Importing necessary libraries

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

In [2]:
```
data = pd.read_csv('/Users/ashleshad/Downloads/Black Friday Sales.csv')
```

In [3]:
```
data.head(4)
```

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_
0	1000001	P00069042	F	0-17	10	A	2	0	3	Na
1	1000001	P00248942	F	0-17	10	A	2	0	1	6
2	1000001	P00087842	F	0-17	10	A	2	0	12	Na
3	1000001	P00085442	F	0-17	10	A	2	0	12	14

In [4]:
```
data.shape
```

Out[4]:
```
(550068, 12)
```

In [5]:
```
data.describe()
```

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]:
```
data.info()
```

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

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
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 [8]:
```
data.nunique()
```

Out[8]:
```
User_ID                       5891
Product_ID                    3631
Gender                           2
Age                              7
Occupation                      21
City_Category                    3
Stay_In_Current_City_Years       5
Marital_Status                   2
Product_Category_1              20
Product_Category_2              17
Product_Category_3              15
Purchase                     18105
dtype: int64
```

In [9]:
```
data['Purchase'].skew()
```

Out[9]:
```
0.6001400037087128
```

In [10]:
```
data['Purchase'].kurtosis()
```

Out[10]:
```
-0.3383775655851702
```

In [11]:
```
data['Product_Category_2'].mean()
```

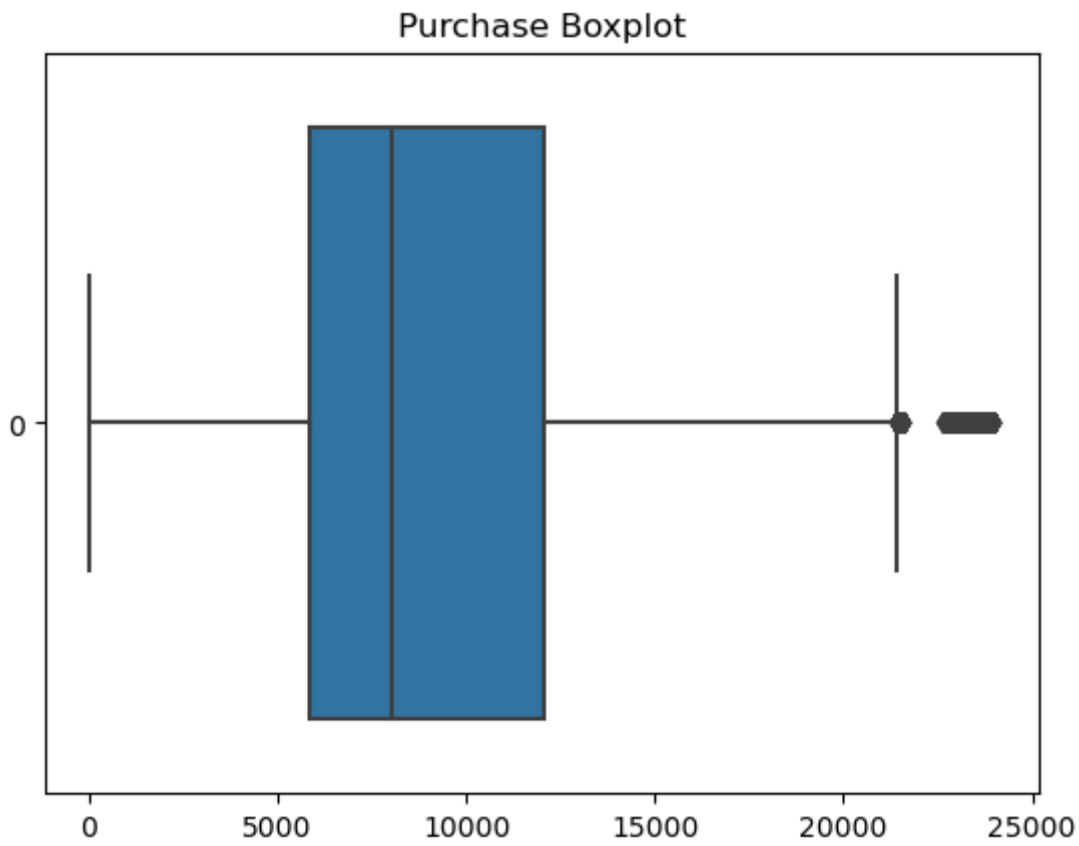
Out[11]:
```
9.842329251122386
```

In [12]:
```
data['Product_Category_3'].mean()
```

Out[12]:
```
12.668243206790512
```

EDA

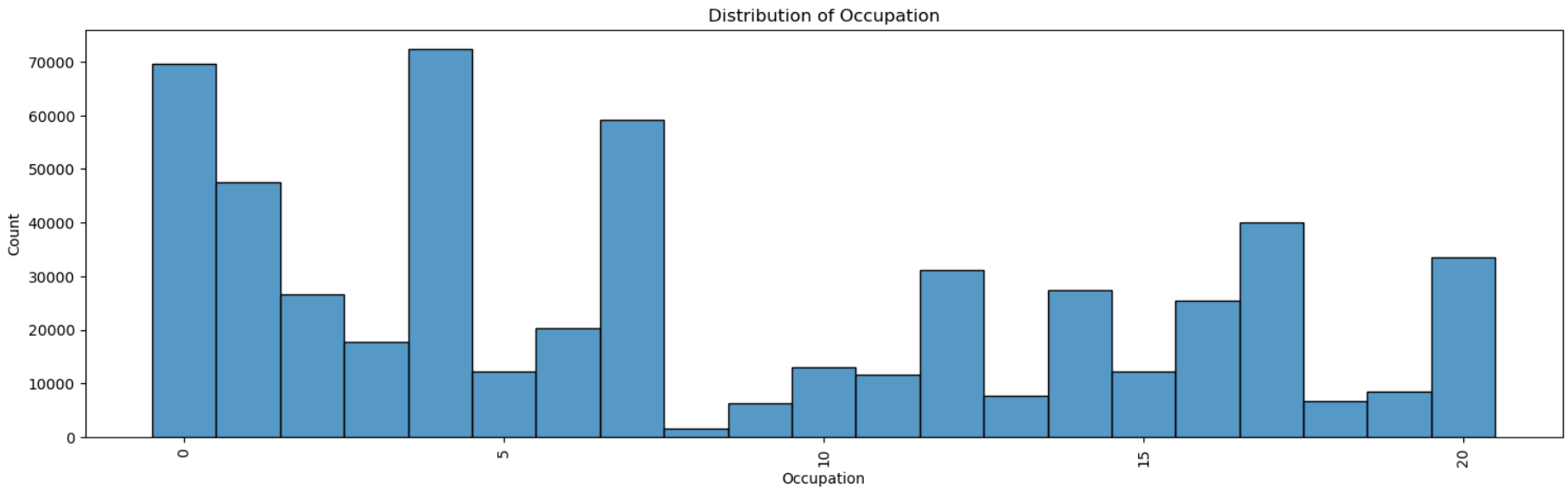
In [13]:
```
sns.boxplot(data["Purchase"], orient='h')
plt.title("Purchase Boxplot")
plt.show()
```



In [14]:
```
data['Gender'].unique()
```

Out[14]:
```
array(['F', 'M'], dtype=object)
```

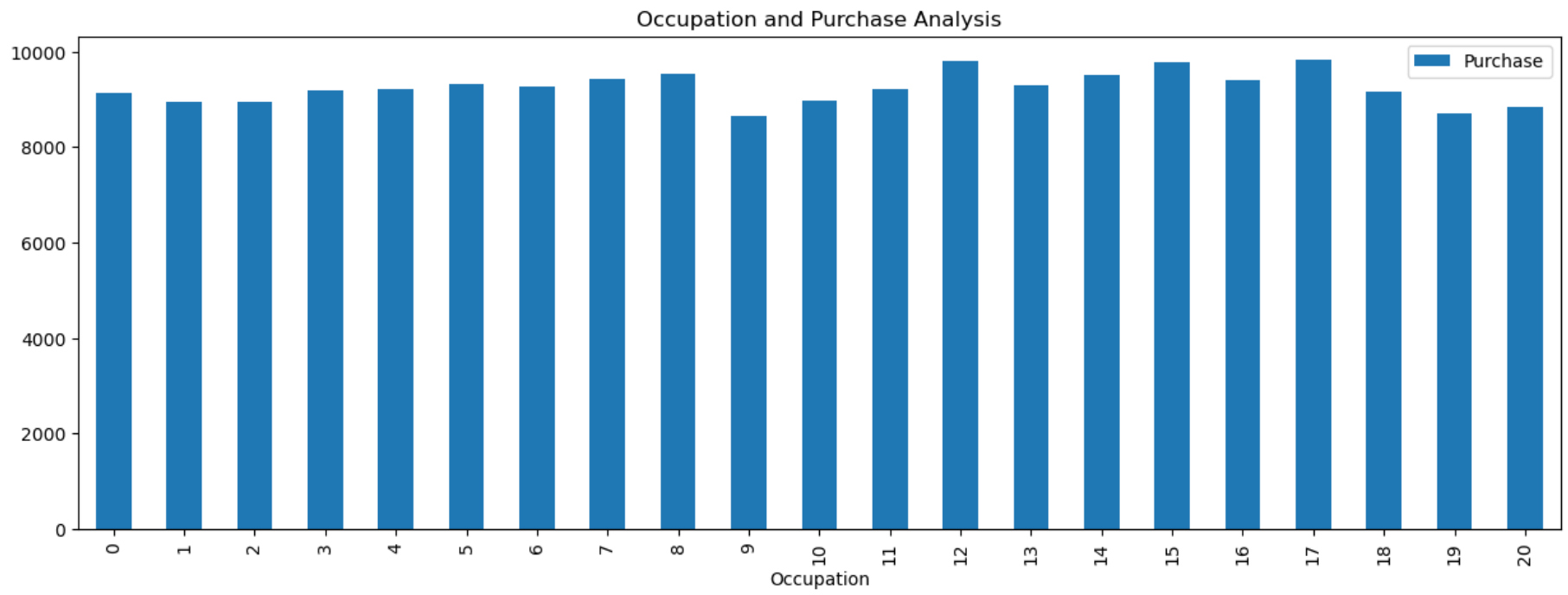
In [15]:
```
plt.figure(figsize=(18, 5))
sns.histplot(data['Occupation'], bins=20, kde=False, discrete=True)
plt.xlabel('Occupation')
plt.ylabel('Count')
plt.title('Distribution of Occupation')
plt.xticks(rotation=90)
plt.show()
```



Occupation has at least 20 different values. Since we don't know each number corresponds to what occupation , it is difficult to make any analysis.

In [16]:
```
occupation_group = pd.DataFrame(data.groupby("Occupation").mean()["Purchase"])
occupation_group.plot(kind='bar',figsize=(15,5))
plt.title("Occupation and Purchase Analysis")
plt.show()
```

```
/var/folders/_j/tzw6wdvd1fv_1s_66tcw6my40000gn/T/ipykernel_1775/1768259721.py:1: FutureWarning: The default
value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will defaul
t to False. Either specify numeric_only or select only columns which should be valid for the function.
  occupation_group = pd.DataFrame(data.groupby("Occupation").mean()["Purchase"])
```



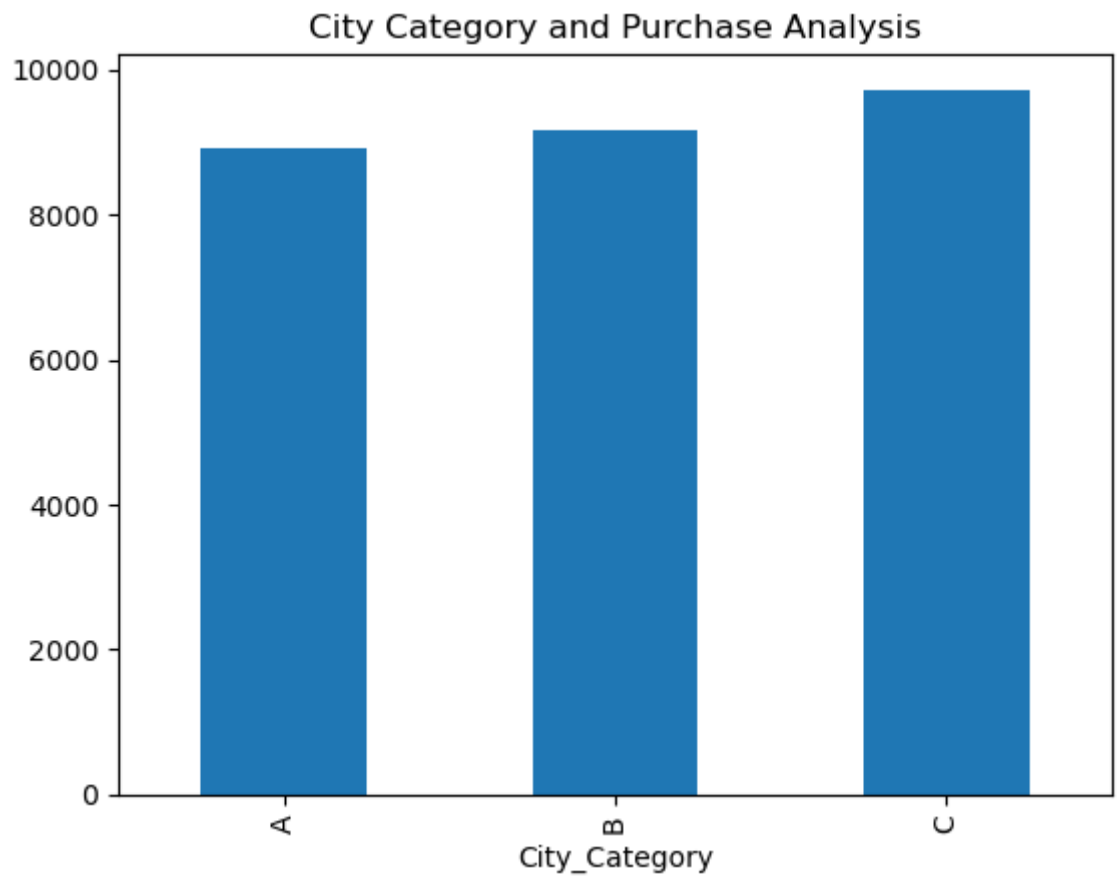
Some occupations which have higher representations, it seems that the amount each customer spends on average is more or less the same for all occupations.

In [17]:
```
# Check the unique values in the 'City_Category' column
print(data['City_Category'].unique())
```

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

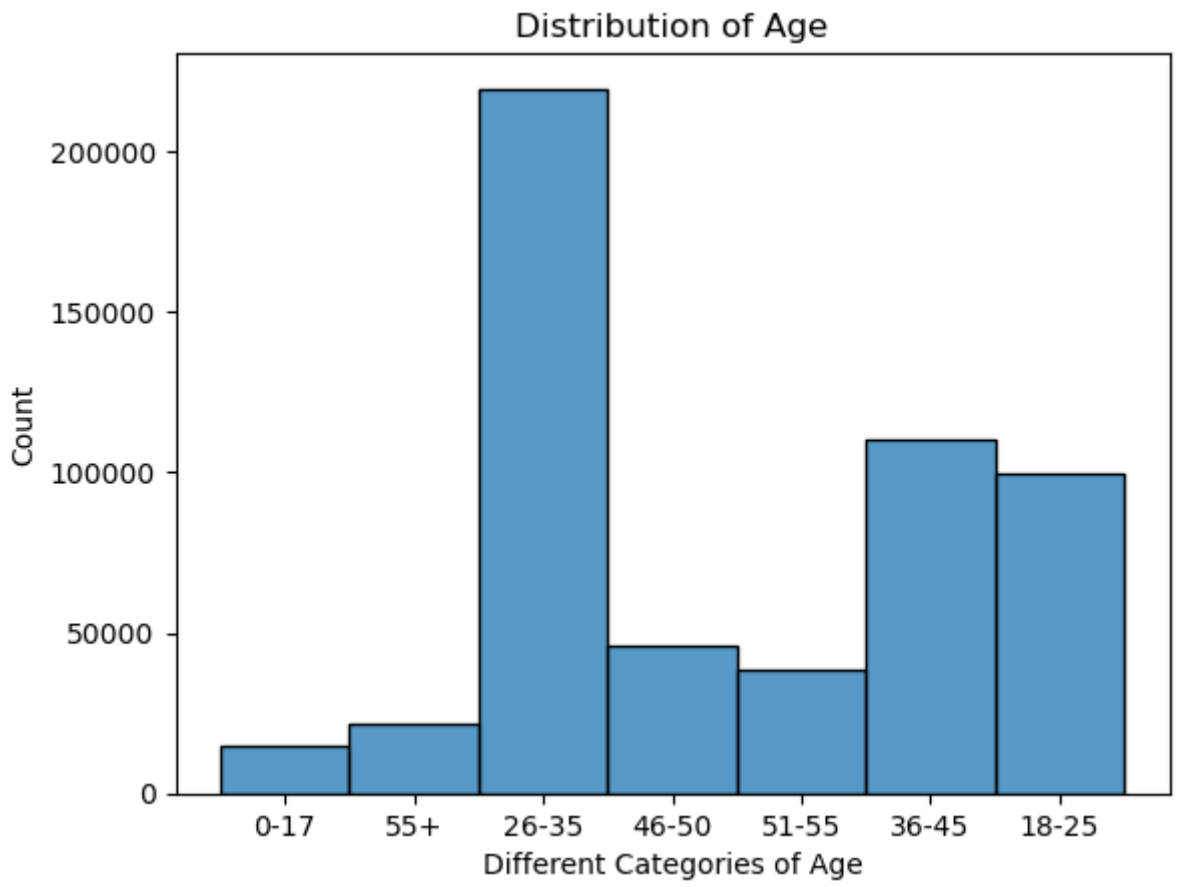
In [18]:
```
data.groupby("City_Category").mean()["Purchase"].plot(kind='bar')
plt.title("City Category and Purchase Analysis")
plt.show()
```

```
/var/folders/_j/tzw6wdvd1fv_1s_66tcw6my40000gn/T/ipykernel_1775/1068414020.py:1: FutureWarning: The default
value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will defaul
t to False. Either specify numeric_only or select only columns which should be valid for the function.
  data.groupby("City_Category").mean()["Purchase"].plot(kind='bar')
```



City whose buyers spend the most is city type ‘C’.

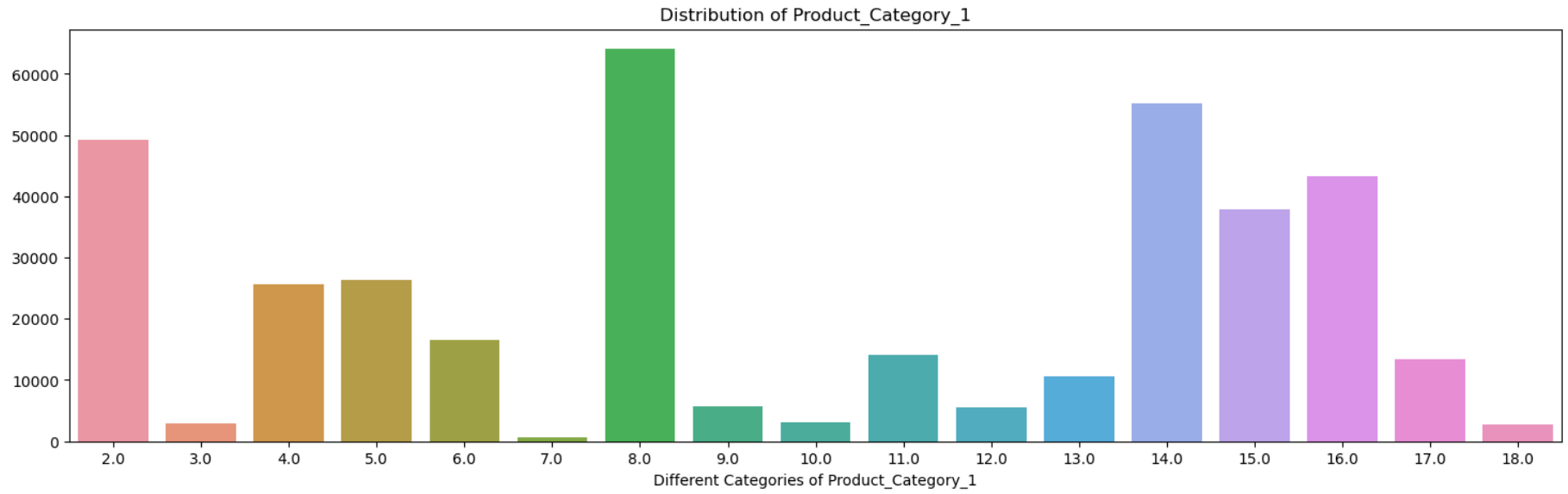
In [19]:
```
sns.histplot(x='Age', data=data)
plt.title('Distribution of Age')
plt.xlabel('Different Categories of Age')
plt.show()
```



26–35 age seems to have more purchase amount

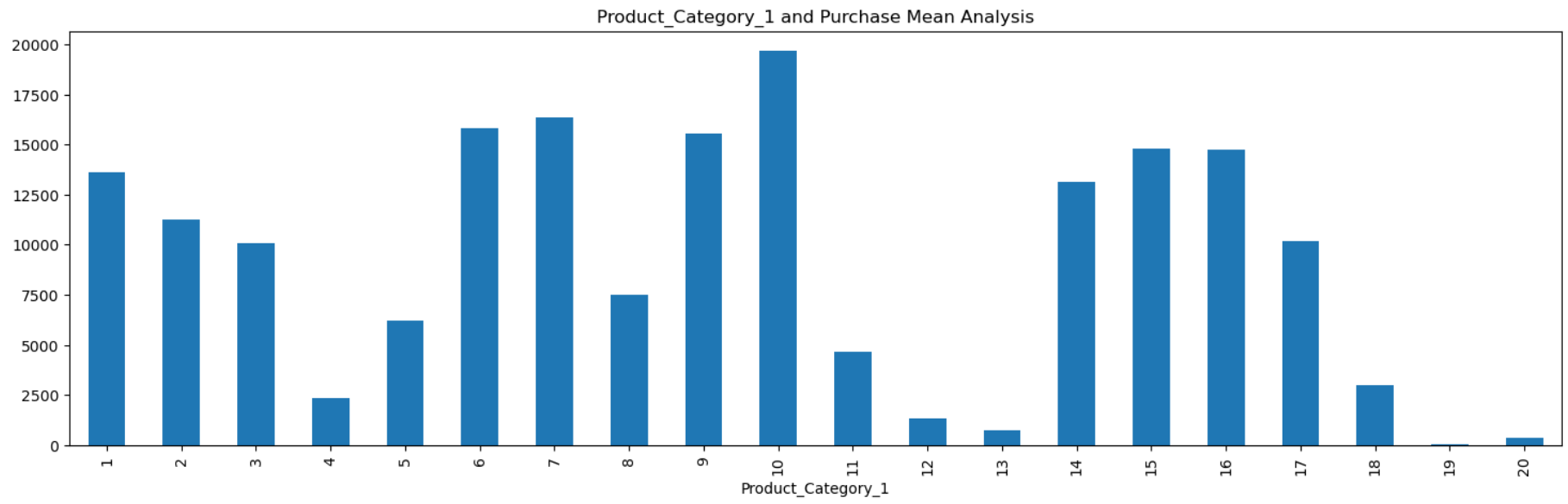
In [20]:
```
Product_Category_1_counts = data['Product_Category_2'].value_counts()

plt.figure(figsize=(18,5))
sns.barplot(x=Product_Category_1_counts.index, y=Product_Category_1_counts.values)
plt.title('Distribution of Product_Category_1')
plt.xlabel('Different Categories of Product_Category_1')
plt.show()
```



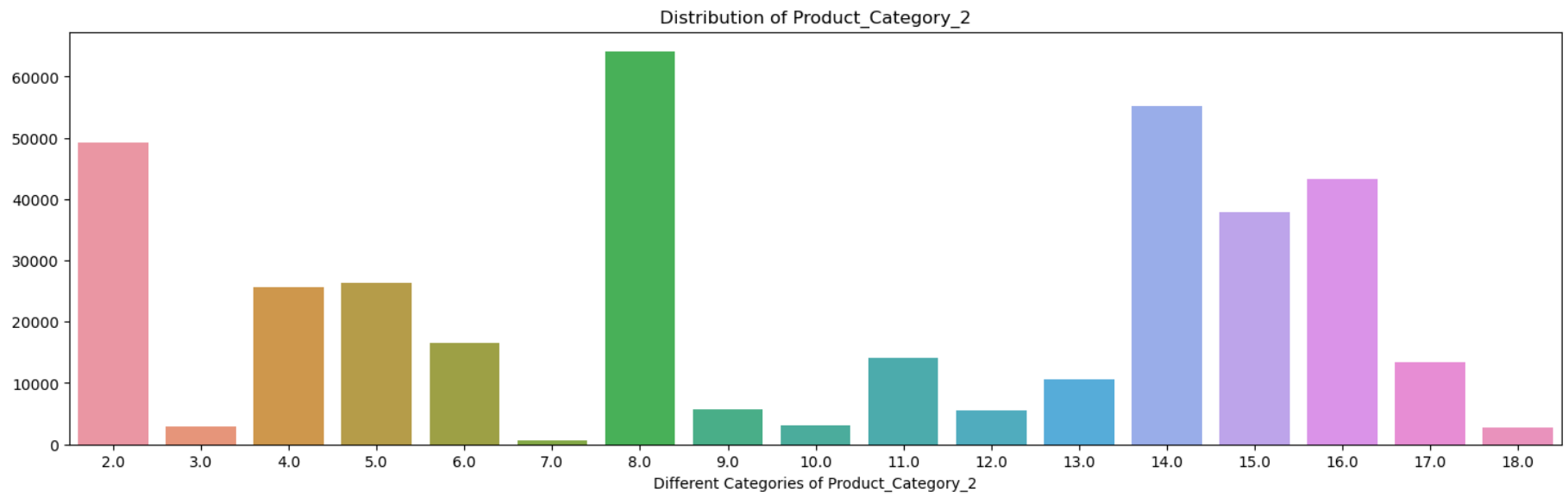
In [21]:
```
data.groupby('Product_Category_1').mean()['Purchase'].plot(kind='bar',figsize=(18,5))
plt.title("Product_Category_1 and Purchase Mean Analysis")
plt.show()
```

```
/var/folders/_j/tzw6wdvd1fv_1s_66tcw6my40000gn/T/ipykernel_1775/3827400744.py:1: FutureWarning: The default
value of numeric_only in DataFrameGroupBy.mean is deprecated. In a future version, numeric_only will defaul
t to False. Either specify numeric_only or select only columns which should be valid for the function.
  data.groupby('Product_Category_1').mean()['Purchase'].plot(kind='bar',figsize=(18,5))
```



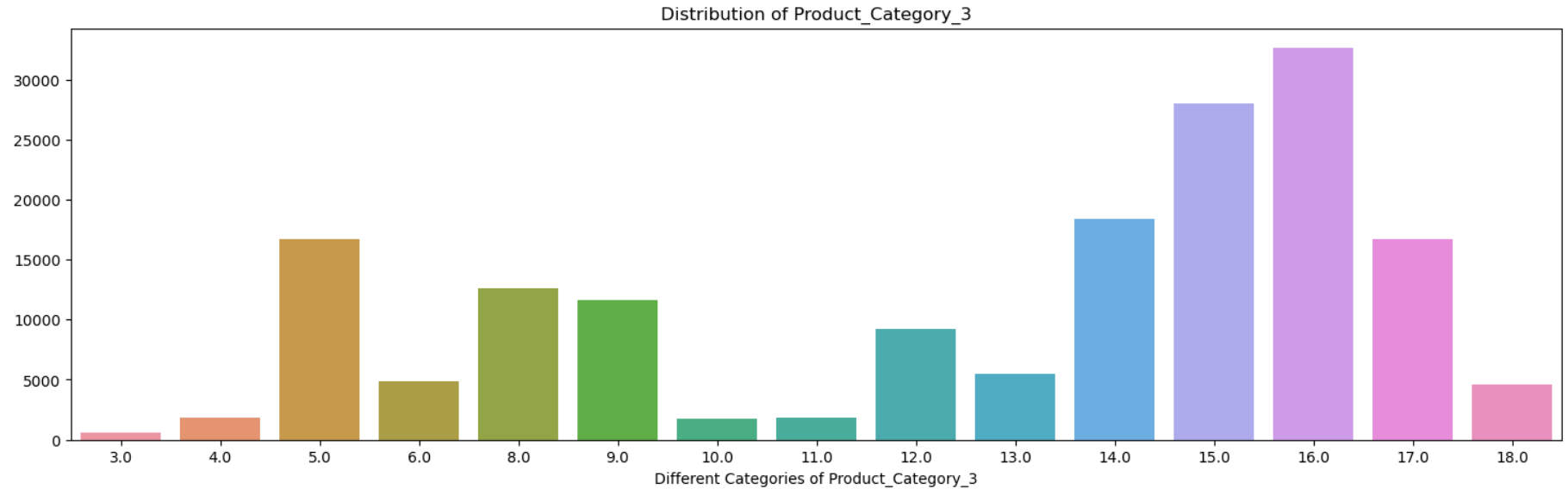
In [22]:
```
Product_Category_2_counts = data['Product_Category_2'].value_counts()

plt.figure(figsize=(18,5))
sns.barplot(x=Product_Category_2_counts.index, y=Product_Category_2_counts.values)
plt.title('Distribution of Product_Category_2')
plt.xlabel('Different Categories of Product_Category_2')
plt.show()
```



In [23]:
```
Product_Category_3_counts = data['Product_Category_3'].value_counts()

plt.figure(figsize=(18,5))
sns.barplot(x=Product_Category_3_counts.index, y=Product_Category_3_counts.values)
plt.title('Distribution of Product_Category_3')
plt.xlabel('Different Categories of Product_Category_3')
plt.show()
```



In [24]:
```
data.corr()
```

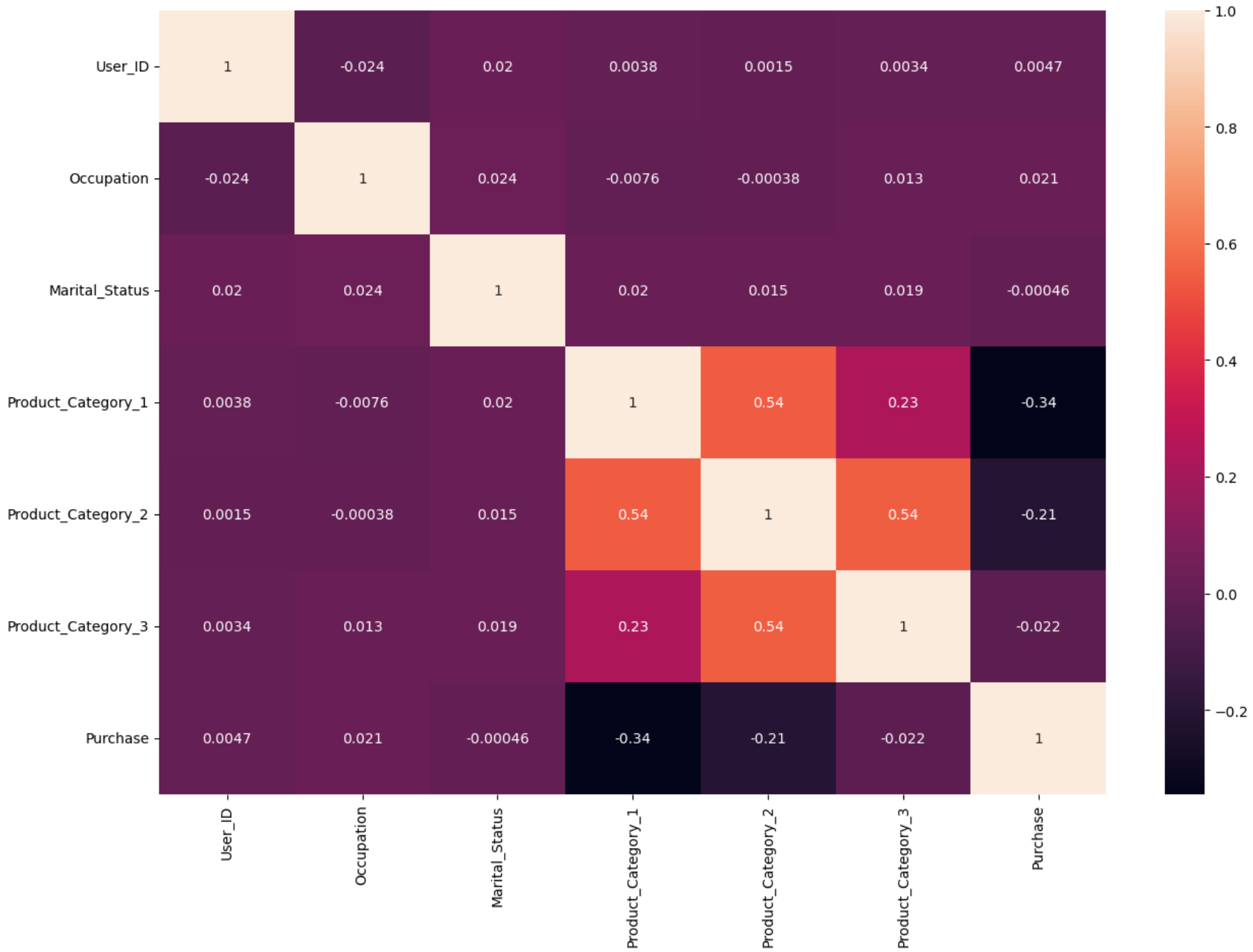
```
/var/folders/_j/tzw6wdvd1fv_1s_66tcw6my40000gn/T/ipykernel_1775/2627137660.py:1: FutureWarning: The default
value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Selec
t only valid columns or specify the value of numeric_only to silence this warning.
  data.corr()
```

Out[24]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
User_ID	1.000000	-0.023971	0.020443	0.003825	0.001529	0.003419	0.004716
Occupation	-0.023971	1.000000	0.024280	-0.007618	-0.000384	0.013263	0.020833
Marital_Status	0.020443	0.024280	1.000000	0.019888	0.015138	0.019473	-0.000463
Product_Category_1	0.003825	-0.007618	0.019888	1.000000	0.540583	0.229678	-0.343703
Product_Category_2	0.001529	-0.000384	0.015138	0.540583	1.000000	0.543649	-0.209918
Product_Category_3	0.003419	0.013263	0.019473	0.229678	0.543649	1.000000	-0.022006
Purchase	0.004716	0.020833	-0.000463	-0.343703	-0.209918	-0.022006	1.000000

In [25]:
```
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(),annot=True)
plt.show()
```

```
/var/folders/_j/tzw6wdvd1fv_1s_66tcw6my40000gn/T/ipykernel_1775/291057077.py:2: FutureWarning: The default
value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Selec
t only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(),annot=True)
```



In [26]:
```
from sklearn.preprocessing import LabelEncoder
lr = LabelEncoder()
data['Gender'] = lr.fit_transform(data['Gender'])
data['Age'] = lr.fit_transform(data['Age'])
data['City_Category'] = lr.fit_transform(data['City_Category'])
data.head()
```

Out[26]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category
0	1000001	P00069042	0	0	10	0	2	0	3	Na
1	1000001	P00248942	0	0	10	0	2	0	1	6
2	1000001	P00087842	0	0	10	0	2	0	12	Na
3	1000001	P00085442	0	0	10	0	2	0	12	14
4	1000002	P00285442	1	6	16	2	4+	0	8	Na

Checking for null values

In [27]:
```
data.isnull().sum()
```

Out[27]:
```
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 [28]:
```
data['Product_Category_2'].mean().round(2)
```

Out[28]:
```
9.84
```

In [29]:
```
data['Product_Category_3'].mean().round()
```

Out[29]:
```
13.0
```

In [30]:
```
data=data.drop(['User_ID','Product_ID','Stay_In_Current_City_Years'],axis=1)
```

Filling null values with mean

In [31]:
```
data['Product_Category_2'].fillna(9.84, inplace=True)
data['Product_Category_3'].fillna(13.0, inplace=True)
```

In [32]:
```
data.head()
```

Out[32]:

	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
0	0	0	10	0	0	3	9.84	13.0	8370
1	0	0	10	0	0	1	6.00	14.0	15200
2	0	0	10	0	0	12	9.84	13.0	1422
3	0	0	10	0	0	12	14.00	13.0	1057
4	1	6	16	2	0	8	9.84	13.0	7969

In [33]:
```
data.isnull().sum()
```

Out[33]:
```
Gender                 0
Age                    0
Occupation             0
City_Category          0
Marital_Status         0
Product_Category_1     0
Product_Category_2     0
Product_Category_3     0
Purchase               0
dtype: int64
```

Splitting data into independent and dependent variables

In [34]:
```
X = data.drop("Purchase",axis=1)
```

In [35]:
```
y=data['Purchase']
```

In [36]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=41)
```

ML Model

Linear Regression

In [37]:
```
from sklearn.linear_model import LinearRegression
```

In [38]:
```
lr = LinearRegression()
lr.fit(X_train,y_train)
```

Out[38]:
```
▾ LinearRegression

LinearRegression()
```

In [39]:
```
y_pred = lr.predict(X_test)
```

In [40]:
```
y_pred
```

Out[40]:
```
array([ 9156.97183037,  9042.57960256,  7359.77194994, ...,
        6162.66673352,  9366.99082864, 10704.05847464])
```

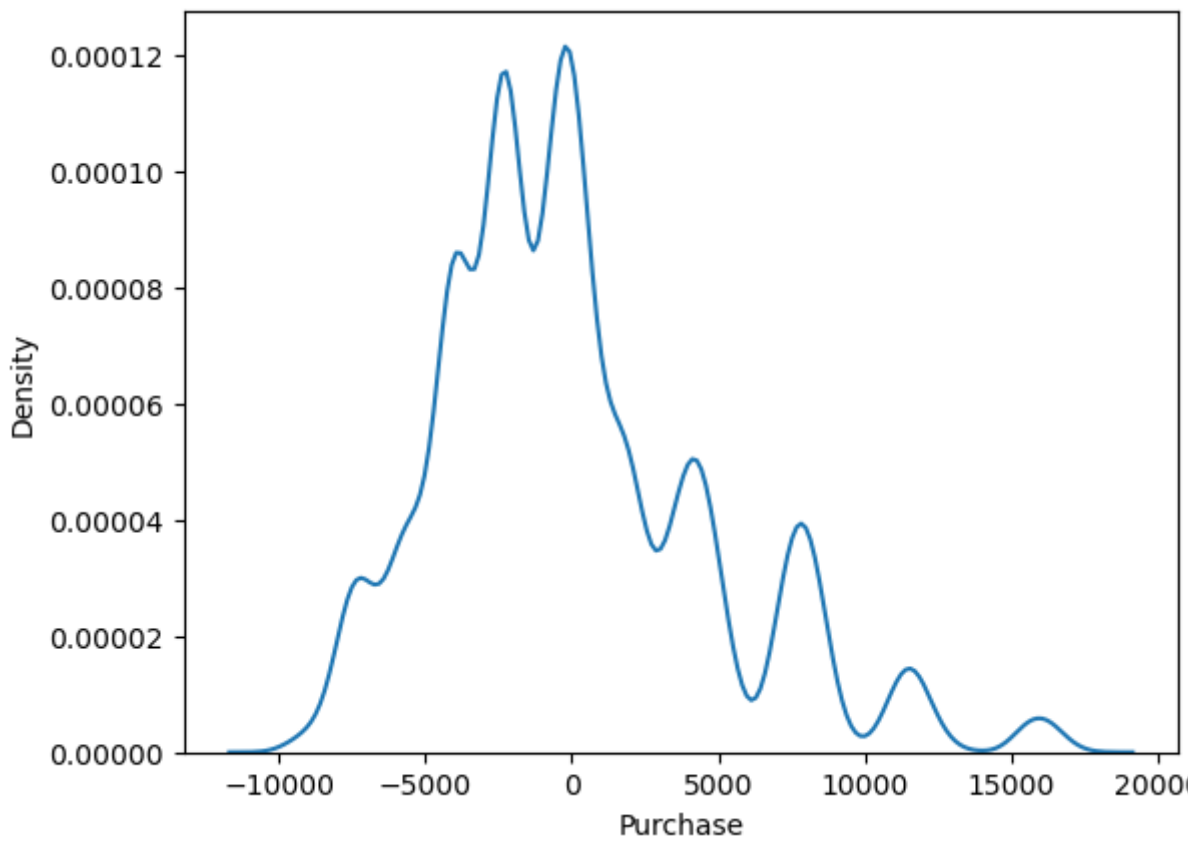
In [41]:
```
y_test
```

Out[41]:
```
522367     7143
171727     6862
239520     7569
265009     7084
371594    16253
          ...
49797      3901
67002      9754
516894     7384
536007     7071
259884     2871
Name: Purchase, Length: 165021, dtype: int64
```

In [42]:
```
residuals = y_test - y_pred
```

In [43]:
```
sns.kdeplot(residuals)
```

Out[43]:
```
<Axes: xlabel='Purchase', ylabel='Density'>
```



In [44]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error, r2_score
```

In [45]:
```
mean_absolute_error(y_test, y_pred)
```

Out[45]:
```
3592.353438936207
```

In [46]:
```
mean_squared_error(y_test, y_pred)
```

Out[46]:
```
22023371.237467762
```

In [47]:
```
r2_score(y_test, y_pred)
```

Out[47]:
```
0.1265548783541176
```

In [48]:
```
from math import sqrt
print("RMSE of Linear Regression Model is ",sqrt(mean_squared_error(y_test, y_pred)))
```

```
RMSE of Linear Regression Model is  4692.906480792874
```

DecisionTreeRegressor

In [49]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [50]:
```
DTC = DecisionTreeRegressor()
```

In [51]:
```
DTC.fit(X_train, y_train)
```

Out[51]:
```
DecisionTreeRegressor
DecisionTreeRegressor()
```

In [52]:
```
y_pred_dtc = DTC.predict(X_test)
```

In [53]:
```
mean_absolute_error(y_test, y_pred_dtc)
```

Out[53]:
```
2306.8656359484653
```

In [54]:
```
mean_squared_error(y_test, y_pred_dtc)
```

Out[54]:
```
10194960.323624445
```

In [55]:
```
r2_score(y_test, y_pred_dtc)
```

Out[55]:
```
0.5956686983101974
```

In [56]:
```
print("RMSE of Linear Regression Model is ",sqrt(mean_squared_error(y_test, y_pred_dtc)))
```

```
RMSE of Linear Regression Model is  3192.9547951113314
```

Random Forest Regressor

In [57]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [58]:
```
RFR = RandomForestRegressor(random_state = 4)
```

In [59]:
```
RFR.fit(X_train, y_train)
```

Out[59]:
```
RandomForestRegressor
RandomForestRegressor(random_state=4)
```

In [60]:
```
y_pred_rfr = RFR.predict(X_test)
```

In [61]:
```
mean_absolute_error(y_test, y_pred_rfr)
```

Out[61]:
```
2235.5304953426935
```

In [62]:
```
mean_squared_error(y_test, y_pred_rfr)
```

Out[62]:
```
9248741.084503233
```

In [63]:
```
r2_score(y_test, y_pred_rfr)
```

Out[63]:
```
0.6331956767871278
```

In [64]:
```
print("RMSE of Linear Regression Model is ",sqrt(mean_squared_error(y_test, y_pred_rfr)))
```

```
RMSE of Linear Regression Model is  3041.1742936739474
```