hue='Gender',data=df)
```

```
C:\Users\vcyad\anaconda3\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 k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[47]:

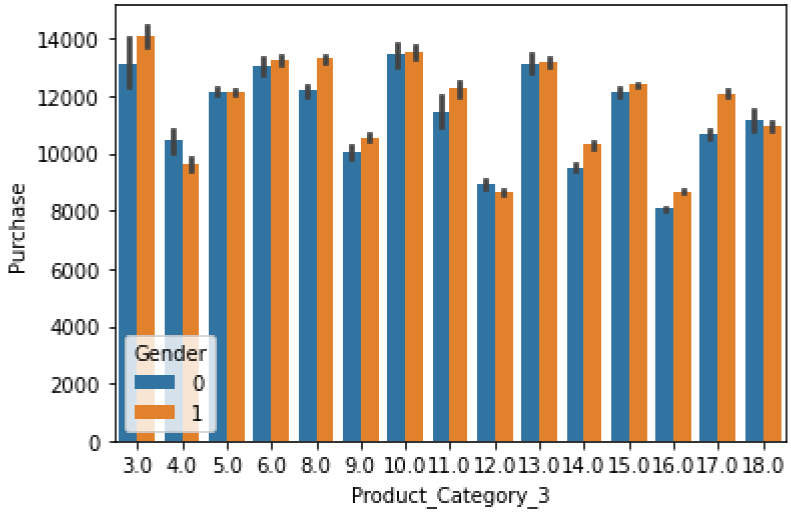
```
<AxesSubplot:xlabel='Product_Category_2', ylabel='Purchase'>
```



```
In [48]: sns.barplot('Product_Category_3','Purchase',hue='Gender',data=df)
```

```
C:\Users\vcyad\anaconda3\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 k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

```
Out[48]: <AxesSubplot:xlabel='Product_Category_3', ylabel='Purchase'>
```


Purchase
Gender
Product_Category_3