ML Project-Linear Regression

cleaning and preparing the data for model training

In [87]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [88]:
```
# importing dataset
df_train=pd.read_csv('train.csv')
df_train.head()
```

Out[88]:

	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 [89]:
```
df_train.shape
```

Out[89]:
```
(550068, 12)
```

In [90]:
```
# import test data
df_test=pd.read_csv('test.csv')
df_test
# here no target
```

Out[90]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Ma
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+	
3	1000010	P00145342	F	36-45	1	B	4+	
4	1000011	P00053842	F	26-35	1	C	1	
...	...	...	...	...	...	...	...	
233594	1006036	P00118942	F	26-35	15	B	4+	
233595	1006036	P00254642	F	26-35	15	B	4+	
233596	1006036	P00031842	F	26-35	15	B	4+	
233597	1006037	P00124742	F	46-50	1	C	4+	
233598	1006039	P00316642	F	46-50	0	B	4+	

233599 rows × 11 columns

In [91]:

```
# how do we combine these 2 datasets?
# we use append here
pd.merge(df_train,df_test,)
df=df_train.append(df_test)
df
```

```
C:\Users\smm\AppData\Local\Temp\ipykernel_16868\1959080800.py:4: FutureWarning: The f
rame.append method is deprecated and will be removed from pandas in a future version.
Use pandas.concat instead.
  df=df_train.append(df_test)
```

Out[91]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	M
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+	
...	...	...	...	...	...	...	...	
233594	1006036	P00118942	F	26-35	15	B	4+	
233595	1006036	P00254642	F	26-35	15	B	4+	
233596	1006036	P00031842	F	26-35	15	B	4+	
233597	1006037	P00124742	F	46-50	1	C	4+	
233598	1006039	P00316642	F	46-50	0	B	4+	

783667 rows × 12 columns

In [92]:
```
df_train.shape[0] + df_test.shape[0]
```

Out[92]:
```
783667
```

In [93]:
```
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 [94]:
```
df.describe()
```

Out[94]:

	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 [95]:
```
# we are dropping the column User_ID since its not that useful
df.drop(['User_ID'],axis=1,inplace=True)
```

In [96]:
```
df.head()
```

Out[96]:

	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	
3	P00085442	F	0-17	10	A	2	0	
4	P00285442	M	55+	16	C	4+	0	

Dealing categorical features in the dataset

Gender

In [97]:
```
# use map function
df['Gender']=df['Gender'].map({'F':0,"M":1})
df.head(5)
```

Out[97]:

	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	

Age

In [98]:
```
df["Age"].unique()
```

Out[98]:
```
array(['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25'],
      dtype=object)
```

use odinal encoding here.also can use labelEncoding(but here we combined both test and train data)

In [99]:
```
df["Age"]=df["Age"].map({'0-17':1,'18-25':2,'26-35':3,'36-45':4,'46-50':5,'51-55':6,'5
df.head(5)
```

Out[99]:

	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	

City_Category

In [100…
```
df['City_Category'].unique()
```

Out[100]:
```
array(['A', 'C', 'B'], dtype=object)
```

In [101…
```
df['City_Category']=df['City_Category'].map({'A':1, 'C':2, 'B':3})
df.head(5)
```

Out[101]:

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

missing values

In [102…

```
df.isnull().sum()
```

Out[102]:

```
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           245982
Product_Category_3           545809
Purchase                     233599
dtype: int64
```

Lots of missing values in product_cateogories. the null values in purchase is because we combined test data with train data

In [103…

```
## focus on replacing missing values
# on Product_Category_2 , Product_Category_3
```

on Product_Category_2

In [104…

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

Out[104]:

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

this is discrete feature and . here it contains a null values.

In [105…

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

Out[105]:
```
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 [106…
```
# replace mising value with MODE here
```

In [107…
```
df['Product_Category_2'].mode()
```

Out[107]:
```
0    8.0
Name: Product_Category_2, dtype: float64
```

In [108…
```
df['Product_Category_2'].mode()[0]
```

Out[108]:
```
8.0
```

In [109…
```
df['Product_Category_2']=df['Product_Category_2'].fillna(df['Product_Category_2'].mode
```

In [110…
```
df['Product_Category_2'].unique()
```

Out[110]:
```
array([ 8.,  6., 14.,  2., 15., 16., 11.,  5.,  3.,  4., 12.,  9., 10.,
       17., 13.,  7., 18.])
```

In [111…
```
df['Product_Category_2'].isnull().sum()
```

Out[111]:
```
0
```

on Product_Category_3

In [112…
```
df['Product_Category_3'].unique()
```

Out[112]:
```
array([nan, 14., 17.,  5.,  4., 16., 15.,  8.,  9., 13.,  6., 12.,  3.,
       18., 11., 10.])
```

In [113…
```
df['Product_Category_3'].value_counts()
```

Out[113]:
```
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 [114…
```
df['Product_Category_3'].mode()
```

Out[114]:
```
0    16.0
Name: Product_Category_3, dtype: float64
```

In [115…
```
df['Product_Category_3'].mode()[0]
```

Out[115]:
```
16.0
```

In [116…
```
df['Product_Category_3']=df['Product_Category_3'].fillna(df['Product_Category_3'].mode
```

In [117…
```
df['Product_Category_3'].unique()
```

Out[117]:
```
array([16., 14., 17.,  5.,  4., 15.,  8.,  9., 13.,  6., 12.,  3., 18.,
       11., 10.])
```

In [118…
```
df['Product_Category_3'].isnull().sum()
```

Out[118]:
```
0
```

In [119…
```
df.shape
```

Out[119]:
```
(783667, 11)
```

In [120…
```
df['Stay_In_Current_City_Years'].unique()
```

Out[120]:
```
array(['2', '4+', '3', '1', '0'], dtype=object)
```

In [121…
```
df['Stay_In_Current_City_Years']=df['Stay_In_Current_City_Years'].str.replace('+','')
```

```
C:\Users\smm\AppData\Local\Temp\ipykernel_16868\2063355665.py:1: FutureWarning: The d
efault 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 [122…
```
df.head()
```

Out[122]:

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

In [123…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 11 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   City_Category               783667 non-null  int64
 5   Stay_In_Current_City_Years  783667 non-null  object
 6   Marital_Status              783667 non-null  int64
 7   Product_Category_1          783667 non-null  int64
 8   Product_Category_2          783667 non-null  float64
 9   Product_Category_3          783667 non-null  float64
 10  Purchase                    550068 non-null  float64
dtypes: float64(3), int64(6), object(2)
memory usage: 71.7+ MB
```

Stay_In_Current_City_Years is still an integer, so convert that

In [124…

```
## convert object into integers

df['Stay_In_Current_City_Years']=df['Stay_In_Current_City_Years'].astype(int)
```

In [125…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 783667 entries, 0 to 233598
Data columns (total 11 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   City_Category               783667 non-null  int64  
 5   Stay_In_Current_City_Years  783667 non-null  int32  
 6   Marital_Status              783667 non-null  int64  
 7   Product_Category_1          783667 non-null  int64  
 8   Product_Category_2          783667 non-null  float64
 9   Product_Category_3          783667 non-null  float64
 10  Purchase                    550068 non-null  float64
dtypes: float64(3), int32(1), int64(6), object(1)
memory usage: 68.8+ MB
```

Visualization

In [126…

```
# sns.pairplot(df)
```

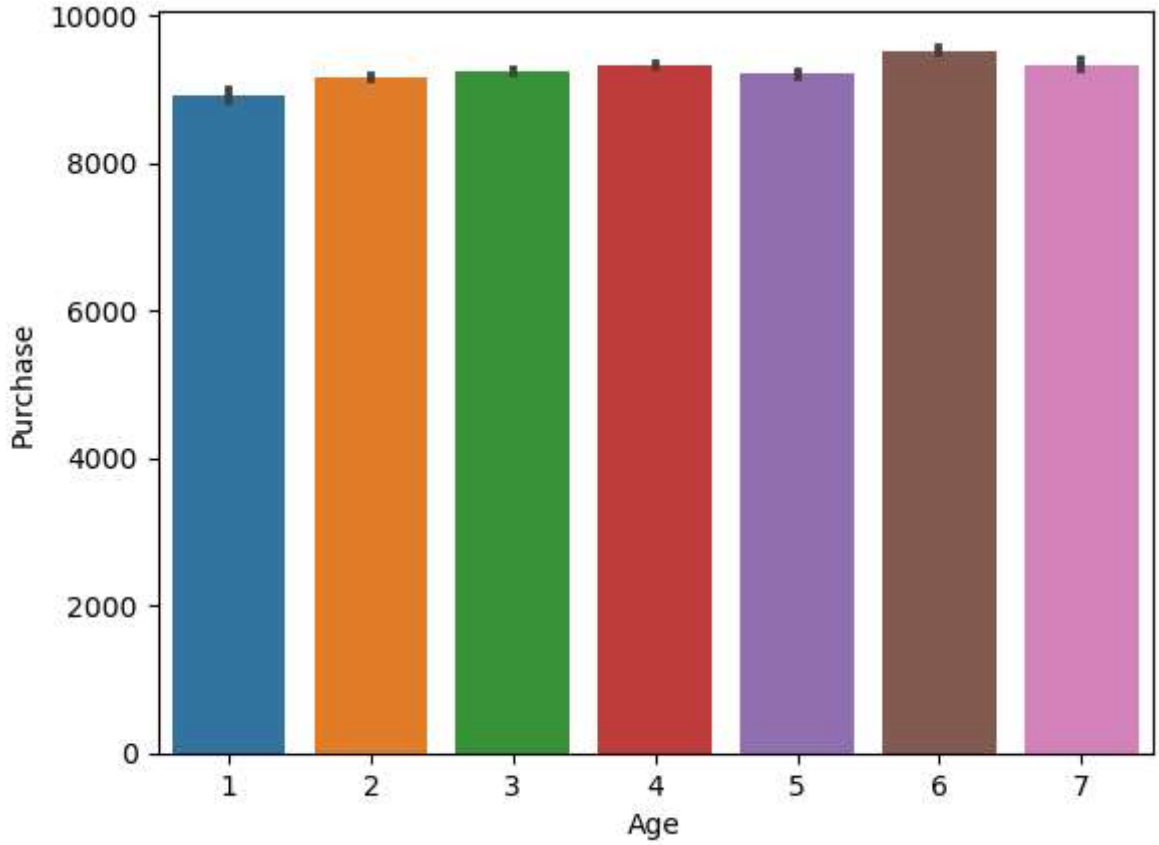
In [127…

```
sns.barplot('Age','Purchase',data=df)
```

```
C:\Users\smm\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pa
ss the following variables as keyword args: x, y. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[127]:

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



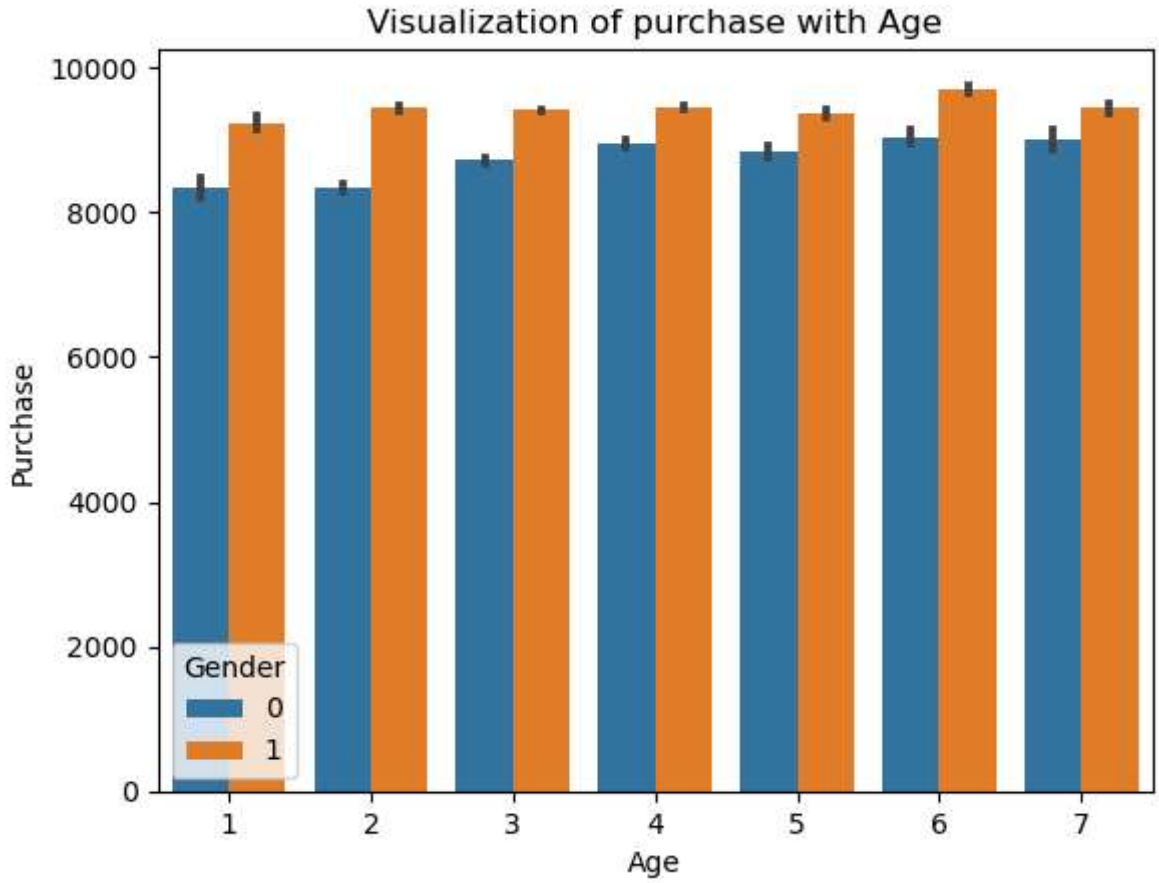
In [128…

```
sns.barplot('Age','Purchase',data=df,hue='Gender')
plt.title('Visualization of purchase with Age')
```

```
C:\Users\smm\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pa
ss the following variables as keyword args: x, y. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[128]:

```
Text(0.5, 1.0, 'Visualization of purchase with Age')
```



The maximum amount of purchase is done by Male customers.

Purchasing range of each age is almost equal.

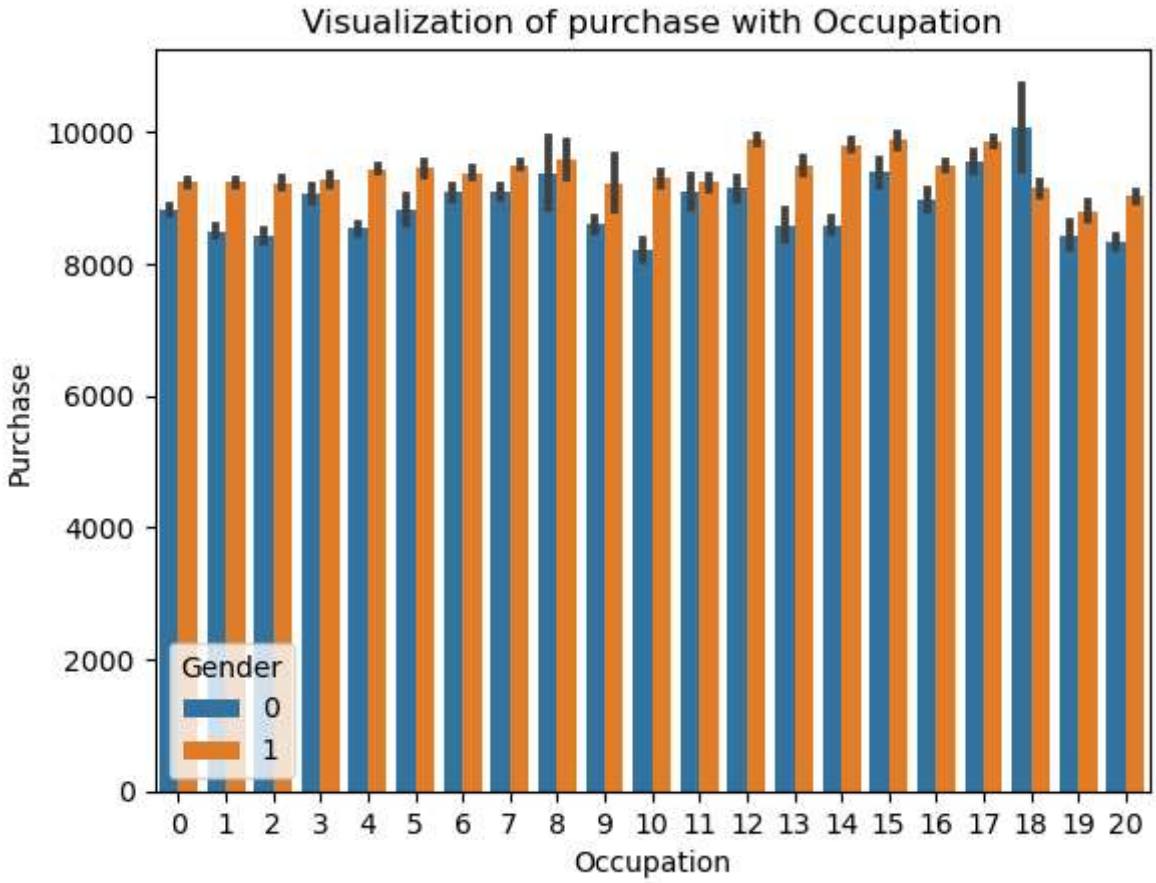
In [129…

```
sns.barplot('Occupation','Purchase',data=df,hue='Gender')
plt.title('Visualization of purchase with Occupation')
```

```
C:\Users\smm\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pa
ss the following variables as keyword args: x, y. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[129]:

```
Text(0.5, 1.0, 'Visualization of purchase with Occupation')
```



Almost every occupation range has equal purchasing rate.

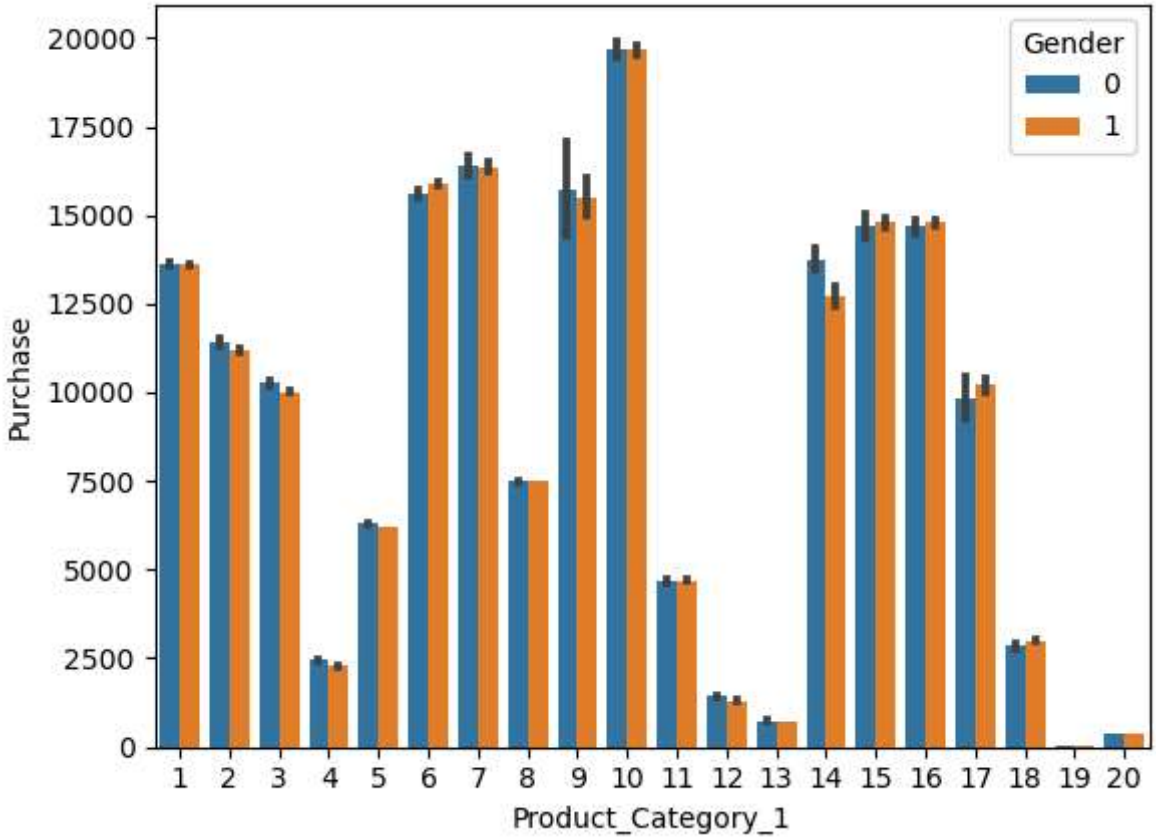
In [130…

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

```
C:\Users\smm\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pa
ss the following variables as keyword args: x, y. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[130]:

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



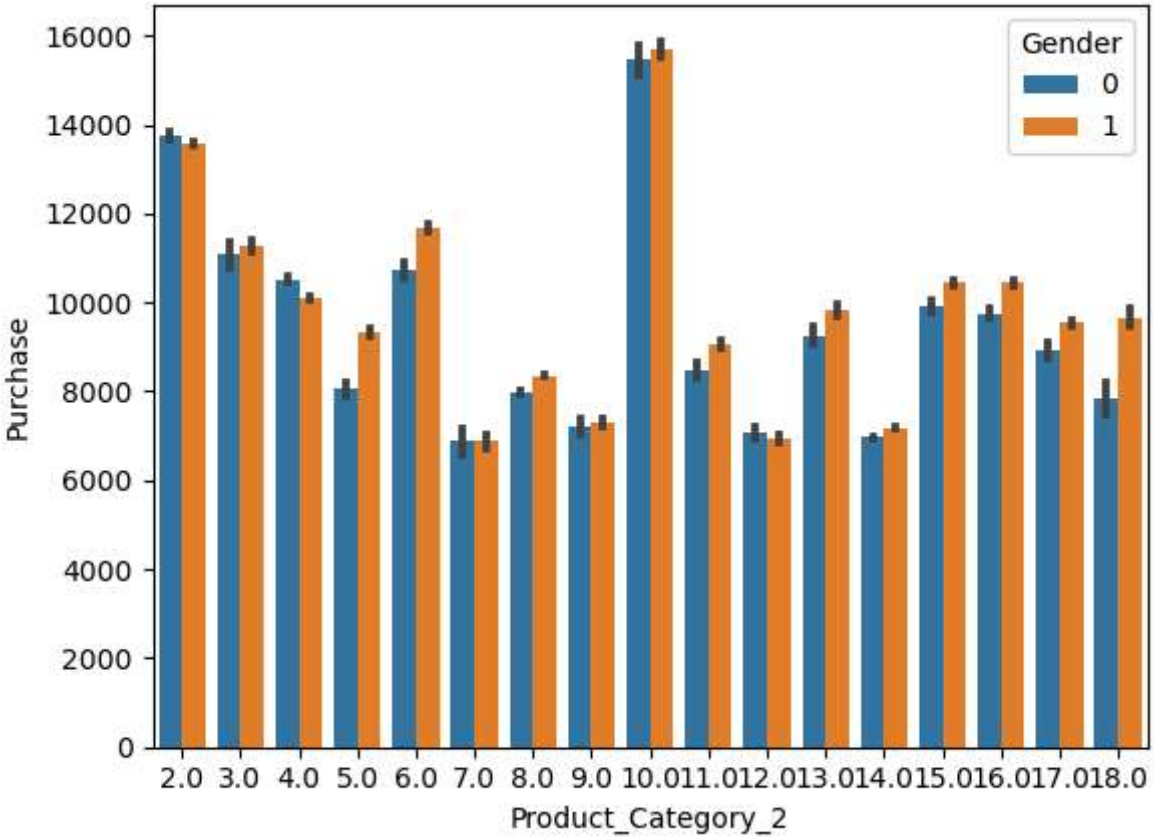
In [131…

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

```
C:\Users\smm\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pa
ss the following variables as keyword args: x, y. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[131]:

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



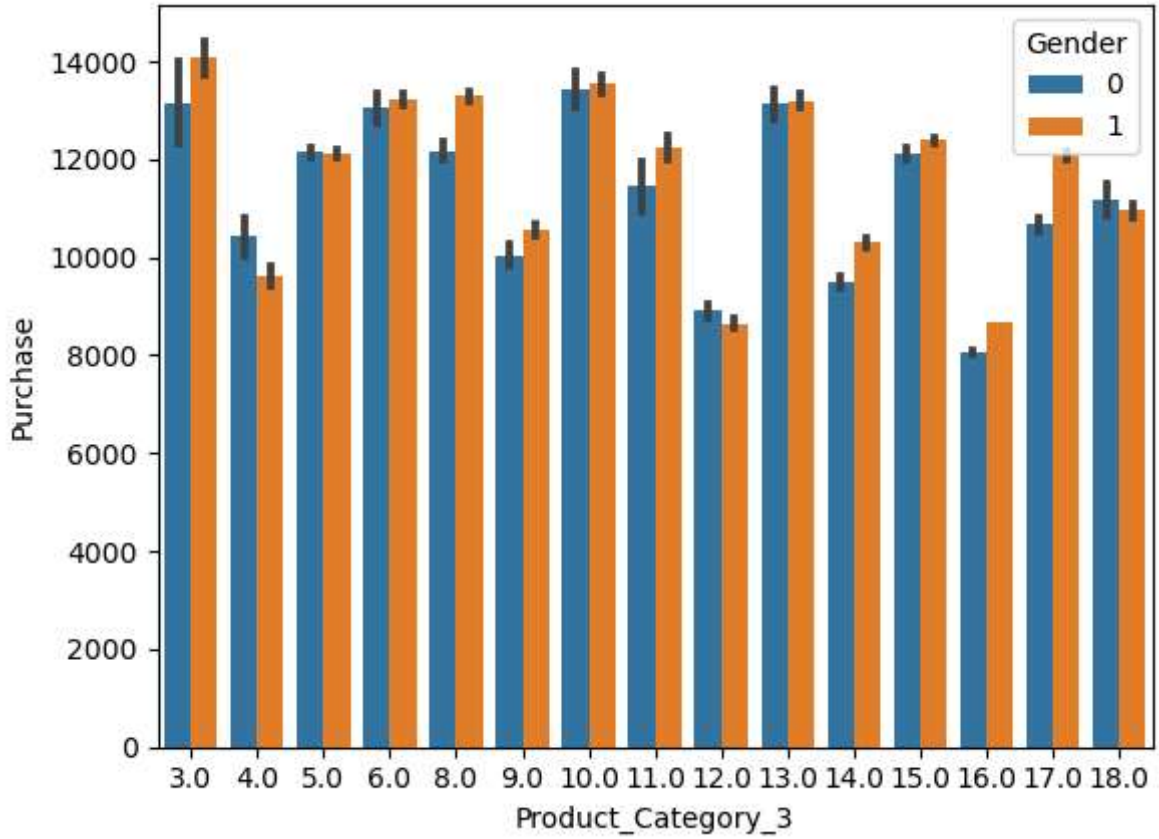
In [132…

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

```
C:\Users\smm\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pa
ss the following variables as keyword args: x, y. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[132]:

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



Product_Category_1 is the most bought product. Product_Category_3 is the least bought product.

In [133…
```
df=df.drop('Product_ID',axis=1)
```

In [134…
```
df.head()
```

Out[134]:

	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Categc
0	0	1	10	1	2	0	
1	0	1	10	1	2	0	
2	0	1	10	1	2	0	
3	0	1	10	1	2	0	
4	1	7	16	2	4	0	

Feature scaling

In [135…
```
df_test=df[df['Purchase'].isnull()]
df_train=df[~df['Purchase'].isnull()]
```

In [136…
```
x=df_train.drop('Purchase',axis=1)
y=df_train['Purchase']
```

In [144…
```
y.shape
```
Out[144]:
```
(550068,)
```

In [145…
```
x.shape
```
Out[145]:
```
(550068, 9)
```

In [137…
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [138…
```
x_train.shape
```
Out[138]:
```
(440054, 9)
```

In [139…
```
y_train
```
Out[139]:
```
178247     7800.0
196647     8677.0
418590     7966.0
408727     9852.0
216416    15804.0
           ...
110268     1862.0
259178     8623.0
365838     6954.0
131932     6151.0
121958     7146.0
Name: Purchase, Length: 440054, dtype: float64
```

In [140…
```
x_test.shape
```
Out[140]:
```
(110014, 9)
```

In [141…
```
y_test.shape
```
Out[141]:
```
(110014,)
```

Train our model

In [143…
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
df_train=sc.fit_transform(df_train)
df_test=sc.transform(df_test)
```

In [148…
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
fit_model=model.fit(x_train,y_train)
y_pred=model.predict(x_test)
```

In [149…
```
y_pred
```
Out[149]:
```
array([11491.26044746, 10621.51205414,  7603.29977449, ...,
        8278.98843806,  6992.67575465,  9347.01837475])
```

In [150…
```
y_test
```

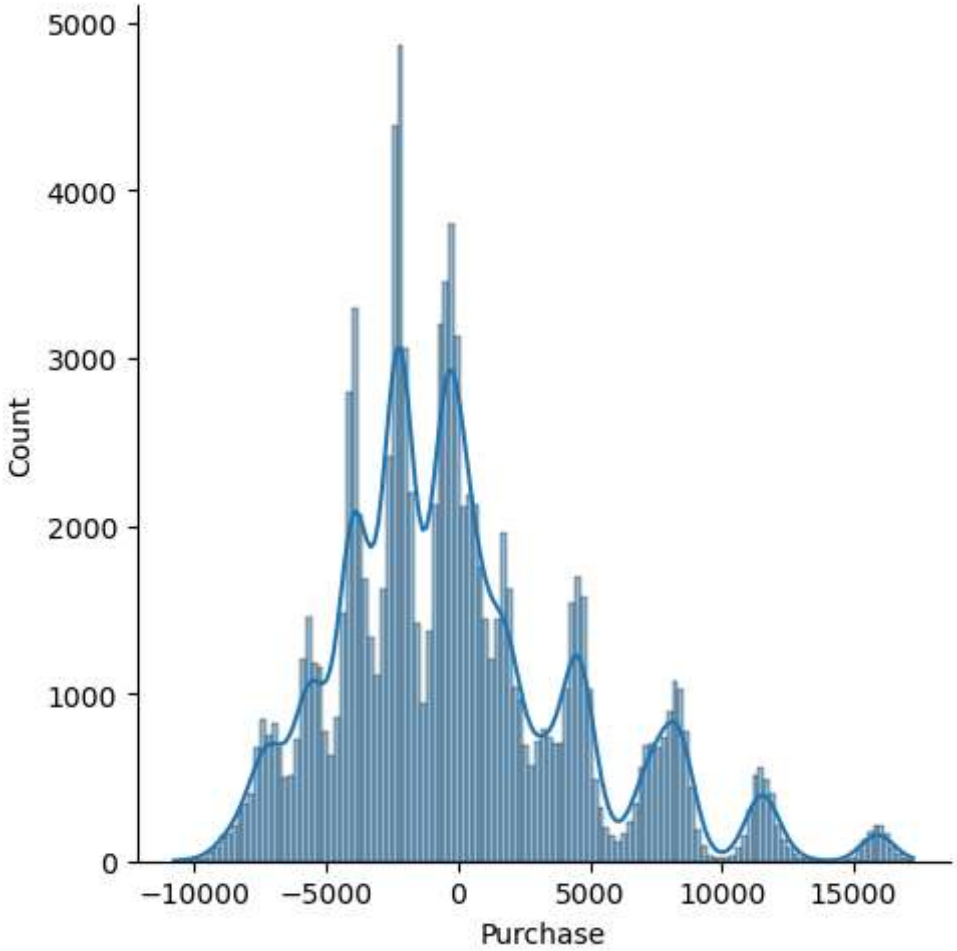
Out[150]:
```
84432     19142.0
72724     15513.0
197032     7802.0
353704    15455.0
91198      4492.0
           ...
133324    19139.0
138718     7856.0
22886      4037.0
137110     7467.0
515193     8002.0
Name: Purchase, Length: 110014, dtype: float64
```

In [153…
```
residuals=y_test - y_pred
residuals
```

Out[153]:
```
84432     7650.739553
72724     4891.487946
197032     198.700226
353704    4695.022437
91198    -4289.240418
             ...
133324    7578.319797
138718     232.372787
22886    -4241.988438
137110     474.324245
515193   -1345.018375
Name: Purchase, Length: 110014, dtype: float64
```

In [154…
```
sns.displot(residuals,kde=True)
```

Out[154]:
```
<seaborn.axisgrid.FacetGrid at 0x17e9f9fc3d0>
```



In [155…

```
from sklearn.metrics import mean_squared_error
print('mean_squared_error ---> ',mean_squared_error(y_test,y_pred))
```

```
mean_squared_error --->  21926829.679565676
```

In [156…

```
from sklearn.metrics import r2_score
score=r2_score(y_test,y_pred)
score
```

Out[156]:

```
0.12733134413466551
```

In [ ]: