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

In [2]:
```
df = pd.read_csv("Black Friday Sales.csv")
```

In [3]:
```
df.head()
```

Out[3]:

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

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

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

In [5]:
```
df.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 [6]:
```
df.describe()
```

Out[6]:

	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 [7]:
```
df.dtypes
```

Out[7]:
```
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
```

In [8]:
```
df.isnull().sum()
```

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

In [9]:
```
df['Product_Category_2'] = df['Product_Category_2'].fillna(df['Product_Category_2'].median())
```

In [10]:
```
df['Product_Category_3'] = df['Product_Category_3'].fillna(df['Product_Category_3'].median())
```

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
User_ID                     0
Product_ID                  0
Gender                      0
Age                         0
Occupation                  0
City_Category               0
Stay_In_Current_City_Years  0
Marital_Status              0
Product_Category_1          0
Product_Category_2          0
Product_Category_3          0
Purchase                    0
dtype: int64
```

In [12]:
```
df.head()
```

Out[12]:

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

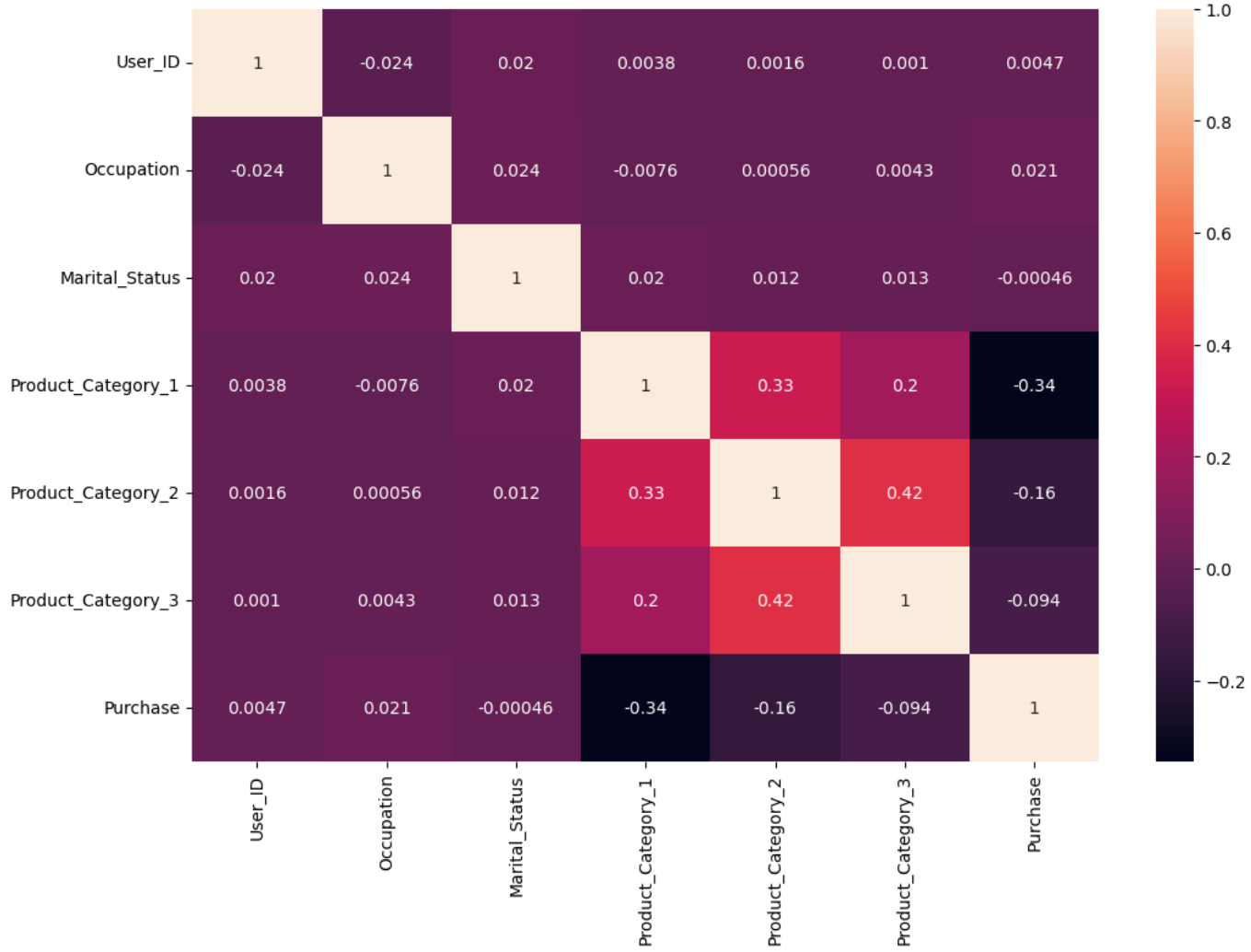
In [13]:
```
df.nunique()
```

Out[13]:
```
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
```

DATA VISUALZITION

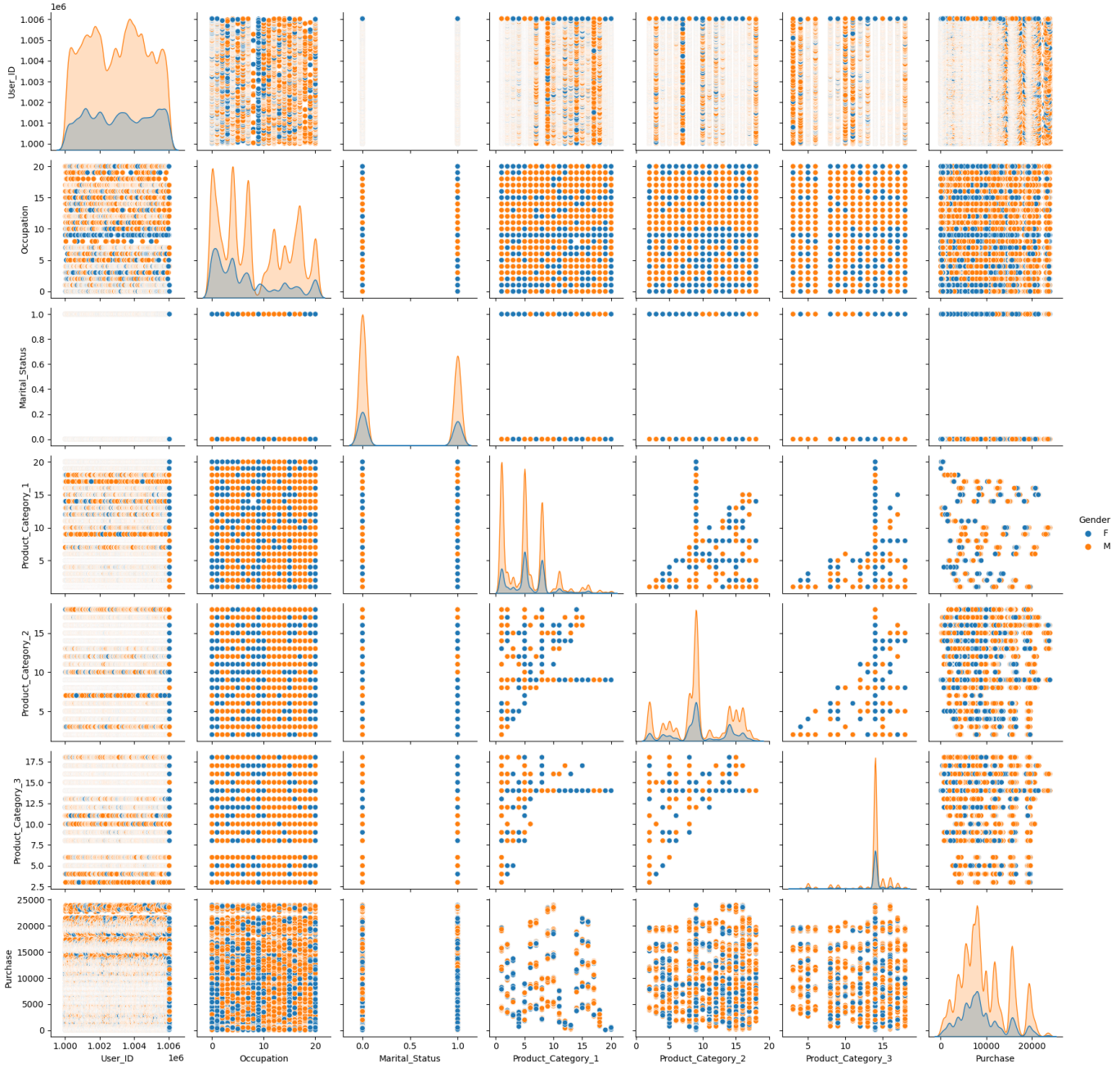
In [14]:
```
corr = df.corr()
plt.figure(figsize=(12,8))
sns.heatmap(corr,annot=True)
plt.show()
```

```
C:\Users\Abdul\AppData\Local\Temp\ipykernel_22404\2000153649.py:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select onl
y valid columns or specify the value of numeric_only to silence this warning.
  corr = df.corr()
```

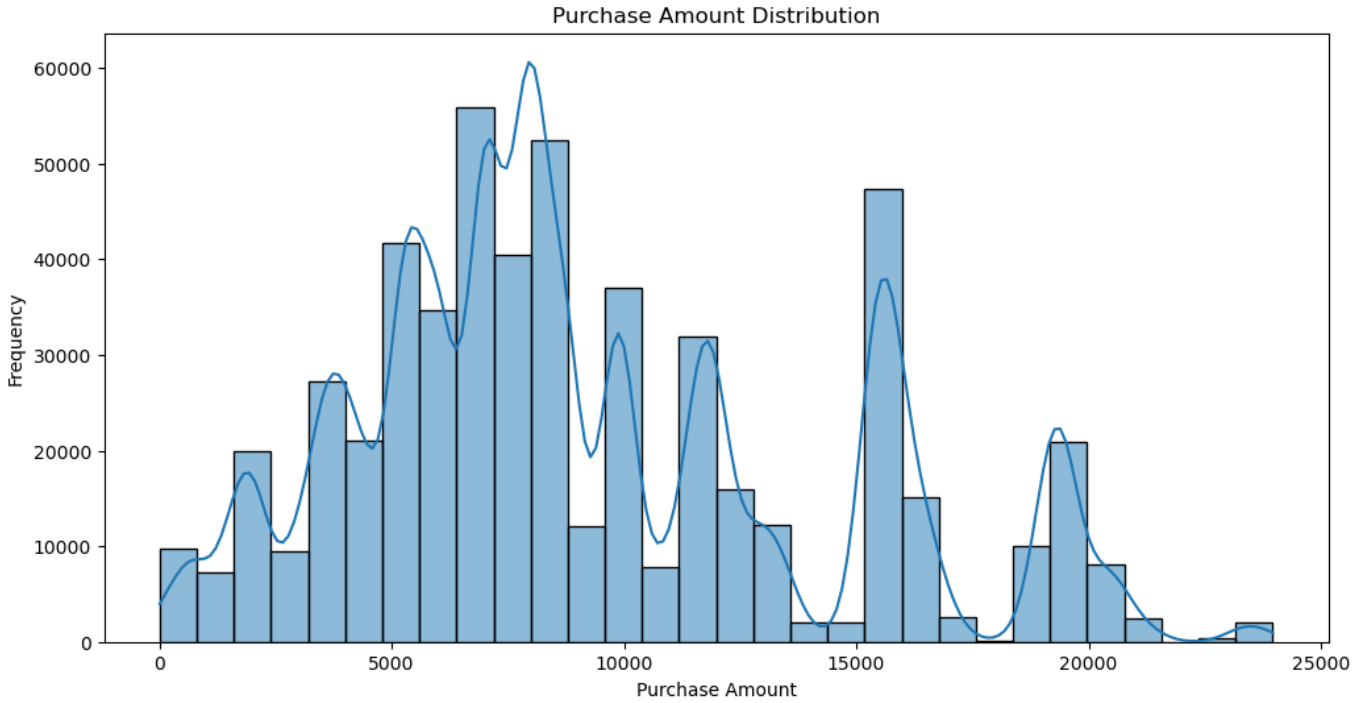


In [39]:
```
sns.pairplot(data=df , hue='Gender')
```

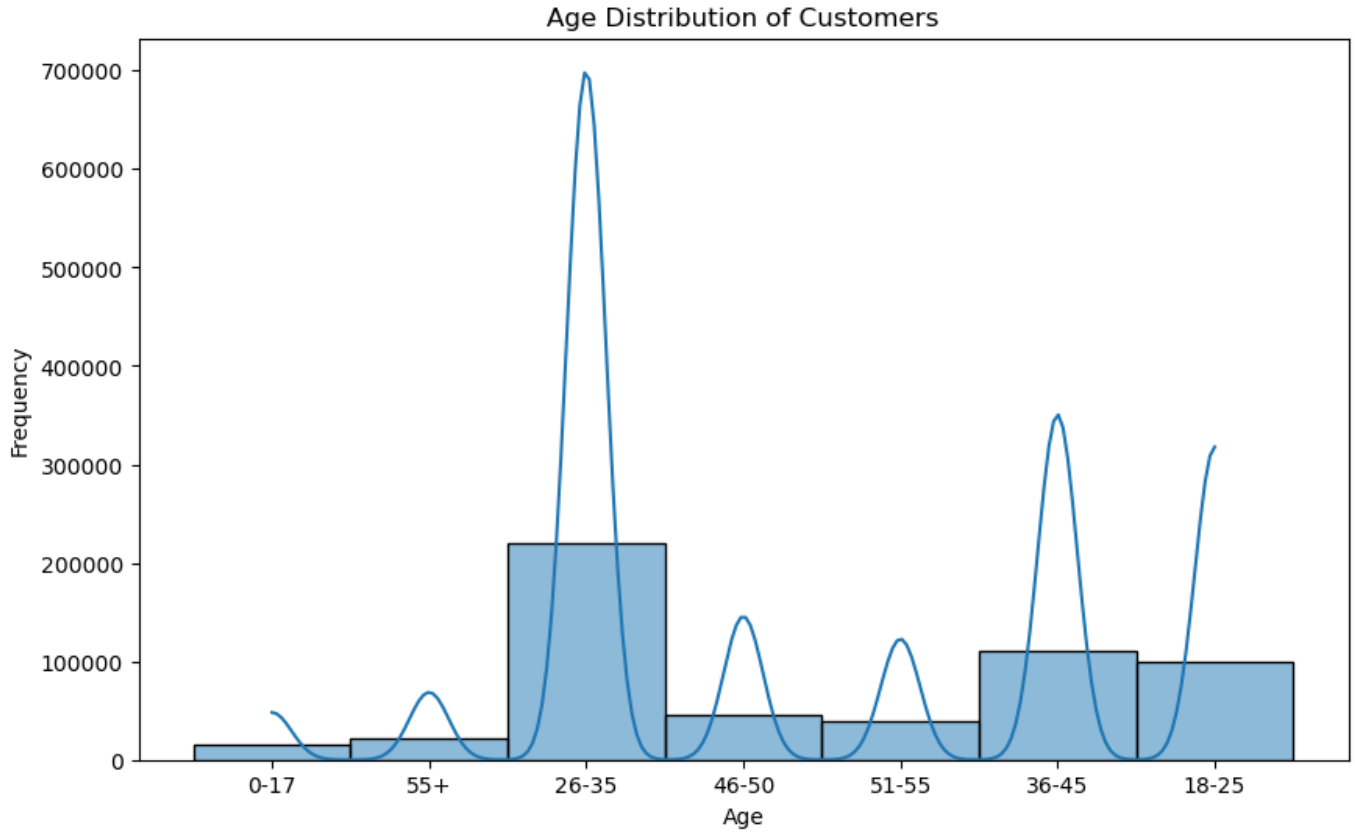
Out[39]:
```
<seaborn.axisgrid.PairGrid at 0x1dc91195bd0>
```



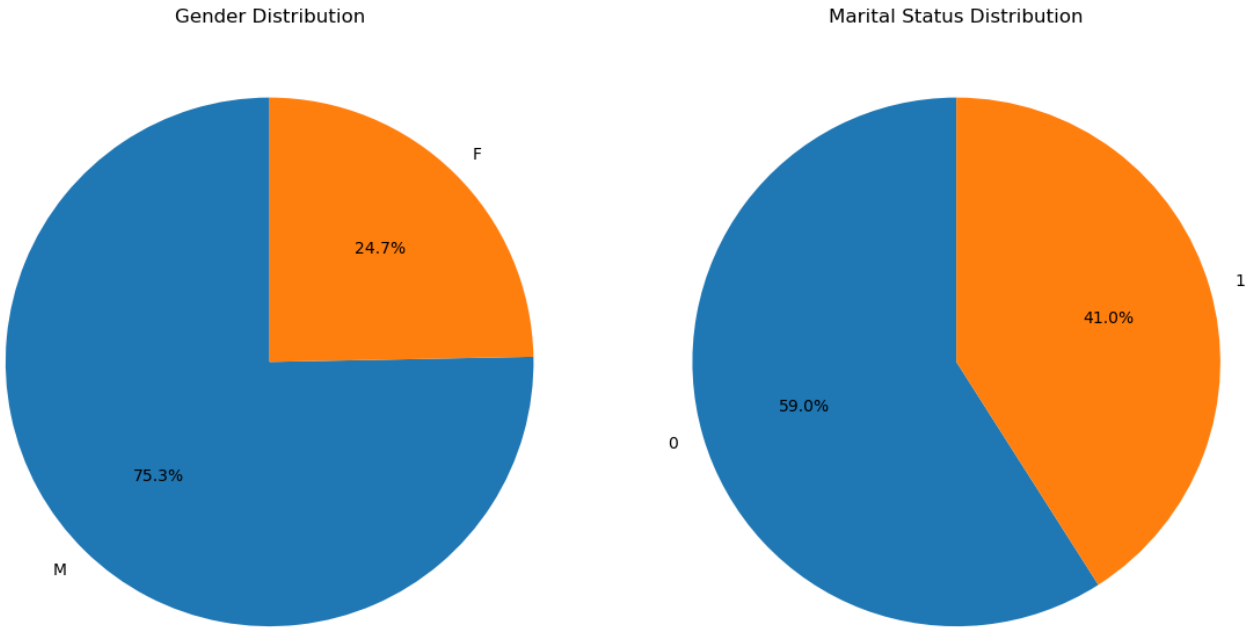
In [15]:
```
# Univariate Visualization
plt.figure(figsize=(12, 6))
sns.histplot(df['Purchase'], bins=30, kde=True)
plt.title('Purchase Amount Distribution')
plt.xlabel('Purchase Amount')
plt.ylabel('Frequency')
plt.show()
```



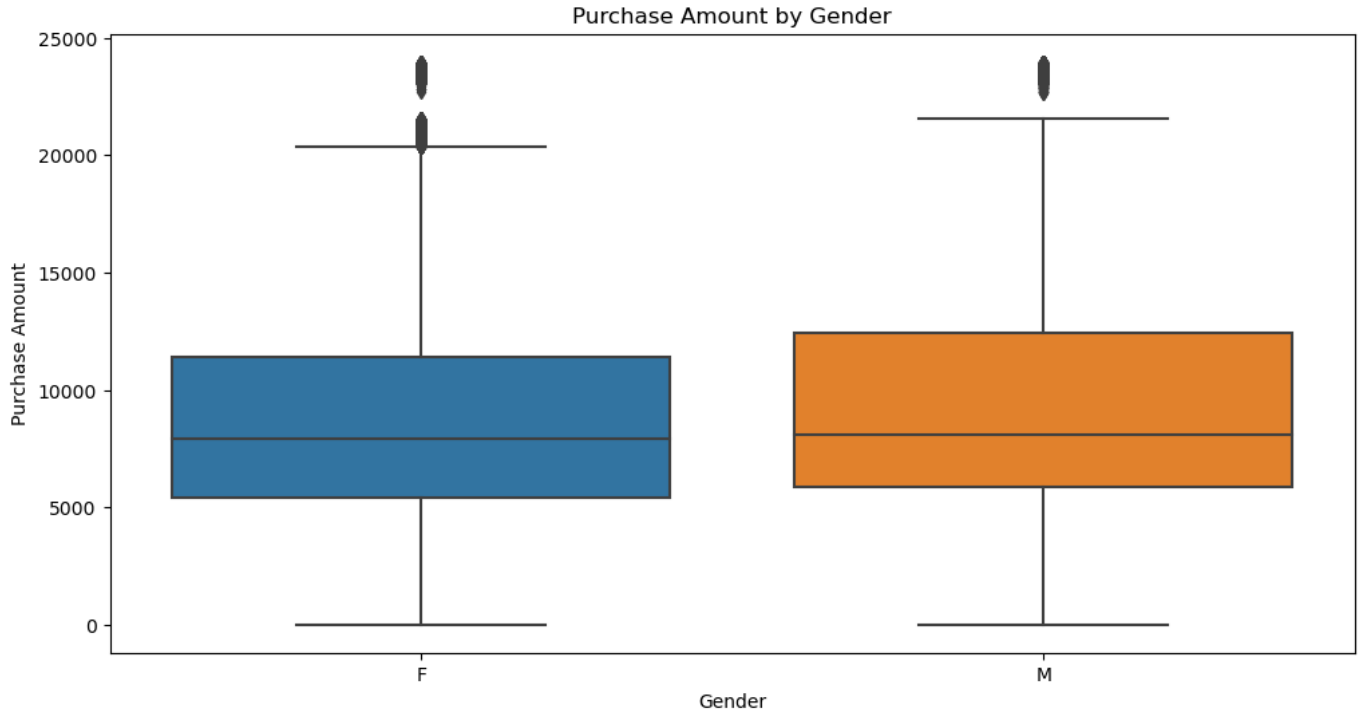
In [16]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df['Age'], bins=10, kde=True)
plt.title('Age Distribution of Customers')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



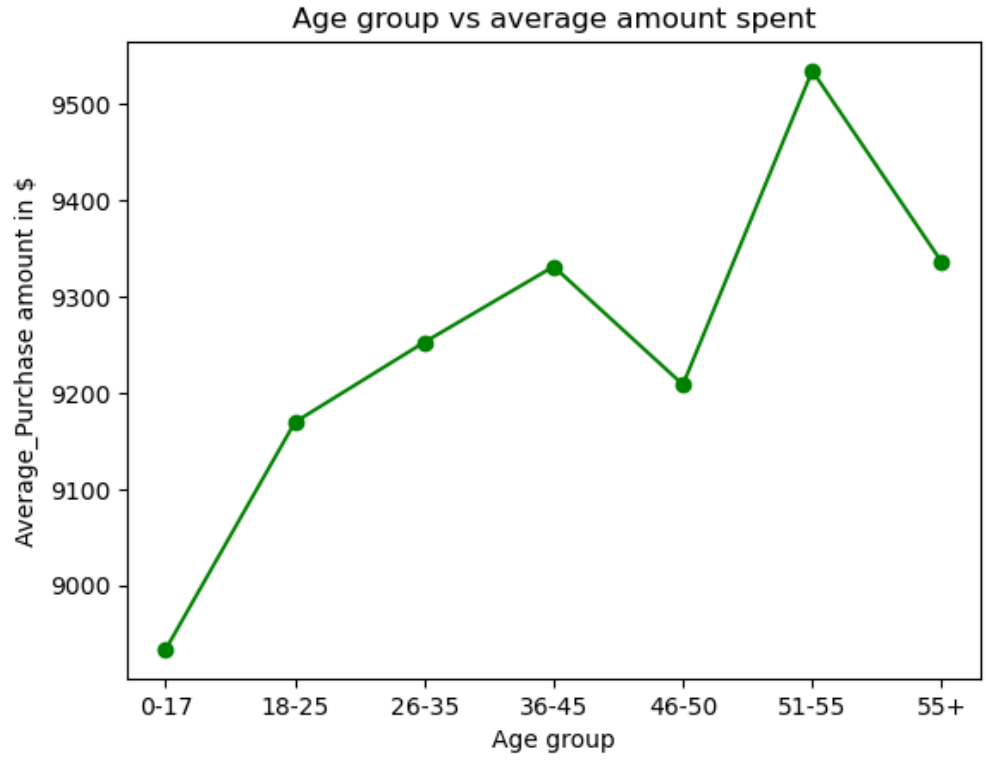
In [17]:
```
gender_counts = df['Gender'].value_counts()
marital_status_counts = df['Marital_Status'].value_counts()

#Pie chart for Gender distribution
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Gender Distribution')

#Pie chart for Marital Status distribution
plt.subplot(1, 2, 2)
plt.pie(marital_status_counts, labels=marital_status_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Marital Status Distribution')

plt.tight_layout()
plt.show()
```



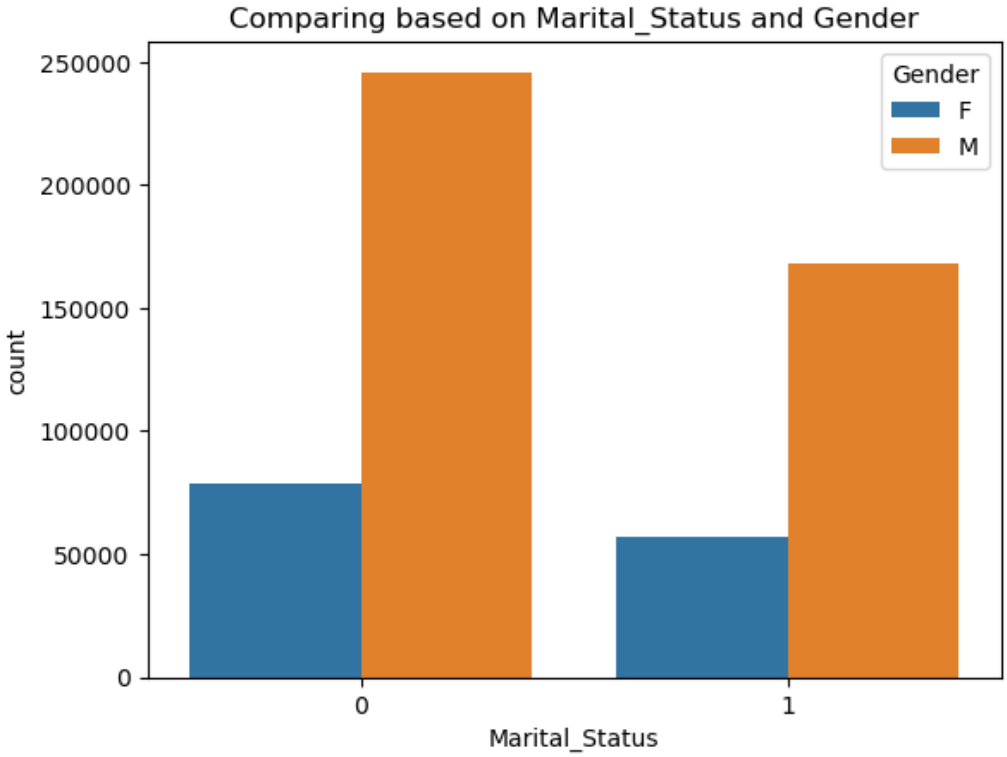
In [18]:
```
# Bivariate Visualization
plt.figure(figsize=(12, 6))
sns.boxplot(x='Gender', y='Purchase', data=df)
plt.title('Purchase Amount by Gender')
plt.xlabel('Gender')
plt.ylabel('Purchase Amount')
plt.show()
```



In [19]:
```
# Avearge amount spend by different age groups

data = df.groupby('Age')['Purchase'].mean()
plt.plot(data.index,data.values,marker='o',color='g')
plt.xlabel('Age group');
plt.ylabel('Average_Purchase amount in $');
plt.title('Age group vs average amount spent');
plt.show()
```

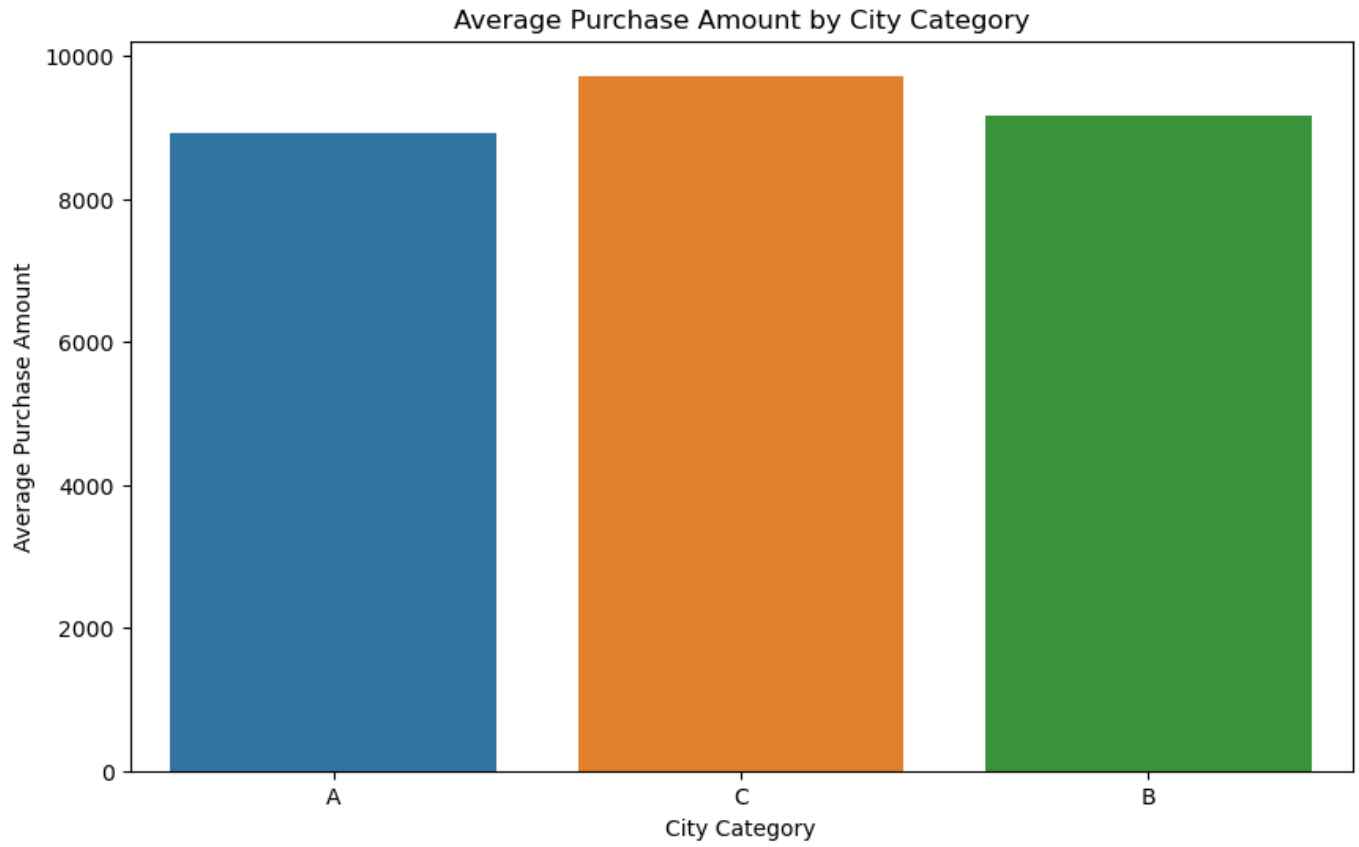


In [20]:
```
#comparing based on Marital_Status and Gender

sns.countplot(x='Marital_Status',data=df,hue='Gender')
plt.title('Comparing based on Marital_Status and Gender')
plt.show()
```

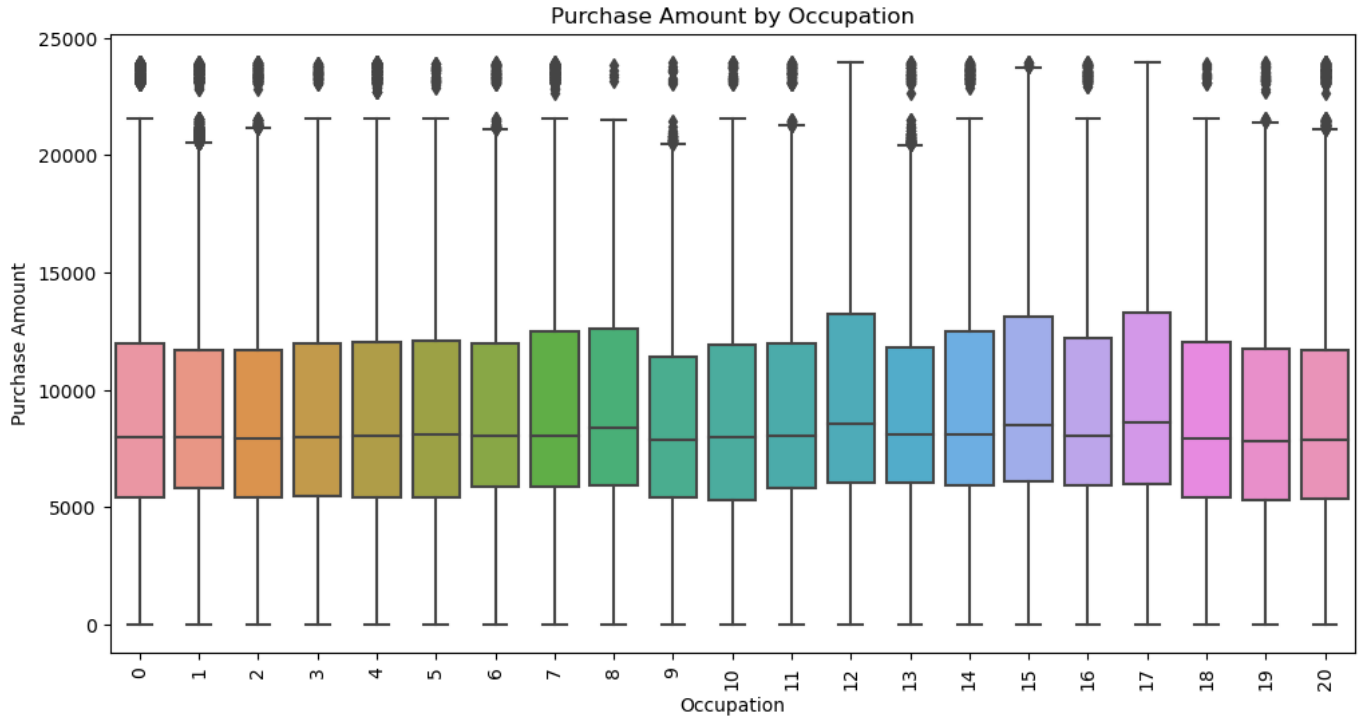


In [21]:
```
plt.figure(figsize=(10, 6))
sns.barplot(x='City_Category', y='Purchase', data=df, ci=None)
plt.title('Average Purchase Amount by City Category')
plt.xlabel('City Category')
plt.ylabel('Average Purchase Amount')
plt.show()
```

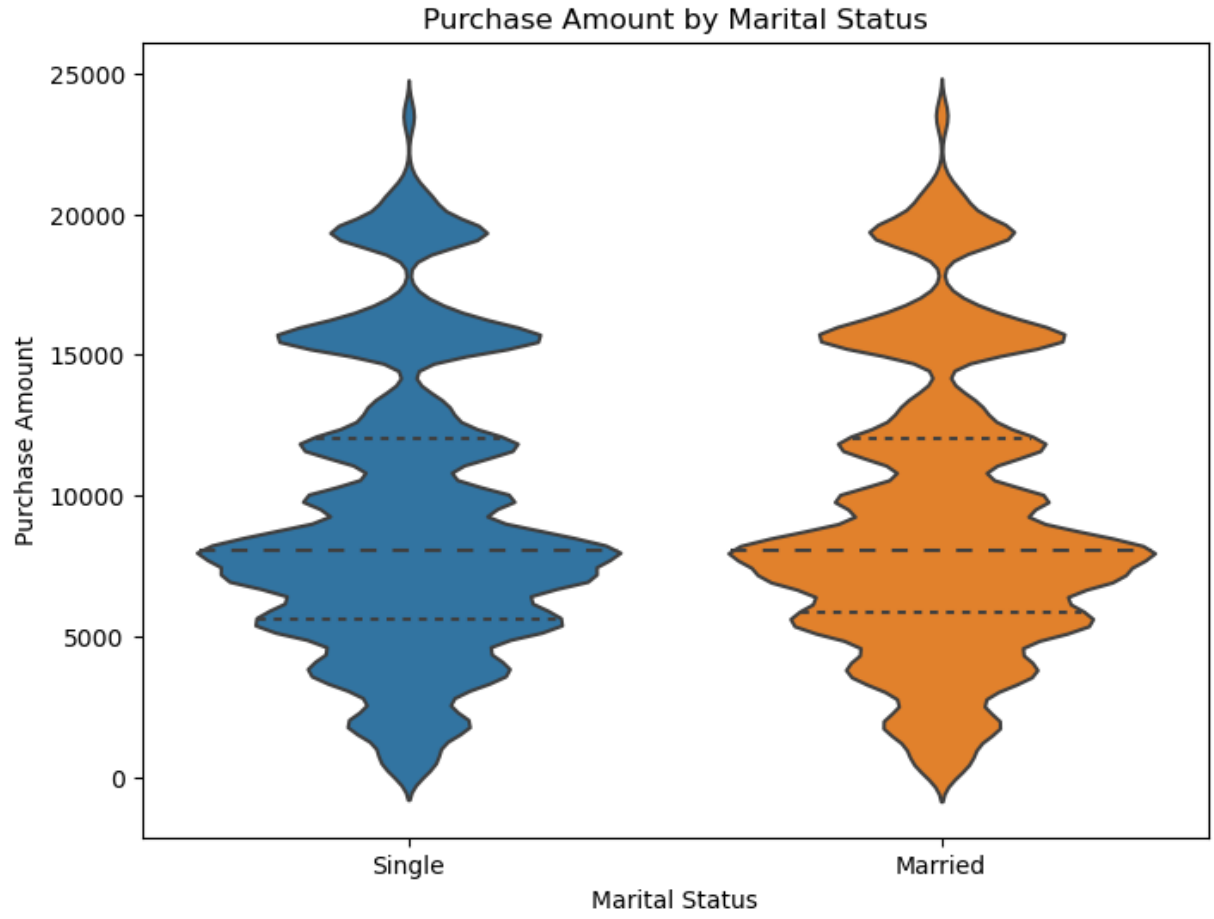
```
C:\Users\Abdul\AppData\Local\Temp\ipykernel_22404\2580027714.py:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='City_Category', y='Purchase', data=df, ci=None)
```



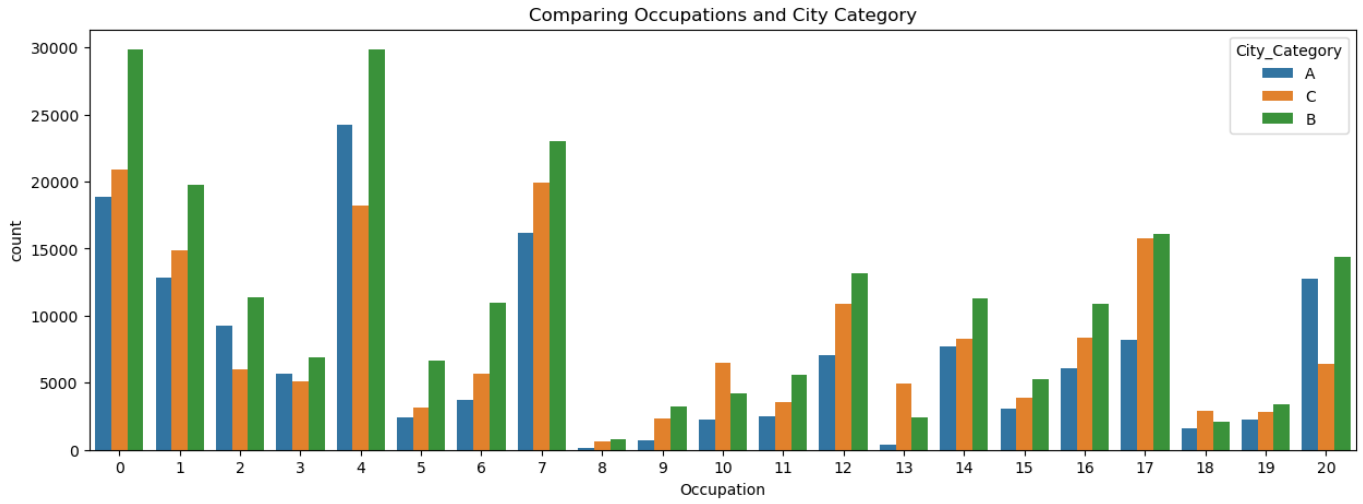
In [22]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Occupation', y='Purchase', data=df)
plt.title('Purchase Amount by Occupation')
plt.xlabel('Occupation')
plt.ylabel('Purchase Amount')
plt.xticks(rotation=90)
plt.show()
```



In [23]:
```
plt.figure(figsize=(8, 6))
sns.violinplot(x='Marital_Status', y='Purchase', data=df, inner='quart')
plt.title('Purchase Amount by Marital Status')
plt.xlabel('Marital Status')
plt.ylabel('Purchase Amount')
plt.xticks([0, 1], ['Single', 'Married'])
plt.show()
```

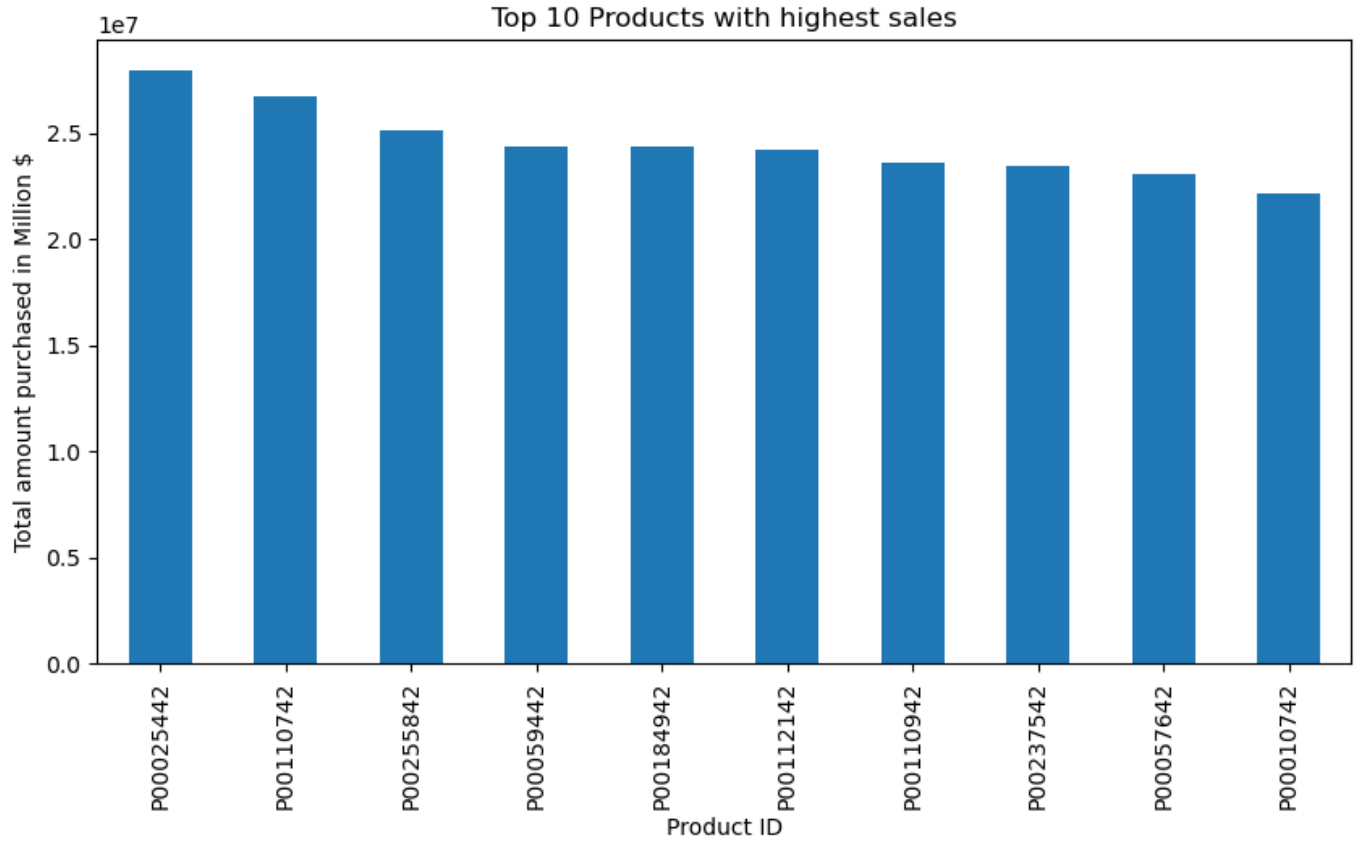


In [24]:
```
#Occupations and City Category

plt.figure(figsize=(15,5))
sns.countplot(x='Occupation',data=df,hue='City_Category')
plt.title('Comparing Occupations and City Category')
plt.show()
```



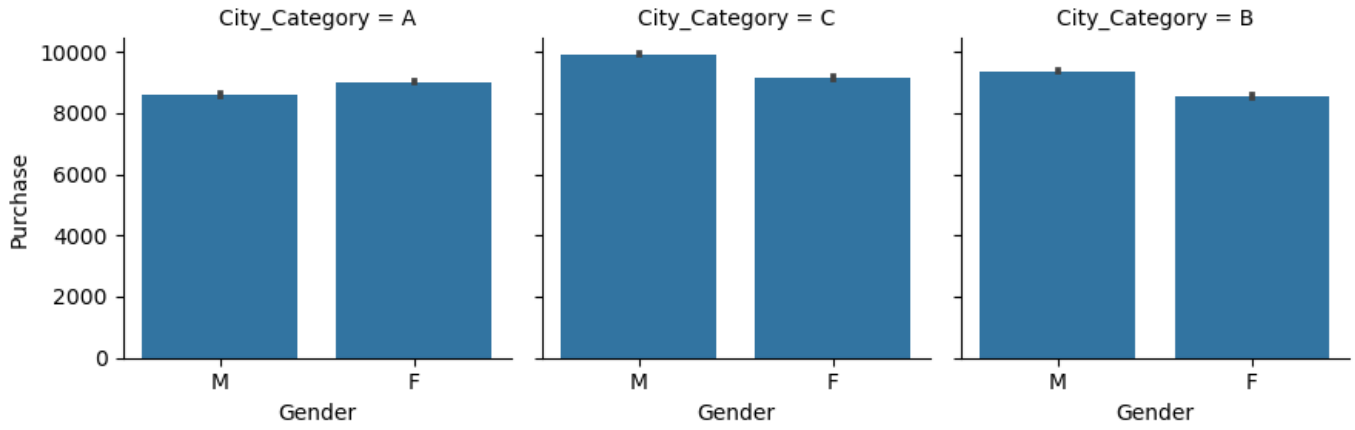
In [25]:
```
# Top 10 products which made the highest sales

data = df.groupby("Product_ID").sum()['Purchase']

plt.figure(figsize=(10,5))
data.sort_values(ascending=False)[0:10].plot(kind='bar')
plt.xticks(rotation=90)
plt.xlabel('Product ID')
plt.ylabel('Total amount purchased in Million $')
plt.title('Top 10 Products with highest sales')
plt.show()
```

```
C:\Users\Abdul\AppData\Local\Temp\ipykernel_22404\3003506316.py:3: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to F
alse. Either specify numeric_only or select only columns which should be valid for the function.
  data = df.groupby("Product_ID").sum()['Purchase']
```



In [26]:
```
#the purchase habits of different genders across the different city categories.

g = sns.FacetGrid(df,col="City_Category")
g.map(sns.barplot, "Gender", "Purchase")
plt.show()
```

```
C:\Users\Abdul\anaconda3\lib\site-packages\seaborn\axisgrid.py:712: UserWarning: Using the barplot func
tion without specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```



In [27]:
```
df.drop(['User_ID','Product_ID'], axis=1, inplace=True)
```

In [28]:
```
cat_var = ['Gender','Age','Occupation','City_Category','Stay_In_Current_City_Years','Marital_Status']
```

In [29]:
```
for x in cat_var:
    print('***************',x,'*****************')
    print(df[x].value_counts())
print('************************************')
```

```
*************** Gender ***************
M    414259
F    135809
Name: Gender, dtype: int64
*************** Age ***************
26-35    219587
36-45    110013
18-25     99660
46-50     45701
51-55     38501
55+       21504
0-17      15102
Name: Age, dtype: int64
*************** Occupation ***************
4     72308
0     69638
7     59133
1     47426
17    40043
20    33562
12    31179
14    27309
2     26588
16    25371
6     20355
3     17650
10    12930
5     12177
15    12165
11    11586
19     8461
13     7728
18     6622
9      6291
8      1546
Name: Occupation, dtype: int64
*************** City_Category ***************
B    231173
C    171175
A    147720
Name: City_Category, dtype: int64
*************** Stay_In_Current_City_Years ***************
1     193821
2     101838
3      95285
4+     84726
0      74398
Name: Stay_In_Current_City_Years, dtype: int64
*************** Marital_Status ***************
0    324731
1    225337
Name: Marital_Status, dtype: int64
************************************
```

In [30]:
```
df['Stay_In_Current_City_Years'] = df['Stay_In_Current_City_Years'].replace(to_replace="4+",value="4")
```

In [31]:
```
df['Stay_In_Current_City_Years'].value_counts()
```

Out[31]:
```
1    193821
2    101838
3     95285
4     84726
0     74398
Name: Stay_In_Current_City_Years, dtype: int64
```

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

Out[32]:

	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2
0	F	0-17	10	A	2	0	3	9.0
1	F	0-17	10	A	2	0	1	6.0
2	F	0-17	10	A	2	0	12	9.0
3	F	0-17	10	A	2	0	12	14.0
4	M	55+	16	C	4	0	8	9.0

In [33]:
```
gender_dict = {'F':0, 'M':1}
df['Gender'] = df['Gender'].apply(lambda x: gender_dict[x])
df.head()
```

Out[33]:

	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2
0	0	0-17	10	A	2	0	3	9.0
1	0	0-17	10	A	2	0	1	6.0
2	0	0-17	10	A	2	0	12	9.0
3	0	0-17	10	A	2	0	12	14.0
4	1	55+	16	C	4	0	8	9.0

In [34]:
```
cat_var = ['Age', 'City_Category', 'Stay_In_Current_City_Years']
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for var in cat_var:
    df[var] = le.fit_transform(df[var])
```

In [35]:
```
df.head(5)
```

Out[35]:

	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2
0	0	0	10	0	2	0	3	9.0
1	0	0	10	0	2	0	1	6.0
2	0	0	10	0	2	0	12	9.0
3	0	0	10	0	2	0	12	14.0
4	1	6	16	2	4	0	8	9.0

In [36]:
```
X = df.drop("Purchase",axis=1)
Y = df["Purchase"]
```

In [37]:
```
X.head()
```

Out[37]:

	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2
0	0	0	10	0	2	0	3	9.0
1	0	0	10	0	2	0	1	6.0
2	0	0	10	0	2	0	12	9.0
3	0	0	10	0	2	0	12	14.0
4	1	6	16	2	4	0	8	9.0

In [38]:
```
Y.head()
```

Out[38]:
```
0     8370
1    15200
2     1422
3     1057
4     7969
Name: Purchase, dtype: int64
```

In [39]:
```
X.shape
```

Out[39]:
```
(550068, 9)
```

In [40]:
```
Y.shape
```

Out[40]:
```
(550068,)
```

In [72]:
```
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 [73]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [74]:
```
#Linear regression

from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train,Y_train)
```

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

In [75]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

Y_pred = lr.predict(X_test)

print("Linear Regression: ")
print('rmse:', np.sqrt(mean_squared_error(Y_test,Y_pred)))
print('r2_score:', r2_score(Y_test,Y_pred))
```

```
Linear Regression:
rmse: 4686.373005187138
r2_score: 0.12592793684162085
```

In [76]:
```
# KNN Regressor
from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor()
```

In [77]:
```
knn.fit(X_train, Y_train)
```

Out[77]:
```
KNeighborsRegressor
KNeighborsRegressor()
```

In [78]:
```
Y_pred_knn = knn.predict(X_test)
```

In [79]:
```
print("KNN regression: ")
print("RMSE:",np.sqrt(mean_squared_error(Y_test, Y_pred_knn)))
print("R2 score:", r2_score(Y_test, Y_pred_knn))
```

```
KNN regression:
RMSE: 3499.7488290381266
R2 score: 0.5125306794266962
```

In [80]:
```
# Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
dec_tree = DecisionTreeRegressor()
```

In [81]:
```
dec_tree.fit(X_train, Y_train)
```

Out[81]:
```
DecisionTreeRegressor
DecisionTreeRegressor()
```

In [82]:
```
Y_pred_dec = dec_tree.predict(X_test)
print("Decision tree regression: ")
print("RMSE:",np.sqrt(mean_squared_error(Y_test, Y_pred_dec)))
print("R2 score:", r2_score(Y_test, Y_pred_dec))
```

```
Decision tree regression:
RMSE: 3334.237206232025
R2 score: 0.5575476347064576
```

In [83]:
```
#Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
ran_for = RandomForestRegressor()
```

In [84]:
```
ran_for.fit(X_train, Y_train)
```

Out[84]:
```
RandomForestRegressor
RandomForestRegressor()
```

In [86]:
```
Y_pred_ran_for = ran_for.predict(X_test)
print("Random forest regression: ")
print("RMSE:",np.sqrt(mean_squared_error(Y_test, Y_pred_ran_for)))
print("R2 score:", r2_score(Y_test, Y_pred_ran_for))
```

```
Random forest regression:
RMSE: 3061.4509295099183
R2 score: 0.6269834167254187
```

In [92]:
```
#XGB Regressor
!pip install xgboost
from xgboost.sklearn import XGBRegressor
xgb_reg = XGBRegressor(learning_rate=1.0, max_depth=6, min_child_weight=40,seed=0)
```

```
Collecting xgboost
  Downloading xgboost-2.0.0-py3-none-win_amd64.whl (99.7 MB)
     ---------------------------------------- 99.7/99.7 MB 3.7 MB/s eta 0:00:00
Requirement already satisfied: scipy in c:\users\abdul\anaconda3\lib\site-packages (from xgboost) (1.10.0)
Requirement already satisfied: numpy in c:\users\abdul\anaconda3\lib\site-packages (from xgboost) (1.23.5)
Installing collected packages: xgboost
Successfully installed xgboost-2.0.0
```

In [93]:
```
xgb_reg.fit(X_train, Y_train)
```

Out[93]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=1.0, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=40, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
```

In [95]:
```
Y_pred_xgb = xgb_reg.predict(X_test)
print("XGB regression: ")
print("RMSE:",np.sqrt(mean_squared_error(Y_test, Y_pred_xgb)))
print("R2 score:", r2_score(Y_test, Y_pred_xgb))
```

```
XGB regression:
RMSE: 2897.762592083295
R2 score: 0.6658056235176206
```

In [96]:
```
#Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV
```

In [97]:
```
max_depth = [int(x) for x in np.linspace(start = 5, stop = 20, num = 15)]
learning_rate = ['0.01', '0.05', '0.1', '0.25', '0.5', '0.75', '1.0']
min_child_weight = [int(x) for x in np.linspace(start = 45, stop = 70, num = 15)]
```

In [98]:
```
params = {
 "learning_rate"    : learning_rate,
 "max_depth"        : max_depth,
 "min_child_weight" : min_child_weight,
 "gamma"            : [0.0, 0.1, 0.2 , 0.3, 0.4],
 "colsample_bytree" : [0.3, 0.4, 0.5 , 0.7]
}
```

In [99]:
```
xgb_tune = XGBRegressor(verbosity = 0, random_state = 42)
```

In [100]:
```
xgb_cv = RandomizedSearchCV(xgb_tune, param_distributions = params, cv = 5, random_state = 42)
```

In [101]:
```
xgb_cv.fit(X_train, Y_train)
```

Out[101]:
```
RandomizedSearchCV
estimator: XGBRegressor
XGBRegressor
```

In [102]:
```
xgb_cv.best_score_
```

Out[102]:
```
0.67145979791069
```

In [103]:
```
xgb_cv.best_params_
```

Out[103]:
```
{'min_child_weight': 53,
 'max_depth': 18,
 'learning_rate': '0.5',
 'gamma': 0.3,
 'colsample_bytree': 0.5}
```

In [108]:
```
xgb_best = XGBRegressor(min_child_weight= 53,max_depth=18,learning_rate= 0.5,gamma= 0.3,colsample_bytree
```

In [109]:
```
xgb_best.fit(X_train, Y_train)
```

Out[109]:
```
XGBRegressor

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.5, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0.3, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.5, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=18, max_leaves=None,
             min_child_weight=53, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
```

In [111]:
```
Y_pred_xgb_best = xgb_best.predict(X_test)
print("XGB regression: ")
print("RMSE:",np.sqrt(mean_squared_error(Y_test, Y_pred_xgb_best)))
print("R2 score:", r2_score(Y_test, Y_pred_xgb_best))
```

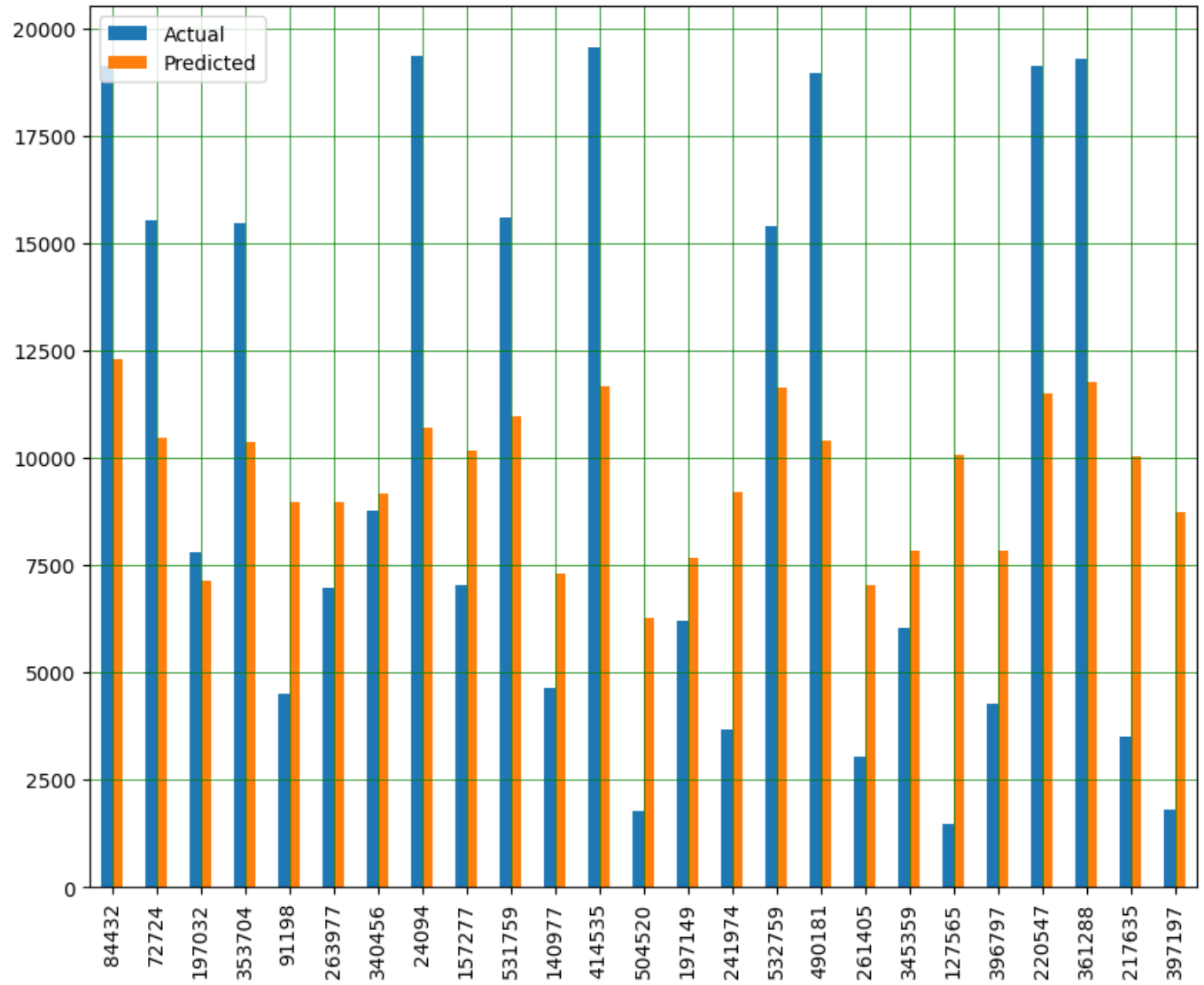
```
XGB regression:
RMSE: 2894.6689236175666
R2 score: 0.6665188183908286
```

In [112]:
```
df = pd.DataFrame({'Actual': Y_test, 'Predicted': Y_pred})
df1 = df.head(25)
df1.head()
```

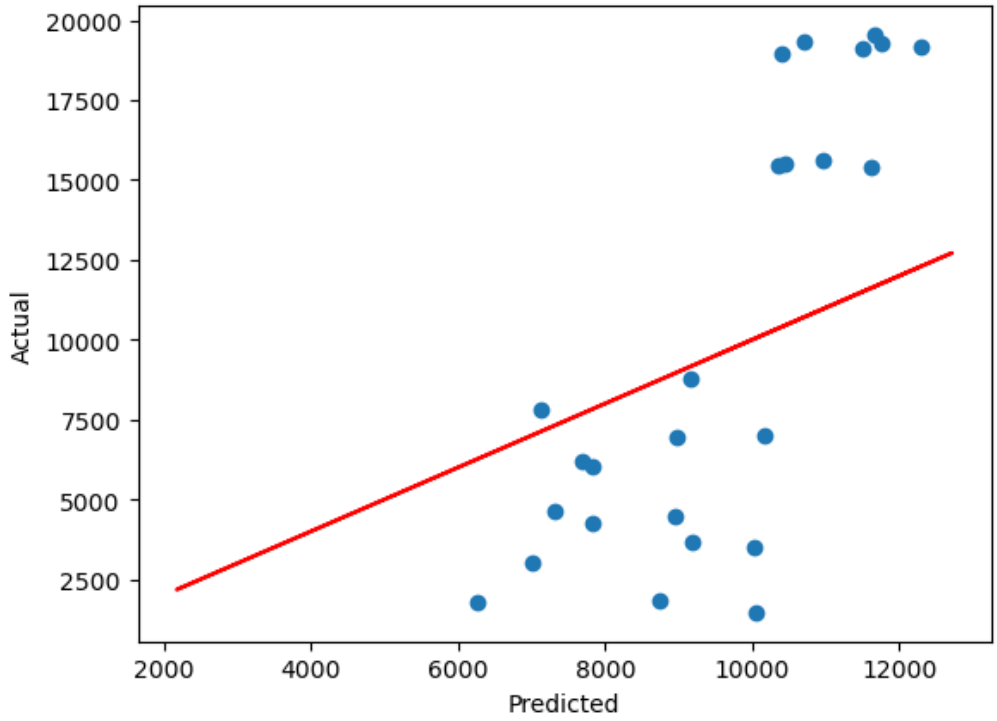
Out[112]:

	Actual	Predicted
84432	19142	12284.856371
72724	15513	10448.486159
197032	7802	7122.129971
353704	15455	10352.757284
91198	4492	8951.038701

In [113]:
```
df1.plot(kind='bar',figsize=(10,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



In [115]:
```
plt.scatter(df1.Predicted,df1.Actual)
plt.plot(Y_pred,Y_pred,'r')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [ ]: