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

In [2]:
```
df = pd.read_csv(r"C:\Users\asus\Desktop\Black Friday Sales.csv")          #loading data
df
```

Out[2]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Prod
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	
...	...	...	...	...	...	...	...	...	...	
550063	1006033	P00372445	M	51-55	13	B	1	1	20	
550064	1006035	P00375436	F	26-35	1	C	3	0	20	
550065	1006036	P00375436	F	26-35	15	B	4+	1	20	
550066	1006038	P00375436	F	55+	1	C	2	0	20	
550067	1006039	P00371644	F	46-50	0	B	4+	1	20	

550068 rows × 12 columns

In [3]:
```
df.head()  #to view top 5 rows of the data
```

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_C
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 [4]:
```
df.info()  # to check the datatype and null value count of the data
```

```
<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 [5]:
```
df.describe()  # to check the central tendency of the data
```

Out[5]:

	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 [6]:
```
df.columns  # to check the name of columns present in the data
```

Out[6]:
```
Index(['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'],
      dtype='object')
```

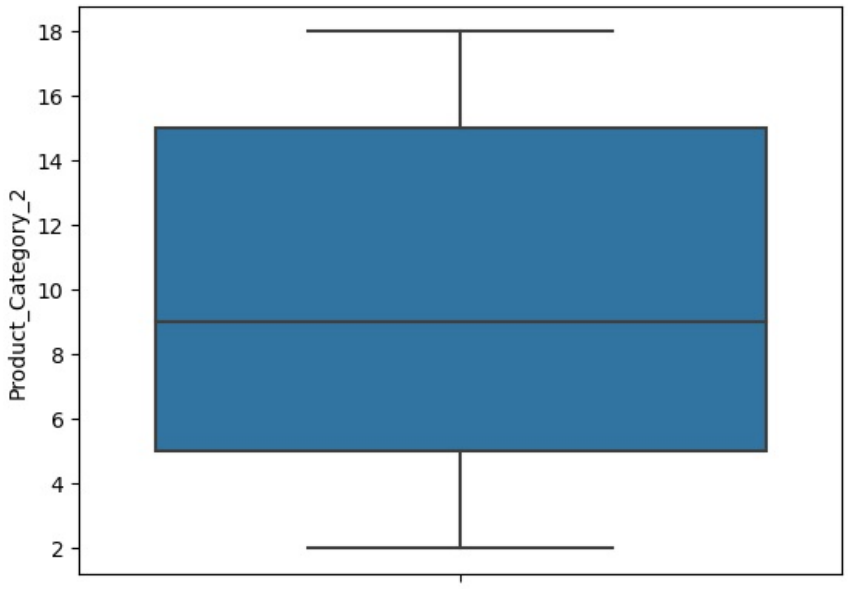
In [7]:
```
df1 = df.sample(50000)
```

In [8]:
```
df.isnull().sum()  # to check the total null value count in data
```

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]:
```
sns.boxplot(data = df,y = 'Product_Category_2')   # to check the outlier to fill the null value
```

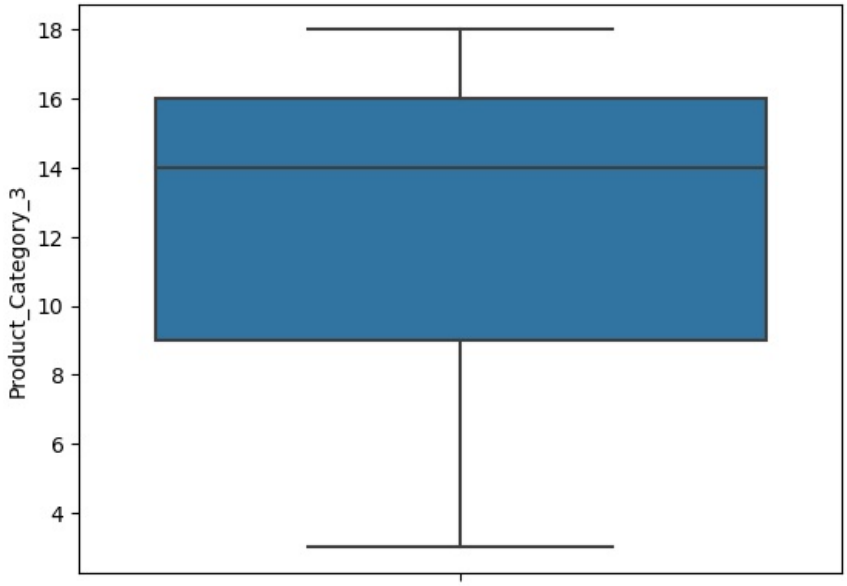
Out[9]:
```
<AxesSubplot:ylabel='Product_Category_2'>
```



In [10]:
```
df['Product_Category_2'] = df.Product_Category_2.fillna(df.Product_Category_2.median())
```

In [11]:
```
sns.boxplot(data = df,y = 'Product_Category_3')
```

Out[11]:
```
<AxesSubplot:ylabel='Product_Category_3'>
```



In [12]:
```
df['Product_Category_3'] = df.Product_Category_3.fillna(df.Product_Category_3.median())
```

In [13]:
```
df.isnull().sum()   # to check the null value count again after filling missing values
```

Out[13]:
```
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          0
Product_Category_3          0
Purchase                    0
dtype: int64
```

In [14]:
```
df['Product_Category_2']=df['Product_Category_2'].astype(int)      # to convert float datatype into int
df['Product_Category_3']=df['Product_Category_3'].astype(int)
df.dtypes
```

Out[14]:
```
User_ID                      int64
Product_ID                  object
Gender                      object
Age                         object
Occupation                   int64
City_Category               object
Stay_In_Current_City_Years  object
Marital_Status               int64
Product_Category_1           int64
Product_Category_2           int32
Product_Category_3           int32
Purchase                     int64
dtype: object
```

In [15]:
```
from sklearn.preprocessing import LabelEncoder                # convert categorical column into numerical
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df
```

Out[15]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Prod
0	1000001	P00069042	0	0-17	10	A	2	0	3	
1	1000001	P00248942	0	0-17	10	A	2	0	1	
2	1000001	P00087842	0	0-17	10	A	2	0	12	
3	1000001	P00085442	0	0-17	10	A	2	0	12	
4	1000002	P00285442	1	55+	16	C	4+	0	8	
...	...	...	...	...	...	...	...	...	...	
550063	1006033	P00372445	1	51-55	13	B	1	1	20	
550064	1006035	P00375436	0	26-35	1	C	3	0	20	
550065	1006036	P00375436	0	26-35	15	B	4+	1	20	
550066	1006038	P00375436	0	55+	1	C	2	0	20	
550067	1006039	P00371644	0	46-50	0	B	4+	1	20	

550068 rows × 12 columns

In [16]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['City_Category'] = le.fit_transform(df['City_Category'])
df
```

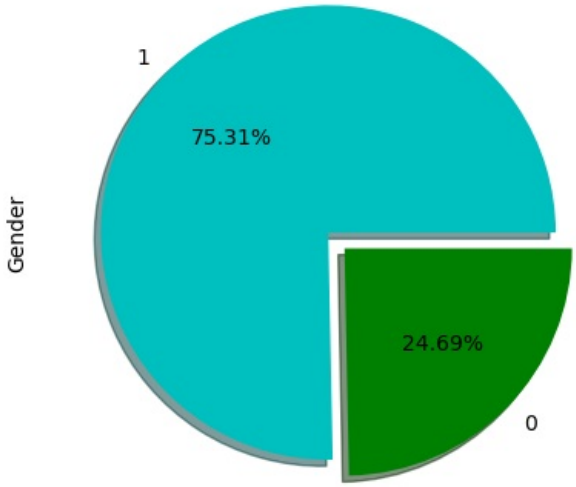
Out[16]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Prod
0	1000001	P00069042	0	0-17	10	0	2	0	3	
1	1000001	P00248942	0	0-17	10	0	2	0	1	
2	1000001	P00087842	0	0-17	10	0	2	0	12	
3	1000001	P00085442	0	0-17	10	0	2	0	12	
4	1000002	P00285442	1	55+	16	2	4+	0	8	
...	...	...	...	...	...	...	...	...	...	
550063	1006033	P00372445	1	51-55	13	1	1	1	20	
550064	1006035	P00375436	0	26-35	1	2	3	0	20	
550065	1006036	P00375436	0	26-35	15	1	4+	1	20	
550066	1006038	P00375436	0	55+	1	2	2	0	20	
550067	1006039	P00371644	0	46-50	0	1	4+	1	20	

550068 rows × 12 columns

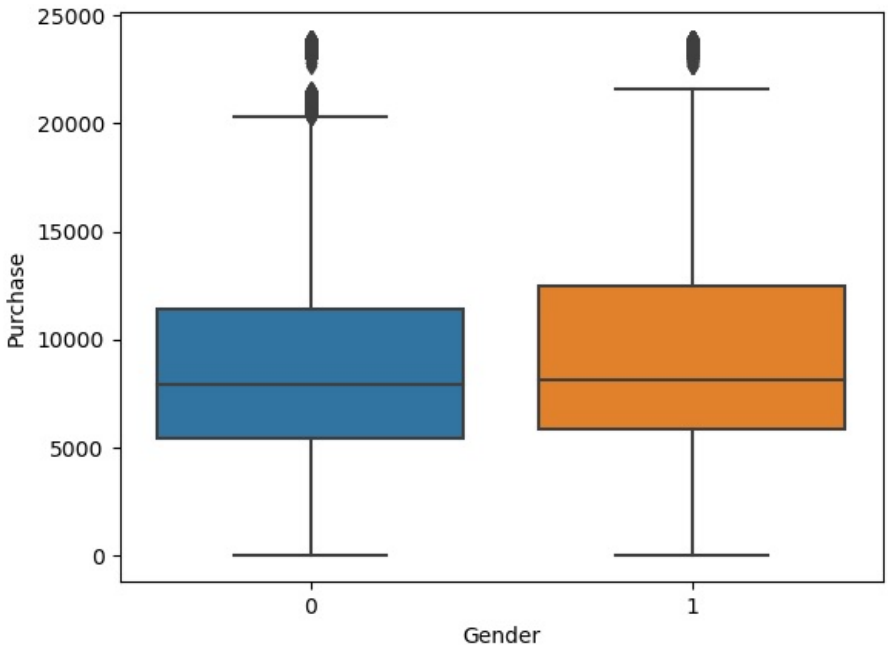
EDA

In [17]:
```
df['Gender'].value_counts().plot(kind='pie',explode=[0,0.1],autopct='%0.2f%%',colors=['c','g'],shadow=True)
plt.show()
```



%75 of customers are male and %25 of customers are female. On average, males spent more money.

In [18]:
```
sns.boxplot(x=df["Gender"], y=df["Purchase"])
plt.show()
```



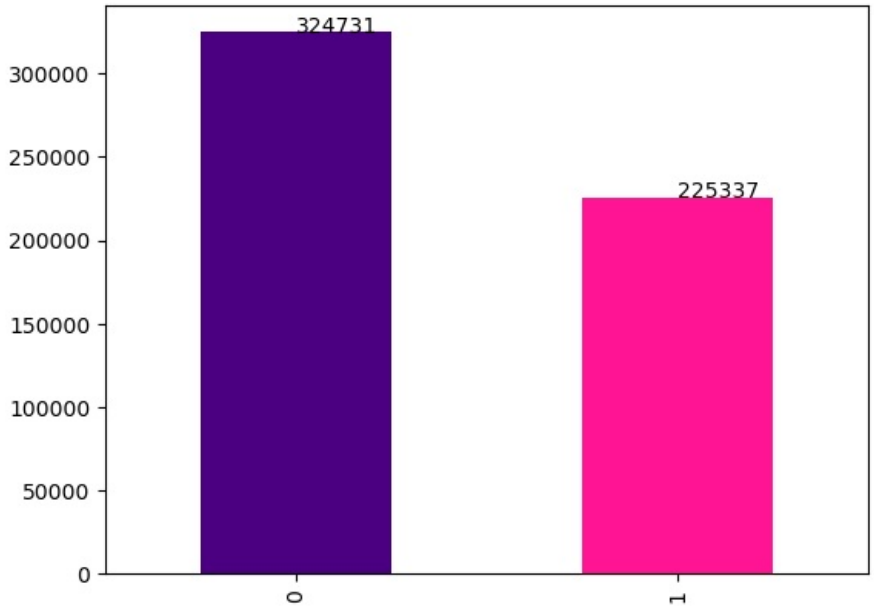
On average the male gender spends more money on purchase contrary to female

In [19]:
```
df['Marital_Status'].value_counts()
```

Out[19]:
```
0    324731
1    225337
Name: Marital_Status, dtype: int64
```

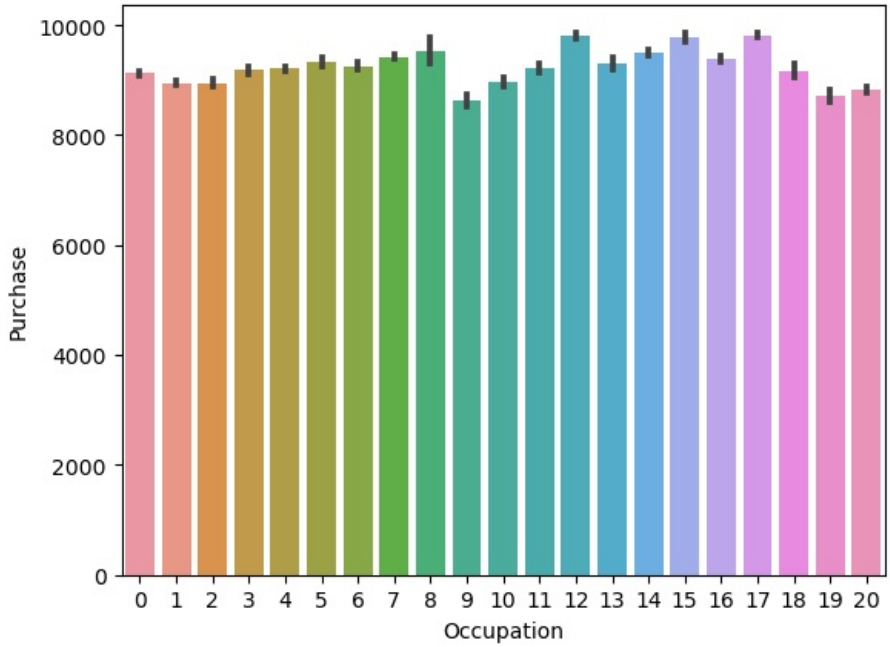
In [20]:
```
df['Marital_Status'].value_counts().plot(kind = 'bar',color=['indigo','deeppink'])
plt.text(0,324735,'324731',color = 'black')
plt.text(1,225340,'225337',color = 'black')
```

Out[20]:
```
Text(1, 225340, '225337')
```



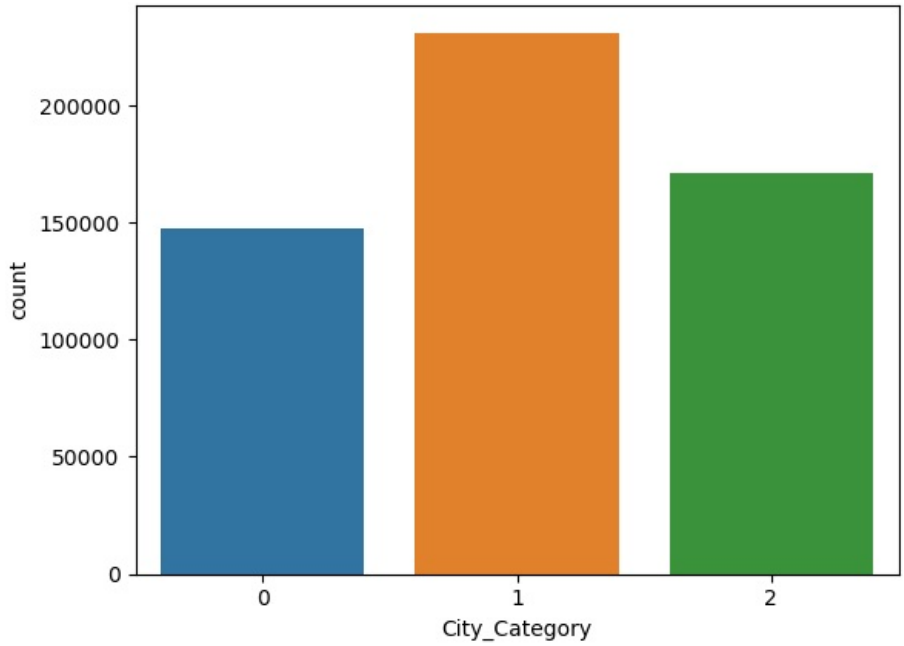
0 represents unmarried population and 1 represents maried population. This is interesting though unmarried people spend more on purchasing.

In [21]:
```
sns.barplot(x=df["Occupation"], y=df["Purchase"])
plt.show()
```



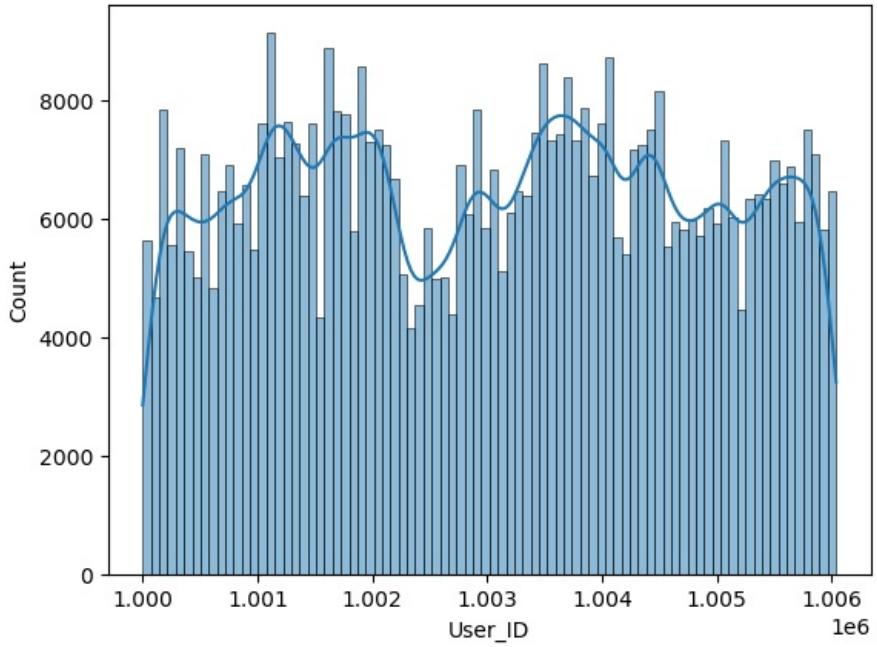
In [22]:
```
sns.countplot(df['City_Category'])
plt.show()
```

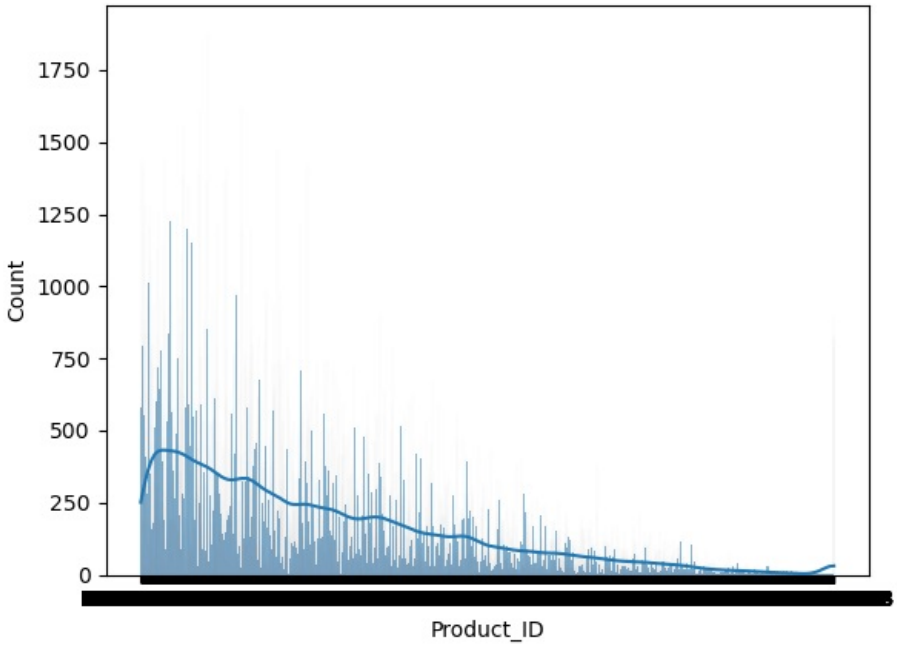
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

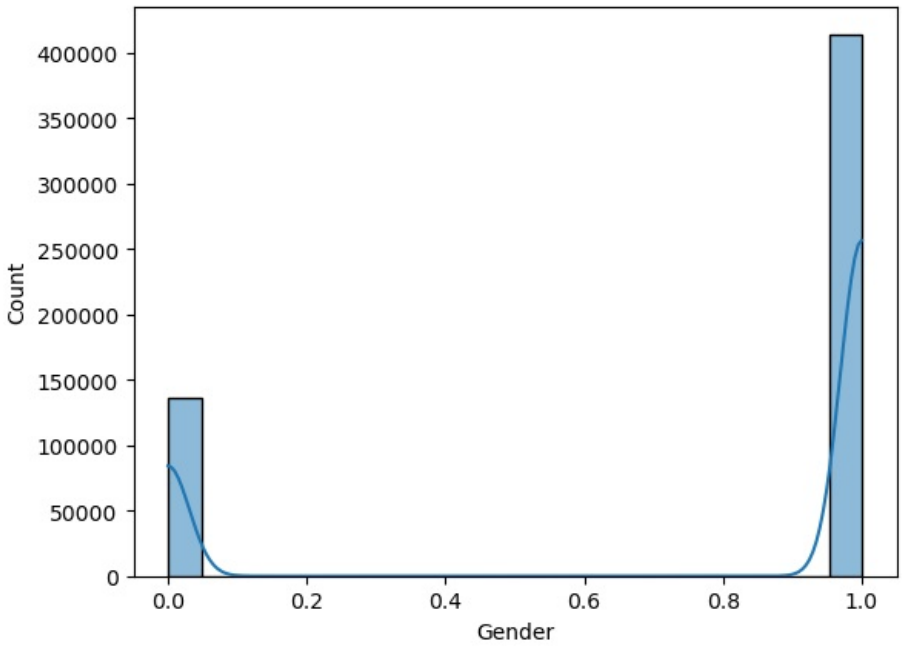


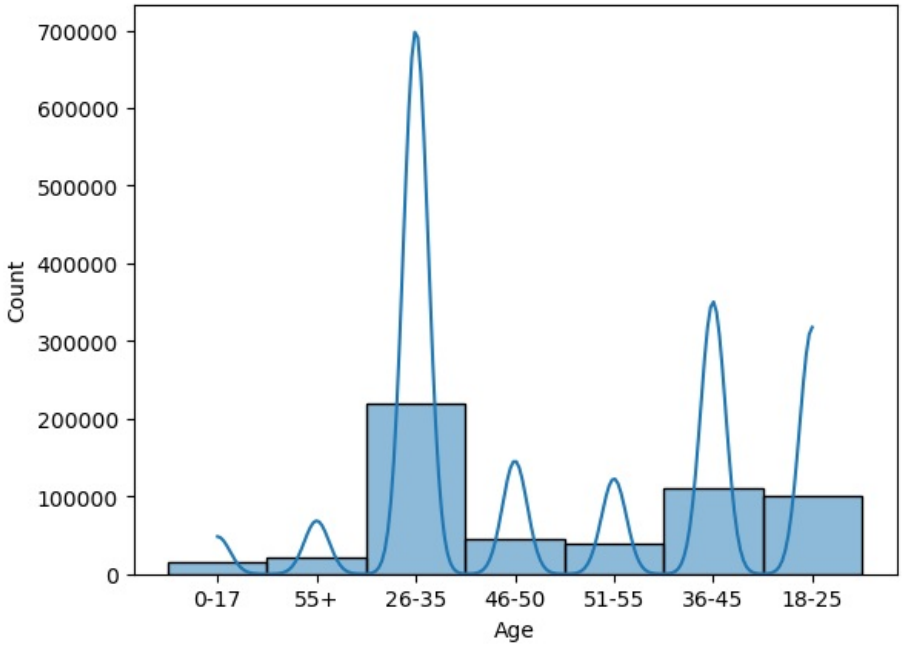
It is observed that city category B has made the most number of purchases.

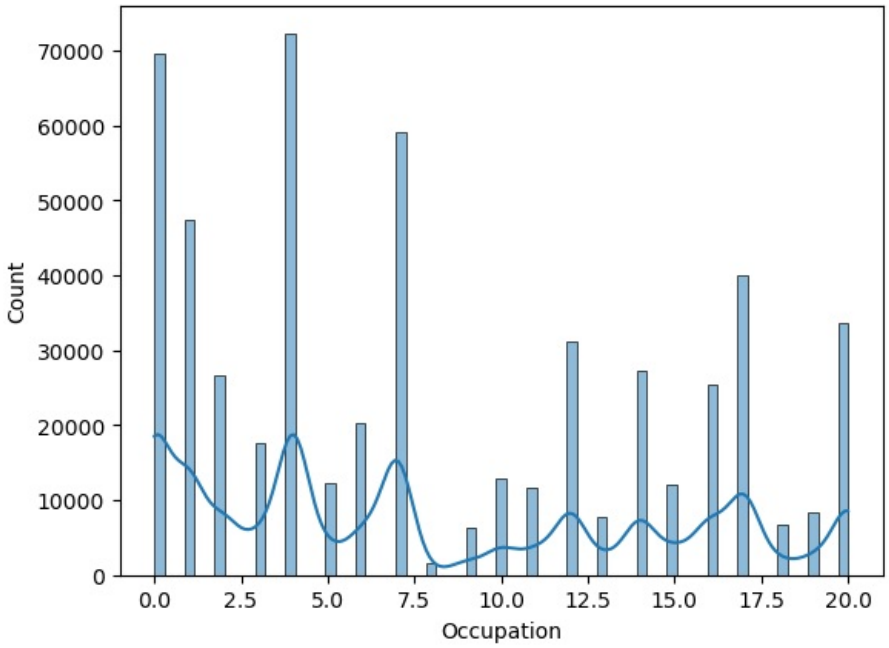
In [23]:
```
for i in df.columns:                                    # to check the skewness of the columns in data
    sns.histplot(data=df, x=i, kde=True)
    plt.show()
```

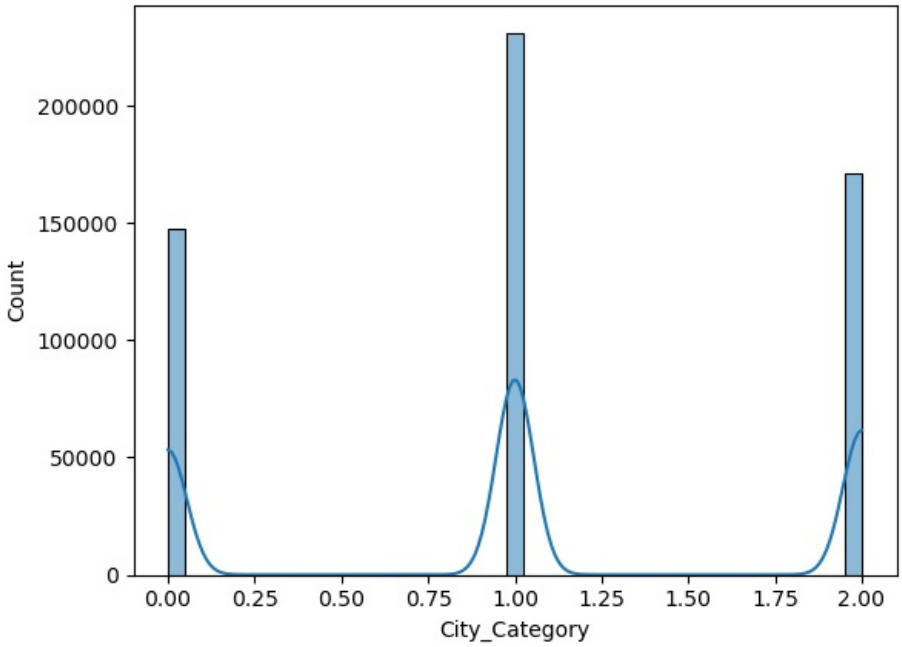


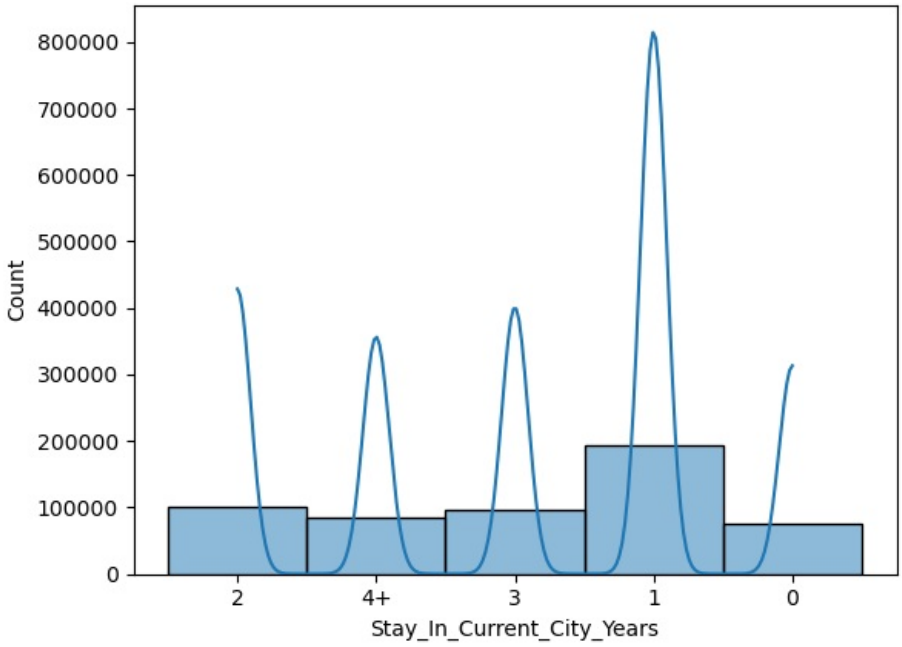

1750
1500
1250
1000
750
500
250
0
Count
Product_ID

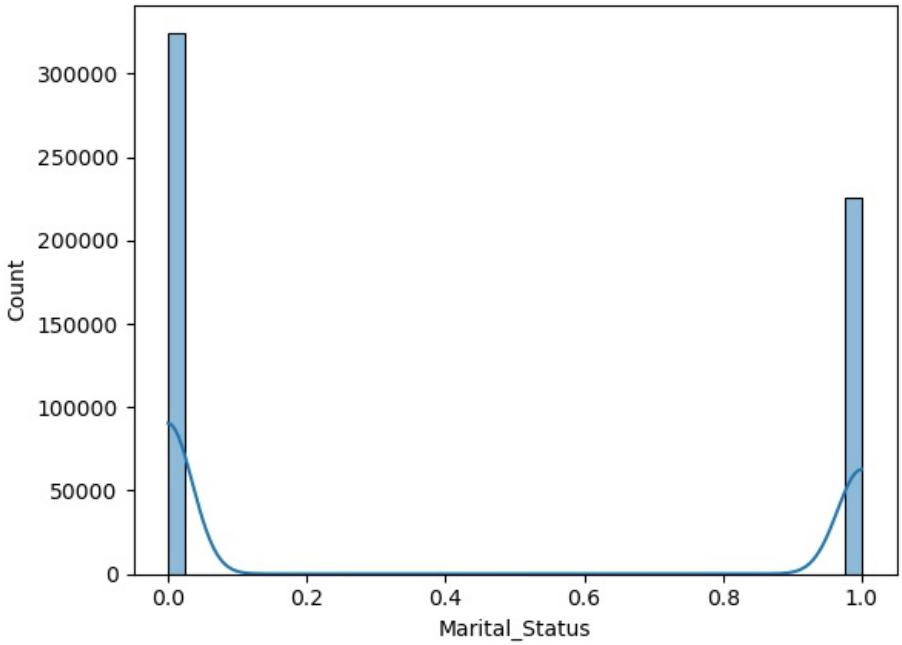

400000
350000
300000
250000
200000
150000
100000
50000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Gender

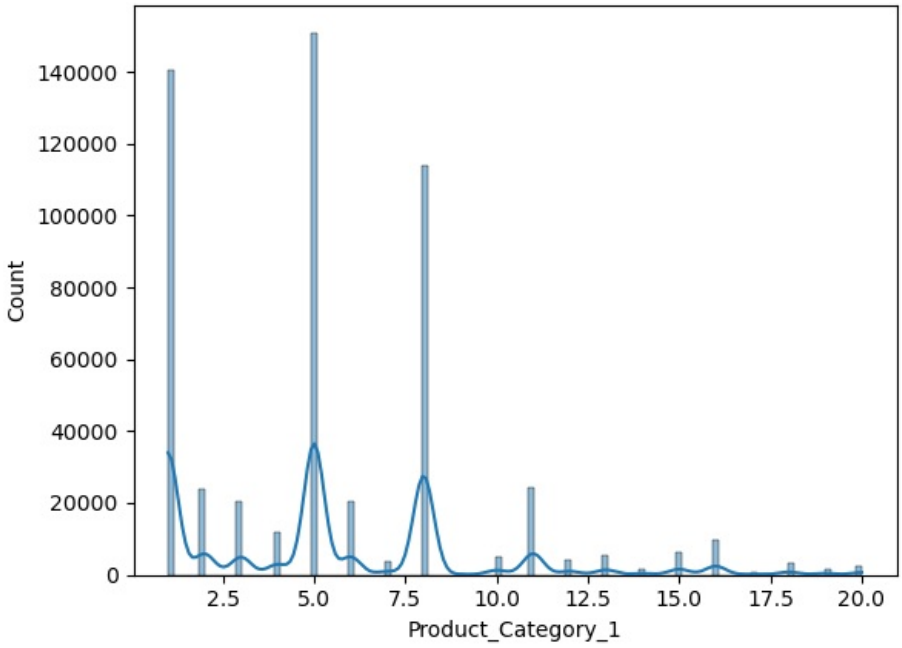

700000
600000
500000
400000
300000
200000
100000
0
Count
0-17
55+
26-35
46-50
51-55
36-45
18-25
Age

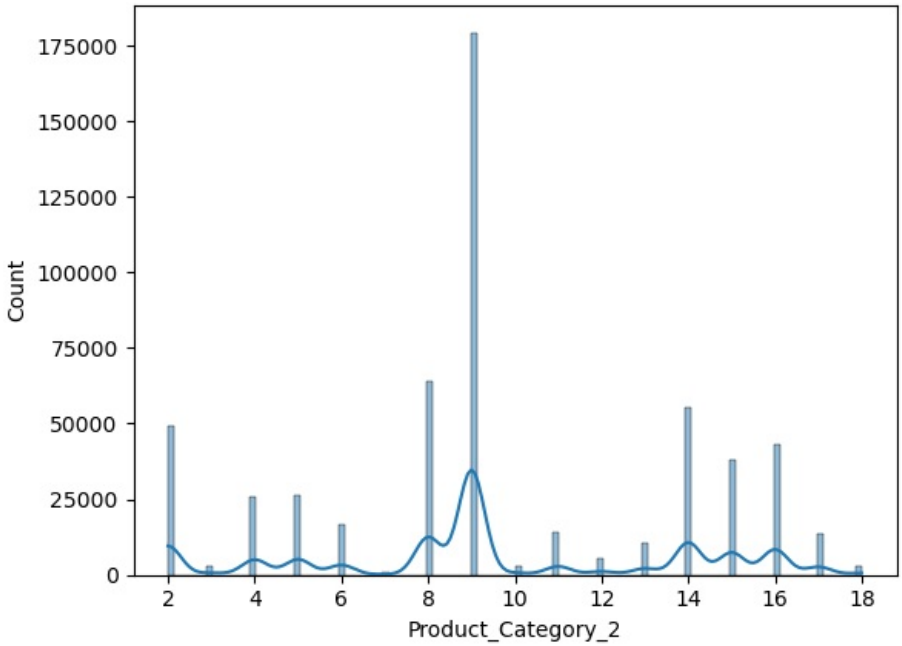

70000
60000
50000
40000
30000
20000
10000
0
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Occupation

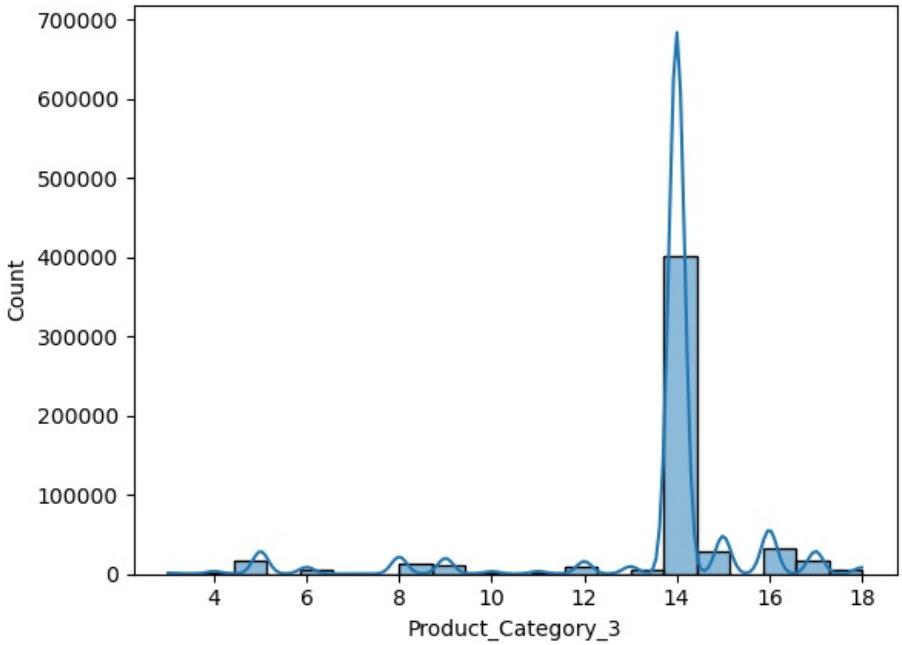

200000
150000
100000
50000
0
Count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
City_Category

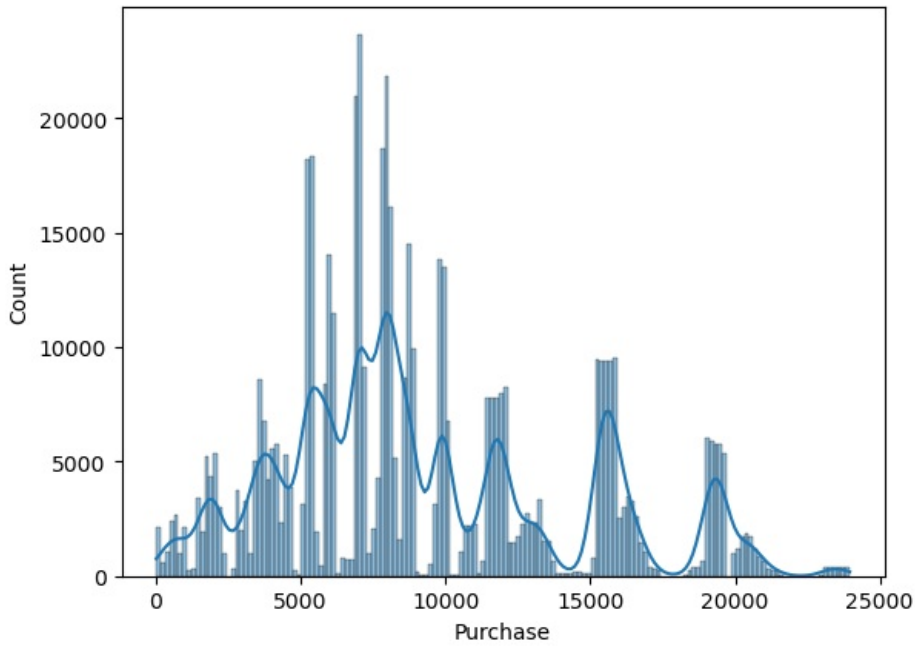

800000
700000
600000
500000
400000
300000
200000
100000
0
Count
2
4+
3
1
0
Stay_In_Current_City_Years


300000
250000
200000
150000
100000
50000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Marital_Status


140000
120000
100000
80000
60000
40000
20000
0
Count
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Product_Category_1

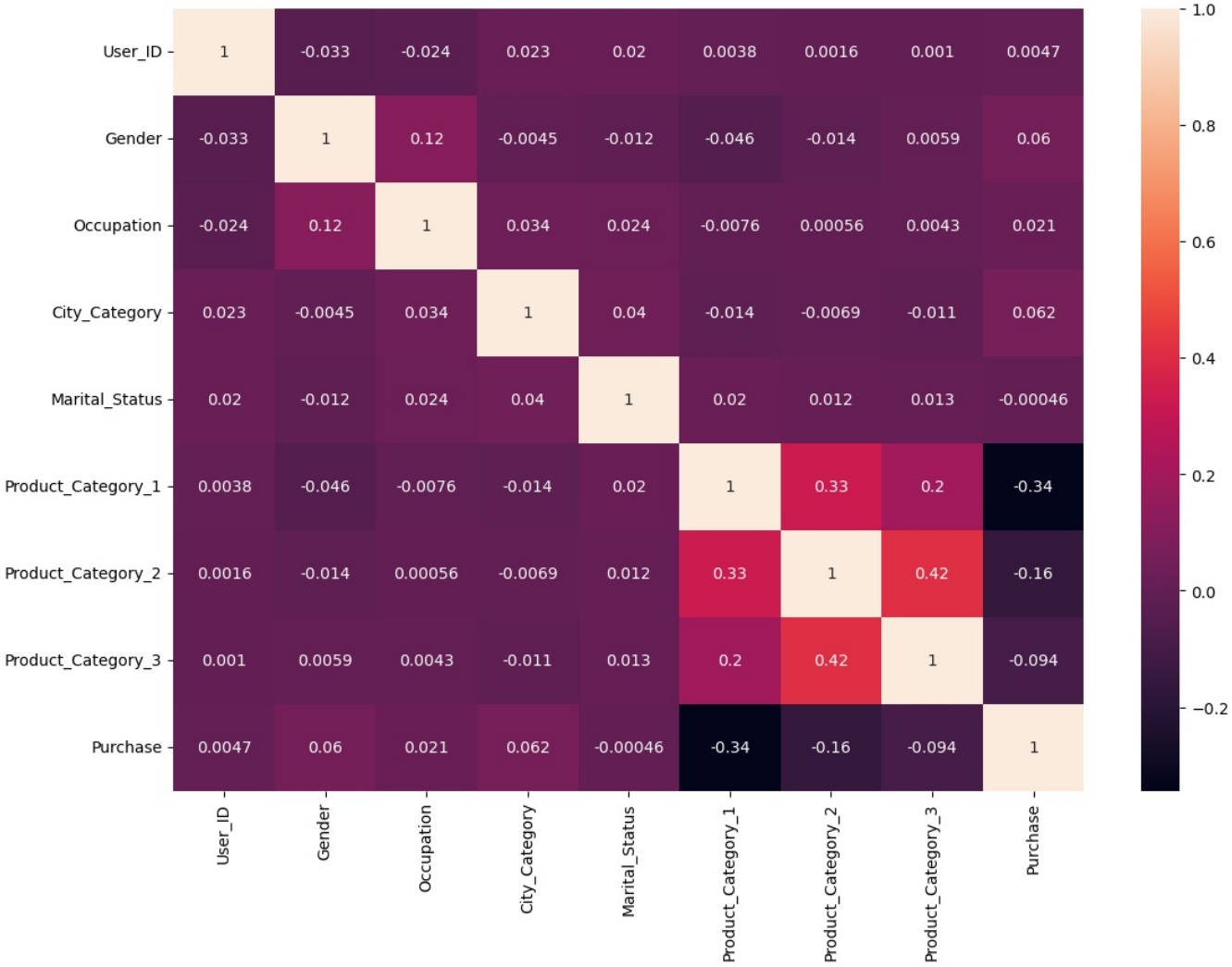

175000
150000
125000
100000
75000
50000
25000
0
Count
2
4
6
8
10
12
14
16
18
Product_Category_2


700000
600000
500000
400000
300000
200000
100000
0
Count
4
6
8
10
12
14
16
18
Product_Category_3



In [24]:
```
plt.figure(figsize = (13,9))                    #to check the correlation
sns.heatmap(df.corr(),annot = True)
```

Out[24]:
```
<AxesSubplot:>
```



In [25]:
```
df2 = df.drop(["Product_ID","Stay_In_Current_City_Years"],axis=1)        #dropping unwanted columns to reduce c
```

In [26]:
```
df2
```

Out[26]:

	User_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Pu
0	1000001	0	0-17	10	0	0	3	9	14	
1	1000001	0	0-17	10	0	0	1	6	14	
2	1000001	0	0-17	10	0	0	12	9	14	
3	1000001	0	0-17	10	0	0	12	14	14	
4	1000002	1	55+	16	2	0	8	9	14	
...	...	...	...	...	...	...	...	...	...	
550063	1006033	1	51-55	13	1	1	20	9	14	
550064	1006035	0	26-35	1	2	0	20	9	14	
550065	1006036	0	26-35	15	1	1	20	9	14	
550066	1006038	0	55+	1	2	0	20	9	14	
550067	1006039	0	46-50	0	1	1	20	9	14	

550068 rows × 10 columns

age {'0-17': 0, '18-25': 1, '26-35': 2, '36-45': 3, '46-50': 4, '51-55': 5, '55+': 6} df2["Age"] = df2["Age"].map(lambda x: int(x.split("-")[0]))

In [27]:
```
df2["Age"] = df2["Age"].map({'0-17': 0, '18-25': 1, '26-35': 2, '36-45': 3, '46-50': 4, '51-55': 5, '55+': 6} )
```

In [28]:
```
df2
```

Out[28]:

	User_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Pu
0	1000001	0	0	10	0	0	3	9	14	
1	1000001	0	0	10	0	0	1	6	14	
2	1000001	0	0	10	0	0	12	9	14	
3	1000001	0	0	10	0	0	12	14	14	
4	1000002	1	6	16	2	0	8	9	14	
...	...	...	...	...	...	...	...	...	...	
550063	1006033	1	5	13	1	1	20	9	14	
550064	1006035	0	2	1	2	0	20	9	14	
550065	1006036	0	2	15	1	1	20	9	14	
550066	1006038	0	6	1	2	0	20	9	14	
550067	1006039	0	4	0	1	1	20	9	14	

550068 rows × 10 columns

In [29]:
```
from sklearn.preprocessing import MinMaxScaler
numerical_columns = df2.select_dtypes(include=['int', 'float','object']).columns       # to normalize the data
min_max = MinMaxScaler()
df2[numerical_columns] = min_max.fit_transform(df2[numerical_columns])
df2
```

Out[29]:

	User_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3
0	0.000000	0.0	0.000000	0.50	0.0	0.0	0.105263	0.4375	0.733333
1	0.000000	0.0	0.000000	0.50	0.0	0.0	0.000000	0.2500	0.733333
2	0.000000	0.0	0.000000	0.50	0.0	0.0	0.578947	0.4375	0.733333
3	0.000000	0.0	0.000000	0.50	0.0	0.0	0.578947	0.7500	0.733333
4	0.000166	1.0	1.000000	0.80	1.0	0.0	0.368421	0.4375	0.733333
...	...	...	...	...	...	...	...	...	...
550063	0.998841	1.0	0.833333	0.65	0.5	1.0	1.000000	0.4375	0.733333
550064	0.999172	0.0	0.333333	0.05	1.0	0.0	1.000000	0.4375	0.733333
550065	0.999338	0.0	0.333333	0.75	0.5	1.0	1.000000	0.4375	0.733333
550066	0.999669	0.0	1.000000	0.05	1.0	0.0	1.000000	0.4375	0.733333
550067	0.999834	0.0	0.666667	0.00	0.5	1.0	1.000000	0.4375	0.733333

550068 rows × 10 columns

In [30]:
```
A = df2.drop(['Purchase'],axis = 1)        #to split the data
Y = df2['Purchase']
```

In [31]:
```
A
```

Out[31]:

	User_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3
0	0.000000	0.0	0.000000	0.50	0.0	0.0	0.105263	0.4375	0.733333
1	0.000000	0.0	0.000000	0.50	0.0	0.0	0.000000	0.2500	0.733333
2	0.000000	0.0	0.000000	0.50	0.0	0.0	0.578947	0.4375	0.733333
3	0.000000	0.0	0.000000	0.50	0.0	0.0	0.578947	0.7500	0.733333
4	0.000166	1.0	1.000000	0.80	1.0	0.0	0.368421	0.4375	0.733333
...	...	...	...	...	...	...	...	...	...
550063	0.998841	1.0	0.833333	0.65	0.5	1.0	1.000000	0.4375	0.733333
550064	0.999172	0.0	0.333333	0.05	1.0	0.0	1.000000	0.4375	0.733333
550065	0.999338	0.0	0.333333	0.75	0.5	1.0	1.000000	0.4375	0.733333
550066	0.999669	0.0	1.000000	0.05	1.0	0.0	1.000000	0.4375	0.733333
550067	0.999834	0.0	0.666667	0.00	0.5	1.0	1.000000	0.4375	0.733333

550068 rows × 9 columns

In [32]:
```
Y
```

Out[32]:
```
0         0.348992
1         0.634181
2         0.058875
3         0.043634
4         0.332248
              ...
550063    0.014865
550064    0.014990
550065    0.005219
550066    0.014740
550067    0.019959
Name: Purchase, Length: 550068, dtype: float64
```

In [33]:
```
A.shape
```

Out[33]:
```
(550068, 9)
```

In [34]:
```
Y.shape
```

Out[34]:
```
(550068,)
```

In [35]:
```
from sklearn.model_selection import train_test_split
train_A,test_A,train_Y,test_Y = train_test_split(A,Y,random_state = 42,test_size = 0.3)
```

In [36]:
```
print('The distribution of target variables in train data: ',train_Y.value_counts()/len(train_Y))
print('The distribution of target variables in test data: ',test_Y.value_counts()/len(test_Y))
```

```
The distribution of target variables in train data:  0.292914    0.000361
0.298718    0.000348
0.292246    0.000348
0.296380    0.000343
0.286734    0.000340
              ...
0.717775    0.000003
0.587081    0.000003
0.197921    0.000003
0.764207    0.000003
0.473047    0.000003
Name: Purchase, Length: 17382, dtype: float64
The distribution of target variables in test data:  0.295670    0.000400
0.299846    0.000394
0.292831    0.000388
0.296422    0.000382
0.298676    0.000382
              ...
0.437972    0.000006
0.872103    0.000006
0.723830    0.000006
0.783457    0.000006
0.613011    0.000006
Name: Purchase, Length: 15453, dtype: float64
```

Linear regression

In [37]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,r2_score
LR1 = LinearRegression()

LR1.fit(train_A, train_Y)
```

Out[37]:
```
LinearRegression()
```

In [38]:
```
pred1 = LR1.predict(test_A)
mae = mean_absolute_error(pred1,test_Y)
print('mean absolute error of test data:',mae)
```

```
mean absolute error of test data: 0.14987254681520512
```

In [39]:
```
pred2 = LR1.predict(train_A)
mae2 = mean_absolute_error(pred2,train_Y)
print('mean absolute error of train data:',mae2)
```

```
mean absolute error of train data: 0.14987659430977243
```

In [40]:
```
print(r2_score(pred1,test_Y))
```

```
-5.847533802165402
```

In [41]:
```
LR1.score(train_A,train_Y)
```

Out[41]:
```
0.12750132486957833
```

In [42]:
```
LR1.score(test_A,test_Y)
```

Out[42]:
```
0.12471944265418167
```

decision tree regression

In [43]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.metrics import mean_squared_error,r2_score
```

In [44]:
```
DTR = DecisionTreeRegressor()
```

In [45]:
```
DTR.fit(train_A,train_Y)
```

Out[45]:
```
DecisionTreeRegressor()
```

In [46]:
```
parameters = {
'max_depth':range(1,200),
'min_samples_leaf':range(1,300),
'max_leaf_nodes':range(1,50)
}
```

In [47]:
```
RSCV = RandomizedSearchCV(DTR,parameters,cv = 10,n_jobs = -1,verbose = 1)
```

In [48]:
```
RSCV.fit(train_A,train_Y)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:372: FitFailedWarning:

20 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
20 fits failed with the following error:
Traceback (most recent call last):
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 680, in _fit_a
nd_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 1315, in fit
    super().fit(
  File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 314, in fit
    raise ValueError(
ValueError: max_leaf_nodes 1 must be either None or larger than 1

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:969: UserWarning: One or more of
the test scores are non-finite: [0.65082218 0.64374883 0.64578806 0.63558274 0.64953865 0.6383433
        nan 0.6460112         nan 0.64894925]
  warnings.warn(
```

Out[48]:
```
RandomizedSearchCV(cv=10, estimator=DecisionTreeRegressor(), n_jobs=-1,
                   param_distributions={'max_depth': range(1, 200),
                                        'max_leaf_nodes': range(1, 50),
                                        'min_samples_leaf': range(1, 300)},
                   verbose=1)
```

In [49]:
```
RSCV.best_params_
```

Out[49]:
```
{'min_samples_leaf': 62, 'max_leaf_nodes': 48, 'max_depth': 141}
```

In [50]:
```
DTR1 = DecisionTreeRegressor(min_samples_leaf= 122, max_leaf_nodes = 46, max_depth = 89)
```

In [51]:
```
DTR1.fit(train_A,train_Y)
```

Out[51]:
```
DecisionTreeRegressor(max_depth=89, max_leaf_nodes=46, min_samples_leaf=122)
```

In [52]:
```
pred1 = DTR1.predict(test_A)
mse = mean_squared_error(pred1,test_Y)
print('mean squared error of test data is:',mse)
```

```
mean squared error of test data is: 0.015594815232106584
```

In [53]:
```
pred2 = DTR.predict(train_A)
mse2 = mean_squared_error(pred2,train_Y)
print('mean squared error of train data is:',mse)
```

```
mean squared error of train data is: 0.015594815232106584
```

In [54]:
```
RSCV.best_score_
```

Out[54]:
```
0.6508221829008229
```

In [55]:
```
print(r2_score(pred1,test_Y))
```

```
0.45443305840201553
```

In [56]:
```
DTR1.score(train_A,train_Y)
```

Out[56]:
```
0.6507455314646828
```

In [57]:
```
DTR1.score(test_A,test_Y)
```

Out[57]:
```
0.6450924387156025
```

randomforestregressor

In [58]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [59]:
```
RFR=RandomForestRegressor()
```

In [60]:
```
RFR.fit(train_A,train_Y)
```

Out[60]:
```
RandomForestRegressor()
```

In [61]:
```
parameters={
    'n_estimators':range(1,5),
    'max_depth':[2,4],
    'min_samples_leaf':[1,2],
    'max_leaf_nodes':[10,20]
```

```
}
```

In [62]:
```
RSCV = RandomizedSearchCV(RFR,parameters,cv = 10,n_jobs = -1,verbose = 1)
```

In [63]:
```
RSCV.fit(train_A,train_Y)
```

Out[63]:
```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
RandomizedSearchCV(cv=10, estimator=RandomForestRegressor(), n_jobs=-1,
                   param_distributions={'max_depth': [2, 4],
                                        'max_leaf_nodes': [10, 20],
                                        'min_samples_leaf': [1, 2],
                                        'n_estimators': range(1, 5)},
                   verbose=1)
```

In [64]:
```
RSCV.best_params_
```

Out[64]:
```
{'n_estimators': 2,
 'min_samples_leaf': 1,
 'max_leaf_nodes': 10,
 'max_depth': 4}
```

In [65]:
```
RFR1 = RandomForestRegressor(n_estimators = 3,min_samples_leaf = 2,max_leaf_nodes = 20,max_depth = 4)
```

In [66]:
```
RFR1.fit(train_A,train_Y)
```

Out[66]:
```
RandomForestRegressor(max_depth=4, max_leaf_nodes=20, min_samples_leaf=2,
                      n_estimators=3)
```

In [67]:
```
pred1 = RFR1.predict(test_A)
MAE = mean_squared_error(test_Y,pred1)
print('accuracy by mean squared error of test data:',MAE)
```

```
accuracy by mean squared error of test data: 0.02254767009905204
```

In [68]:
```
pred2 = RFR1.predict(train_A)
MAE2 = mean_squared_error(train_Y,pred2)
print('accuracy by mean squared error of train data:',MAE2)
```

```
accuracy by mean squared error of train data: 0.022328152752515453
```

In [69]:
```
print(r2_score(pred1,test_Y))
```

```
-0.10048416255113657
```

In [70]:
```
RFR1.score(train_A,train_Y)
```

Out[70]:
```
0.49268449830498606
```

In [71]:
```
RFR1.score(test_A,test_Y)
```

Out[71]:
```
0.4868590304921031
```

KNeighbourRegressor

In [72]:
```
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.neighbors import KNeighborsRegressor
```

In [73]:
```
knn = KNeighborsRegressor()
```

In [74]:
```
knn.fit(train_A,train_Y)
```

Out[74]:
```
KNeighborsRegressor()
```

In [75]:
```
param = {
'n_neighbors':(2,50,3),
'metric':['minkwoski','euclidean','manhanttan','haming'],
'algorithm':['auto','ball_tree','kd_tree','brute']
}
```

In [76]:
```
GSCV = GridSearchCV(knn,param,cv = 10,verbose = 1,n_jobs = -1)
```

In [ ]:
```
GSCV.fit(train_A,train_Y)
```

```
Fitting 10 folds for each of 48 candidates, totalling 480 fits
```

In [ ]:
```
GSCV.best_params_
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: