In [8]:
```
#import libraries
# import pandas seaborn matplotlib numpy

# import pandas library and make it as pd:
import pandas as pd
#import numerical python library and make it as np
import numpy as np
#import seaborn library and make it as sns
import seaborn as sns
#import pyplot from matplotlib library and make it as plt:
import matplotlib.pyplot as plt
# inorder to surpress the warning import filterwarnings:
from warnings import filterwarnings
filterwarnings('ignore')
# import scipy library:
import scipy
from scipy import stats
```

In [9]:
```
# Load the BLACK FRIDAY SALES dataset
df = pd.read_csv('train.csv', nrows = 15000)
```

`Interpretation` : In This Dataset lakhs of rows are there , so we have reduced to 15000 rows

In [10]:
```
df
```

Out[10]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...
14995	1002225	P00192842	M	26-35	5	B	1	1	5	14.0	NaN	5217
14996	1002225	P00310642	M	26-35	5	B	1	1	8	NaN	NaN	1948
14997	1002228	P00070342	M	26-35	12	C	3	1	1	2.0	14.0	15847
14998	1002228	P00002142	M	26-35	12	C	3	1	1	5.0	8.0	11552
14999	1002230	P00208542	F	46-50	1	B	4+	1	8	NaN	NaN	3934

15000 rows × 12 columns

In [11]:
```
# check the size of the dataset:
df.shape
```

Out[11]:
```
(15000, 12)
```

`Interpretation:` There are (15000 rows and 12 columns) present in the Dataset.

In [12]:
```
# Information about the dataset:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   User_ID                     15000 non-null  int64  
 1   Product_ID                  15000 non-null  object 
 2   Gender                      15000 non-null  object 
 3   Age                         15000 non-null  object 
 4   Occupation                  15000 non-null  int64  
 5   City_Category               15000 non-null  object 
 6   Stay_In_Current_City_Years  15000 non-null  object 
 7   Marital_Status              15000 non-null  int64  
 8   Product_Category_1          15000 non-null  int64  
 9   Product_Category_2          10128 non-null  float64
 10  Product_Category_3          4494 non-null   float64
 11  Purchase                    15000 non-null  int64  
dtypes: float64(2), int64(5), object(5)
memory usage: 1.4+ MB
```

`Interpretation:` It gives the information about the Dataset.

In [13]:
```
#  check the variable type:
df.dtypes
```

Out[13]:
```
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            float64
Product_Category_3            float64
Purchase                        int64
dtype: object
```

`Interpretation:` It gives the Variable types in each column.

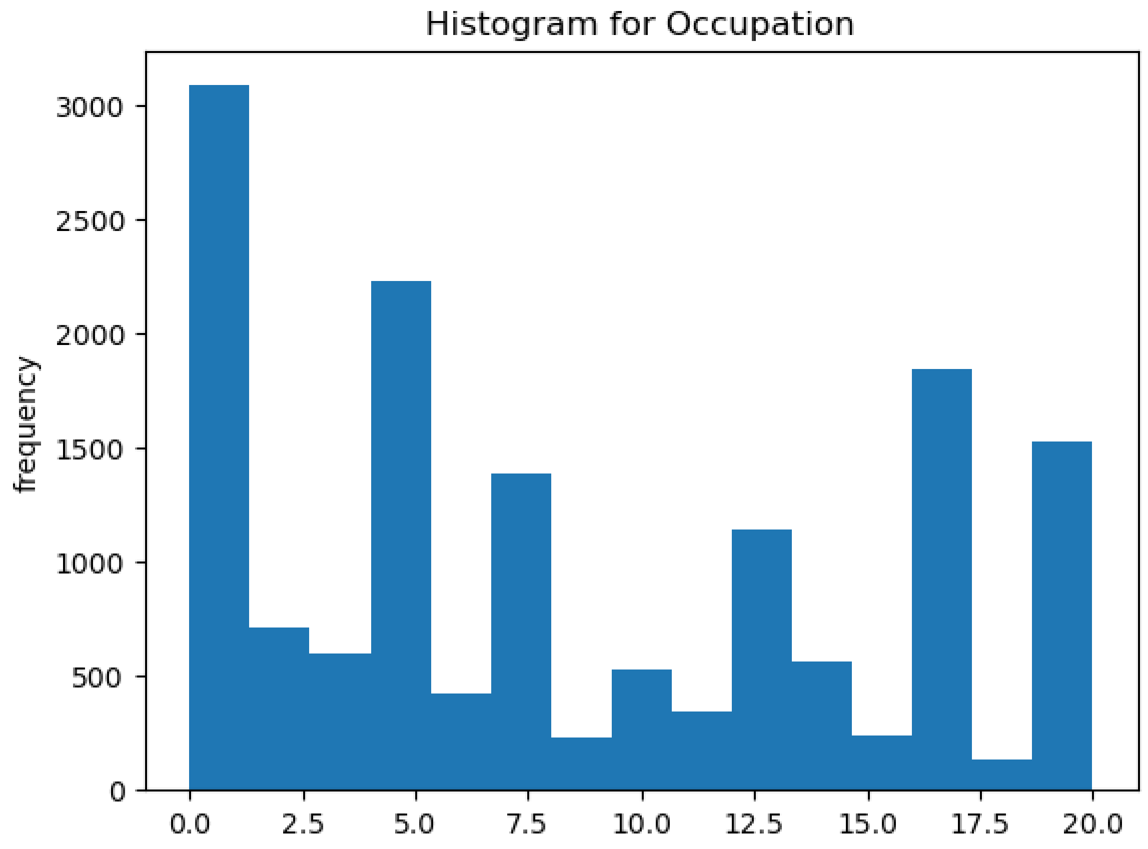
In [14]:
```
# Obtain Occupation :
Occupation = df['Occupation']
```

In [15]:
```
# check the length of the Occupation :
len(Occupation)
```

Out[15]:
```
15000
```

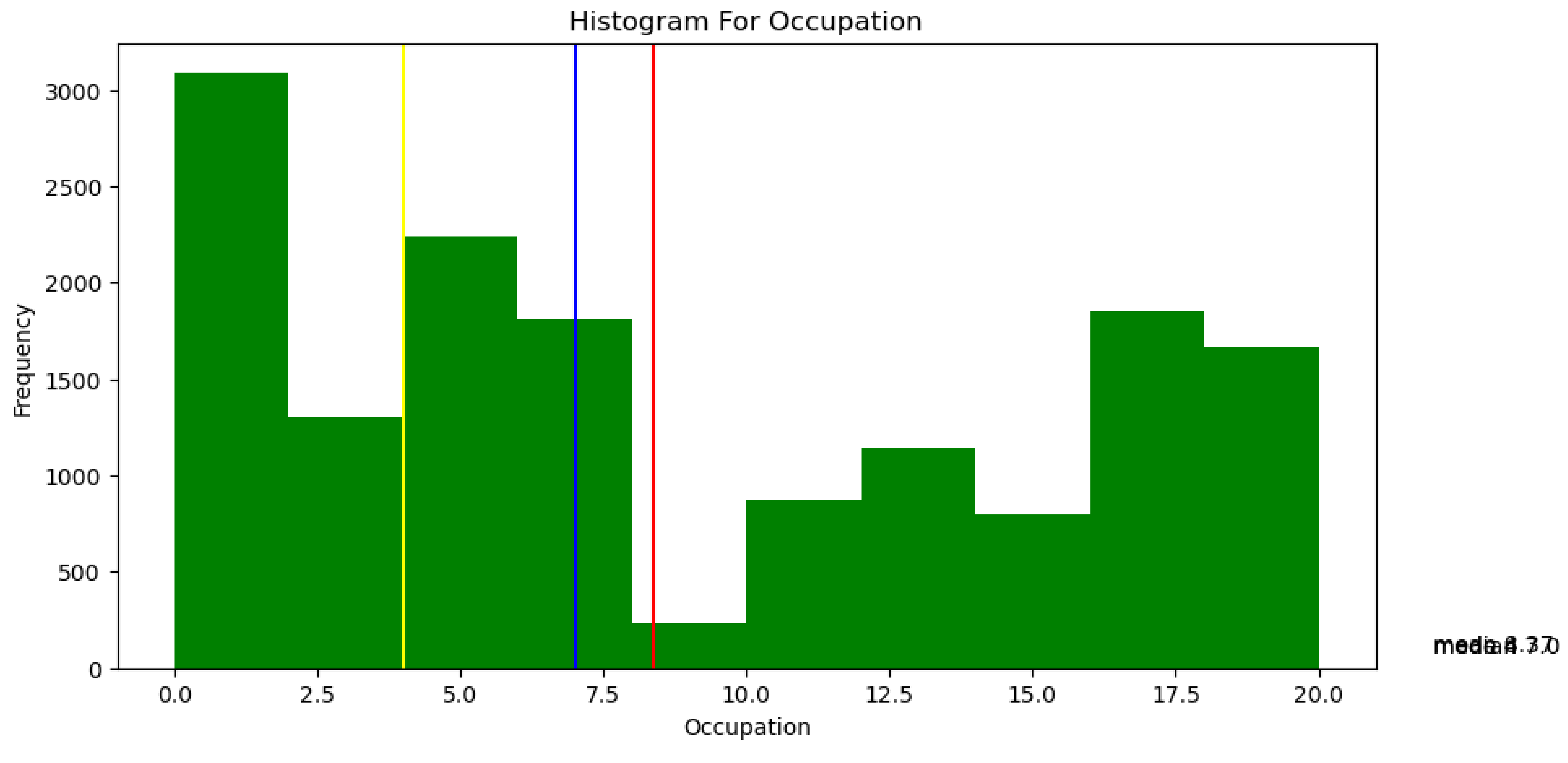
In [16]:

```python
# bins = 15, creates 15 class intervals:

plt.hist(Occupation, bins = 15)

# set the title
plt.title('Histogram for Occupation')

# set label for y - axis:
plt.ylabel('frequency')

# display the plot:
plt.show()
```



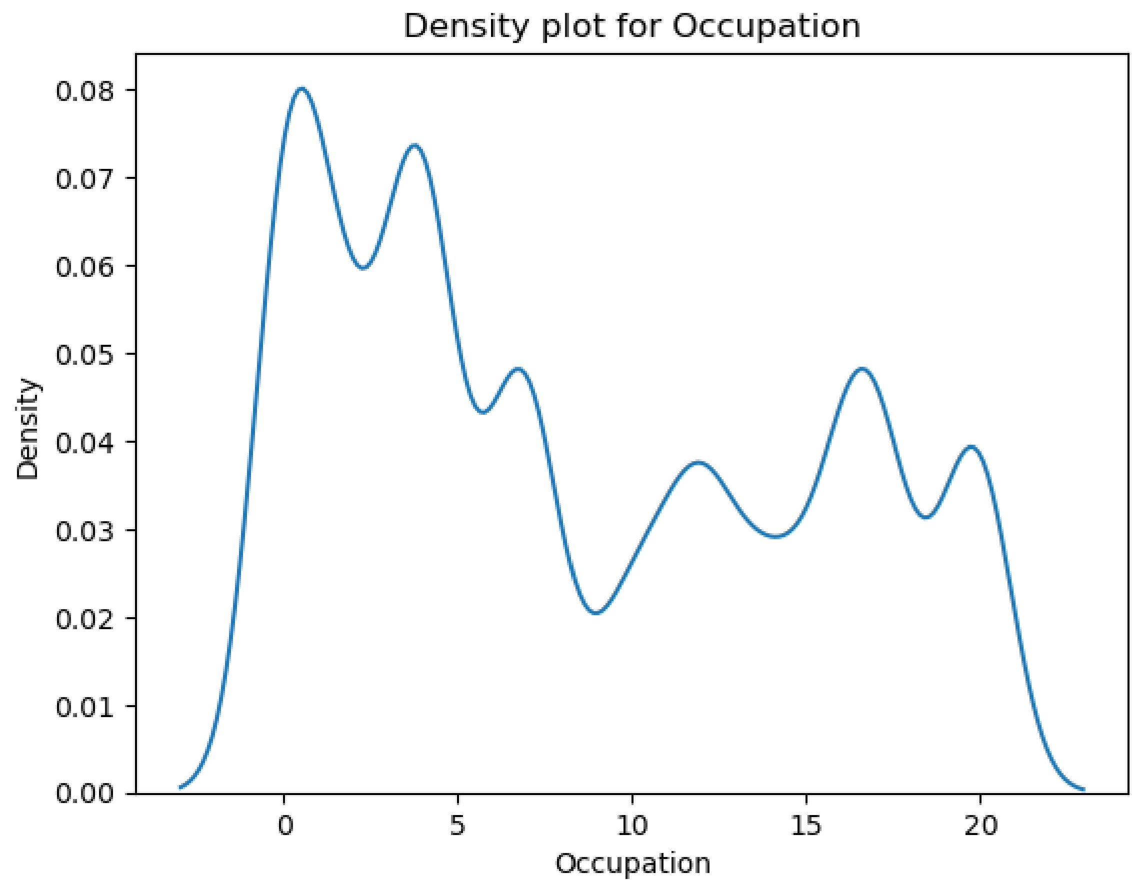
In [17]:

```python
# Set the Figure size:
plt.figure(figsize = (10, 5))
# Bins - 10, Creates the class Interval:
plt.hist(Occupation, bins = 10, color = 'g')
# PLot the Lines of the Mean, Median and Mode on my Histogram PLot:
# Specify Different colors for each Line along by using the 'Color' Parameter
plt.axvline(Occupation.mean(), color = 'red')
plt.axvline(Occupation.median(), color = 'blue')
plt.axvline(Occupation.mode()[0], color = 'Yellow')
# Add the Values in the Plot:
plt.text(22, 90, 'mean'+' '+ str(round(Occupation.mean(),2)))
plt.text(22, 82, 'median'+' '+ str(round(Occupation.median(),2)))
plt.text(22, 75, 'mode'+' '+ str(round(Occupation.mode()[0],2)))
# Set title:
plt.title('Histogram For Occupation')
# Set the Label of x- axis :
plt.xlabel('Occupation')
# Set the Label for y - axis :
plt.ylabel('Frequency')
# Display the Plot:
plt.show()
```

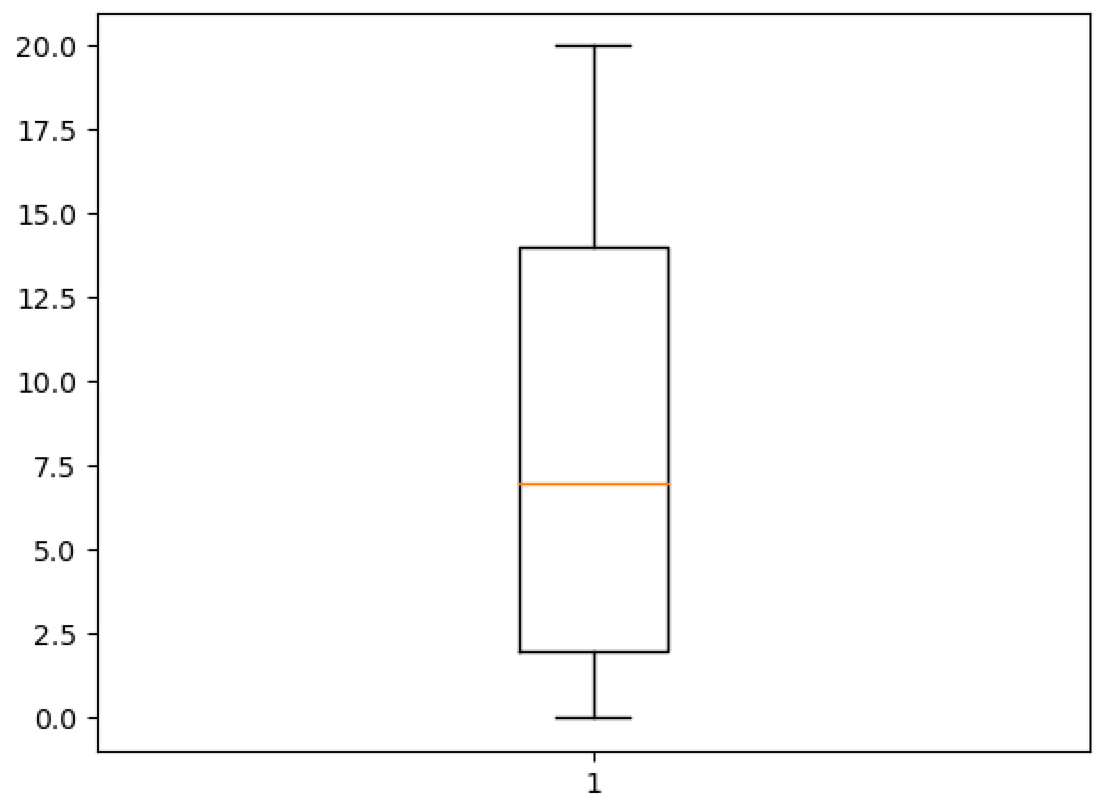


`interpretation` : A histogram is a graphical representation of the distribution of a dataset. It displays the frequencies of the Observation. It clearly shows the Mean, median and mode.

In [18]:
```
# Distplot without Histogram:
# Distplot() - a plot for kernel Density Estimator:
sns.distplot(Occupation, hist = False)
# Set the Title:
plt.title('Density plot for Occupation')
# Display the PLot:
plt.show()
```

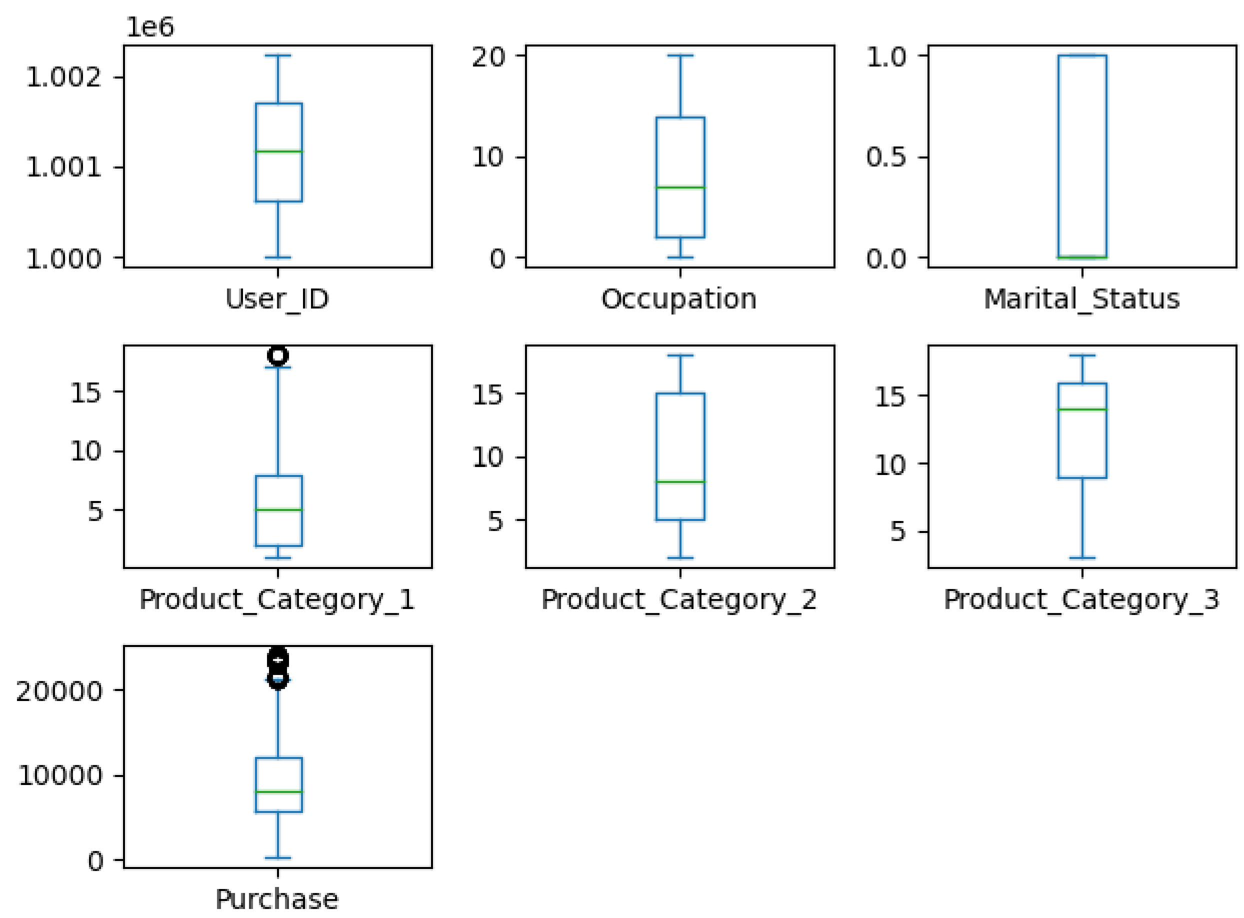


In [19]:
```
# Box plot:
plt.boxplot(Occupation)
# Display the plot:
plt.show()
```



Interpretation : there is no outliers in the Occupation column.

In [20]:
```
df.plot(kind = 'box', subplots = True, layout = (3, 3))
# To give the Specified padding from the subplots:
plt.tight_layout()
# Display the PLot:
plt.show()
```



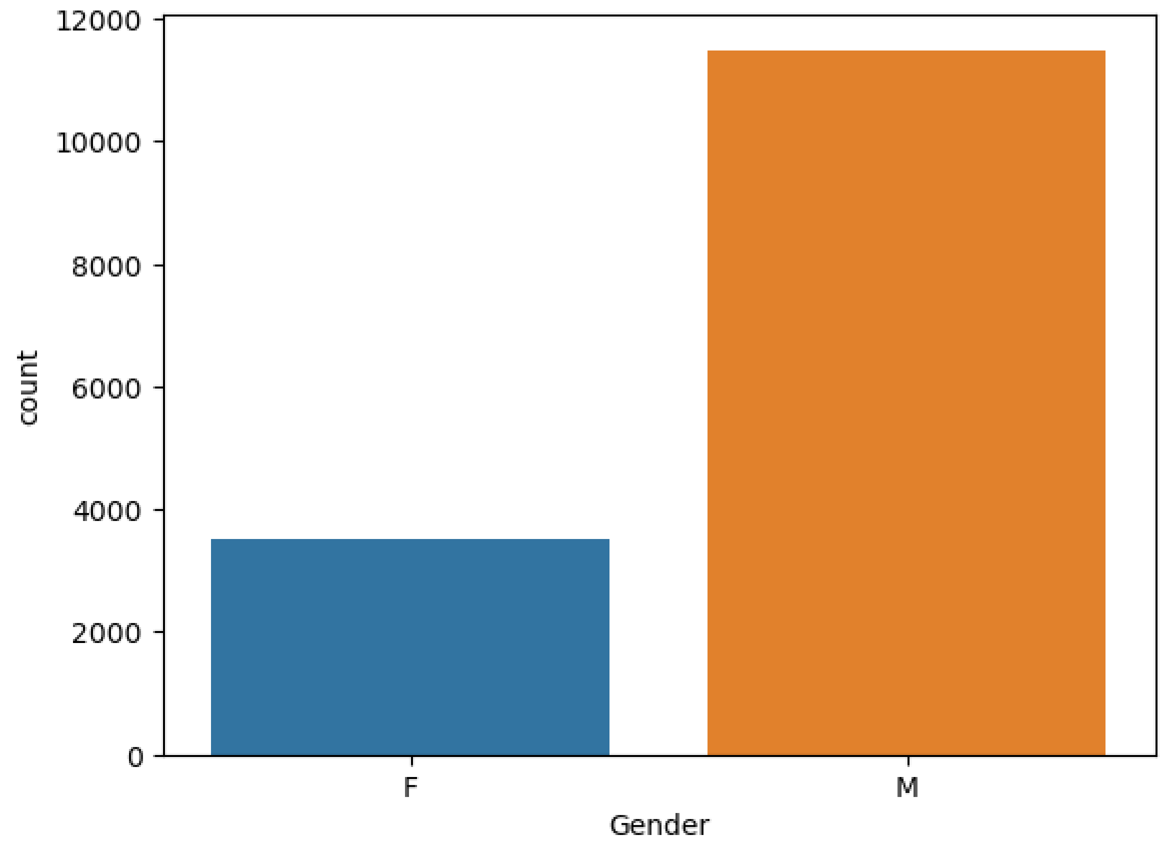
Interpretation : there are product category 1 and purchase has outliers

In [21]:
```
# To check value_counts (Gender column):
df['Gender'].value_counts()
```

Out[21]:
```
M    11490
F     3510
Name: Gender, dtype: int64
```

Interpretation : In Gender column, there are 11490 males and 3510 females in my dataset.

In [22]:
```
# countplot:
sns.countplot(x = 'Gender', data = df)
#Display the plot:
plt.show()
```



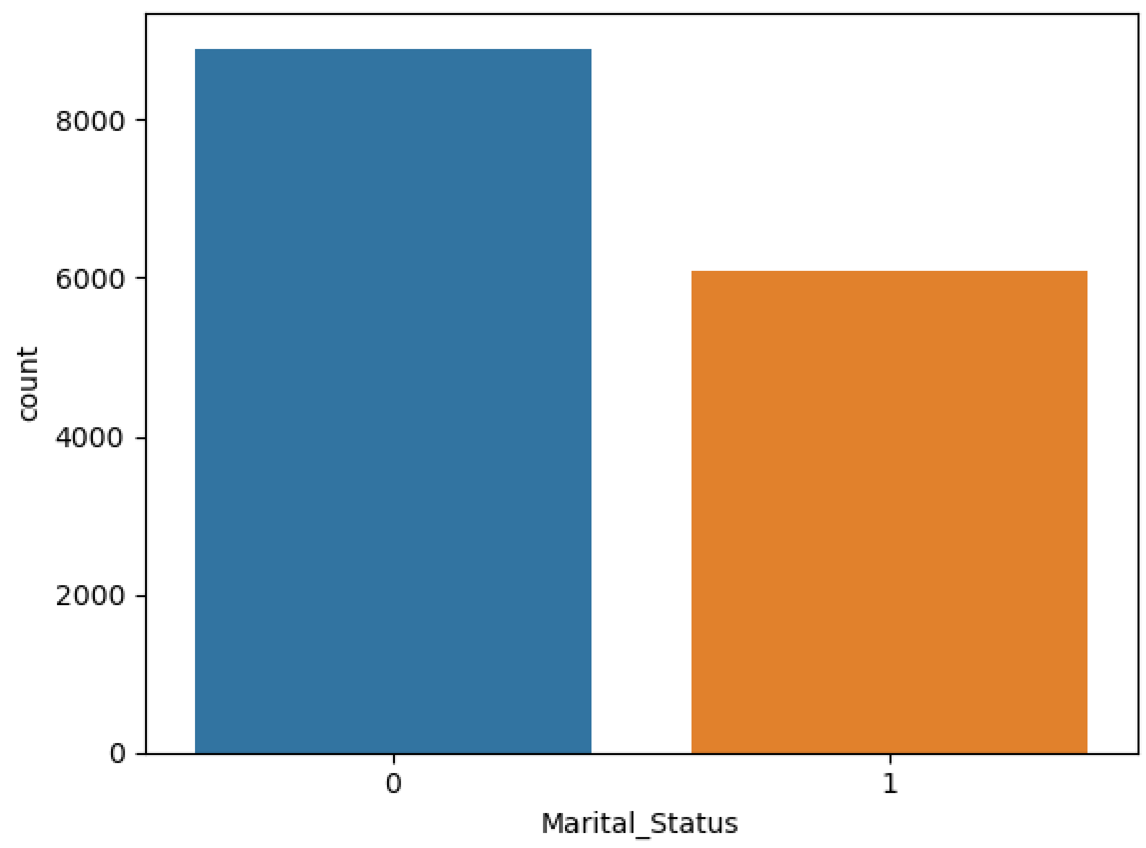
`Interpretation` : **It is visualise clearly using count plot**. The Countplot shows the number of observations for each category.

In [23]:
```
# To check value_counts (marital_status column):
df['Marital_Status'].value_counts()
```

Out[23]:
```
0    8908
1    6092
Name: Marital_Status, dtype: int64
```

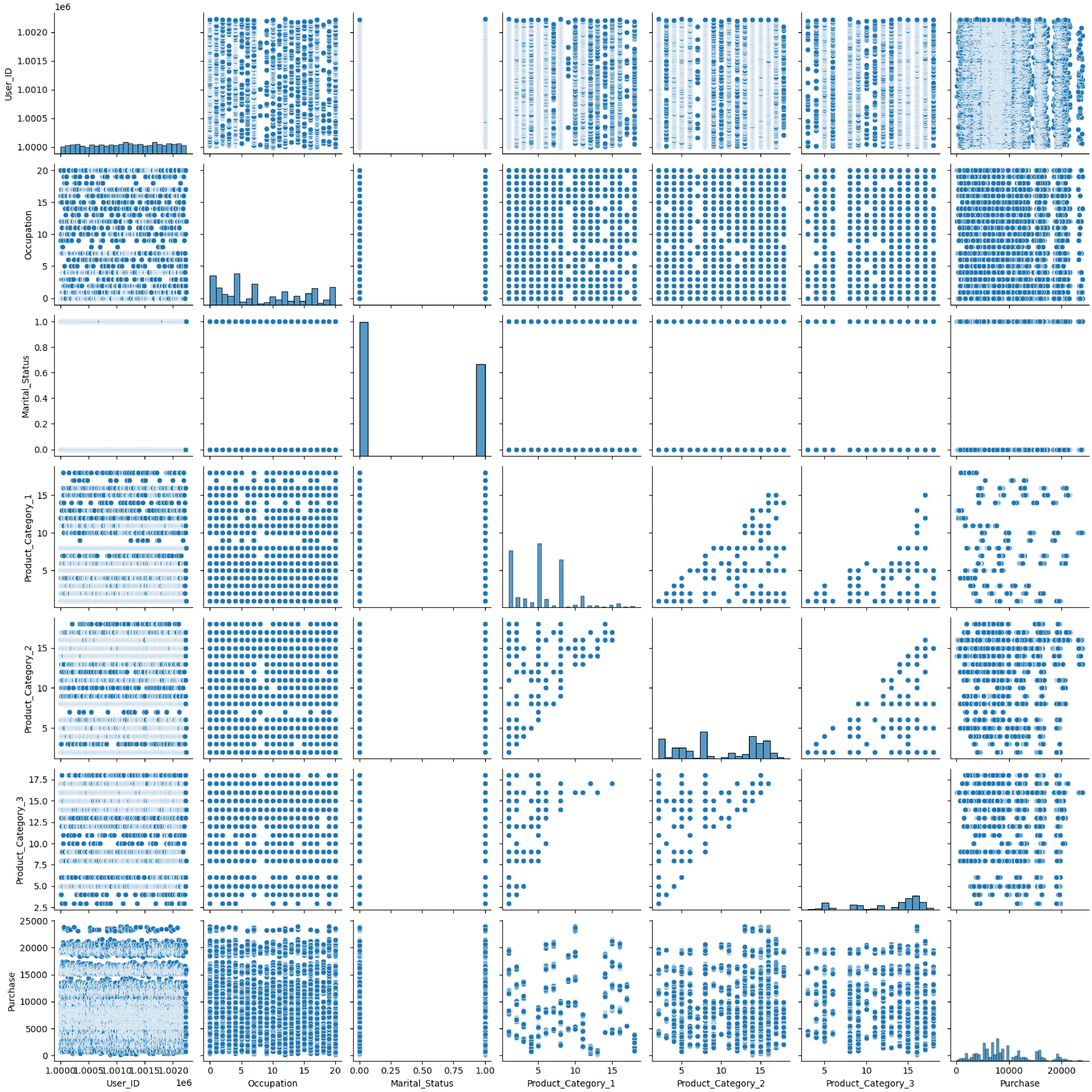
`Interpretation` : In Marital_status column, there are 8908 - 0's and 6092 - 1's in my dataset.

In [24]:
```
# countplot:
sns.countplot(x = 'Marital_Status', data = df)
# Display the plot:
plt.show()
```



`Interpretation` : **It is visualise clearly using count plot.** The Countplot shows the number of observations for each category.

In [25]:
```
# pairplot:
sns.pairplot(data = df)
# display the plot:
plt.show()
```



Interpretation: The pairplot shows the pairwise relationships between all the columns in the Dataset

In [26]:
```
# check for null values:
df.isnull().sum()
```

Out[26]:
```
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          4872
Product_Category_3         10506
Purchase                       0
dtype: int64
```

interpretation there are Product_Category_2 has 4872 Null values and Product_Category_3 has 10506 Null values.

In [27]:
```
# check for total null values:
df.isnull().sum().sum()
```

Out[27]:
```
15378
```

interpretation There are 15378 null values in the dataset

In [28]:
```
df.duplicated().sum()
```

Out[28]:
```
0
```

Interpretaion : There is no Duplicated values in the Dataset.

In [29]:
```
df.duplicated()
```

Out[29]:
```
0        False
1        False
2        False
3        False
4        False
         ...
14995    False
14996    False
14997    False
14998    False
14999    False
Length: 15000, dtype: bool
```

`Interpretaion :` There is no Duplicated values in the Dataset.

In [30]:
```
# Describe Statistics:
# summary of num variables :
df.describe()
```

Out[30]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
count	1.500000e+04	15000.00000	15000.000000	15000.000000	10128.000000	4494.000000	15000.000000
mean	1.001153e+06	8.37040	0.406133	5.300267	9.774585	12.721406	9153.202000
std	6.357349e+02	6.71817	0.491126	3.676204	5.075050	4.093042	4884.921949
min	1.000001e+06	0.00000	0.000000	1.000000	2.000000	3.000000	186.000000
25%	1.000618e+06	2.00000	0.000000	2.000000	5.000000	9.000000	5727.000000
50%	1.001172e+06	7.00000	0.000000	5.000000	8.000000	14.000000	8021.000000
75%	1.001696e+06	14.00000	1.000000	8.000000	15.000000	16.000000	11923.500000
max	1.002230e+06	20.00000	1.000000	18.000000	18.000000	18.000000	23958.000000

`INTERPREATION` It computes summary statistics for numerical columns in the DataFrame, including count, mean, standard deviation, minimum, maximum, and percentiles.

In [31]:
```
# skewness for my entire DataSet:
df.skew()
```

Out[31]:
```
User_ID              -0.090890
Occupation            0.356002
Marital_Status        0.382302
Product_Category_1    0.816648
Product_Category_2   -0.138342
Product_Category_3   -0.808111
Purchase              0.659615
dtype: float64
```

`Interpretaion :` In this dataset Occupation, Marital_Status, Product_Category_1, Purchase are positive skewness and Product_Category_2, Product_Category_3 are negativ skewness

Product_Category_1: The skewness value of 0.816 indicates a moderately right-skewed distribution

In [32]:
```
# obtain kurt():
df.kurt()
```

Out[32]:
```
User_ID              -1.171441
Occupation           -1.279740
Marital_Status       -1.854093
Product_Category_1    0.620056
Product_Category_2   -1.437361
Product_Category_3   -0.724029
Purchase             -0.244646
dtype: float64
```

`Interpretation :` Product_Category_1: The kurtosis value of 0.620 indicates a leptokurtic distribution. and remaining all columns are platykurtic distribution.

In [33]:
```
# obtain co-variance:
cov = df.cov()
```

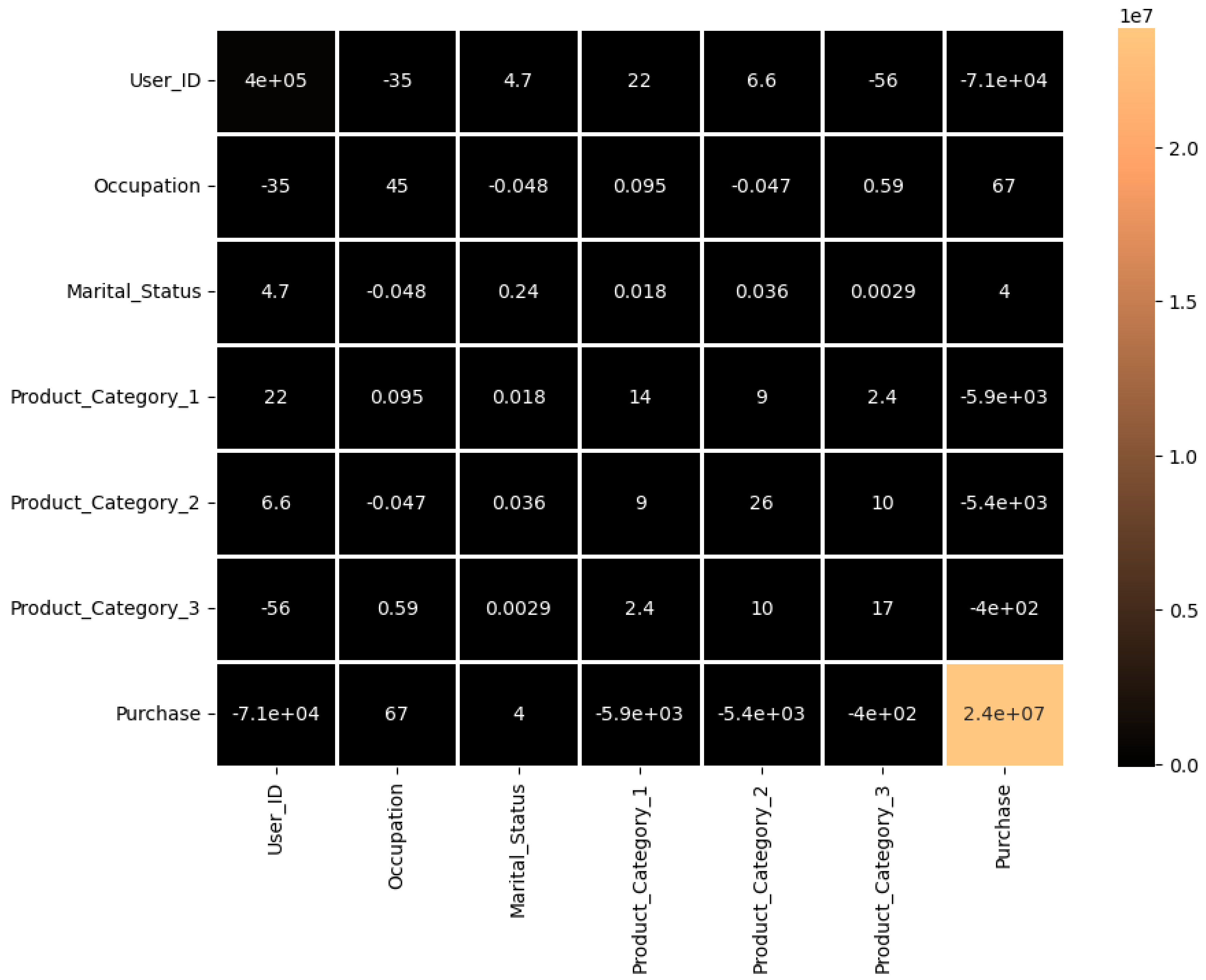
In [34]:
```
# co-variance
cov
```

Out[34]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
User_ID	404158.808095	-34.833668	4.708260	22.274784	6.621372	-55.630329	-7.114361e+04
Occupation	-34.833668	45.133813	-0.048168	0.094521	-0.046910	0.593629	6.728193e+01
Marital_Status	4.708260	-0.048168	0.241205	0.017986	0.036262	0.002915	4.014695e+00
Product_Category_1	22.274784	0.094521	0.017986	13.514474	9.033479	2.414415	-5.861286e+03
Product_Category_2	6.621372	-0.046910	0.036262	9.033479	25.756135	10.057251	-5.415199e+03
Product_Category_3	-55.630329	0.593629	0.002915	2.414415	10.057251	16.752994	-4.045445e+02
Purchase	-71143.613878	67.281931	4.014695	-5861.286073	-5415.199150	-404.544483	2.386246e+07

`Interpretation :` co-variance means b/w the different columns. ex: cov(x,y)

In [36]:
```python
# covariance

cov = df.cov()
cov

# set the plot size:

fig, ax = plt.subplots(figsize = (10, 7))

# plot a heapmap for the covariance matrix
# annot : print values in each cell
# linewidths : specify width of the line and specifying the plot
# vmin minimum value of the variable
# vmax maximum value of the variable
# cmap: colour code for the plot
# fmt : set the decimal place of the annot

sns.heatmap(cov, annot = True, linewidths = 0.95,
           cmap = 'copper', fmt = '.2g')

plt.show()
```



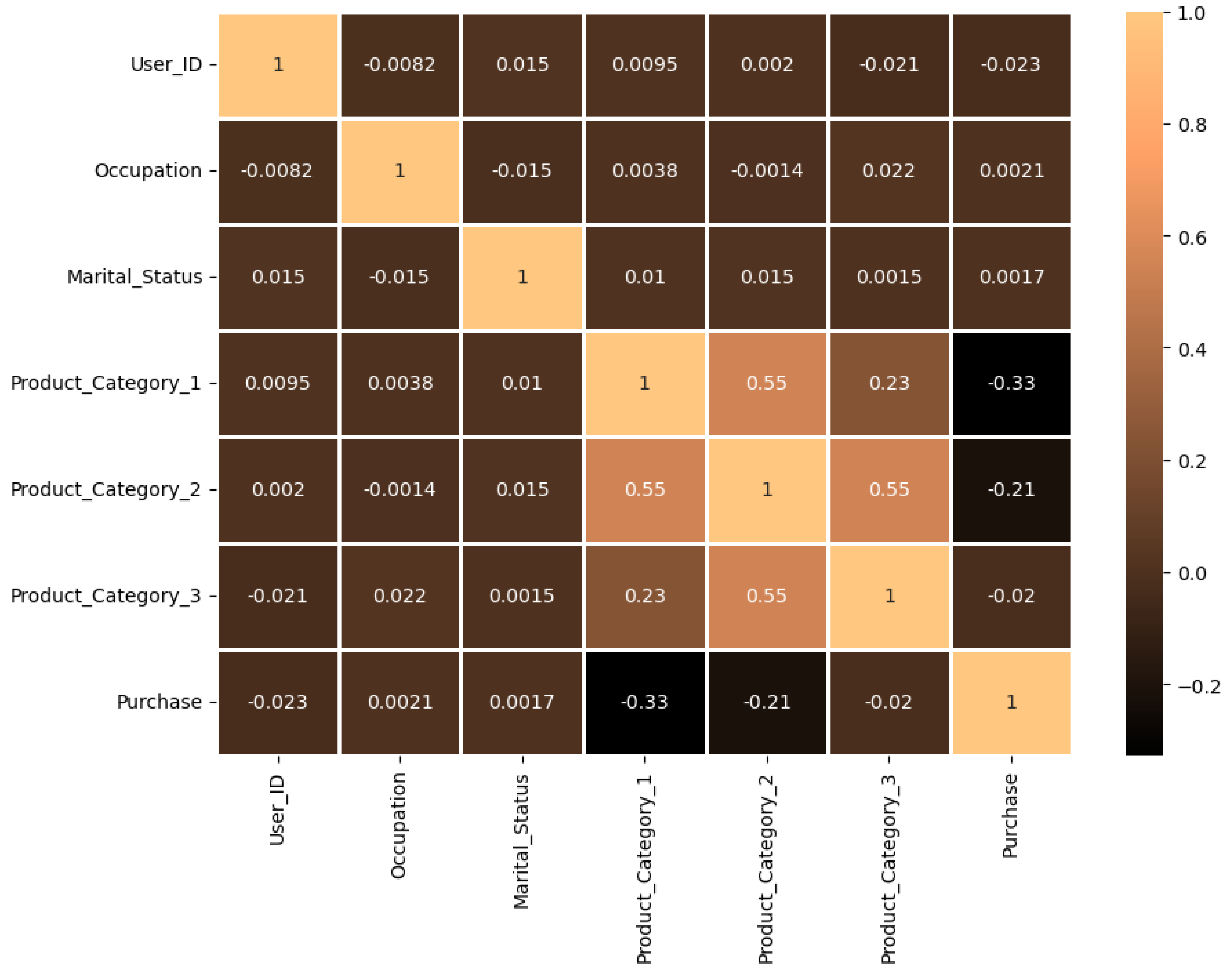
`Interpretation` : It is visualise clearly using subplots

In [37]:
```python
# correlation:

corr =df.corr()
corr
```

Out[37]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
User_ID	1.000000	-0.008156	0.015080	0.009531	0.002045	-0.021192	-0.022909
Occupation	-0.008156	1.000000	-0.014599	0.003827	-0.001388	0.021729	0.002050
Marital_Status	0.015080	-0.014599	1.000000	0.009962	0.014551	0.001460	0.001673
Product_Category_1	0.009531	0.003827	0.009962	1.000000	0.545826	0.228782	-0.326389
Product_Category_2	0.002045	-0.001388	0.014551	0.545826	1.000000	0.548466	-0.209083
Product_Category_3	-0.021192	0.021729	0.001460	0.228782	0.548466	1.000000	-0.019696
Purchase	-0.022909	0.002050	0.001673	-0.326389	-0.209083	-0.019696	1.000000

In [38]:
```python
# set the plot size:

fig, ax = plt.subplots(figsize = (10, 7))

# plot a heapmap for the correlation matrix
# annot : print values in each cell
# linewidths : specify width of the line and specifying the plot
# vmin minimum value of the variable
# vmax maximum value of the variable
# cmap: colour code for the plot
# fmt : set the decimal place of the annot

sns.heatmap(corr, annot = True, linewidths = 0.95,
           cmap = 'copper', fmt = '.2g')
# display the plot:
plt.show()
```



In [39]:
```python
# obtain mean:
df.mean()
```

Out[39]:
```
User_ID               1.001153e+06
Occupation            8.370400e+00
Marital_Status        4.061333e-01
Product_Category_1    5.300267e+00
Product_Category_2    9.774585e+00
Product_Category_3    1.272141e+01
Purchase              9.153202e+03
dtype: float64
```

`interpretation:` we have find the average of the entire dataset. in column wise.

In [40]:
```python
# obtain median:
df.median()
```

Out[40]:
```
User_ID               1001172.0
Occupation                  7.0
Marital_Status              0.0
Product_Category_1          5.0
Product_Category_2          8.0
Product_Category_3         14.0
Purchase                 8021.0
dtype: float64
```

`interpretation:` we have find the median of the entire dataset. in column wise.

In [41]:
```python
# descriptive stats for categorical column:
df.describe(include = 'object')
```

Out[41]:

	Product_ID	Gender	Age	City_Category	Stay_In_Current_City_Years
count	15000	15000	15000	15000	15000
unique	2569	2	7	3	5
top	P00265242	M	26-35	B	1
freq	48	11490	5727	6179	4997

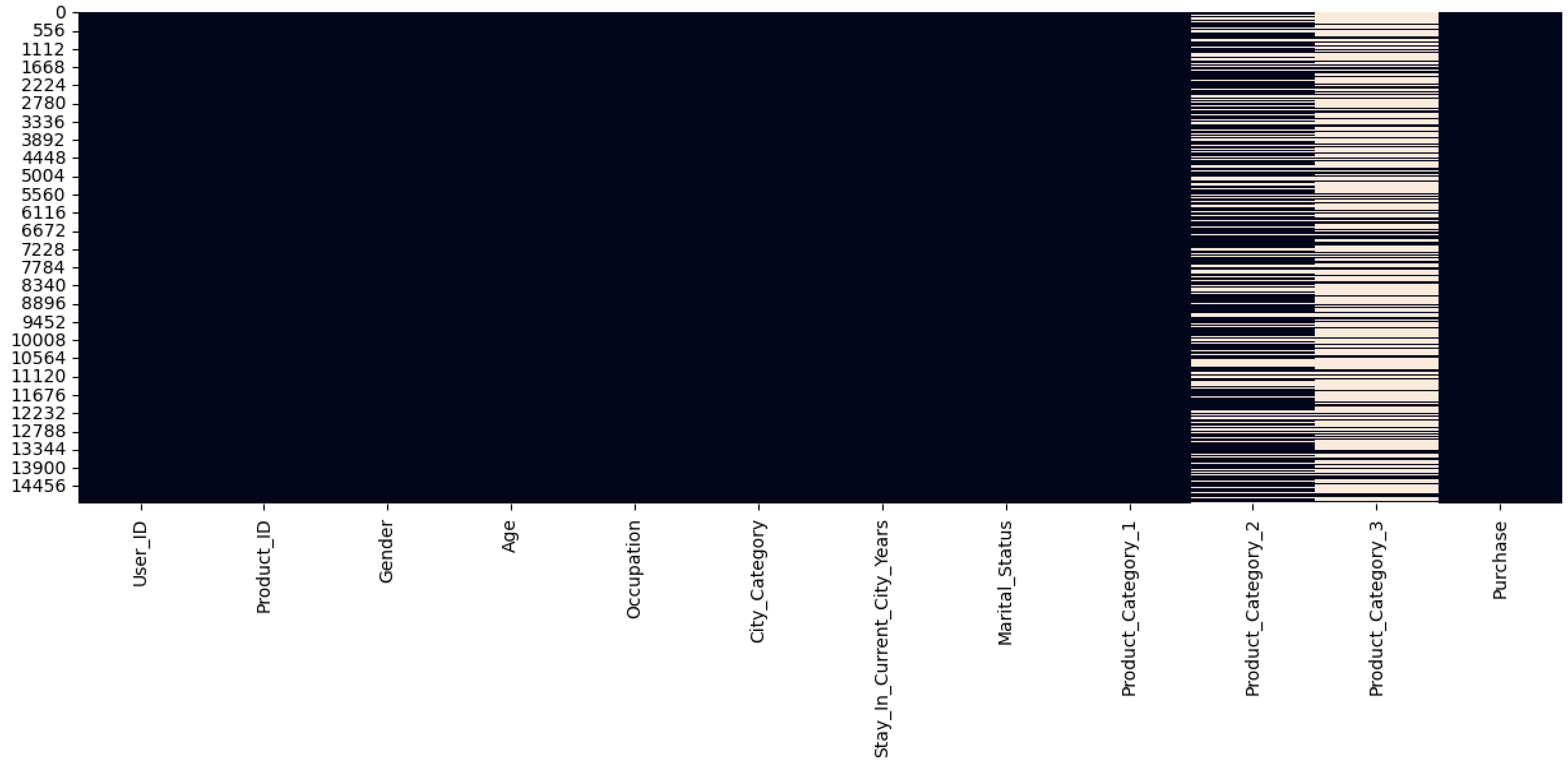
`interpretation` descriptive stats for categorical column:

In [42]:
```python
# sanity check wheather our dataset has null values:
df.isnull().values.any()
```

Out[42]: True

`Interpretation :` True, it means the dataset has null values

In [43]:
```
# let us plot the heatmap to visualize the missing values:

# to make the visualization to firm:

# matplotlib.inline lineremove:

# set the figure size:

plt.rcParams['figure.figsize'] = [15,5]

# plot the heatmap
sns.heatmap(df.isnull(), cbar = False)

# plt the map:
plt.show()
```



interpretation : this heatmap clearly visualizes the null values

In [44]:
```
# get the count of missing values:

missing_values = df.isnull().sum()

# check the missing values:

total = df.isnull().sum().sort_values(ascending = False)

# calculate the percentage of the null values:

percent = (df.isnull().sum()/df.shape[0])*100

# sort the values in the descending order:

percent = percent.sort_values(ascending = False)

# concat the total missing values and percentage of the missing values:

missing_data = pd.concat([total, percent], axis = 1,
                         keys = ['Total', 'Percentage'])

missing_data['Type'] = df[missing_data.index].dtypes


missing_data
```

Out[44]:

	Total	Percentage	Type
Product_Category_3	10506	70.04	float64
Product_Category_2	4872	32.48	float64
User_ID	0	0.00	int64
Product_ID	0	0.00	object
Gender	0	0.00	object
Age	0	0.00	object
Occupation	0	0.00	int64
City_Category	0	0.00	object
Stay_In_Current_City_Years	0	0.00	object
Marital_Status	0	0.00	int64
Product_Category_1	0	0.00	int64
Purchase	0	0.00	int64

interpretation it clearly shows the null values in the table format and added perccentage of the null values and type of the column

In [45]:
```
# let us consider each variable separately for missing value treatments:
# Product_Category_3
# Product_Category_2
```

Product_Category_3

In [46]:
```
# check the sum of null values in Product_Category_3:
df.Product_Category_3.isnull().sum()
```

Out[46]: 10506

Interpretation : The Product_Category_3 has 10506 null values.

In [47]:
```
# check the values counts in Product_Category_3:
df.Product_Category_3.value_counts()
```

Out[47]:
```
16.0    913
15.0    744
14.0    498
17.0    476
5.0     435
8.0     333
9.0     294
12.0    285
13.0    132
6.0     119
18.0     95
11.0     54
4.0      50
10.0     43
3.0      23
Name: Product_Category_3, dtype: int64
```

In [48]:
```
# Product_Category_3 chech head(10) --> starting 10 rows in Product_Category_3(column)
df.Product_Category_3.head(10)
```

Out[48]:
```
0     NaN
1    14.0
2     NaN
3     NaN
4     NaN
5     NaN
6    17.0
7     NaN
8     NaN
9     NaN
Name: Product_Category_3, dtype: float64
```

`Interpretation :` It has standard missing values

In [49]:
```
# obtain describe()
df.Product_Category_3.describe()
```

Out[49]:
```
count    4494.000000
mean       12.721406
std         4.093042
min         3.000000
25%         9.000000
50%        14.000000
75%        16.000000
max        18.000000
Name: Product_Category_3, dtype: float64
```

`Interpretation :` it shows the mean, counts min, max and percentile in the Product_Category_3(column)

In [50]:
```
# check of making median or mean for numerical variable:
```

In [51]:
```
# import necessary labraries:
from matplotlib import gridspec
#set the plot size:
plt.rcParams['figure.figsize'] = [15,5]
# specify the qeometry of the grid that a subplot is placed in:
#split the plot into 2 rows and 3 columns:
gs = gridspec.GridSpec(2,3, width_ratios = [.5,.5,.5], height_ratios = [2,15])
# step 1 : specify the plot location by calling 'gs' initiative above
# step 2 : write the plot text
# write the text in the plot using the text ()
# x : location on x - axis where the text is to be written
# y : location on y - axis where the text is to be written
# s : text to be written
# fontsize: set the font size
# step 3: use.axis('off ') to hide the x and y axes:
# plot the 1st row and 1st column

a11 = plt.subplot(gs[0,0])
a11.text(x = 0.1, y = 0.03, s = 'Original data', fontsize = 15)
a11.axis('off')
# plot the 1st row and 2 column
a12 = plt.subplot(gs[0,1])
a12.text(x = 0.1, y = 0.03, s = 'Data Imputed with mean', fontsize = 15)
a12.axis('off')
#plot the 1 st row and 3rd column
a12=plt.subplot(gs[0,2])
a12.text(x=0.1,y=0.03,s='Data Imputed with Median',fontsize=15)
a12.axis('off')
#summary statistics:
#step=1: specify the plot location by calling 'gs' intiative above
#step=2: specify the text along with location
#step=3: use.axis('off')to hide the x and y axes:
#plot in 2nd row and 1st column
#original data
a12=plt.subplot(gs[1,0])
a12.text(0.05,.3,s=str(df['Product_Category_3'].describe()),fontsize=15)
a12.axis('off')
#fill the missing value with mean
#obtain the mean of the data
mu=df['Product_Category_3'].mean()
a12=plt.subplot(gs[1,1])
a12.text(0.05,0.3,s=str(df['Product_Category_3'].fillna(mu).describe()),fontsize=15)
a12.axis('off')
#fill the missing value with median
me=df['Product_Category_3'].median()
a12=plt.subplot(gs[1,2])
a12.text(0.05,0.3,s=str(df['Product_Category_3'].fillna(me).describe()),fontsize=15)
a12.axis('off')
#display the plot
plt.show()
```

Original data		Data Imputed with mean		Data Imputed with Median	
count	4494.000000	count	15000.000000	count	15000.000000
mean	12.721406	mean	12.721406	mean	13.616933
std	4.093042	std	2.240182	std	2.315488
min	3.000000	min	3.000000	min	3.000000
25%	9.000000	25%	12.721406	25%	14.000000
50%	14.000000	50%	12.721406	50%	14.000000
75%	16.000000	75%	12.721406	75%	14.000000
max	18.000000	max	18.000000	max	18.000000

Name: Product_Category_3, dtype: floatName: Product_Category_3, dtype: floatName: Product_Category_3, dtype: float64

`Interpretaion` mean is minimum values so we take mean

In [52]:
```
# obtain describe()
df.Product_Category_3.describe()
```

Out[52]:
```
count    4494.000000
mean       12.721406
std         4.093042
min         3.000000
25%         9.000000
50%        14.000000
75%        16.000000
max        18.000000
Name: Product_Category_3, dtype: float64
```

In [53]:
```
# obtain mean:
df.Product_Category_3.mean()
```

Out[53]:
```
12.721406319537161
```

`Interpretation :` in the Product_Category_3(column) mean has 12.72

In [54]:
```
# replace all the missing values with '12.72' it is called mean
df.Product_Category_3.replace(np.NaN, '12.72', inplace = True)
```

`Interpretation :` Replace all the null values into (avg 12.72)

In [55]:
```
df.Product_Category_3.isnull().sum()
```

Out[55]:
```
0
```

`Interpretaion` now it have replaced the null values into mean value(12.72)

all null values are filled with the average value (12.72)

`Interpretation` now, no null values are present in the column (Product_Category_3)

In [56]:
```
# get the count of missing values:

missing_values = df.isnull().sum()

# check for missing values:

total = df.isnull().sum().sort_values(ascending = False)

# calculate percentage of the null values:

percent = ((df.isnull().sum()/df.shape[0])*100)

# sort the values in descending order

percent = percent.sort_values(ascending = False)

# concatenate the total missing values and percentage of the missing values:

missing_data = pd.concat([total, percent], axis = 1,
                         keys = ['Total', ' Percentage'])

#

missing_data['Type'] = df[missing_data.index].dtypes

missing_data
```

Out[56]:

	Total	Percentage	Type
Product_Category_2	4872	32.48	float64
User_ID	0	0.00	int64
Product_ID	0	0.00	object
Gender	0	0.00	object
Age	0	0.00	object
Occupation	0	0.00	int64
City_Category	0	0.00	object
Stay_In_Current_City_Years	0	0.00	object
Marital_Status	0	0.00	int64
Product_Category_1	0	0.00	int64
Product_Category_3	0	0.00	object
Purchase	0	0.00	int64

`Interpretation :` In this table only Product_Category_2 has null values.

Product_Category_2

In [58]:
```
# check for sum null values  in the Product_Category_2 column
df.Product_Category_2.isnull().sum()
```

Out[58]:
```
4872
```

`Interpretation :` Product_Category_2 has 4872 null values

In [60]:
```
# check the value counts:
df.Product_Category_2.value_counts()
```

Out[60]:
```
8.0     1767
14.0    1472
2.0     1318
16.0    1160
15.0     992
4.0      717
5.0      692
6.0      471
11.0     377
17.0     357
13.0     268
9.0      158
12.0     140
3.0       85
10.0      76
18.0      60
7.0       18
Name: Product_Category_2, dtype: int64
```

In [61]:
```
df.Product_Category_2.head(10)
```

Out[61]:
```
0     NaN
1     6.0
2     NaN
3    14.0
4     NaN
5     2.0
6     8.0
7    15.0
8    16.0
9     NaN
Name: Product_Category_2, dtype: float64
```

`interpretation` it has standard missing values

In [62]:
```
# obtain discriptive statistics:
df.Product_Category_2.describe()
```

Out[62]:
```
count    10128.000000
mean         9.774585
std          5.075050
min          2.000000
25%          5.000000
50%          8.000000
75%         15.000000
max         18.000000
Name: Product_Category_2, dtype: float64
```

`Interpretation :` it has count,mean, sd, min, max, percentile values shows

In [63]:
```python
#import necessary libraries
from matplotlib import gridspec
#set the plot size:
plt.rcParams['figure.figsize']=[15,5]
#specify the geometry of the grid that a subplot is placed in:
#split the plot into 2 rows and 3 columns
gs=gridspec.GridSpec(2,3,width_ratios=[.5,.5,.5],height_ratios=[2,15])
#step=1: specify the plot location by calling 'gs' iitiative above
#step=2 : write the plot text
#write the text in the plot using the text()
#x: location on x-axis where the text is to be written
#y: location on y-axis where the text is to be written
#s: text to be written
#fontsize: set the font size
#step=3 : use.axis('off') to hide the x and y axes:
#plot the 1 st row and 1 st column

a11=plt.subplot(gs[0,0])
a11.text(x=0.1,y=0.03,s='Original Data',fontsize=15)
a11.axis('off')
#plot the 1 st row and 2nd column
a12=plt.subplot(gs[0,1])
a12.text(x=0.1,y=0.03,s='Data Imputed with Mean',fontsize=15)
a12.axis('off')
#plot the 1 st row and 3rd column
a12=plt.subplot(gs[0,2])
a12.text(x=0.1,y=0.03,s='Data Imputed with Median',fontsize=15)
a12.axis('off')
#summary statistics:
#step=1: specify the plot location by calling 'gs' intiative above
#step=2: specify the text along with location
#step=3: use.axis('off')to hide the x and y axes:
#plot in 2nd row and 1st column
#original data
a12=plt.subplot(gs[1,0])
a12.text(0.05,.3,s=str(df['Product_Category_2'].describe()),fontsize=15)
a12.axis('off')
#fill the missing value with mean
#obtain the mean of the data
mu=df['Product_Category_2'].mean()
a12=plt.subplot(gs[1,1])
a12.text(0.05,0.3,s=str(df['Product_Category_2'].fillna(mu).describe()),fontsize=15)
a12.axis('off')
#fill the missing value with median
me=df['Product_Category_2'].median()
a12=plt.subplot(gs[1,2])
a12.text(0.05,0.3,s=str(df['Product_Category_2'].fillna(me).describe()),fontsize=15)
a12.axis('off')
#display the plot
plt.show()
```

Original Data	Data Imputed with Mean	Data Imputed with Median
count 10128.000000	count 15000.000000	count 15000.000000
mean 9.774585	mean 9.774585	mean 9.198200
std 5.075050	std 4.170130	std 4.252135
min 2.000000	min 2.000000	min 2.000000
25% 5.000000	25% 8.000000	25% 8.000000
50% 8.000000	50% 9.774585	50% 8.000000
75% 15.000000	75% 14.000000	75% 14.000000
max 18.000000	max 18.000000	max 18.000000
Name: Product_Category_2, dtype: float64	Name: Product_Category_2, dtype: float64	Name: Product_Category_2, dtype: float64

interpretation median is minimum so we take median

In [64]:
```python
# Product_Category_2
df['Product_Category_2'].fillna(me, inplace =True)
```

Interpretation : It has fill tha null values

In [65]:
```python
# describe the Product_Category_2:
df.Product_Category_2.describe()
```

Out[65]:
```
count    15000.000000
mean         9.198200
std          4.252135
min          2.000000
25%          8.000000
50%          8.000000
75%         14.000000
max         18.000000
Name: Product_Category_2, dtype: float64
```

In [66]:
```python
# let check the head of the column (head(10) of Product_Category_2:)
df.Product_Category_2.head(10)
```

Out[66]:
```
0     8.0
1     6.0
2     8.0
3    14.0
4     8.0
5     2.0
6     8.0
7    15.0
8    16.0
9     8.0
Name: Product_Category_2, dtype: float64
```

Interpretation : it shows the first 10 rows

In [67]:
```
# check the value counts of the column:

df.Product_Category_2.value_counts()
```

Out[67]:
```
8.0     6639
14.0    1472
2.0     1318
16.0    1160
15.0     992
4.0      717
5.0      692
6.0      471
11.0     377
17.0     357
13.0     268
9.0      158
12.0     140
3.0       85
10.0      76
18.0      60
7.0       18
Name: Product_Category_2, dtype: int64
```

`Interpretation :` all are fill with median values 8.0

In [68]:
```
# obtain the median:
df.Product_Category_2.median()
```

Out[68]:
```
8.0
```

In [69]:
```
# replace all the missing values with '8.0' it is called median:
df.Product_Category_2.replace(np.NaN, '8.0', inplace = True)
```

In [70]:
```
#sanity check for the missing values
df.Product_Category_2.isnull().sum()
```

Out[70]:
```
0
```

`Interpretation :` There is no null values in the Product_Category_2, all null values are filled in median value 8.0

In [71]:
```
# sanity check for the column Product_Category_2:
df.Product_Category_2.head(20)
```

Out[71]:
```
0      8.0
1      6.0
2      8.0
3     14.0
4      8.0
5      2.0
6      8.0
7     15.0
8     16.0
9      8.0
10    11.0
11     8.0
12     8.0
13     2.0
14     8.0
15     5.0
16     3.0
17    14.0
18    14.0
19     5.0
Name: Product_Category_2, dtype: float64
```

`Interpretaion` all missing values are filled in median value(8.0)

In [72]:
```
# check the null values in the (Product_Category_2) column:
df.Product_Category_2.isnull().sum()
```

Out[72]:
```
0
```

In [73]:
```
# check the null values in the dataset:
df.isnull().sum()
```

Out[73]:
```
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
```

`interpretation` now, there is no null values in the dataset

In [74]:
```
# get the count of missing values:

missing_values = df.isnull().sum()

# check for missing values:

total = df.isnull().sum().sort_values(ascending = False)

# calculate percentage of the null values:

percent = ((df.isnull().sum()/df.shape[0])*100)

# sort the values in descending order

percent = percent.sort_values(ascending = False)

# concatenate the total missing values and percentage of the missing values:

missing_data = pd.concat([total, percent], axis = 1,
                        keys = ['Total', ' Percentage'])

#

missing_data['Type'] = df[missing_data.index].dtypes

missing_data
```

Out[74]:

	Total	Percentage	Type
User_ID	0	0.0	int64
Product_ID	0	0.0	object
Gender	0	0.0	object
Age	0	0.0	object
Occupation	0	0.0	int64
City_Category	0	0.0	object
Stay_In_Current_City_Years	0	0.0	object
Marital_Status	0	0.0	int64
Product_Category_1	0	0.0	int64
Product_Category_2	0	0.0	float64
Product_Category_3	0	0.0	object
Purchase	0	0.0	int64

`Interpretaion` there is no missing values in the dataset.

In [75]:
```
# sanity check :
missing_values = df.isnull().sum()
missing_values
```

Out[75]:
```
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
```

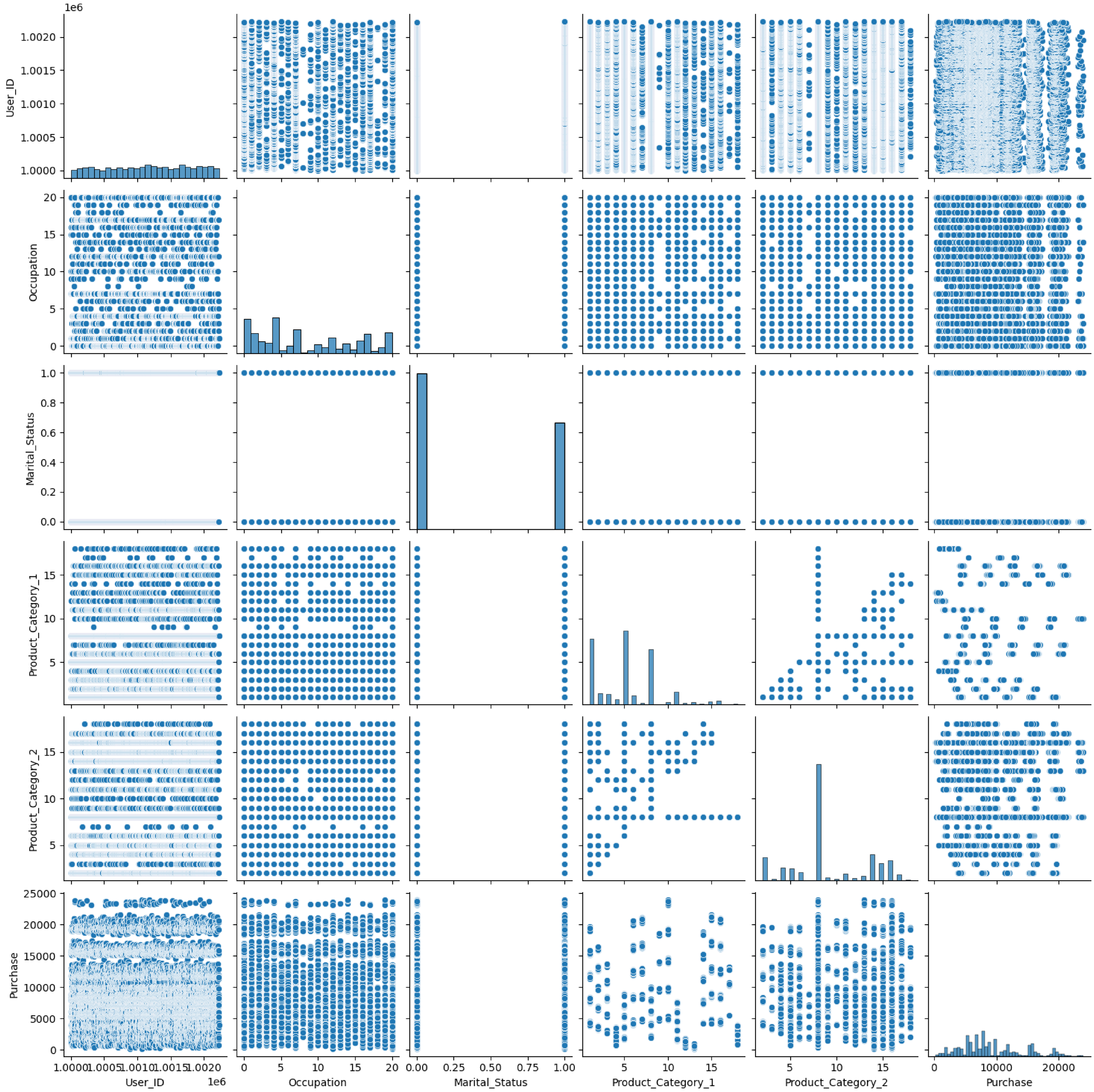
`Interpretation` there is no missing values

In [76]:
```
# check the head()
df.head()
```

Out[76]:

	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	8.0	12.72	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	8.0	12.72	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	12.72	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	8.0	12.72	7969

In [77]:
```
# pairplot:
sns.pairplot(data = df)
# display the plot:
plt.show()
```



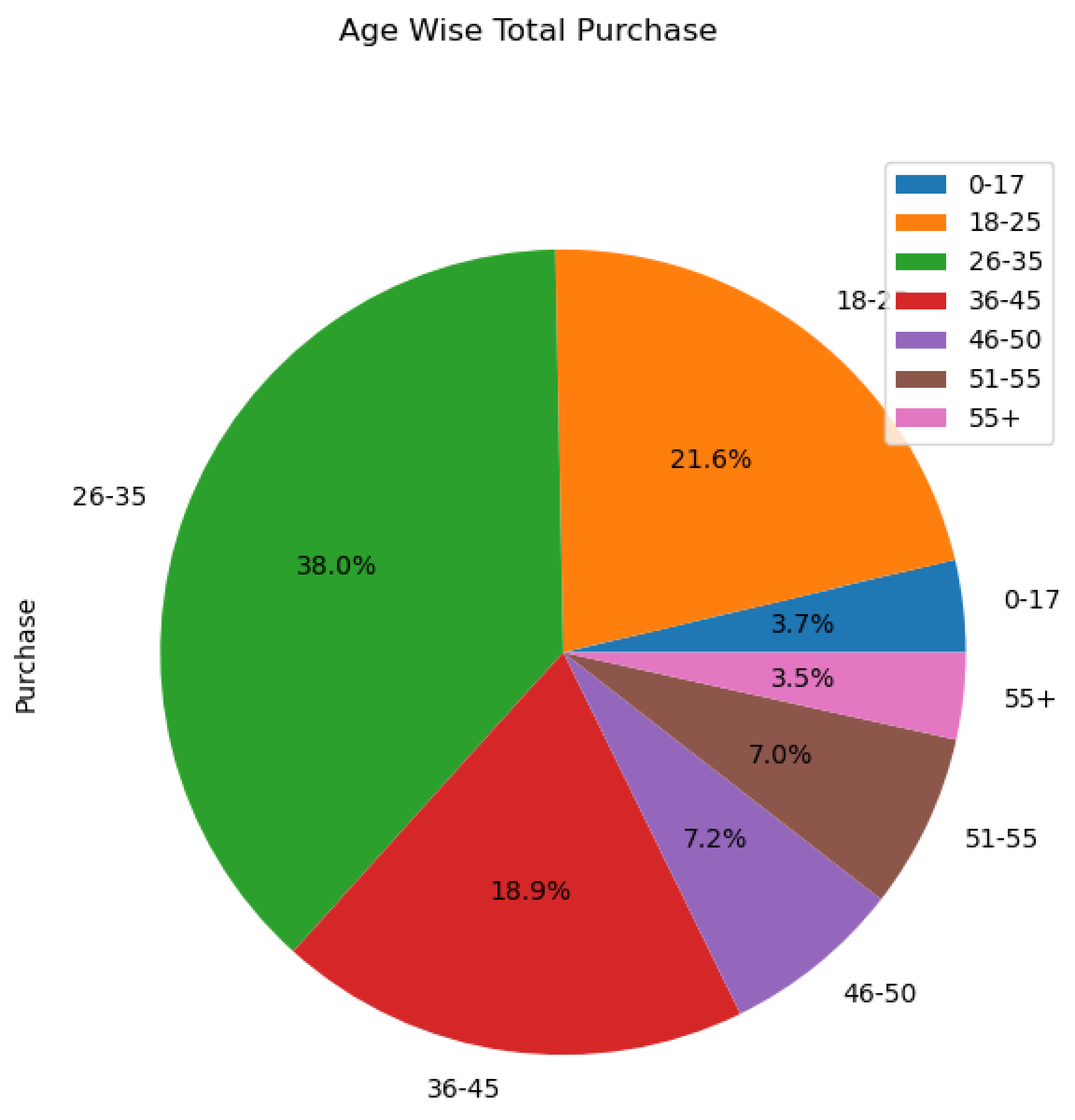
In [78]:
```
# obtain Age column to check the value counts:
df['Age'].value_counts()
```

Out[78]:
```
26-35    5727
18-25    3272
36-45    2825
46-50    1078
51-55    1017
0-17      555
55+       526
Name: Age, dtype: int64
```

`interpretaion` : the value counts of Age column is clearly shows, to count the values.

In [80]:
```
# Pie chart to visualize the age wise distribution of total purchase:

x = df[['Age','Purchase']].groupby('Age').sum()
x.plot(kind='pie',autopct='%1.1f%%',subplots=True,figsize=(15,7),title='Age Wise Total Purchase')
```

Out[80]:
```
array([<AxesSubplot:ylabel='Purchase'>], dtype=object)
```



Interpretation : The pie chart clearly visualise the Age wise total purchase.

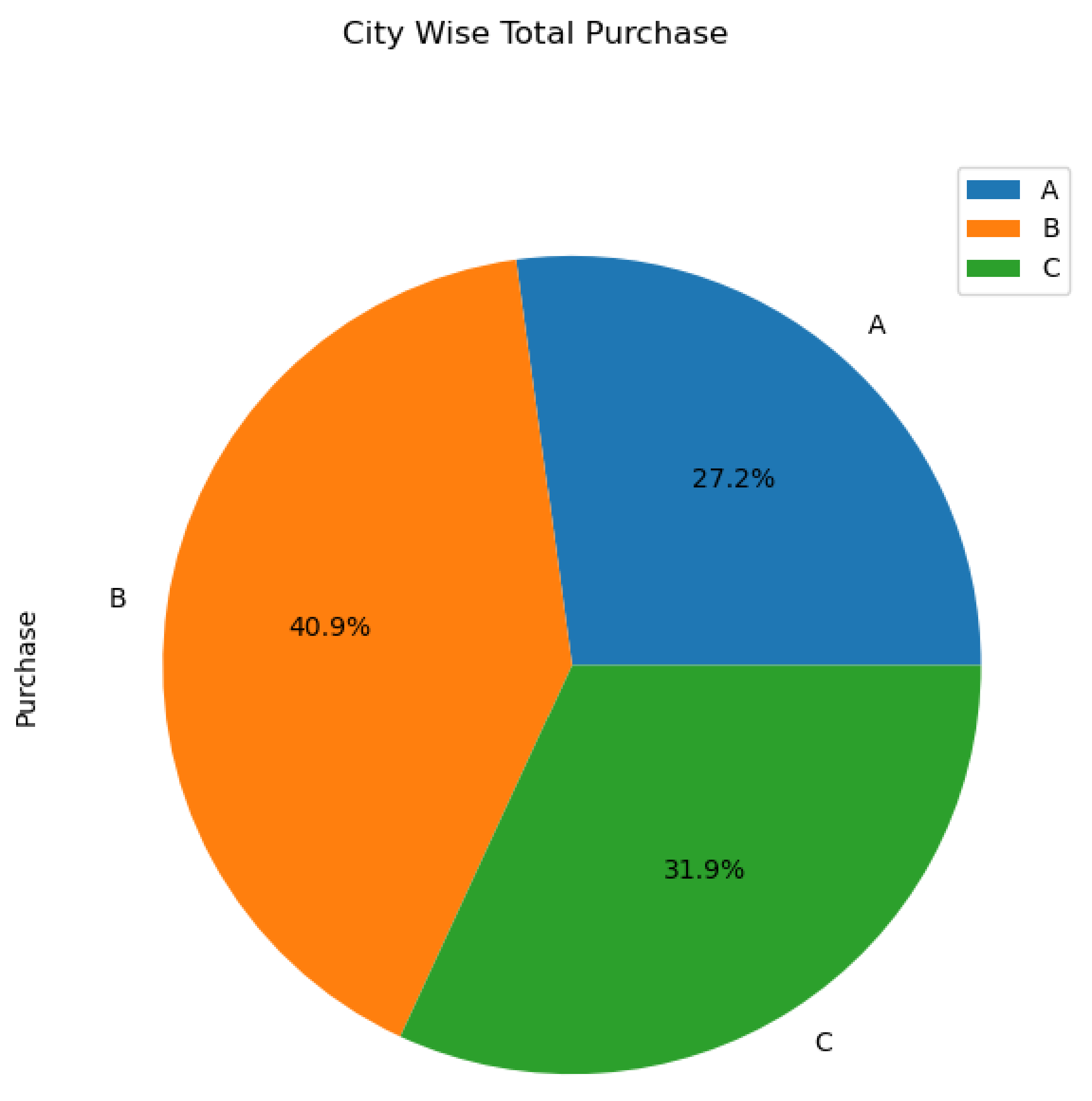
In [81]:
```
# to check the value counts City_Category(column):
df['City_Category'].value_counts()
```

Out[81]:
```
B    6179
C    4511
A    4310
Name: City_Category, dtype: int64
```

Interpretation : The city category has A, B, and C.

In [82]:
```
# Pie chart to visualize the City Category Wise distribution of total purchase:

y = df[['City_Category', 'Purchase']].groupby('City_Category').sum()
y.plot(kind='pie',autopct='%1.1f%%',subplots=True,figsize=(15,7),title='City Wise Total Purchase')
```

Out[82]:
```
array([<AxesSubplot:ylabel='Purchase'>], dtype=object)
```



Interpretation : The pie chart clearly visualise the City wise total purchase.

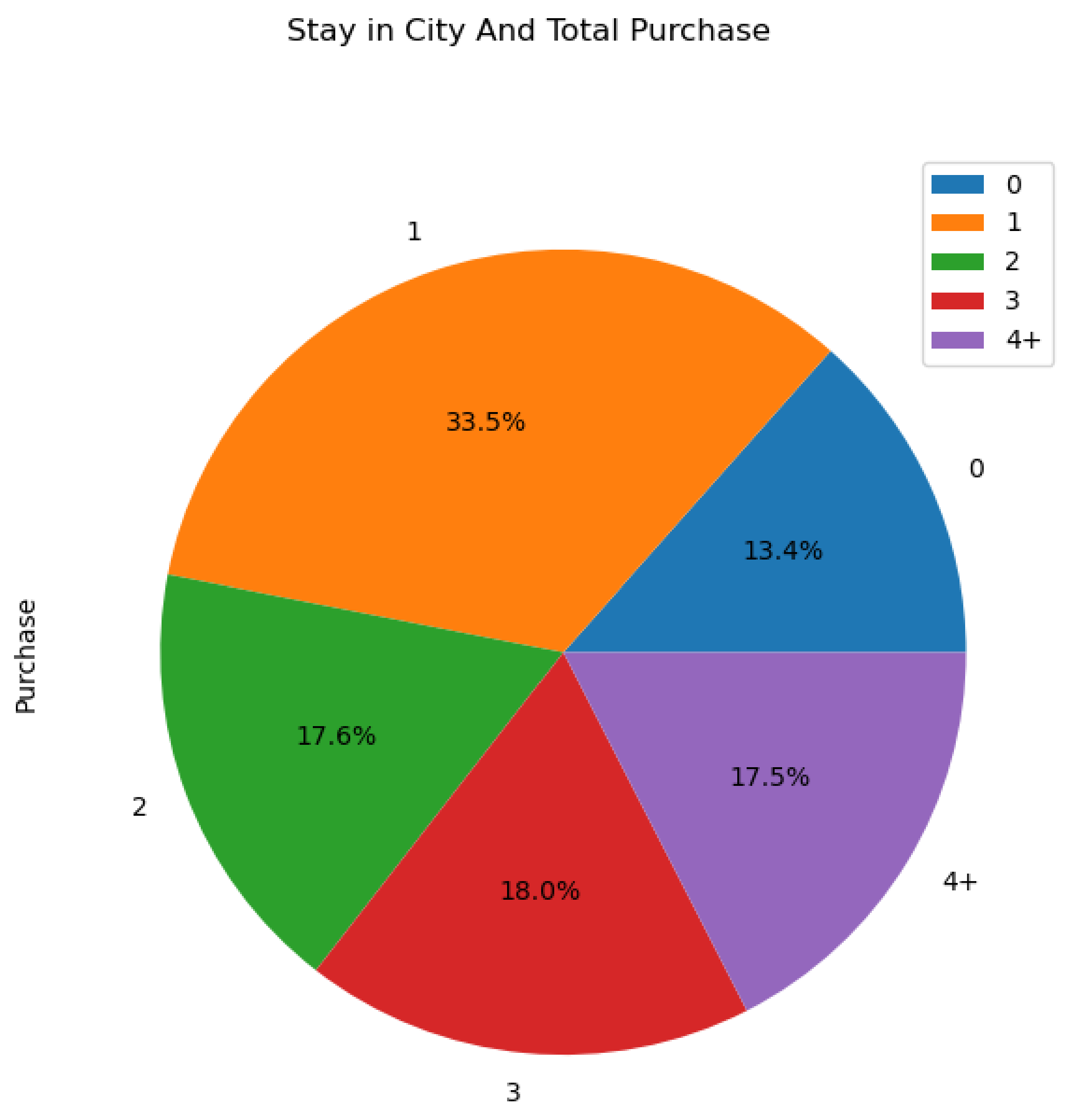
In [83]:
```
# to check the value counts Stay_In_Current_City_Years(column):
df['Stay_In_Current_City_Years'].value_counts()
```

Out[83]:
```
1     4997
3     2762
4+    2601
2     2562
0     2078
Name: Stay_In_Current_City_Years, dtype: int64
```

Interpretation the Stay_In_Current_City_Years has years counts 0,1,2,3, and 4+ years.

In [84]:
```python
# Pie chart to visualize the stay in city wise distribution of total purchase:

z= df[['Stay_In_Current_City_Years', 'Purchase']].groupby('Stay_In_Current_City_Years').sum()
z.plot(kind='pie',autopct='%1.1f%%',subplots=True,figsize=(15,7),title='Stay in City And Total Purchase')
```

Out[84]:
```
array([<AxesSubplot:ylabel='Purchase'>], dtype=object)
```



`Interpretation :` The pie chart clearly visualise the stay in city and total purchase.

ENCODING

n-1 dummy encoding

In [85]:
```python
# check head() the dataset:
df.head()
```

Out[85]:

	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	8.0	12.72	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	8.0	12.72	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	12.72	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	8.0	12.72	7969

In [86]:
```python
df.columns
```

Out[86]:
```
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 [87]:
```python
# create dummy variable for 'City_Category' :
# drop _first = 'True' creates (n - 1) dummy variables from categories:
pd.get_dummies(df, columns = ['City_Category'], drop_first = True)
```

Out[87]:

	User_ID	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	City_Category_B	City_Category_C
0	1000001	P00069042	F	0-17	10	2	0	3	8.0	12.72	8370	0	0
1	1000001	P00248942	F	0-17	10	2	0	1	6.0	14.0	15200	0	0
2	1000001	P00087842	F	0-17	10	2	0	12	8.0	12.72	1422	0	0
3	1000001	P00085442	F	0-17	10	2	0	12	14.0	12.72	1057	0	0
4	1000002	P00285442	M	55+	16	4+	0	8	8.0	12.72	7969	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
14995	1002225	P00192842	M	26-35	5	1	1	5	14.0	12.72	5217	1	0
14996	1002225	P00310642	M	26-35	5	1	1	8	8.0	12.72	1948	1	0
14997	1002228	P00070342	M	26-35	12	3	1	1	2.0	14.0	15847	0	1
14998	1002228	P00002142	M	26-35	12	3	1	1	5.0	8.0	11552	0	1
14999	1002230	P00208542	F	46-50	1	4+	1	8	8.0	12.72	3934	1	0

15000 rows × 13 columns

`Interpretation :` In this n-1 dummy encoding the city category has converted into two new columns city category 2 and city category 3, THE n-1 dummy encoding is categorical values is converted into numerical values

OneHotEncoder :

In [88]:
```python
# OneHotEncoder :

pd.get_dummies(df, columns = ['Stay_In_Current_City_Years'])
```

Out[88]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	Stay_In_Current_City_Years_0	Stay_In_Current_City_Years_1
0	1000001	P00069042	F	0-17	10	A	0	3	8.0	12.72	8370	0	0
1	1000001	P00248942	F	0-17	10	A	0	1	6.0	14.0	15200	0	0
2	1000001	P00087842	F	0-17	10	A	0	12	8.0	12.72	1422	0	0
3	1000001	P00085442	F	0-17	10	A	0	12	14.0	12.72	1057	0	0
4	1000002	P00285442	M	55+	16	C	0	8	8.0	12.72	7969	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
14995	1002225	P00192842	M	26-35	5	B	1	5	14.0	12.72	5217	0	1
14996	1002225	P00310642	M	26-35	5	B	1	8	8.0	12.72	1948	0	1
14997	1002228	P00070342	M	26-35	12	C	1	1	2.0	14.0	15847	0	0
14998	1002228	P00002142	M	26-35	12	C	1	1	5.0	8.0	11552	0	0
14999	1002230	P00208542	F	46-50	1	B	1	8	8.0	12.72	3934	0	0

15000 rows × 16 columns

Interpretation : Stay_In_Current_City_Years has 0,1,2,3,4+ these types of years are present in the dataset. In the dataset we use onehotencoding it obviosely converts into separate columns (like Stay_In_Current_City_Years_1,Stay_In_Current_City_Years_1, Stay_In_Current_City_Years_2, Stay_In_Current_City_Years_3, Stay_In_Current_City_Years_4+))

In [89]:
```python
#sklearn library
# import the OneHotEncoder:

from sklearn.preprocessing import OneHotEncoder
```

In [90]:
```python
# creating an instance of one hot encoder:

encode = OneHotEncoder()
```

In [91]:
```python
#fit transform : fit to data and return a transformed version:
# toarray() : Returns the Numpy array
# columns : add the column names

df_encode = pd.DataFrame(encode.fit_transform(df[['Stay_In_Current_City_Years']]).toarray(),
columns = ['Years_0','Years_1','Years_2','Years_3','Years_4+'])

# merge with main data dataframe (df_car):
# Axis = 1 : it stands for columns

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

df_encode
```

Out[91]:

	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	Years_0	Years_1	Years_2
0	1000001	P00069042	F	0-17	10	A	2	0	3	8.0	12.72	8370	0.0	0.0	1.0
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200	0.0	0.0	1.0
2	1000001	P00087842	F	0-17	10	A	2	0	12	8.0	12.72	1422	0.0	0.0	1.0
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	12.72	1057	0.0	0.0	1.0
4	1000002	P00285442	M	55+	16	C	4+	0	8	8.0	12.72	7969	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
14995	1002225	P00192842	M	26-35	5	B	1	1	5	14.0	12.72	5217	0.0	1.0	0.0
14996	1002225	P00310642	M	26-35	5	B	1	1	8	8.0	12.72	1948	0.0	1.0	0.0
14997	1002228	P00070342	M	26-35	12	C	3	1	1	2.0	14.0	15847	0.0	0.0	0.0
14998	1002228	P00002142	M	26-35	12	C	3	1	1	5.0	8.0	11552	0.0	0.0	0.0
14999	1002230	P00208542	F	46-50	1	B	4+	1	8	8.0	12.72	3934	0.0	0.0	0.0

15000 rows × 17 columns

Interpretation : The One-hot encoding creates new binary columns for each unique value in the original categorical column.

In [92]:
```python
# to shows the first five rows:
df_encode.head()
```

Out[92]:

	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	Years_0	Years_1	Years_2	Years_3
0	1000001	P00069042	F	0-17	10	A	2	0	3	8.0	12.72	8370	0.0	0.0	1.0	0.0
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200	0.0	0.0	1.0	0.0
2	1000001	P00087842	F	0-17	10	A	2	0	12	8.0	12.72	1422	0.0	0.0	1.0	0.0
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	12.72	1057	0.0	0.0	1.0	0.0
4	1000002	P00285442	M	55+	16	C	4+	0	8	8.0	12.72	7969	0.0	0.0	0.0	0.0

Interpretation : It shows the encode.head() --> first five rows shows.

In [96]:
```
df_encode.head(2)
```

Out[96]:

	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	Years_0	Years_1	Years_2	Years_3
0	1000001	P00069042	F	0-17	10	A	2	0	3	8.0	12.72	8370	0.0	0.0	1.0	0.0
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200	0.0	0.0	1.0	0.0

In [97]:
```
#  Label encoding:
# use sk learn library:

from sklearn.preprocessing import LabelEncoder

# create an instance:

labelencoder = LabelEncoder()

# fit the encoder:

df['Gender'] = labelencoder.fit_transform(df.Gender)

# display the data:

df.head(3)
```

Out[97]:

	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	0	0-17	10	A	2	0	3	8.0	12.72	8370
1	1000001	P00248942	0	0-17	10	A	2	0	1	6.0	14.0	15200
2	1000001	P00087842	0	0-17	10	A	2	0	12	8.0	12.72	1422

In [98]:
```
df.Gender.value_counts()
```

Out[98]:
```
1    11490
0     3510
Name: Gender, dtype: int64
```

Interpretation : The Label encoding is converting categorical data into numerical data. It is the gender has M (male) and F (female) is converting into male has --> 1 and female has --> 0

In [99]:
```
# example
# use sk learn library:

from sklearn.preprocessing import LabelEncoder

# create an instance:

labelencoder = LabelEncoder()

# fit the encoder:

df['City_Category'] = labelencoder.fit_transform(df.City_Category)

# display the data:

df.head(3)
```

Out[99]:

	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	0	0-17	10	0	2	0	3	8.0	12.72	8370
1	1000001	P00248942	0	0-17	10	0	2	0	1	6.0	14.0	15200
2	1000001	P00087842	0	0-17	10	0	2	0	12	8.0	12.72	1422

Interpretation : The Label encoding is converting categorical data into numerical data. City category has (A, B, C) the categorical data is converting into numerical data (0, 1, 2)

In [100]:
```
# for value counts (city_category)
df.City_Category.value_counts()
```

Out[100]:
```
1    6179
2    4511
0    4310
Name: City_Category, dtype: int64
```

Interpretation : City category has (A, B, C) the categorical data is converting into numerical data (0, 1, 2)

StandardScaler

In [101]:
```
from sklearn.preprocessing import StandardScaler

# minimum and maximum values of 'Purchase':
# '\n' - add space

print('minimum value before transformation : ', df.Purchase.min(), '\n'
     'maximum value before transformation :', df.Purchase.max(), '\n')

# insantinate the standardscaler:

standard_scale = StandardScaler()

#fit the standardscaler:
# fit_transform() : returns a transformed data

df['Scaled_Purchase'] = standard_scale.fit_transform(df[['Purchase']])

print('minimum value after transformation : ', df['Scaled_Purchase'].min(), '\n'
     'maximum value after transformation :', df['Scaled_Purchase'].max(), '\n')
```

```
minimum value before transformation :  186
maximum value before transformation : 23958

minimum value after transformation :  -1.8357511165050533
maximum value after transformation : 3.030814345225164
```

Interpretation : # scaled value mean = 0 , # scaled value Sd = 1

In [102]:
```
print('Mean: ', df['Scaled_Purchase'].mean())
print('\n')
print('Standard deviation :', df['Scaled_Purchase'].std())
```

```
Mean:  9.792167077193881e-17


Standard deviation : 1.000033335000094
```

Interpretation : The transformed values are stored in a new column called 'Scaled_Purchase' in the DataFrame.

In [103]:
```
# head of the dataset:
df.head()
```

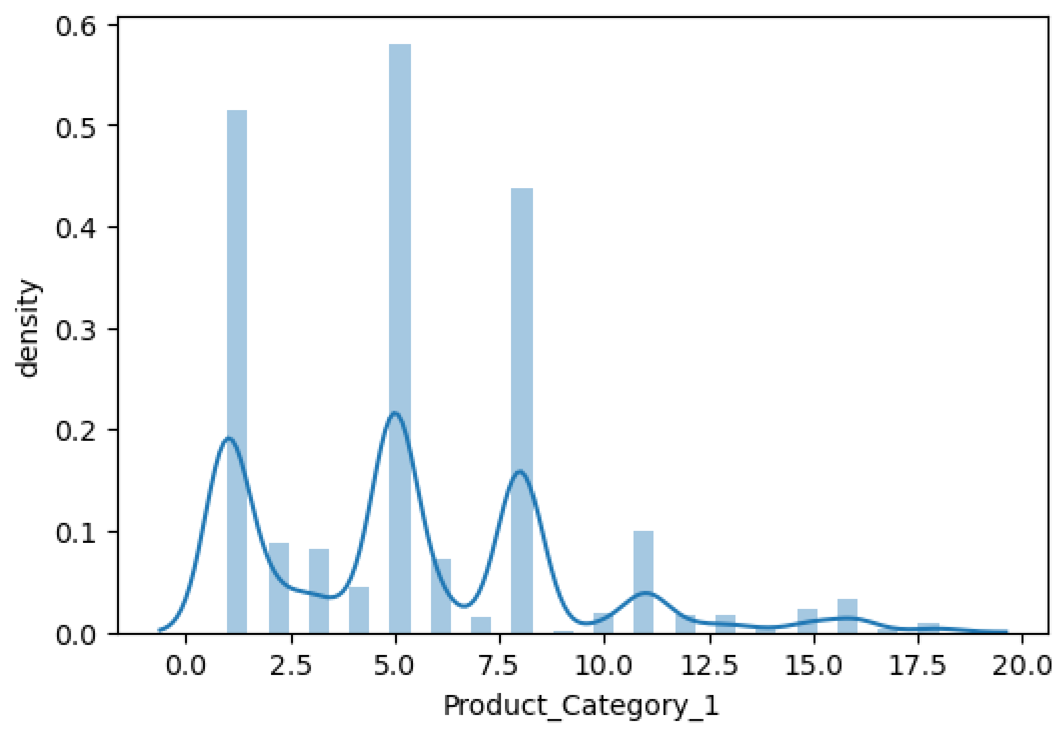
Out[103]:

	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	Scaled_Purchase
0	1000001	P00069042	0	0-17	10	0	2	0	3	8.0	12.72	8370	-0.160336
1	1000001	P00248942	0	0-17	10	0	2	0	1	6.0	14.0	15200	1.237891
2	1000001	P00087842	0	0-17	10	0	2	0	12	8.0	12.72	1422	-1.582719
3	1000001	P00085442	0	0-17	10	0	2	0	12	14.0	12.72	1057	-1.657441
4	1000002	P00285442	1	55+	16	2	4+	0	8	8.0	12.72	7969	-0.242428

DATA TRANSFORMATION

In [104]:
```
# set the figure size:
plt.rcParams['figure.figsize'] = [6,4]

# distribution of the displacement:
sns.distplot(df['Product_Category_1'])

plt.ylabel('density')

# coefficient of skewness:
print('Skewness: ', df['Product_Category_1'].skew())

plt.show()
```

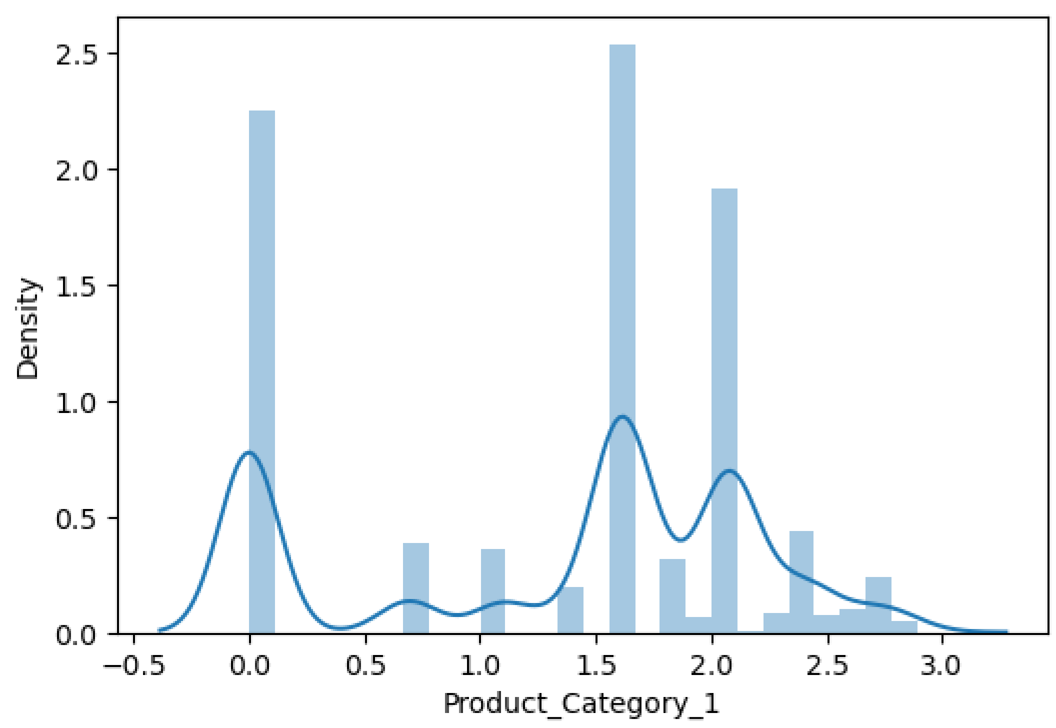
```
Skewness:  0.8166481551781835
```



Interpretation : In the above dataset the product category has 0.8166 positive skewness and to visualise clearly by using distplot.

In [105]:
```
# apply natural log transformation with (base 'e')
log_Product_Category_1 = np.log(df['Product_Category_1'])

# coefficient of skewness:
print('skewness:', log_Product_Category_1.skew())

# distribution of log_transformed variable:
sns.distplot(log_Product_Category_1)

# set label for y - axis:
plt.ylabel('Density')

# display the plot:
plt.show()
```

```
skewness: -0.5161452988650036
```



Interpretation : In the above skewness is reduced to -0.5141

Outlier Detection

In [106]:
```
# outlier detection based on boxplot:
```

In [107]:
```
# obtain numerical
df_num = df.select_dtypes(include = [np.number])
```

In [108]:
```
# df_num head of the dataset :
df_num.head()
```

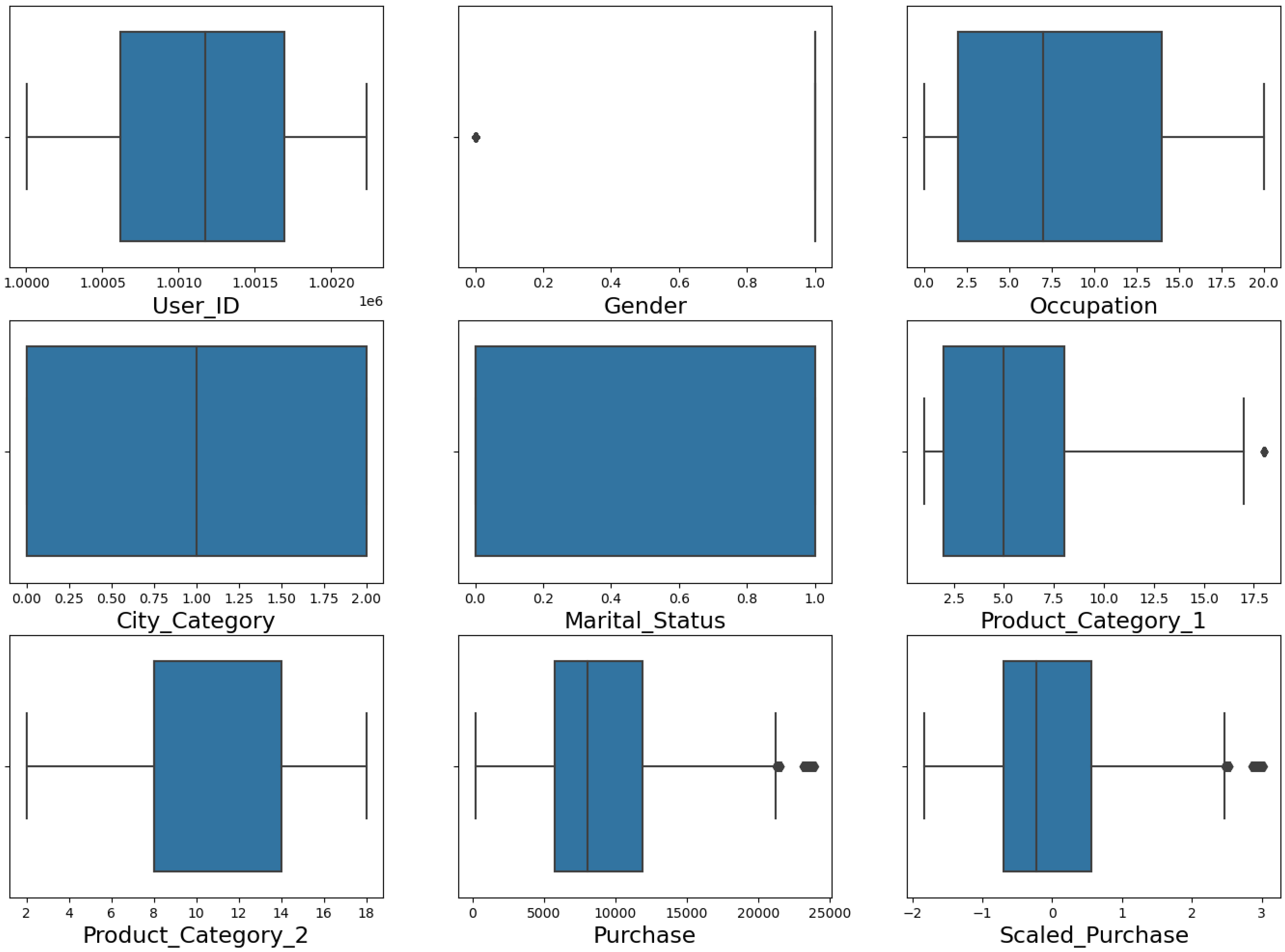
Out[108]:

	User_ID	Gender	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Purchase	Scaled_Purchase
0	1000001	0	10	0	0	3	8.0	8370	-0.160336
1	1000001	0	10	0	0	1	6.0	15200	1.237891
2	1000001	0	10	0	0	12	8.0	1422	-1.582719
3	1000001	0	10	0	0	12	14.0	1057	-1.657441
4	1000002	1	16	2	0	8	8.0	7969	-0.242428

In [104]:
```
# display the column names:
df_num.columns
```

Out[104]:
```
Index(['User_ID', 'Gender', 'Occupation', 'City_Category', 'Marital_Status',
       'Product_Category_1', 'Product_Category_2', 'Purchase',
       'Scaled_Purchase'],
      dtype='object')
```

In [105]:
```
# plot the boxplot for each columns:
# subplots() : plot subplots:
# figsize(): set the figure size:

fig, ax = plt.subplots(3, 3, figsize = (17, 12))



# plot the boxplot using boxplot() from seaborn library:
# Z : let the variable z defines the boxplot:
# x : data from which the boxplot is to be predicted:
# orient : 'h' specifies the horizontal boxplot
# whisker : proportion of the IQR , it past the low and high quartiles to extend the plot
# ax : specifies the axes object to draw the plot
# set_xlabel() : set the x axis label
# fontsize () : set the font size of the x- axis
for variable, subplot in zip(df_num.columns, ax.flatten()):
    z = sns.boxplot(x = df_num[variable], orient = 'h', whis = 1.5, ax = subplot)
    z.set_xlabel(variable, fontsize = 17)
```



`Interpretation :` In this dataset (purchase , scaled_purchase , and product category has outliers)

In [ ]:
```
# based on IQR method
# based on first quartile:
Q1 = df_num.quantile(0.25)

# obtain the quartile:
Q3 = df_num.quantile(0.75)

# to obtain the IQR:
IQR =  Q3 - Q1

# print the IQR:
print(IQR)
```

In [ ]:
```
`Interpretation :` Interquartile range  IQR method clearly
```

In [107]:
```
# filter out the outlier values:
#  ~ : select all the rows which do not satisfy the condition
#  any( ) : returns whether the elements is True over the columns
# axis = 1 : indicates should select the alternate columns('0' for index positions)

from warnings import filterwarnings
filterwarnings('ignore')

# IQR  formula:
df_sales_iqr = df[~((df < (Q1 - 1.5 * IQR ))| (df > (Q3 + 1.5 * IQR))).any(axis = 1)]
```

Interpretation : IQR formula iqr = df[~((df < (Q1 - 1.5 * IQR))| (df > (Q3 + 1.5 * IQR))).any(axis = 1)]

In [108]:
```
# modified
df_sales_iqr.shape
# all outliers are removed
```

Out[108]:
```
(11375, 13)
```

In [ ]:
```
`Interpretation :` all outliers are removed by using IQR method. after removed the outliers the shape of the dataset is (11375,13)
```

In [109]:
```
df.shape
```

Out[109]:
```
(15000, 13)
```

In [ ]:
```
`Interpretation :` The original shape of the dataset is (15000,13)
```

In [110]:
```
# based on z score
```

In [111]:
```
df.head()
```

Out[111]:

	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	Scaled_Purchase
0	1000001	P00069042	0	0-17	10	0	2	0	3	8.0	12.72	8370	-0.160336
1	1000001	P00248942	0	0-17	10	0	2	0	1	6.0	14.0	15200	1.237891
2	1000001	P00087842	0	0-17	10	0	2	0	12	8.0	12.72	1422	-1.582719
3	1000001	P00085442	0	0-17	10	0	2	0	12	14.0	12.72	1057	-1.657441
4	1000002	P00285442	1	55+	16	2	4+	0	8	8.0	12.72	7969	-0.242428

In [112]:
```
df.tail()
```

Out[112]:

	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	Scaled_Purchase
14995	1002225	P00192842	1	26-35	5	1	1	1	5	14.0	12.72	5217	-0.805813
14996	1002225	P00310642	1	26-35	5	1	1	1	8	8.0	12.72	1948	-1.475037
14997	1002228	P00070342	1	26-35	12	2	3	1	1	2.0	14.0	15847	1.370344
14998	1002228	P00002142	1	26-35	12	2	3	1	1	5.0	8.0	11552	0.491078
14999	1002230	P00208542	0	46-50	1	1	4+	1	8	8.0	12.72	3934	-1.068467

In [113]:
```
# check the value counts of purchase column:
df.Purchase.value_counts()
```

Out[113]:
```
7992     12
7187     12
5415     12
7068     11
5267     11
         ..
15231     1
12587     1
16584     1
4197      1
4365      1
Name: Purchase, Length: 7308, dtype: int64
```

In [114]:
```
# import library.
import scipy

# from scipy import stats module:
from scipy import stats

# z -score are defined for each observation in a variable
# compute the z - score using the method z score from the scipy library
z_scores_price = scipy.stats.zscore(df_num['Purchase'])

# display the z - score:
z_scores_price
```

Out[114]:
```
0       -0.160336
1        1.237891
2       -1.582719
3       -1.657441
4       -0.242428
           ...
14995   -0.805813
14996   -1.475037
14997    1.370344
14998    0.491078
14999   -1.068467
Name: Purchase, Length: 15000, dtype: float64
```

In [115]:
```
# printing the rows where the z - score is less than -3
row_index_less = np.where(z_scores_price < -3)

print(row_index_less)
```

```
(array([], dtype=int64),)
```

In [ ]:
```
`Interpretaion :` there is no row index less by using the less than -3
```

In [116]:
```
row_index_great = np.where(z_scores_price > 3)

print(row_index_great)
```

```
(array([ 1445,  6543,  6585,  6911,  7542,  9201, 10016, 13013],
      dtype=int64),)
```

In [ ]:
```
`Interpretaion :` there are 8 outliers in the row index great. by usnig the greater than +3
```

In [117]:
```
# count of outliers in the var representing purchare:

len(row_index_less[0]) + len(row_index_great[0])
```
Out[117]:
```
8
```

`Interpretaion :` there are total 8 outliers

In [118]:
```
# filter out the outlier values:
# ~ : select all the rows which do not satisfy the condition

df_Zscore_puurchase = df['Purchase'][~((z_scores_price < -3)|(z_scores_price > 3))]
```

`Interpretaion :` z - score of a value is the diff b/w that values and the mean divided by the standard deviation . if the z - score greater than +3 or less than -3

In [119]:
```
# check for the shape
df_Zscore_purchase.shape
```
Out[119]:
```
(14992,)
```

`Interpretaion :` the shape of the purchase column is 14992 . there are 8 outliers are removed.

In [120]:
```
# original dataset shape
df.shape
```
Out[120]:
```
(15000, 13)
```

In [ ]:
```
`Interpretaion :` the shape of the dataset is 15000 rows and 13 columns
```

In [121]:
```
# from sklearn library
import sklearn

# from testtrain
from sklearn.model_selection import train_test_split
```

`Interpretation :` The process of splitting a dataset into training and testing sets is known as "train-test split."

In [122]:
```
# select the target column:
Y = df['Purchase']

# select the independent column:
# by drop the target column:
X = df.drop(['Purchase'], axis = 1)
```

In [123]:
```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25,
                                                    random_state = 100)

print('X_train : ', x_train.shape)
print('X_test : ', x_test.shape)
print('Y_train : ', y_train.shape)
print('Y_train : ', y_train.shape)
```
```
X_train :  (11250, 12)
X_test :  (3750, 12)
Y_train :  (11250,)
Y_train :  (11250,)
```

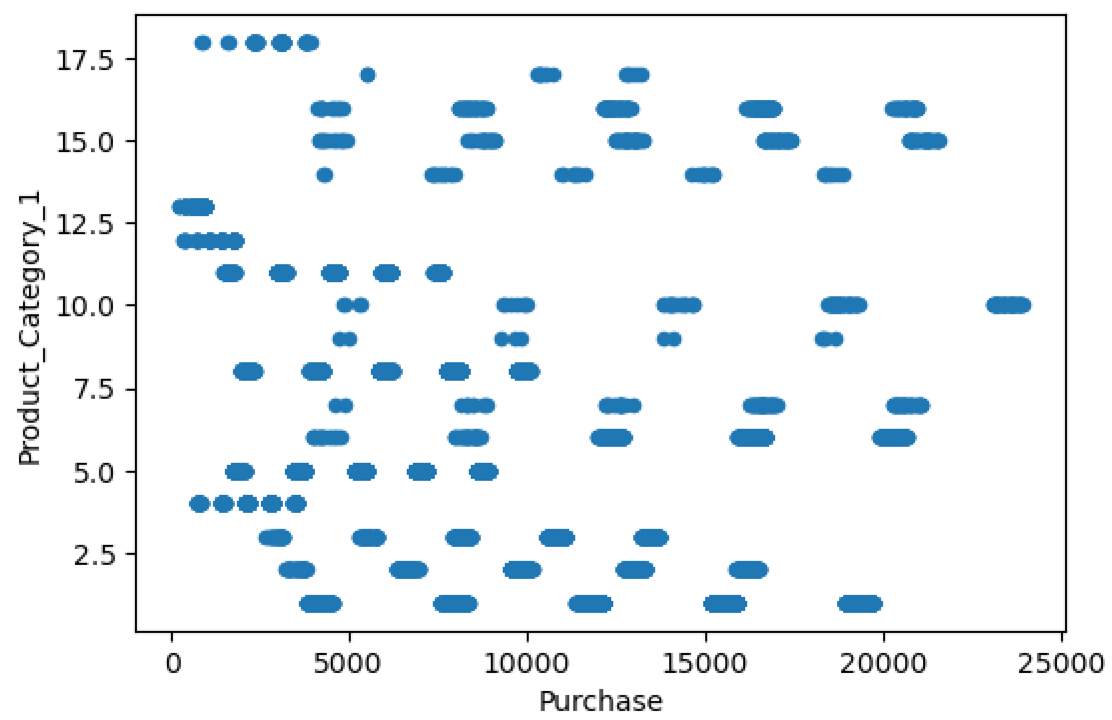
In [124]:
```
#Purchase is a target column
df.Purchase.value_counts()
```
Out[124]:
```
7992     12
7187     12
5415     12
7068     11
5267     11
         ..
15231     1
12587     1
16584     1
4197      1
4365      1
Name: Purchase, Length: 7308, dtype: int64
```

LINEAR REGRESSION

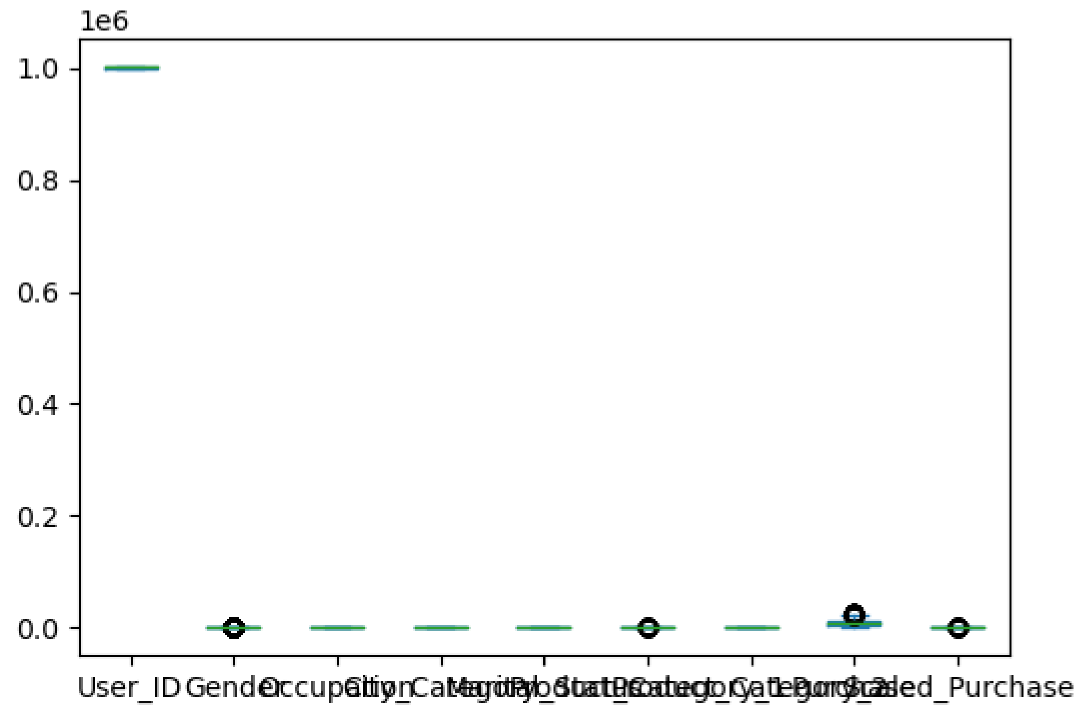
In [125]:
```
df.head()
```
Out[125]:

	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	Scaled_Purchase
0	1000001	P00069042	0	0-17	10	0	2	0	3	8.0	12.72	8370	-0.160336
1	1000001	P00248942	0	0-17	10	0	2	0	1	6.0	14.0	15200	1.237891
2	1000001	P00087842	0	0-17	10	0	2	0	12	8.0	12.72	1422	-1.582719
3	1000001	P00085442	0	0-17	10	0	2	0	12	14.0	12.72	1057	-1.657441
4	1000002	P00285442	1	55+	16	2	4+	0	8	8.0	12.72	7969	-0.242428

In [126]:
```


df.plot(kind = 'scatter', x = 'Purchase', y = 'Product_Category_1')
plt.show()
```



In [127]:
```
df.plot(kind = 'box')
plt.show()
```



In [128]:
```
df.corr()
```

Out[128]:

	User_ID	Gender	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Purchase	Scaled_Purchase
User_ID	1.000000	-0.002963	-0.008156	0.061503	0.015080	0.009531	0.001414	-0.022909	-0.022909
Gender	-0.002963	1.000000	0.140781	-0.016628	0.006261	-0.046522	0.000729	0.061357	0.061357
Occupation	-0.008156	0.140781	1.000000	0.020314	-0.014599	0.003827	-0.002444	0.002050	0.002050
City_Category	0.061503	-0.016628	0.020314	1.000000	-0.002945	-0.042062	-0.009424	0.082377	0.082377
Marital_Status	0.015080	0.006261	-0.014599	-0.002945	1.000000	0.009962	0.011479	0.001673	0.001673
Product_Category_1	0.009531	-0.046522	0.003827	-0.042062	0.009962	1.000000	0.307734	-0.326389	-0.326389
Product_Category_2	0.001414	0.000729	-0.002444	-0.009424	0.011479	0.307734	1.000000	-0.129082	-0.129082
Purchase	-0.022909	0.061357	0.002050	0.082377	0.001673	-0.326389	-0.129082	1.000000	1.000000
Scaled_Purchase	-0.022909	0.061357	0.002050	0.082377	0.001673	-0.326389	-0.129082	1.000000	1.000000

In [109]:
```
# change to the dataframe variable:
occupation = pd.DataFrame(df['Occupation'])
purchase = pd.DataFrame(df['Purchase'])
```

`Interpretation :` occupation is the independent column and Purchase is the target column.

In [110]:
```
import sklearn

from sklearn.linear_model import LinearRegression
```

In [111]:
```
# making instances:

lm = LinearRegression()
model = lm.fit(occupation, purchase)
```

In [112]:
```
model.coef_
```

Out[112]:
```
array([[1.4907212]])
```

In [113]:
```
model.intercept_
```

Out[113]:
```
array([9140.7240673])
```

In [114]:
```
model.score(occupation, purchase)
```

Out[114]:
```
4.203195765883905e-06
```

In [115]:
```
import statsmodels
import statsmodels.api as sm
```

In [116]:
```
# predict the target columns
# predict the new value of weight:
from warnings import filterwarnings
filterwarnings('ignore')

occupation_new = np.array([97])
occupation_new = occupation_new.reshape(-1, 1)
purchase_predict = model.predict(occupation_new)
purchase_predict
```

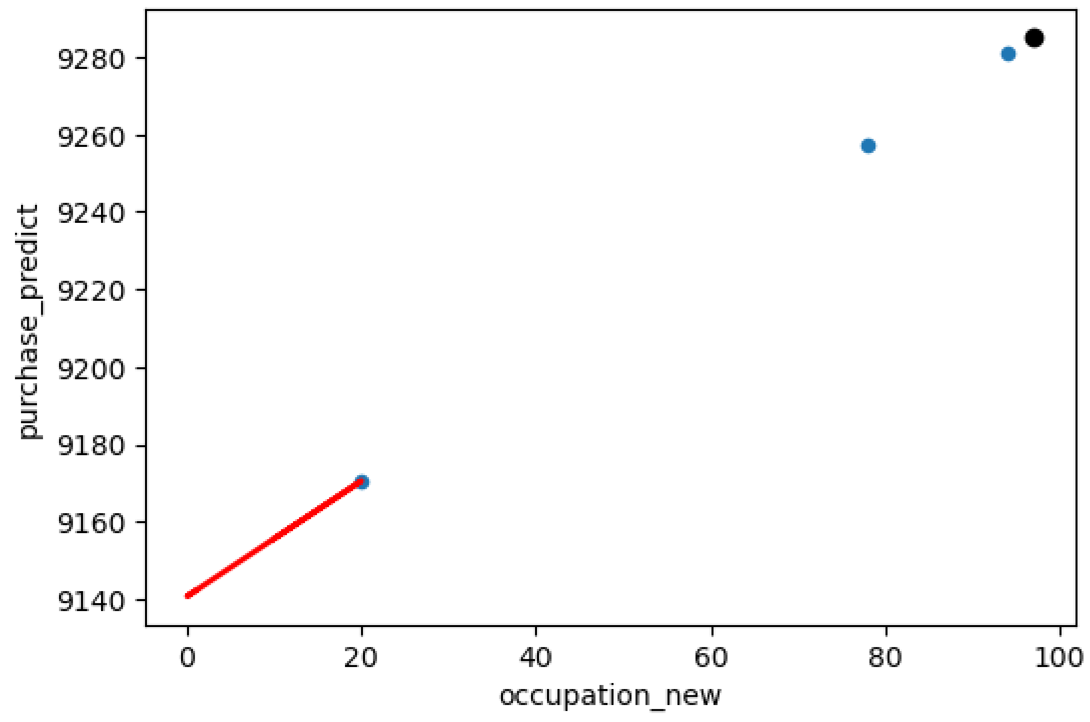
Out[116]:
```
array([[9285.32402329]])
```

In [117]:
```
# predict more values:
X = ([20,78,94])
X = pd.DataFrame(X)
Y = model.predict(X)
Y = pd.DataFrame(Y)

df = pd.concat([X, Y], axis = 1, keys = ['occupation_new', 'purchase_predict'] )
df
```

Out[117]:

	occupation_new	purchase_predict
	0	0
0	20	9170.538491
1	78	9257.000321
2	94	9280.851860

In [120]:
```
# visual the result :
df.plot(kind = 'scatter', x = 'occupation_new', y = 'purchase_predict')

# plot the regressin line:
plt.plot(occupation, model.predict(occupation), color = 'r', linewidth = 2)

# plotting the predicted values:
plt.scatter(occupation_new,purchase_predict , color = 'black')
plt.show()
```



In [121]:
```
model = sm.OLS(X, Y)
```

In [122]:
```
fit = model.fit()
```

In [123]:
```
fit.pvalues
```

Out[123]:
```
0    0.102135
dtype: float64
```

In [124]:
```
fit.summary()
```

Out[124]:

OLS Regression Results

Dep. Variable:	0	**R-squared (uncentered):**	0.806
Model:	OLS	**Adj. R-squared (uncentered):**	0.709
Method:	Least Squares	**F-statistic:**	8.318
Date:	Wed, 01 Mar 2023	**Prob (F-statistic):**	0.102
Time:	23:20:28	**Log-Likelihood:**	-14.603
No. Observations:	3	**AIC:**	31.21
Df Residuals:	2	**BIC:**	30.30
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
0	0.0069	0.002	2.884	0.102	-0.003	0.017

Omnibus:	nan	**Durbin-Watson:**	1.194
Prob(Omnibus):	nan	**Jarque-Bera (JB):**	0.447
Skew:	-0.575	**Prob(JB):**	0.800
Kurtosis:	1.500	**Cond. No.**	1.00

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

PROJECT SUMMARY

We have take Black Friday Sales Dataset from kaggle.

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, citytype, stayincurrentcity), product details (productid and product category) and Total purchaseamount 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.