# Cleaning and preparing the data for model trining

In [2]:

```
import pandas as pd
import numpy as np
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 ot build a model to predict and purchase amount of customer against various products which will help them to create personalized offer for customers against different products.

In [3]:

```
# importing the dataset
df_train = pd.read_csv('blackfriday_train.csv')
```

In [4]:
```
df_train.head(10)
```

Out[4]:

| | User_ID | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | Product_Catego |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1000001 | P00069042 | F | 0-17 | 10 | A | 2 | 0 | 3 | |
| **1** | 1000001 | P00248942 | F | 0-17 | 10 | A | 2 | 0 | 1 | |
| **2** | 1000001 | P00087842 | F | 0-17 | 10 | A | 2 | 0 | 12 | |
| **3** | 1000001 | P00085442 | F | 0-17 | 10 | A | 2 | 0 | 12 | |
| **4** | 1000002 | P00285442 | M | 55+ | 16 | C | 4+ | 0 | 8 | |
| **5** | 1000003 | P00193542 | M | 26-35 | 15 | A | 3 | 0 | 1 | |
| **6** | 1000004 | P00184942 | M | 46-50 | 7 | B | 2 | 1 | 1 | |
| **7** | 1000004 | P00346142 | M | 46-50 | 7 | B | 2 | 1 | 1 | |
| **8** | 1000004 | P0097242 | M | 46-50 | 7 | B | 2 | 1 | 1 | |
| **9** | 1000005 | P00274942 | M | 26-35 | 20 | A | 1 | 1 | 8 | |

In [5]:
```
df_train.shape
```

Out[5]:
```
(550068, 12)
```

In [6]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   User_ID                     550068 non-null  int64  
 1   Product_ID                  550068 non-null  object 
 2   Gender                      550068 non-null  object 
 3   Age                         550068 non-null  object 
 4   Occupation                  550068 non-null  int64  
 5   City_Category               550068 non-null  object 
 6   Stay_In_Current_City_Years  550068 non-null  object 
 7   Marital_Status              550068 non-null  int64  
 8   Product_Category_1          550068 non-null  int64  
 9   Product_Category_2          376430 non-null  float64
 10  Product_Category_3          166821 non-null  float64
 11  Purchase                    550068 non-null  int64  
dtypes: float64(2), int64(5), object(5)
memory usage: 50.4+ MB
```

In [7]:
```
df_train.describe()
```

Out[7]:

| | User_ID | Occupation | Marital_Status | Product_Category_1 | Product_Category_2 | Product_Category_3 | Purchase |
|---|---|---|---|---|---|---|---|
| **count** | 5.500680e+05 | 550068.000000 | 550068.000000 | 550068.000000 | 376430.000000 | 166821.000000 | 550068.000000 |
| **mean** | 1.003029e+06 | 8.076707 | 0.409653 | 5.404270 | 9.842329 | 12.668243 | 9263.968713 |
| **std** | 1.727592e+03 | 6.522660 | 0.491770 | 3.936211 | 5.086590 | 4.125338 | 5023.065394 |
| **min** | 1.000001e+06 | 0.000000 | 0.000000 | 1.000000 | 2.000000 | 3.000000 | 12.000000 |
| **25%** | 1.001516e+06 | 2.000000 | 0.000000 | 1.000000 | 5.000000 | 9.000000 | 5823.000000 |
| **50%** | 1.003077e+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 [8]:
```
df_train.isnull().sum()
```

Out[8]:
```
User_ID                            0
Product_ID                         0
Gender                             0
Age                                0
Occupation                         0
City_Category                      0
Stay_In_Current_City_Years         0
Marital_Status                     0
Product_Category_1                 0
Product_Category_2            173638
Product_Category_3            383247
Purchase                           0
dtype: int64
```

In [9]:
```
## Import the test data
df_test=pd.read_csv("blackfriday_test.csv")
```

In [10]:
```
df_test.head()
```

Out[10]:

| | User_ID | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | Product_Catego |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1000004 | P00128942 | M | 46-50 | 7 | B | 2 | 1 | 1 | |
| **1** | 1000009 | P00113442 | M | 26-35 | 17 | C | 0 | 0 | 3 | |
| **2** | 1000010 | P00288442 | F | 36-45 | 1 | B | 4+ | 1 | 5 | |
| **3** | 1000010 | P00145342 | F | 36-45 | 1 | B | 4+ | 1 | 4 | |
| **4** | 1000011 | P00053842 | F | 26-35 | 1 | C | 1 | 0 | 4 | |

In [11]:
```
df_test.shape
```

Out[11]:
```
(233599, 11)
```

In [12]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 233599 entries, 0 to 233598
Data columns (total 11 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   User_ID                     233599 non-null  int64  
 1   Product_ID                  233599 non-null  object 
 2   Gender                      233599 non-null  object 
 3   Age                         233599 non-null  object 
 4   Occupation                  233599 non-null  int64  
 5   City_Category               233599 non-null  object 
 6   Stay_In_Current_City_Years  233599 non-null  object 
 7   Marital_Status              233599 non-null  int64  
 8   Product_Category_1          233599 non-null  int64  
 9   Product_Category_2          161255 non-null  float64
 10  Product_Category_3          71037 non-null   float64
dtypes: float64(2), int64(4), object(5)
memory usage: 19.6+ MB
```

In [13]:
```
df_test.describe()
```

Out[13]:

| | User_ID | Occupation | Marital_Status | Product_Category_1 | Product_Category_2 | Product_Category_3 |
|---|---|---|---|---|---|---|
| **count** | 2.335990e+05 | 233599.000000 | 233599.000000 | 233599.000000 | 161255.000000 | 71037.000000 |
| **mean** | 1.003029e+06 | 8.085407 | 0.410070 | 5.276542 | 9.849586 | 12.669454 |
| **std** | 1.726505e+03 | 6.521146 | 0.491847 | 3.736380 | 5.094943 | 4.125944 |
| **min** | 1.000001e+06 | 0.000000 | 0.000000 | 1.000000 | 2.000000 | 3.000000 |
| **25%** | 1.001527e+06 | 2.000000 | 0.000000 | 1.000000 | 5.000000 | 9.000000 |
| **50%** | 1.003070e+06 | 7.000000 | 0.000000 | 5.000000 | 9.000000 | 14.000000 |
| **75%** | 1.004477e+06 | 14.000000 | 1.000000 | 8.000000 | 15.000000 | 16.000000 |
| **max** | 1.006040e+06 | 20.000000 | 1.000000 | 18.000000 | 18.000000 | 18.000000 |

In [14]:
```
df_test.isnull().sum()
```

Out[14]:
```
User_ID                            0
Product_ID                         0
Gender                             0
Age                                0
Occupation                         0
City_Category                      0
Stay_In_Current_City_Years         0
Marital_Status                     0
Product_Category_1                 0
Product_Category_2             72344
Product_Category_3            162562
dtype: int64
```

In [15]:
```
## Merging the train and test dataset

df = df_train.append(df_test)
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_16220\2182507668.py:3: FutureWarning: The frame.append method is depreca
ted and will be removed from pandas in a future version. Use pandas.concat instead.
  df = df_train.append(df_test)
```

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

Out[16]:

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

In [17]:
```
# Dropping USER_ID column because it is of no use

df.drop(["User_ID"],axis=1,inplace=True)
```

In [18]:
```
df.head()
```

Out[18]:

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

# Fixing categorical column into Numerical

In [19]:

```
# GENDER Column

df["Gender"]=df["Gender"].map({"F":0,"M":1})
```

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 | Pro |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | P00069042 | 0 | 0-17 | 10 | A | 2 | 0 | 3 | NaN | |
| **1** | P00248942 | 0 | 0-17 | 10 | A | 2 | 0 | 1 | 6.0 | |
| **2** | P00087842 | 0 | 0-17 | 10 | A | 2 | 0 | 12 | NaN | |
| **3** | P00085442 | 0 | 0-17 | 10 | A | 2 | 0 | 12 | 14.0 | |
| **4** | P00285442 | 1 | 55+ | 16 | C | 4+ | 0 | 8 | NaN | |

In [21]:

```
# AGE column

df["Age"].unique()
```

Out[21]:

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

In [22]:

```
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 [23]:

```
df.head()
```

Out[23]:

| | Product_ID | Gender | Age | Occupation | City_Category | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | Product_Category_2 | Pro |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | P00069042 | 0 | 1 | 10 | A | 2 | 0 | 3 | NaN | |
| **1** | P00248942 | 0 | 1 | 10 | A | 2 | 0 | 1 | 6.0 | |
| **2** | P00087842 | 0 | 1 | 10 | A | 2 | 0 | 12 | NaN | |
| **3** | P00085442 | 0 | 1 | 10 | A | 2 | 0 | 12 | 14.0 | |
| **4** | P00285442 | 1 | 7 | 16 | C | 4+ | 0 | 8 | NaN | |

```
### 2nd technique:----------->>>>>> Using LABEL ENCODING
from sklearn import preprocessing

# Label_encoder object knows how to undestand word label
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column "Age"
df["Age"]=label_encoder.fit_transform(df["Age"])

df["Age"].unique()
```

In [24]:

```
# CITY_CATEGORY column

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

In [25]:

```
df_city.head()
```

Out[25]:

|   | B | C |
|---|---|---|
| **0** | 0 | 0 |
| **1** | 0 | 0 |
| **2** | 0 | 0 |
| **3** | 0 | 0 |
| **4** | 0 | 1 |

In [55]:

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

Out[55]:

|   | Product_ID | Gender | Age | Occupation | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | Product_Category_2 | Product_Category_ |
|---|---|---|---|---|---|---|---|---|---|
| **0** | P00069042 | 0 | 1 | 10 | 2 | 0 | 3 | 8.0 | 16. |
| **1** | P00248942 | 0 | 1 | 10 | 2 | 0 | 1 | 6.0 | 14. |
| **2** | P00087842 | 0 | 1 | 10 | 2 | 0 | 12 | 8.0 | 16. |
| **3** | P00085442 | 0 | 1 | 10 | 2 | 0 | 12 | 14.0 | 16. |
| **4** | P00285442 | 1 | 7 | 16 | 4 | 0 | 8 | 8.0 | 16. |

In [56]:

```
###### now dropping city_category

df.drop('City_Category',axis=1,inplace=True)


## already DROPPED this columnn thats why showing error
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[56], line 3
      1 ###### now dropping city_category
----> 3 df.drop('City_Category',axis=1,inplace=True)

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decora
te.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:5399, in DataFrame.drop(self, labels, axis, index, columns,
level, inplace, errors)
   5251 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
   5252 def drop(  # type: ignore[override]
   5253     self,
   (...)
   5260     errors: IgnoreRaise = "raise",
   5261 ) -> DataFrame | None:
   5262     """
   5263     Drop specified labels from rows or columns.
   5264 
   (...)
   5397             weight  1.0     0.8
   5398     """
-> 5399     return super().drop(
   5400         labels=labels,
   5401         axis=axis,
   5402         index=index,
   5403         columns=columns,
   5404         level=level,
   5405         inplace=inplace,
   5406         errors=errors,
   5407     )

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decora
```

```
te.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:4505, in NDFrame.drop(self, labels, axis, index, columns,
level, inplace, errors)
   4503 for axis, labels in axes.items():
   4504     if labels is not None:
-> 4505         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4507 if inplace:
   4508     self._update_inplace(obj)

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:4575, in NDFrame._drop_axis(self, labels, axis, level, err
ors, only_slice)
   4573     labels_missing = (axis.get_indexer_for(labels) == -1).any()
   4574     if errors == "raise" and labels_missing:
-> 4575         raise KeyError(f"{labels} not found in axis")
   4577 if is_extension_array_dtype(mask.dtype):
   4578     # GH#45860
   4579     mask = mask.to_numpy(dtype=bool)

KeyError: "['City_Category'] not found in axis"
```

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

# MISSING VALUES

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
```

## we now focus on replacing missing values

In [30]:

```
### replacing the "missing values" using mode for "PRODUCT_CATEGORY_2"
```

In [31]:

```
df["Product_Category_2"].unique()
```

Out[31]:

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

```
In [32]: df['Product_Category_2'].value_counts()
```

```
Out[32]: 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 [33]: # filling the "missing values" using mode for "PRODUCT_CATEGORY_2"

         df["Product_Category_2"]=df["Product_Category_2"]\
         .fillna(df['Product_Category_2'].mode()[0])
```

```
In [34]: df["Product_Category_2"].isnull().sum()
```

```
Out[34]: 0
```

```
In [35]: # replacing the "missing values" using mode for "PRODUCT_CATEGORY_3"
```

```
In [36]: df["Product_Category_3"].unique()
```

```
Out[36]: array([nan, 14., 17.,  5.,  4., 16., 15.,  8.,  9., 13.,  6., 12.,  3.,
                18., 11., 10.])
```

```
In [37]: df["Product_Category_3"].value_counts()
```

```
Out[37]: 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 [38]: # filling the "missing values" using mode for "PRODUCT_CATEGORY_3"

         df["Product_Category_3"]=df["Product_Category_3"].fillna(df["Product_Category_3"].mode()[0])
```

```
In [39]: df["Product_Category_3"].isnull().sum()
```

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

In [40]:
```
df.head()
```

Out[40]:

| | Product_ID | Gender | Age | Occupation | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | Product_Category_2 | Product_Category_ |
|---|---|---|---|---|---|---|---|---|---|
| **0** | P00069042 | 0 | 1 | 10 | 2 | 0 | 3 | 8.0 | 16. |
| **1** | P00248942 | 0 | 1 | 10 | 2 | 0 | 1 | 6.0 | 14. |
| **2** | P00087842 | 0 | 1 | 10 | 2 | 0 | 12 | 8.0 | 16. |
| **3** | P00085442 | 0 | 1 | 10 | 2 | 0 | 12 | 14.0 | 16. |
| **4** | P00285442 | 1 | 7 | 16 | 4+ | 0 | 8 | 8.0 | 16. |

In [41]:
```
## Removing "+" in "Stay_In_Current_City_Years"

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

Out[41]:
```
array(['2', '4+', '3', '1', '0'], dtype=object)
```

In [42]:
```
df['Stay_In_Current_City_Years']=df['Stay_In_Current_City_Years'].str.replace('+','')
```

```
C:\Users\harsh\AppData\Local\Temp\ipykernel_16220\2063355665.py:1: FutureWarning: The default value of regex will ch
ange 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('+','')
```

In [43]:
```
df.head()
```

Out[43]:

| | Product_ID | Gender | Age | Occupation | Stay_In_Current_City_Years | Marital_Status | Product_Category_1 | Product_Category_2 | Product_Category_ |
|---|---|---|---|---|---|---|---|---|---|
| **0** | P00069042 | 0 | 1 | 10 | 2 | 0 | 3 | 8.0 | 16. |
| **1** | P00248942 | 0 | 1 | 10 | 2 | 0 | 1 | 6.0 | 14. |
| **2** | P00087842 | 0 | 1 | 10 | 2 | 0 | 12 | 8.0 | 16. |
| **3** | P00085442 | 0 | 1 | 10 | 2 | 0 | 12 | 14.0 | 16. |
| **4** | P00285442 | 1 | 7 | 16 | 4 | 0 | 8 | 8.0 | 16. |

# CONVERTING Dtype

In [44]:
```
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 [45]:

```
## Converting Dtype from 'Object' to "int64" of Column "Stay_In_Current_City_Years"

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 [46]:

```
## ## Converting Dtype from 'uint8' to "int" of Column "B & C"

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

In [47]:

```
df.info()
```

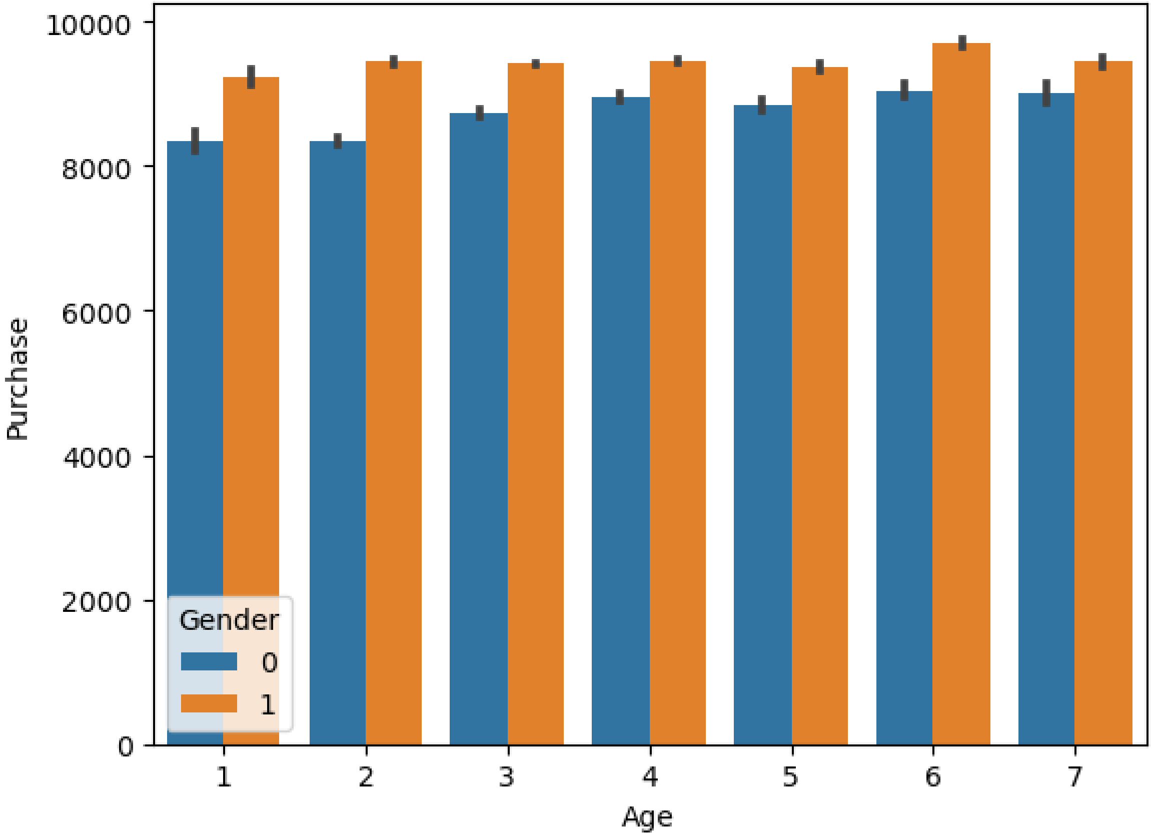
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   Product_ID                  783667 non-null  object 
 1   Gender                      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 [58]:
```
## Age Vs Purchase


sns.barplot(df, x='Age', y='Purchase',hue='Gender')
```

Out[58]:
```
<Axes: xlabel='Age', ylabel='Purchase'>
```



```
Inference: Purchasing of men is high then women
```

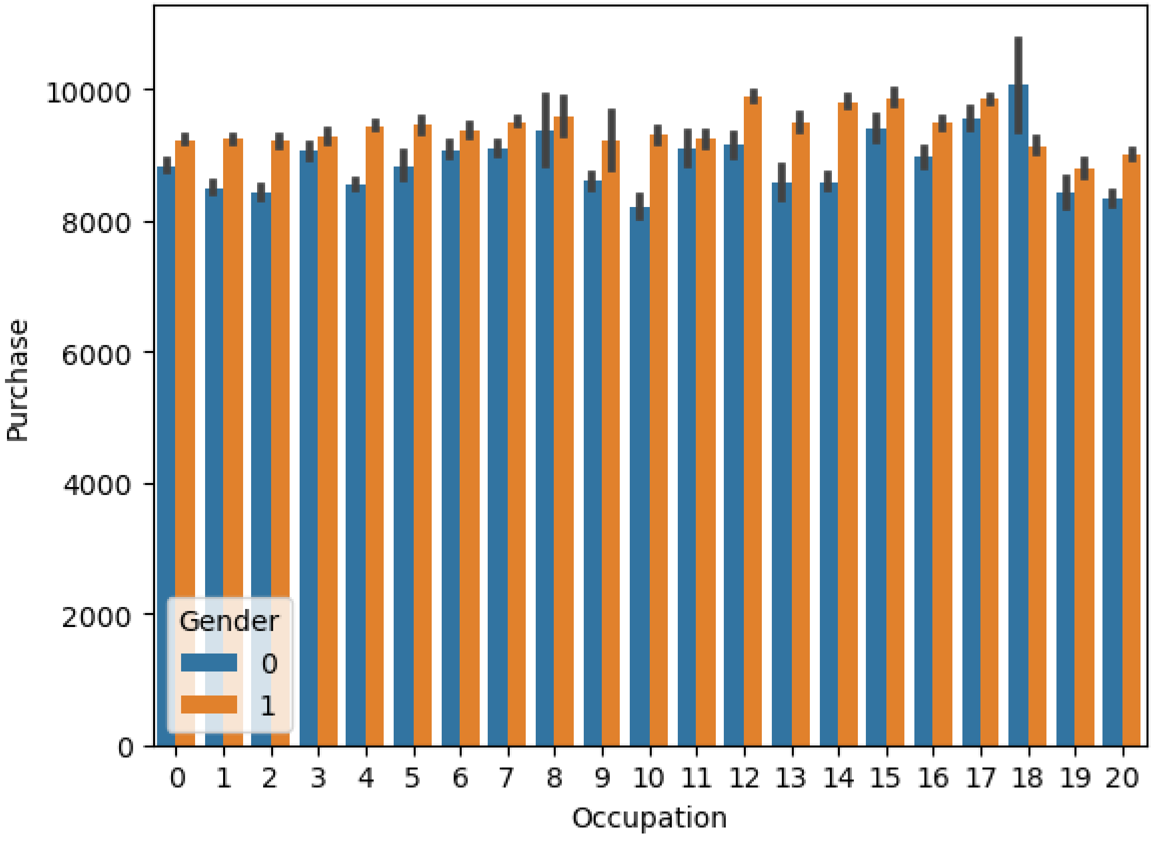
In [63]:

```
## Purchase with Occupation

sns.barplot(df,x='Occupation',y='Purchase',hue='Gender')
```

Out[63]:

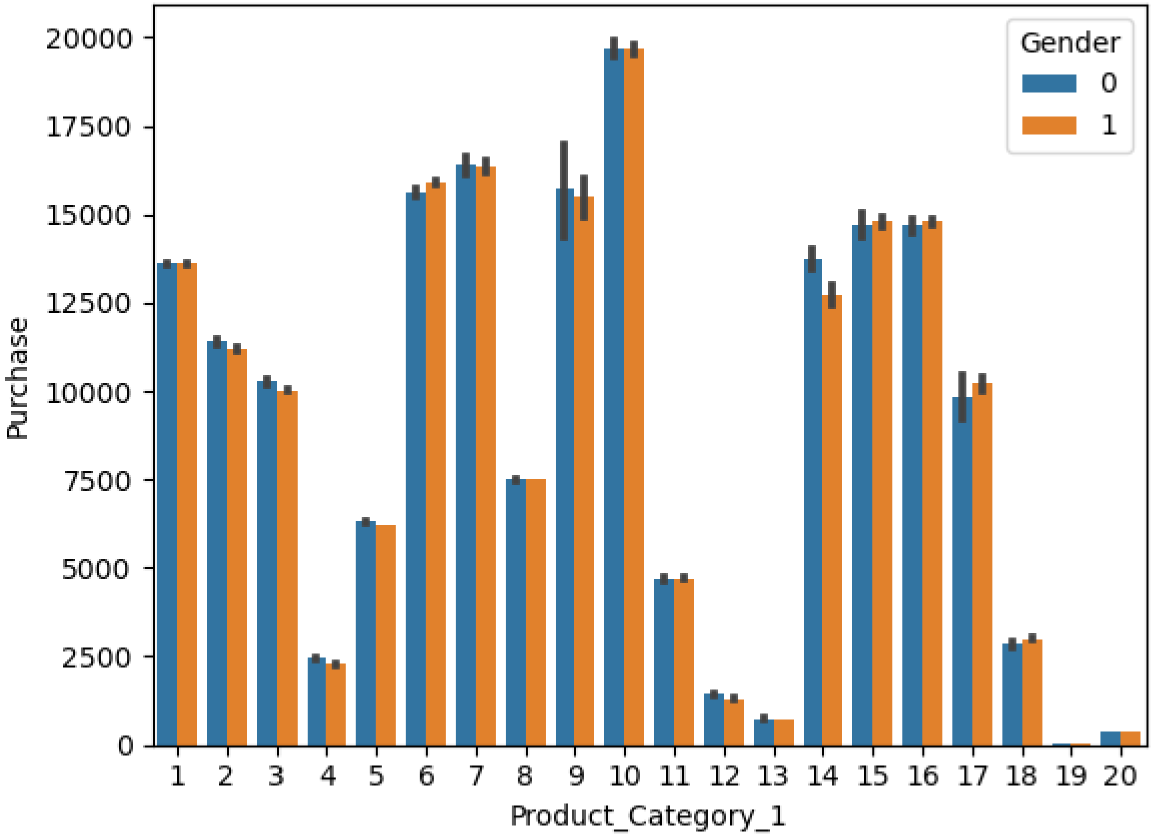
```
<Axes: xlabel='Occupation', ylabel='Purchase'>
```



In [64]:
```
## Product_Category_1 with Purchase

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

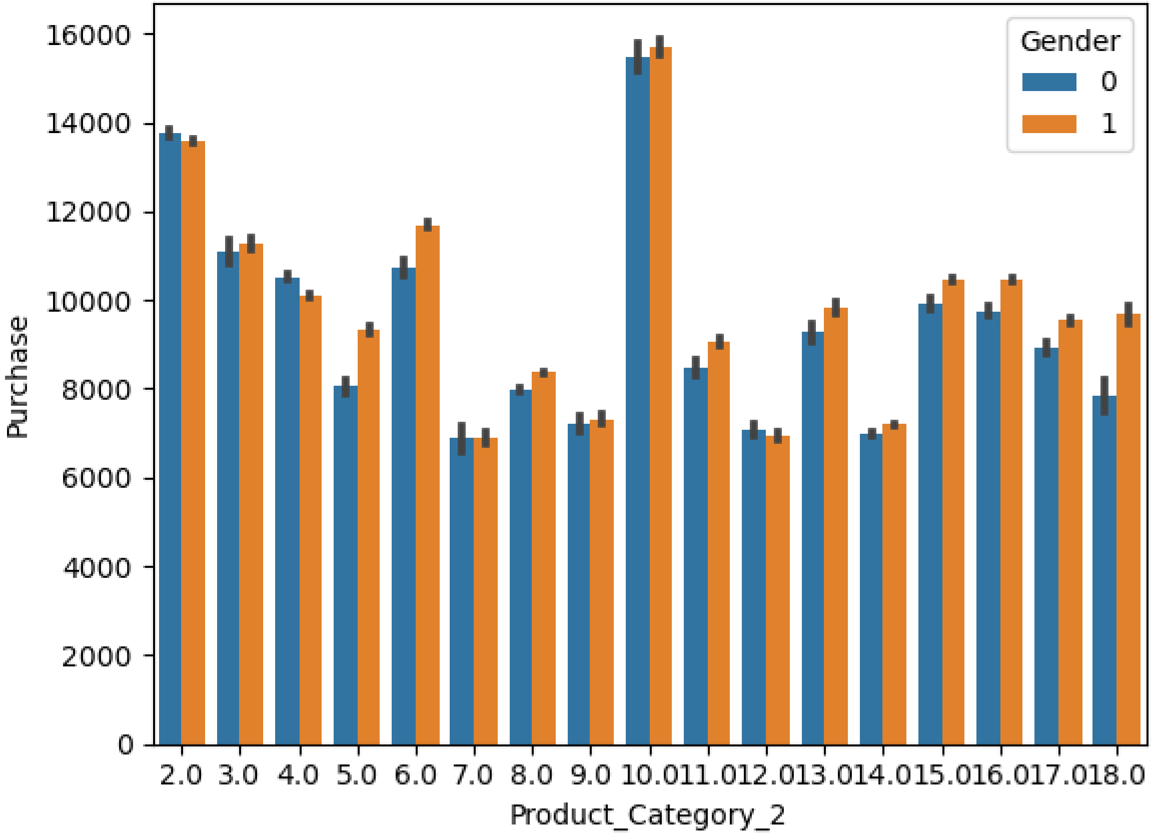
Out[64]:
```
<Axes: xlabel='Product_Category_1', ylabel='Purchase'>
```



In [65]:
```
## Product_Category_2 with Purchase

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

Out[65]:
```
<Axes: xlabel='Product_Category_2', ylabel='Purchase'>
```



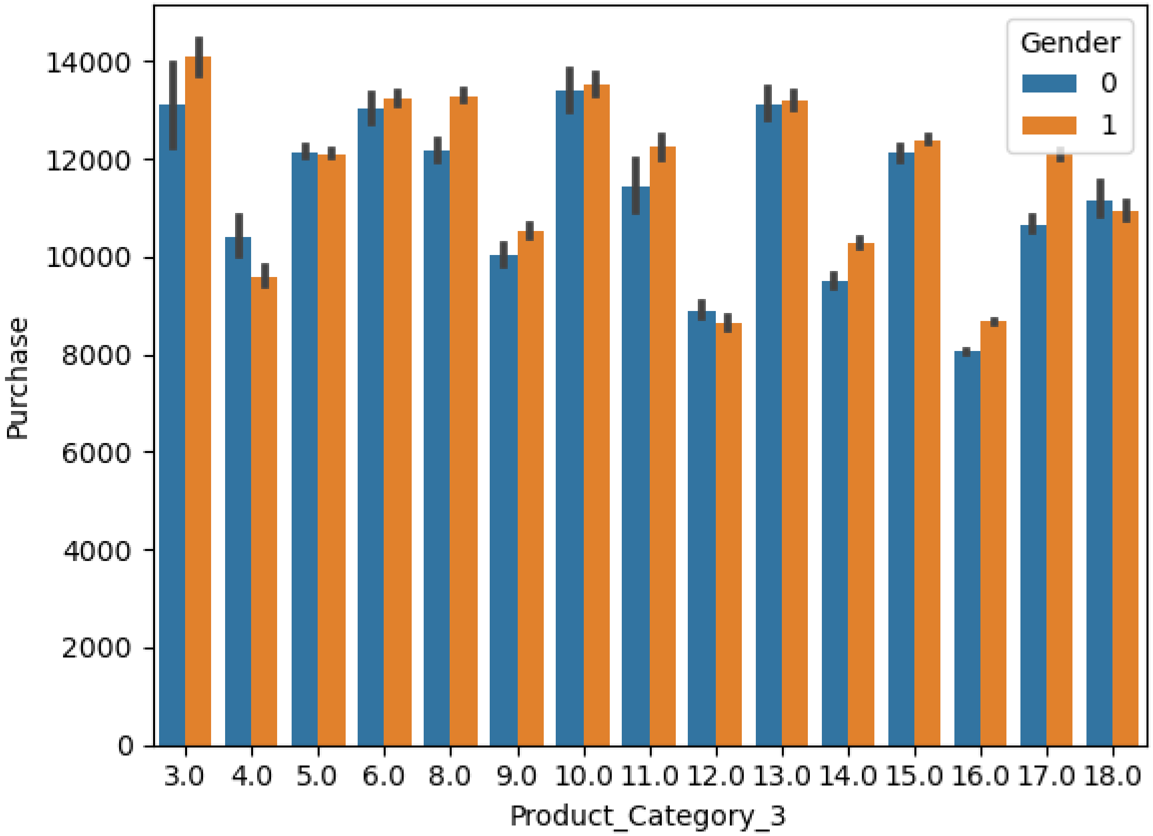
In [67]:

```
## Product_Category_3 with Purchase

sns.barplot(data=df,x='Product_Category_3',y='Purchase',hue='Gender')
```

Out[67]:

```
<Axes: xlabel='Product_Category_3', ylabel='Purchase'>
```



In [ ]:

```
## Purchase with null values

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

# FEATURE SCALING

In [ ]:

```
## Purchase without null values

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

In [ ]:

```
### feature scaling

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [ ]:

```
df
```

In [ ]:

In [ ]:

In [ ]: