```
import pandas as pd
import xgboost as xgb
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import warnings
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.tree import DecisionTreeRegressor
from statsmodels.stats.outliers_influence import variance_inflation_factor
warnings.filterwarnings('ignore')
## The above libraries are needed for performing the machine learning and evaluati
```

Loading the data

```
train = pd.read_csv('/content/train.csv')
train_copy = train
# We're Reading the data using pd.read_csv since the dataset is in the csv format.
```

```
train
# A peek into the dataset.
```

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

Data Understanding

```
train.shape
# The dataframe has 550068 rows and 12 columns.
```

```
(550068, 12)
```

550066	1006038	P00375436	F	55+	1	C

```
train.describe()
# The 'describe function' shows us the descriptive statistics of our data and the
```

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Cate
count	5.500680e+05	550068.000000	550068.000000	550068.000000	37643(
mean	1.003029e+06	8.076707	0.409653	5.404270	(
std	1.727592e+03	6.522660	0.491770	3.936211	!
min	1.000001e+06	0.000000	0.000000	1.000000	:
25%	1.001516e+06	2.000000	0.000000	1.000000	!
50%	1.003077e+06	7.000000	0.000000	5.000000	(
75%	1.004478e+06	14.000000	1.000000	8.000000	1!
max	1.006040e+06	20.000000	1.000000	20.000000	1!

```
train.info()
# Using the 'info function' we can see below that there are a few null values pres
```

```
<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
```

```
train.isnull().sum()
## The isnull().sum() function will help in finding all the null values present in
## There are 173638 missing values and 383247 missing values in the Product_catego
```

```
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
```

```
train.groupby(['Product_ID'])['Purchase'].mean().sort_values(ascending = True)
```

```
Product_ID
P00370293       36.675159
P00370853       37.393643
P00371644      362.911012
P00375436      374.266585
P00372445      374.930705
                  ...    
P00119342    20448.756494
P00116142    20463.791277
P00200642    20468.773234
P00085342    20980.268116
P00086242    21256.505495
Name: Purchase, Length: 3631, dtype: float64
```

```
1 train.groupby(['Marital_Status'])['Purchase'].mean()
```

```
Marital_Status
0    9265.907619
1    9261.174574
Name: Purchase, dtype: float64
```

```
1 train.groupby('Product_Category_1')['Purchase'].mean().sort_values(ascending = Tru
```

```
Product_Category_1
19       37.041797
20      370.481176
13      722.400613
12     1350.859894
4      2329.659491
18     2972.864320
11     4685.268456
5      6240.088178
8      7498.958078
3     10096.705734
17    10170.759516
2     11251.935384
14    13141.625739
1     13606.218596
16    14766.037037
15    14780.451828
9     15537.375610
6     15838.478550
7     16365.689600
10    19675.570927
Name: Purchase, dtype: float64
```

```
1 train.groupby('Product_Category_2')['Purchase'].mean().sort_values(ascending = Tru
```

```
Product_Category_2
7.0      6884.683706
12.0     6975.472504
14.0     7105.264916
9.0      7277.006851
11.0     8940.580515
5.0      9027.821574
18.0     9352.440433
17.0     9421.576577
13.0     9683.352388
4.0     10215.192001
8.0     10273.259518
16.0    10295.681933
15.0    10357.077691
3.0     11235.359570
6.0     11503.551379
2.0     13619.356401
10.0    15648.729543
Name: Purchase, dtype: float64
```

```
1 train.groupby('Product_Category_3')['Purchase'].mean().sort_values(ascending = Tru
```

```
Product_Category_3
12.0     8715.512762
4.0      9794.386667
14.0    10052.594530
9.0     10431.697210
18.0    10993.980773
17.0    11769.943001
16.0    11981.890642
11.0    12091.437673
5.0     12117.786889
15.0    12339.369900
8.0     13024.918882
13.0    13185.118703
6.0     13194.311043
10.0    13505.813441
3.0     13939.696574
Name: Purchase, dtype: float64
```

```
1 train.groupby('Age')['Purchase'].mean().sort_values(ascending = True)
```

```
Age
0-17     8933.464640
18-25    9169.663606
46-50    9208.625697
26-35    9252.690633
36-45    9331.350695
55+      9336.280459
51-55    9534.808031
Name: Purchase, dtype: float64
```

```
1 train.groupby('City_Category')['Purchase'].mean().sort_values(ascending = True)
```

```
City_Category
A    8911.939216
B    9151.300563
C    9719.920993
Name: Purchase, dtype: float64
```

```
1 train.groupby('Gender')['Purchase'].mean().sort_values(ascending = True)
```

```
Gender
F    8734.565765
M    9437.526040
Name: Purchase, dtype: float64
```

```
1 train.groupby('Occupation')['Purchase'].mean().sort_values(ascending = True)
```

```
Occupation
9     8637.743761
19    8710.627231
```

```
20    8836.494905
2     8952.481683
1     8953.193270
10    8959.355375
0     9124.428588
18    9169.655844
3     9178.593088
11    9213.845848
4     9213.980251
6     9256.535691
13    9306.351061
5     9333.149298
16    9394.464349
7     9425.728223
14    9500.702772
8     9532.592497
15    9778.891163
12    9796.640239
17    9821.478236
Name: Purchase, dtype: float64
```

```
1 train.groupby('Stay_In_Current_City_Years')['Purchase'].mean().sort_values(ascendi
```

```
Stay_In_Current_City_Years
0     9180.075123
1     9250.145923
4+    9275.598872
3     9286.904119
2     9320.429810
Name: Purchase, dtype: float64
```

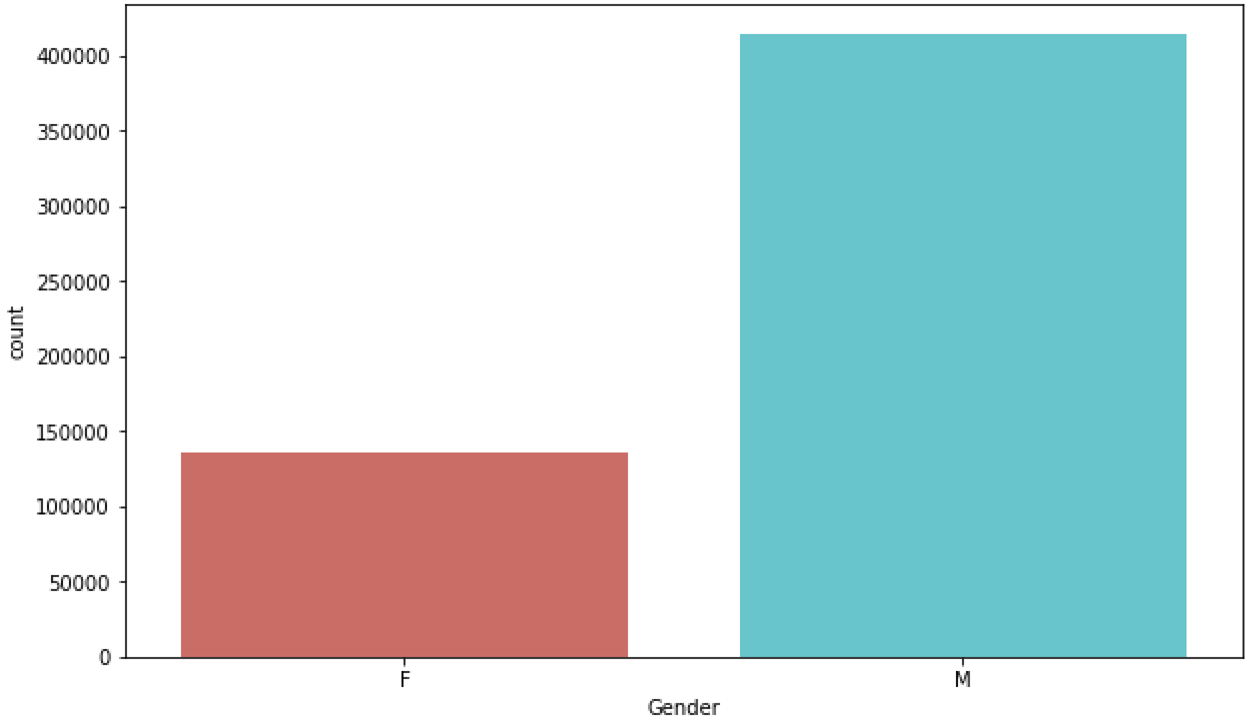
```
1 train.groupby(['Product_ID'])['Product_ID'].count()
```

```
Product_ID
P00000142    1152
P00000242     376
P00000342     244
P00000442      92
P00000542     149
             ...
P0099442      200
P0099642       13
P0099742      126
P0099842      102
P0099942       14
Name: Product_ID, Length: 3631, dtype: int64
```

Data Visualization

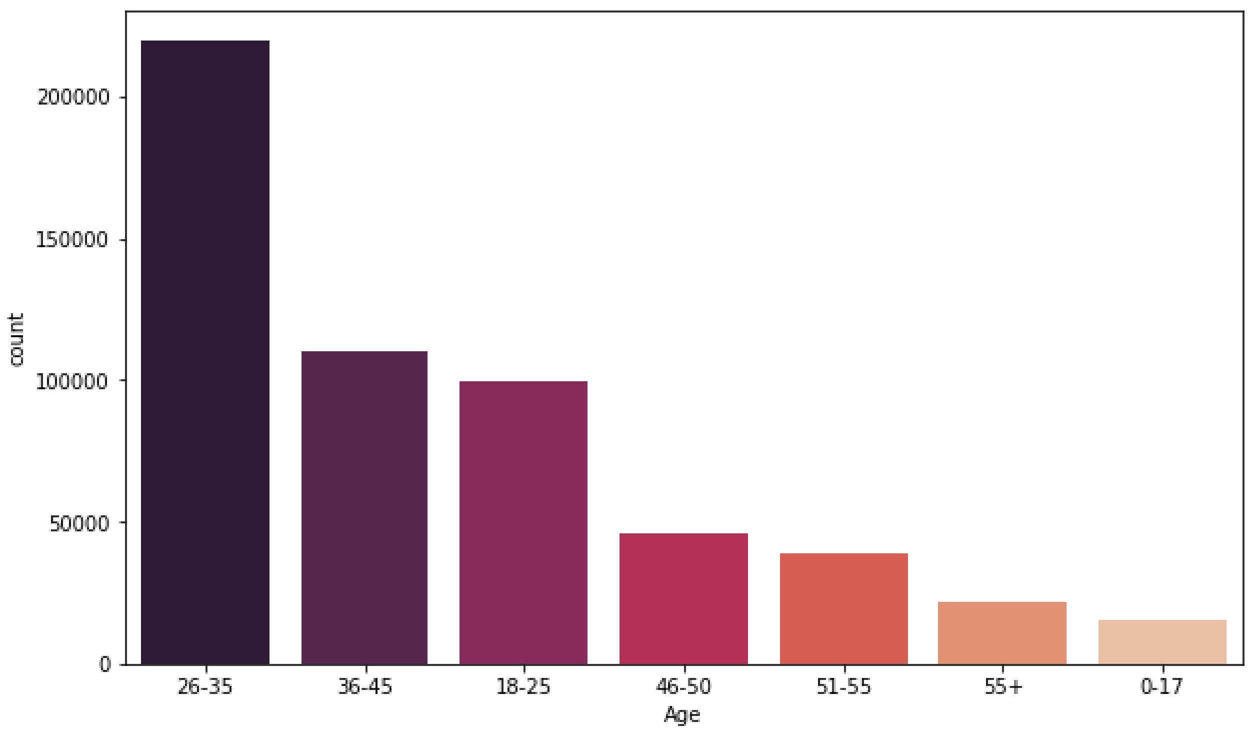
```
1 plt.figure(figsize = (10, 6))
2 sns.countplot(data = train, x = 'Gender', palette = 'hls')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5046fdf9d0>
```



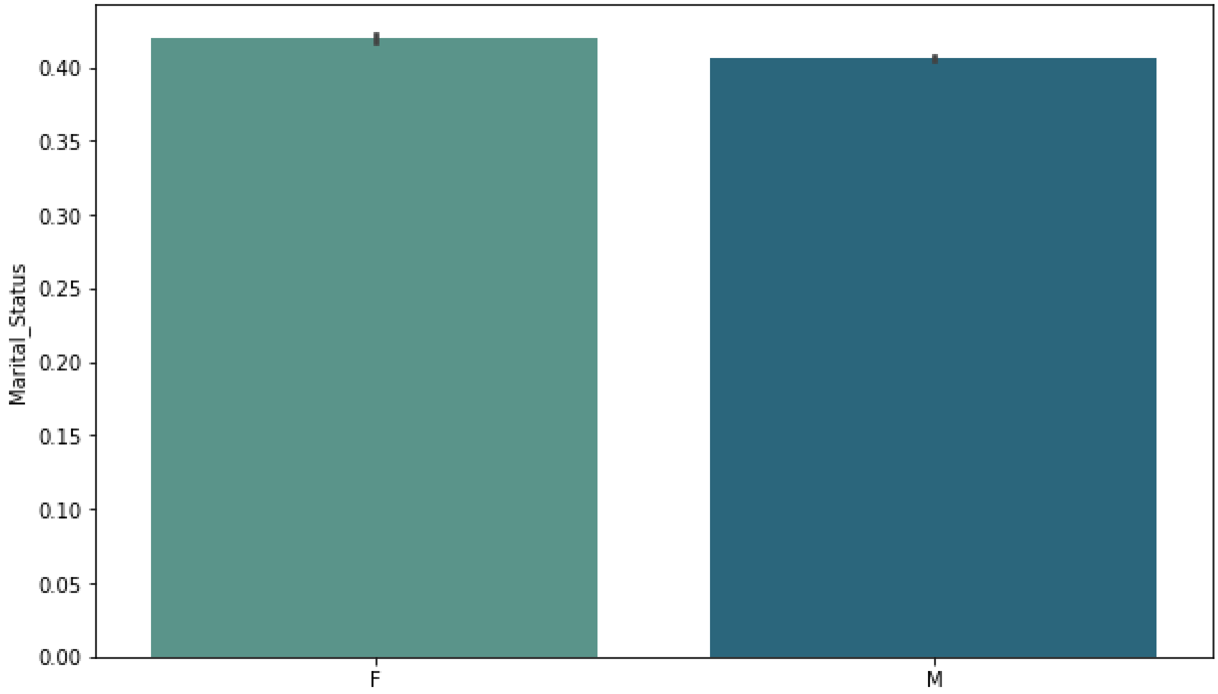

```
1 plt.figure(figsize = (10, 6))
2 sns.countplot(data = train, x = 'Age', palette = 'rocket', order = train['Age'].va
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50458a8f10>
```



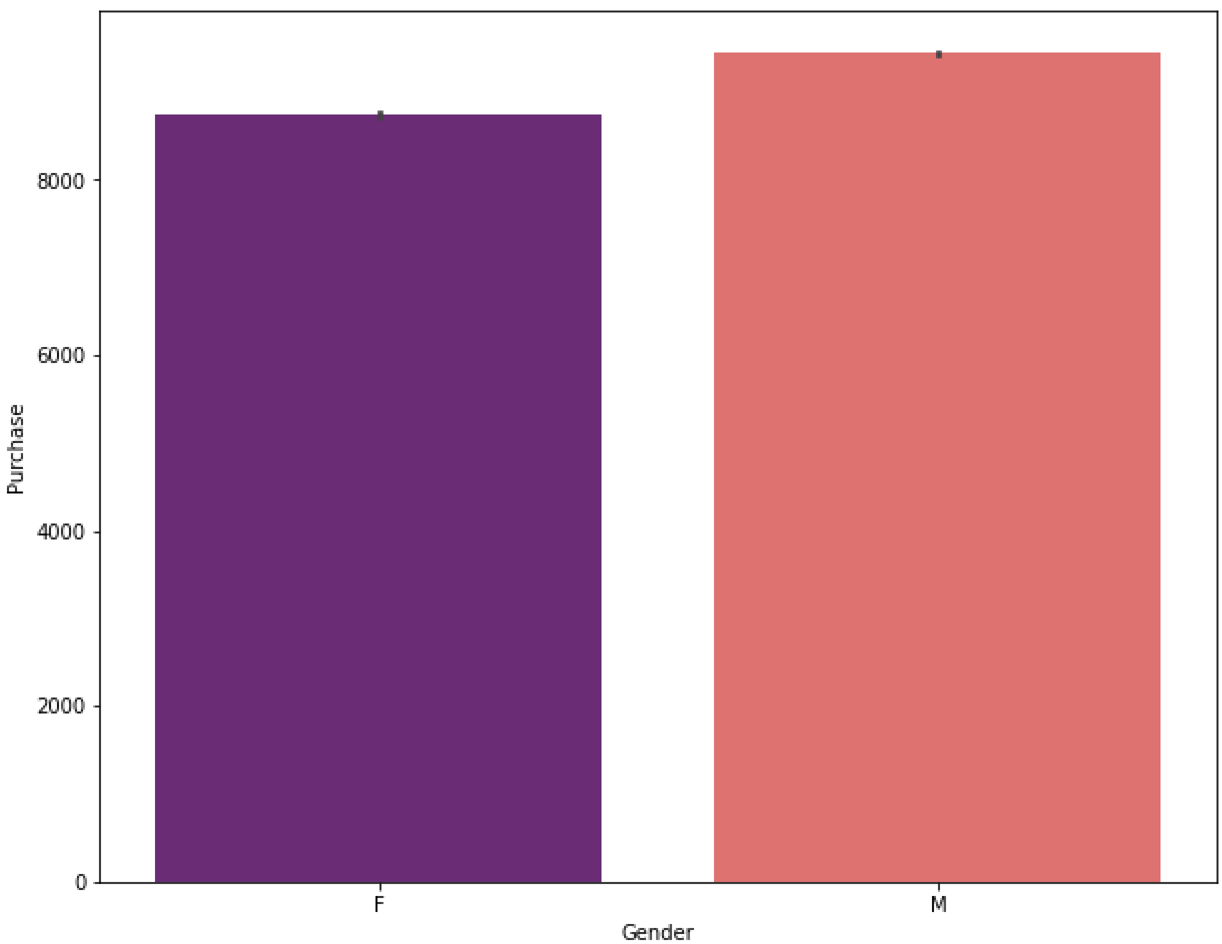

```
1 plt.figure(figsize = (10, 6))
2 sns.barplot(data = train, x = 'Gender', y = 'Marital_Status', palette = 'crest')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50458fec40>
```



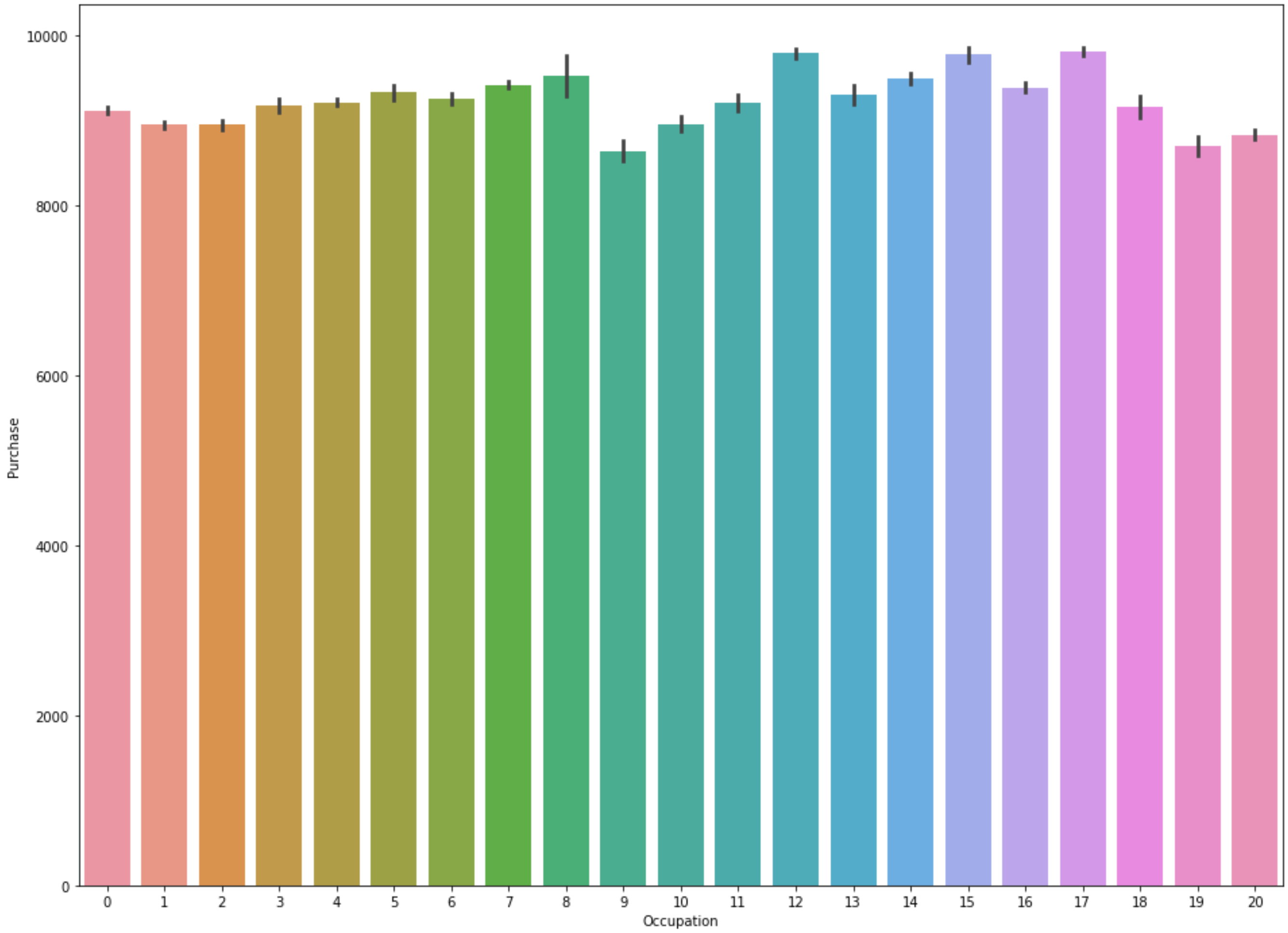

```
1 plt.figure(figsize = (10, 8))
2 sns.barplot(data = train, x = 'Gender', y = 'Purchase', palette = 'magma')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50453e5520>
```



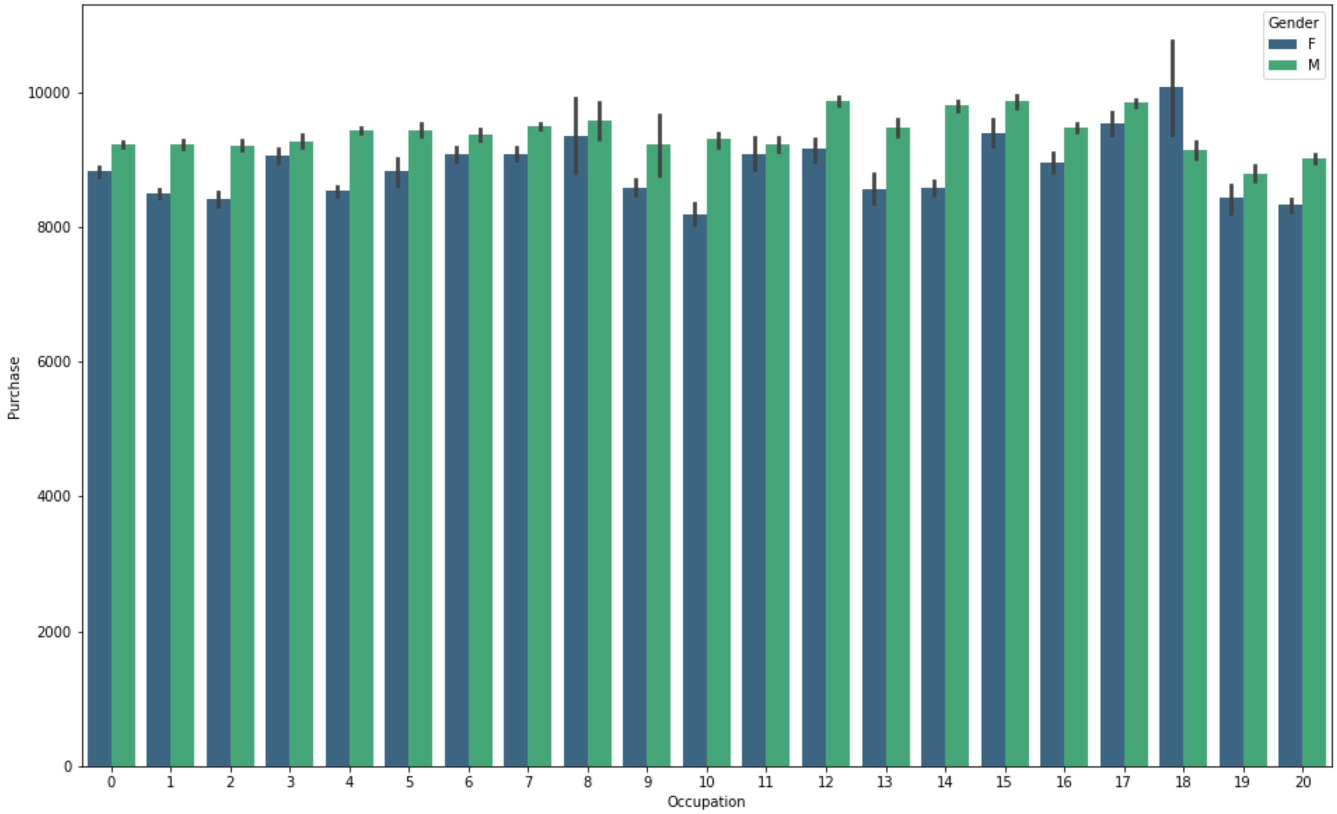
```
plt.figure(figsize = (16, 12))
sns.barplot(data = train, x = 'Occupation', y = 'Purchase')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50459a9250>
```




```
plt.figure(figsize = (16, 10))
sns.barplot(data = train, x = 'Occupation', y = 'Purchase', hue = 'Gender', palett
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f5045263ee0>



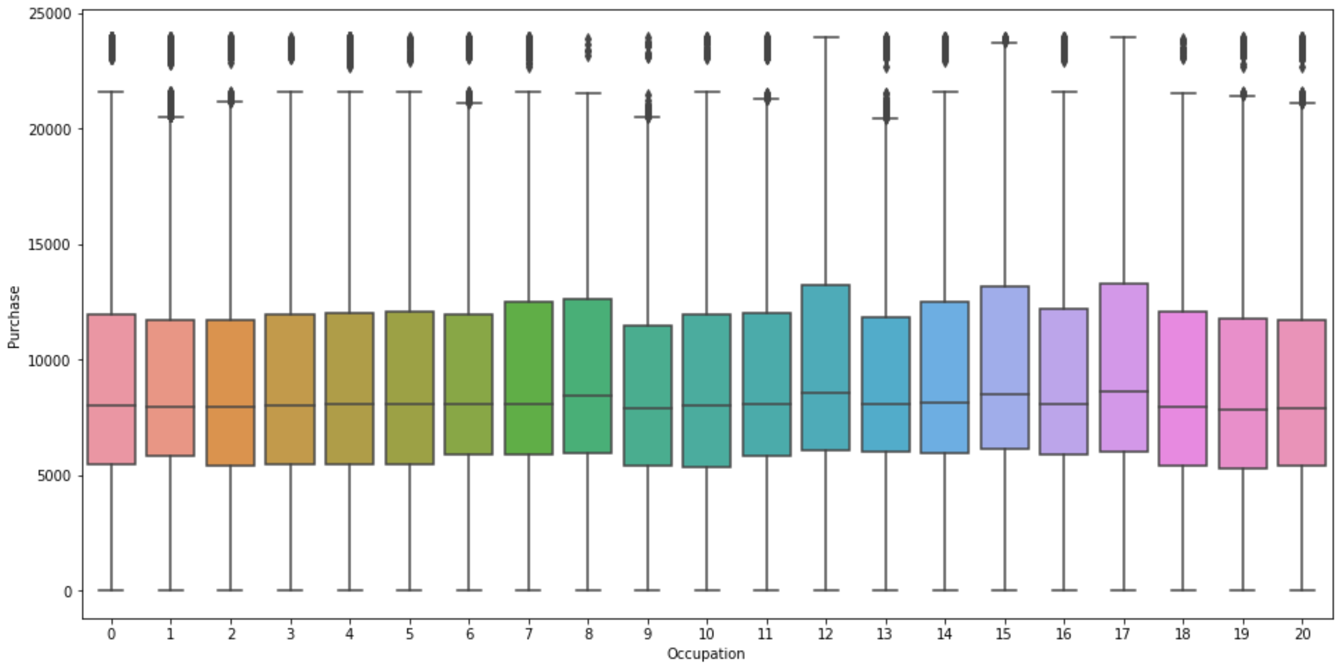
Outlier Detection

```
1 plt.figure(figsize = (10, 6))
2 sns.boxplot(data = train, x = "Gender", y = "Purchase", palette = 'viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f504508a670>
```

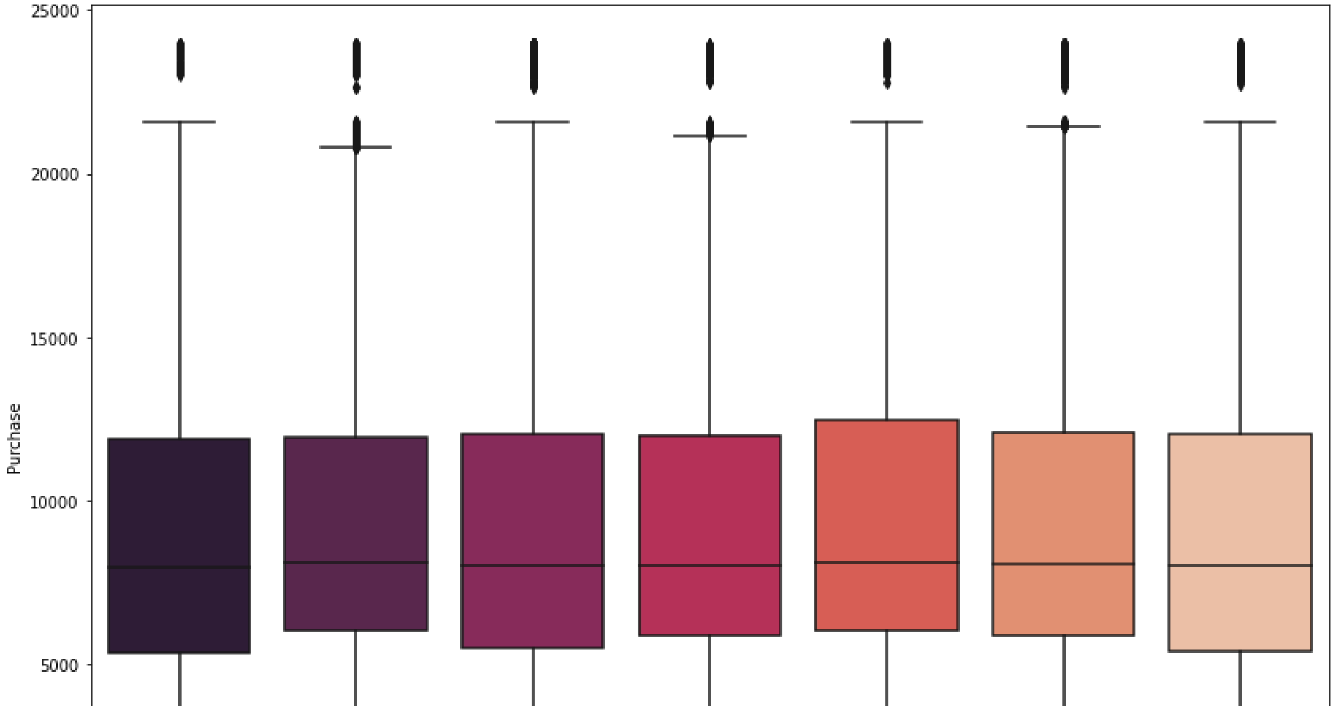

```
plt.figure(figsize = (16, 8))
sns.boxplot(data = train, x = "Occupation", y = "Purchase")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5044ffaa60>
```



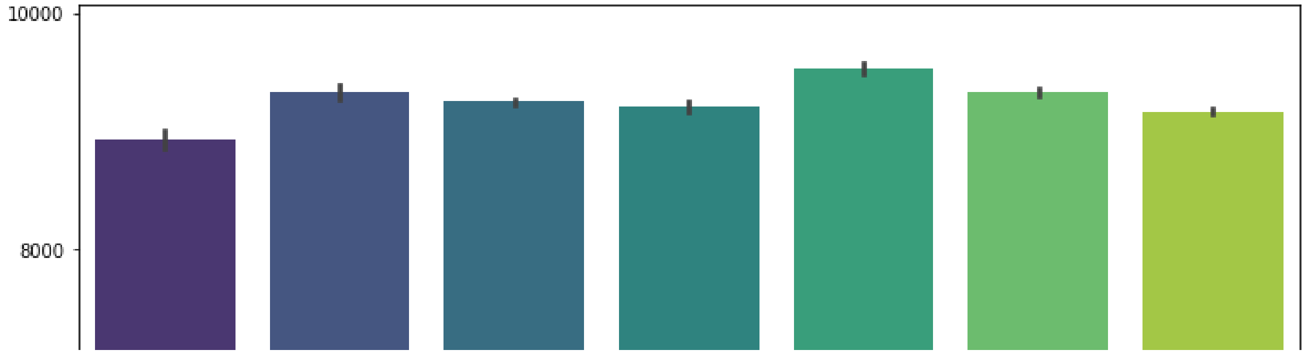

```
plt.figure(figsize = (14, 10))
sns.boxplot(data = train, x = "Age", y = "Purchase", palette = 'rocket')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50459124f0>
```



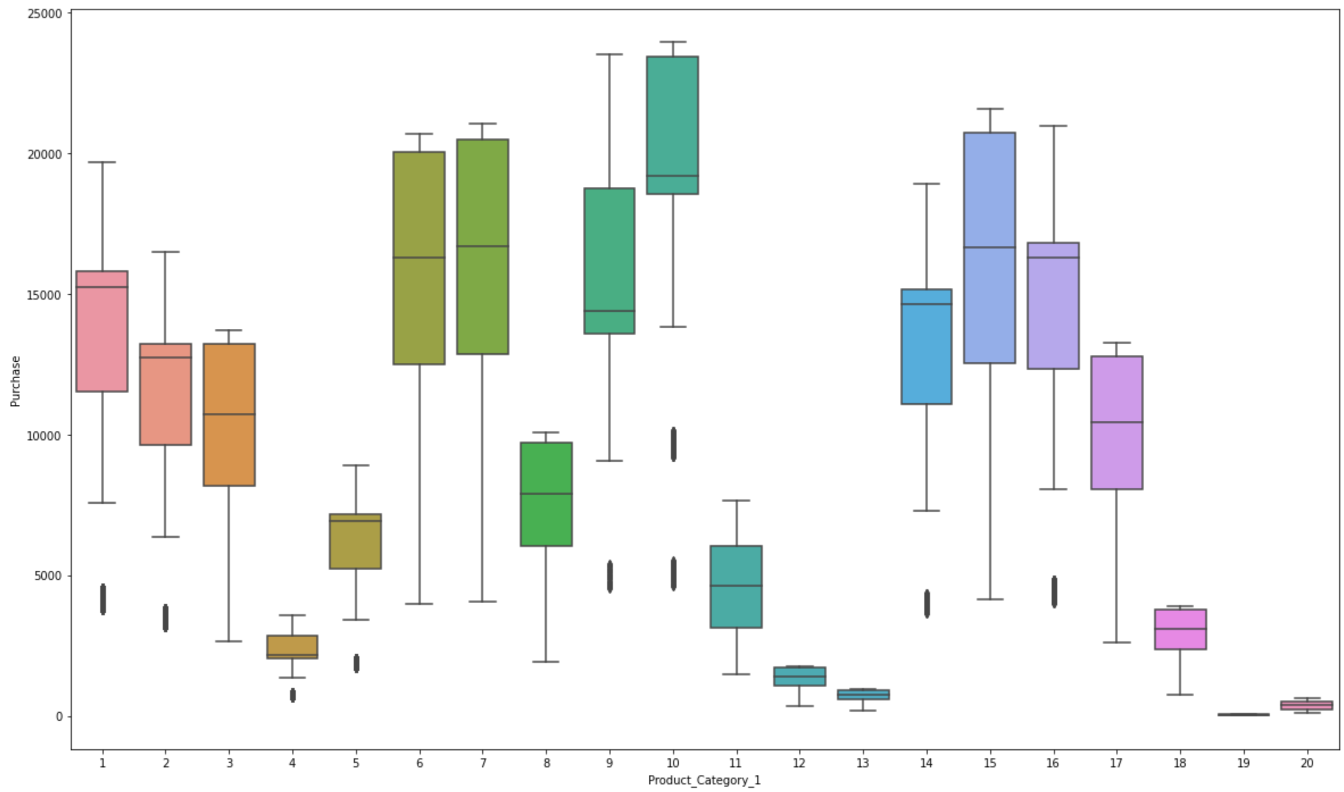

```
plt.figure(figsize = (12, 12))
sns.barplot(data = train, x = 'Age', y = 'Purchase', palette = 'viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5044d9cd30>
```



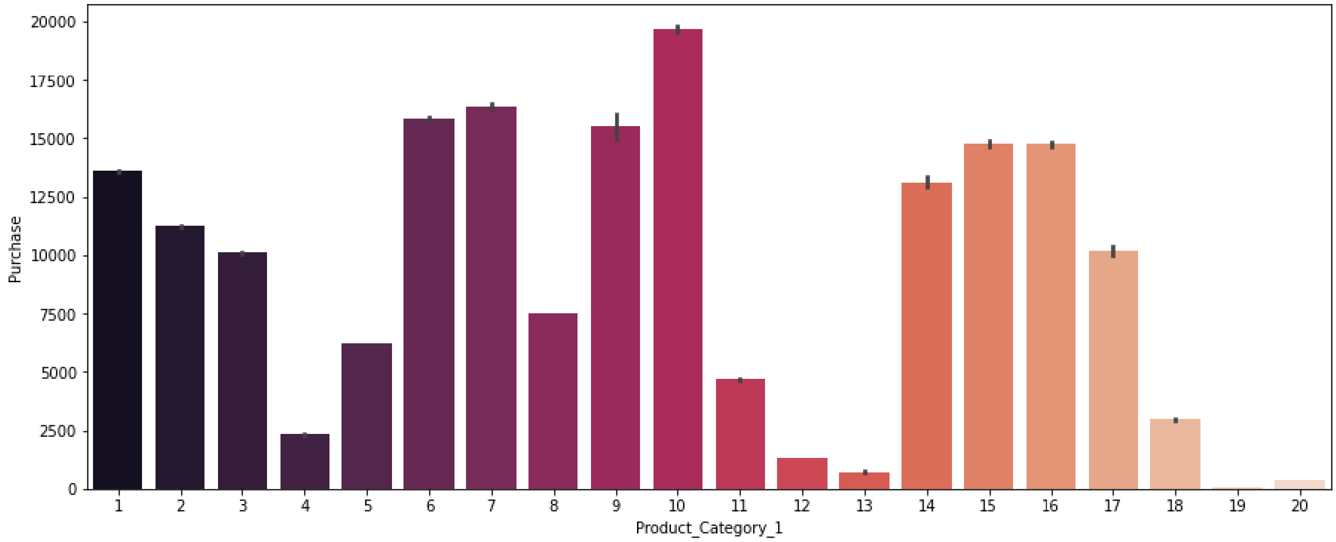

```
plt.figure(figsize = (20, 12))
sns.boxplot(data = train, x = "Product_Category_1", y = "Purchase")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5044da6760>
```



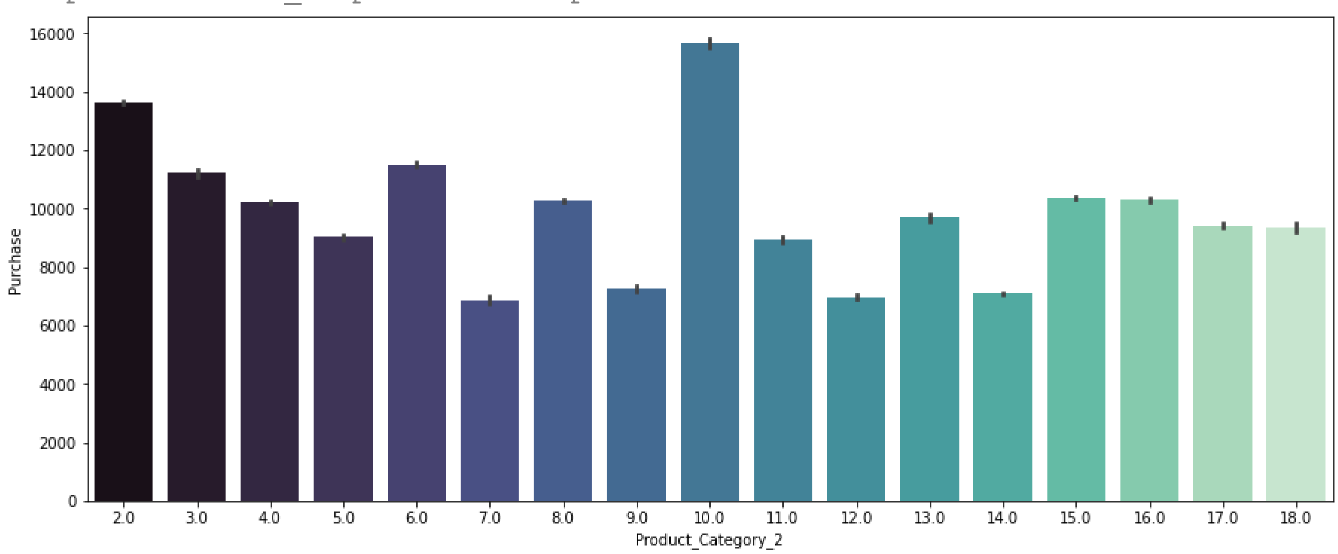

```
plt.figure(figsize = (15, 6))
sns.barplot(data = train, x = 'Product_Category_1', y = 'Purchase', palette = 'roc
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5044b0d1f0>
```



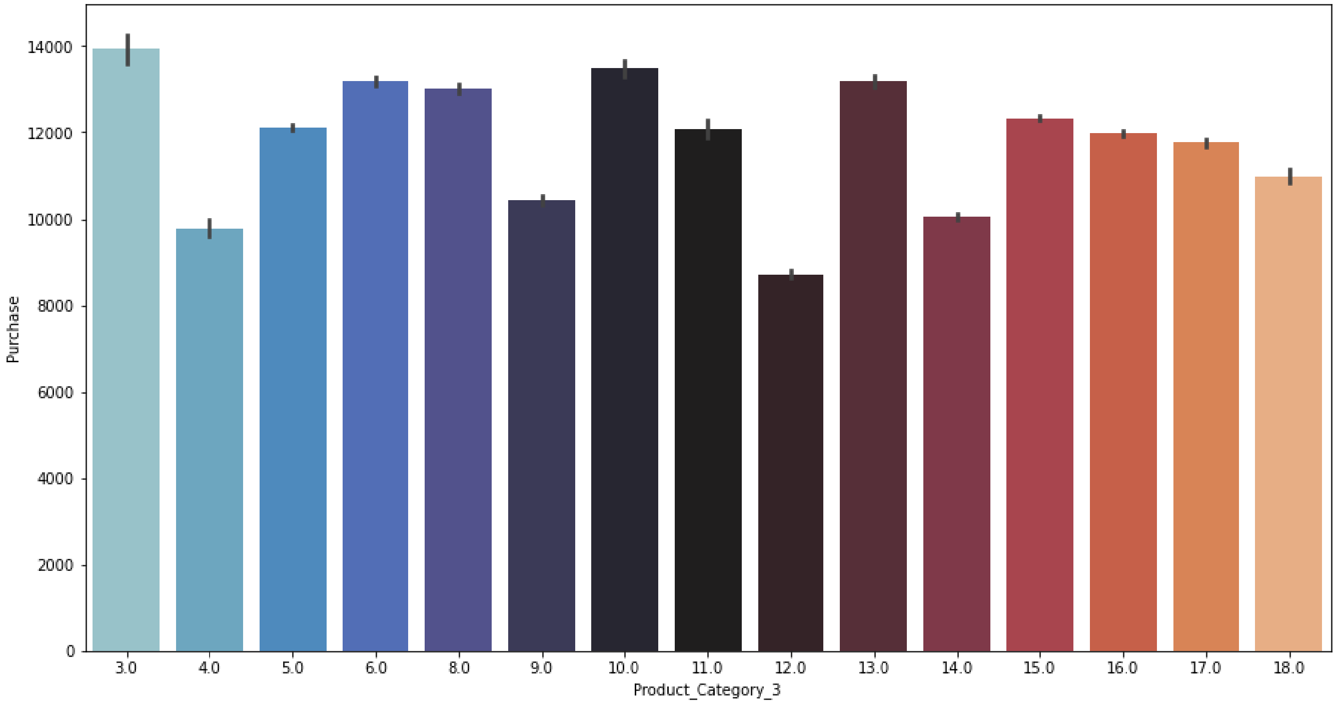

```
1 plt.figure(figsize = (15, 6))
2 sns.barplot(data = train, x = 'Product_Category_2', y = 'Purchase', palette = 'mak
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5044ad3a30>
```



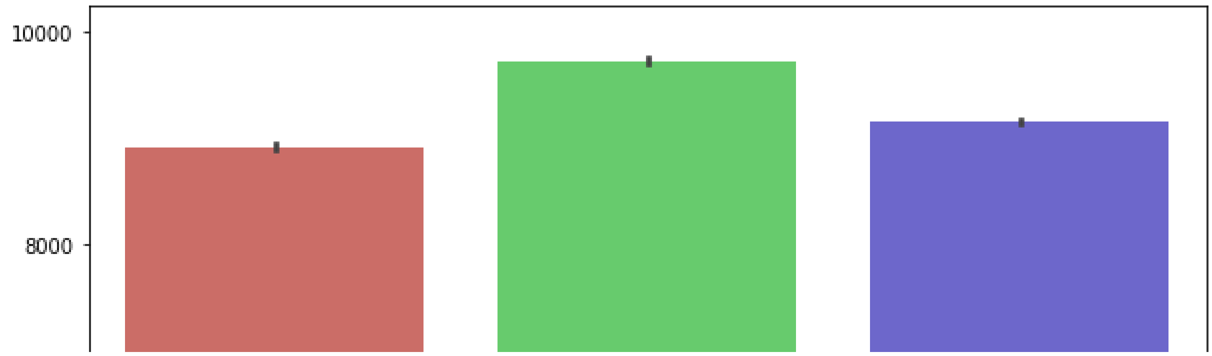

```
1 plt.figure(figsize = (15, 8))
2 sns.barplot(data = train, x = 'Product_Category_3', y = 'Purchase', palette = 'ice
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5044b66280>
```



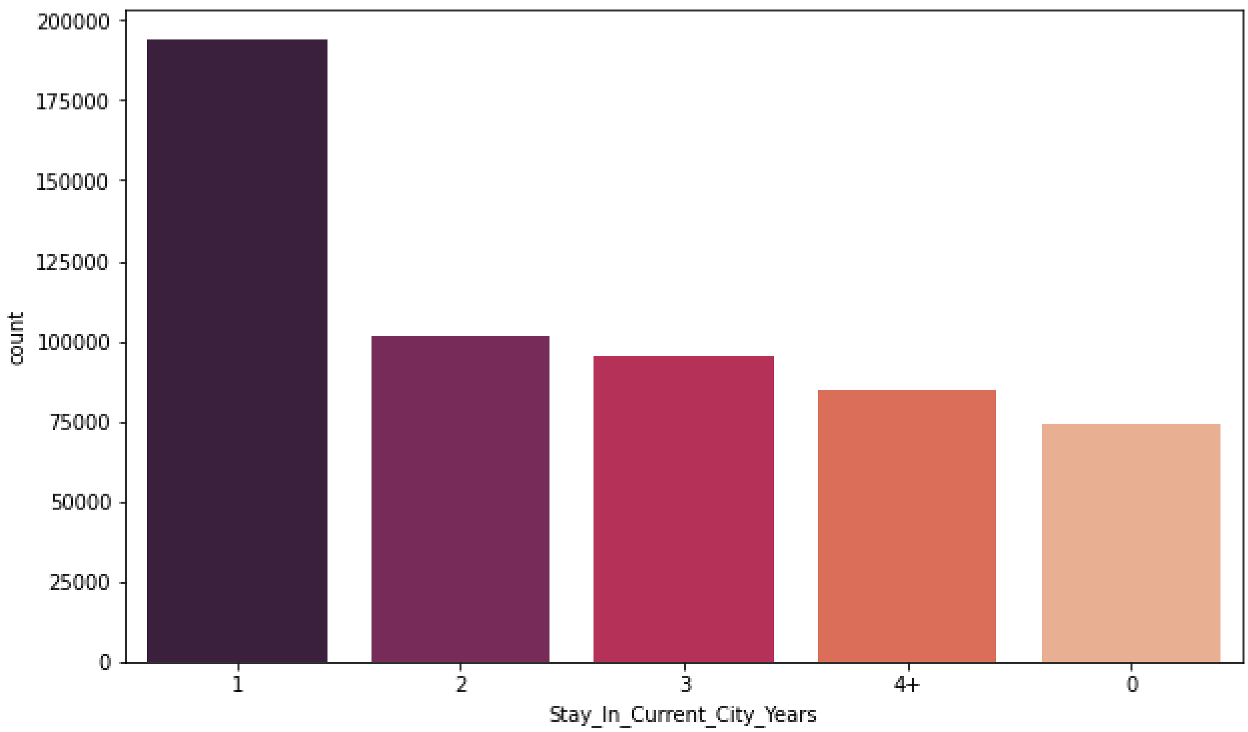

```
1 plt.figure(figsize = (10, 10))
2 sns.barplot(data = train, x = 'City_Category', y = 'Purchase', palette = 'hls')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50448e3eb0>
```



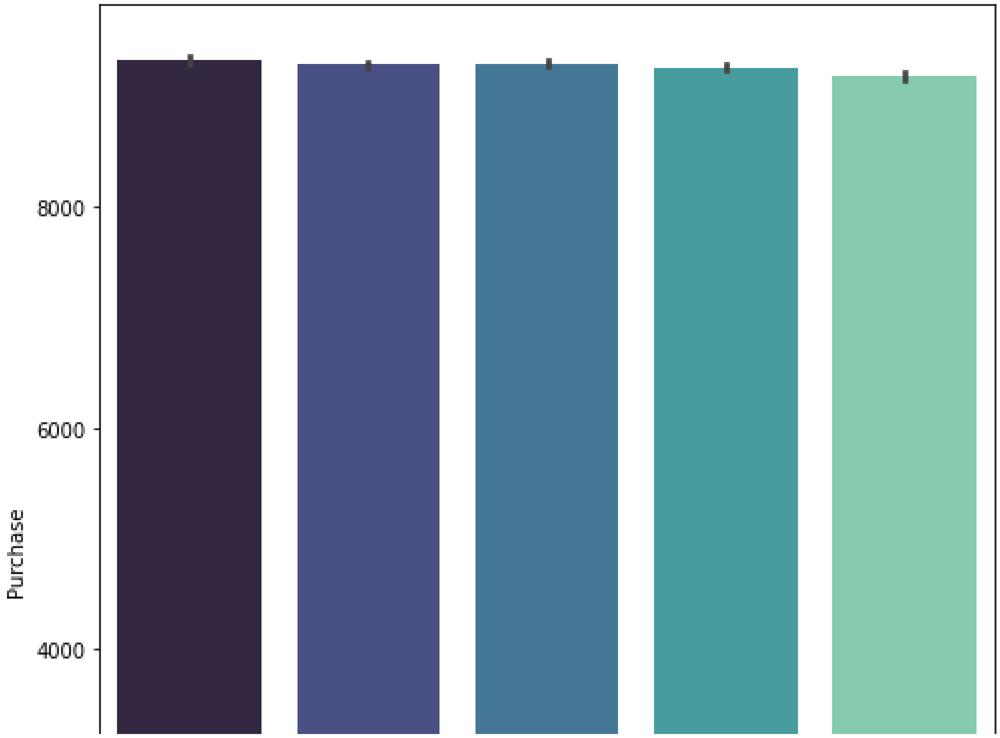

```
plt.figure(figsize = (10, 6))
sns.countplot(data = train, x = 'Stay_In_Current_City_Years', palette = 'rocket',
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5044a63490>
```



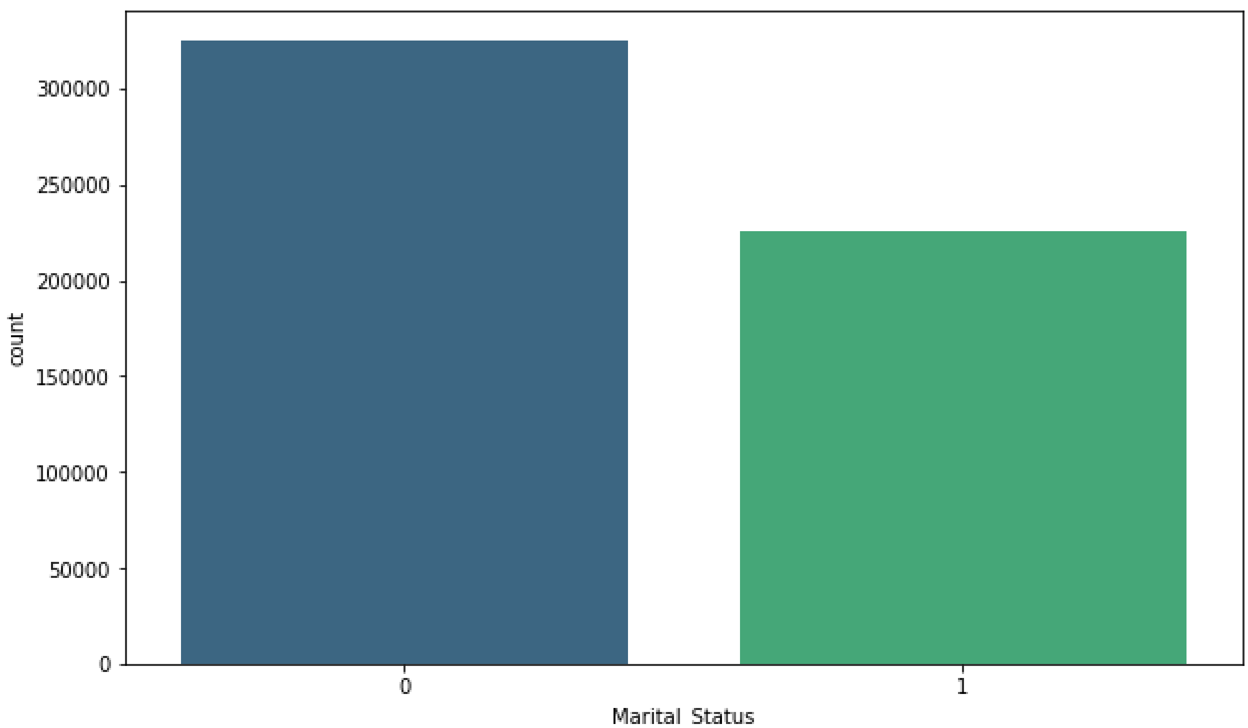

```
plt.figure(figsize = (8, 10))
sns.barplot(data = train, x = 'Stay_In_Current_City_Years', y = 'Purchase', palett
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f50449a4fa0>

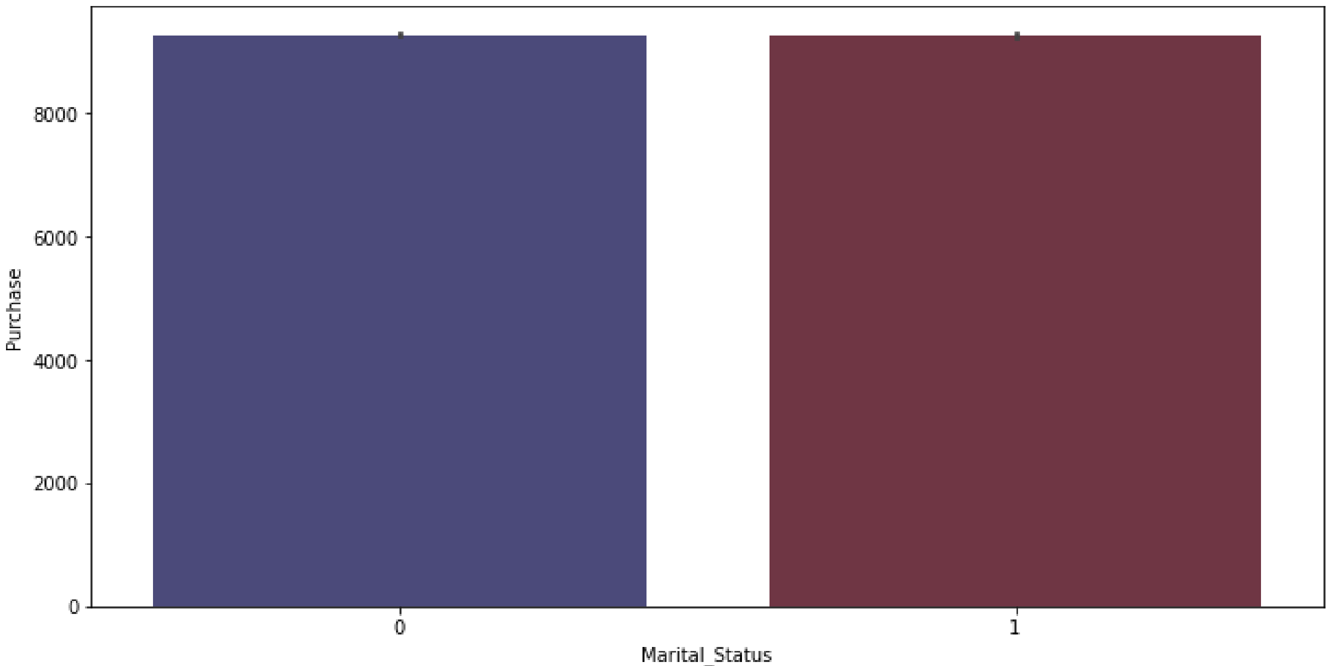



```
1 plt.figure(figsize = (10, 6))
2 sns.countplot(data = train, x = 'Marital_Status', palette = 'viridis')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f5044e2c580>

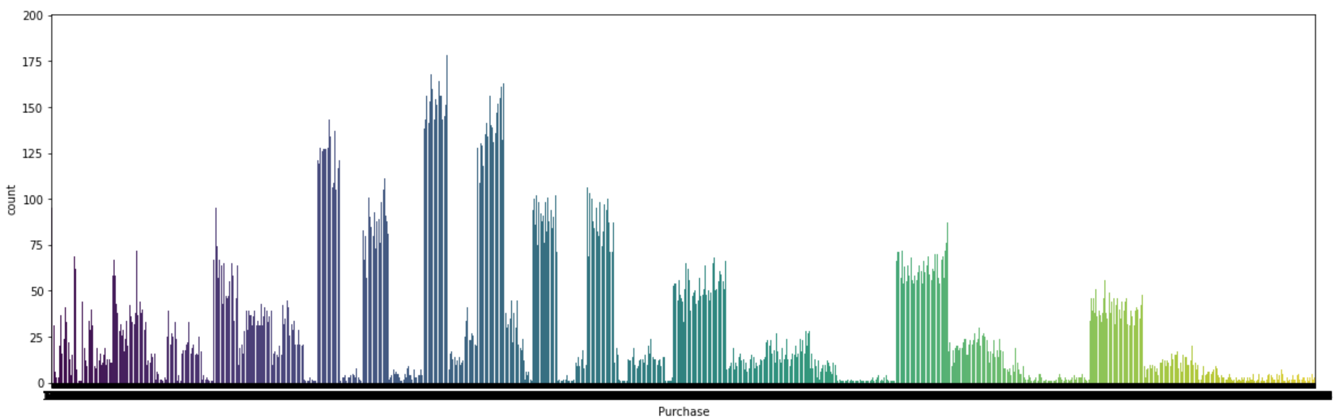



```
1 plt.figure(figsize = (12, 6))
2 sns.barplot(data = train, x = 'Marital_Status', y = 'Purchase', palette = 'icefire
```



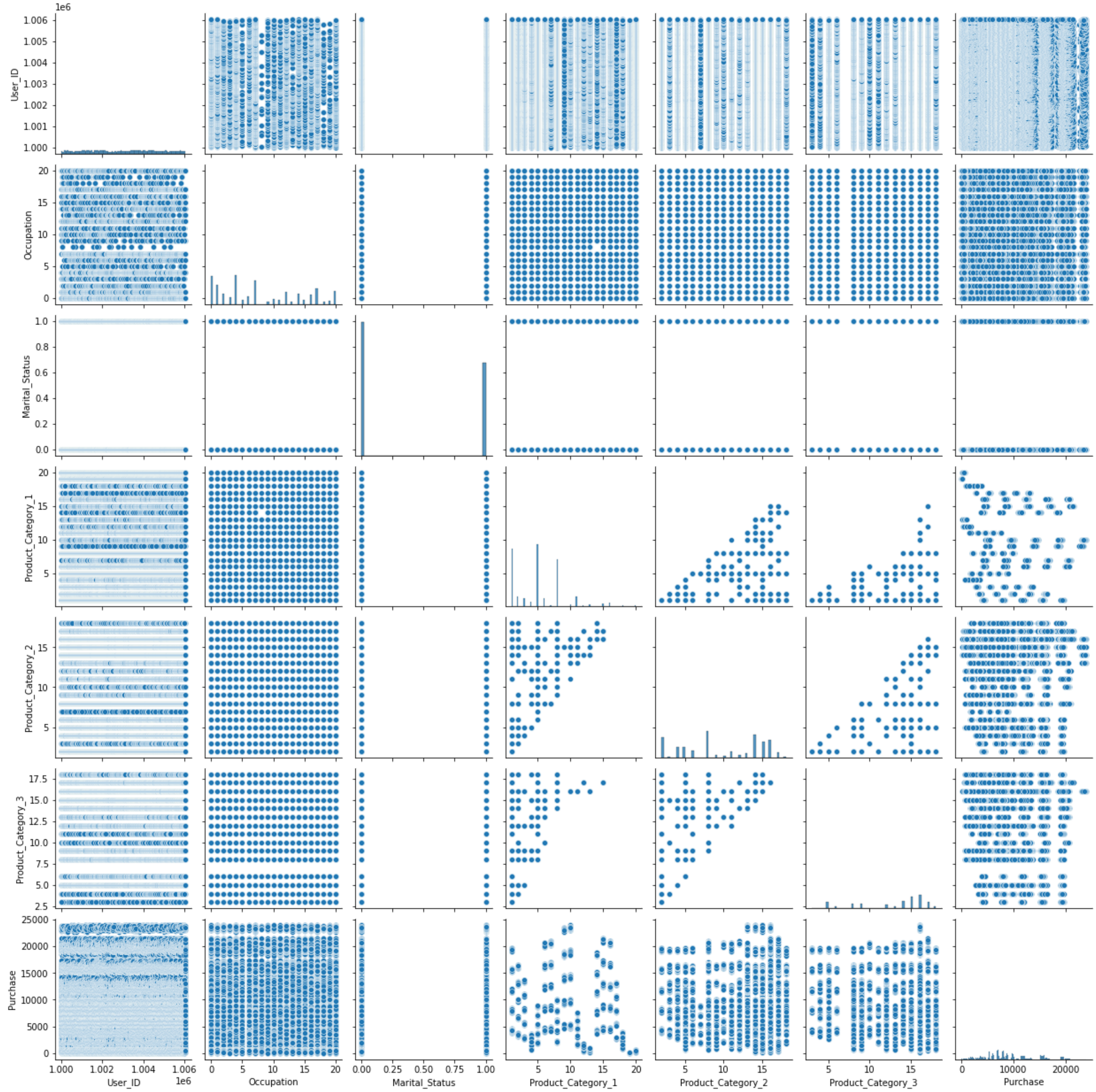

```
1 plt.figure(figsize = (20, 6))
2 sns.countplot(data = train, x = 'Purchase', palette = 'viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f50446f4250>
```




```
1 sns.pairplot(data = train, height = 2.5, aspect = 1)
```

```
<seaborn.axisgrid.PairGrid at 0x7f5044d11d00>
```




```
1 train
```

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

550068 rows × 12 columns

```
# We are replacing 'P00' with no value and scaling the ProductID column.
train['Product_ID'] = train['Product_ID'].str.replace('P00', '')
StScale = StandardScaler()
train['Product_ID'] = StScale.fit_transform(train['Product_ID'].values.reshape(-1,
```

```
# There are more than 50 percent missing values present in the Product_category_co
train.drop(['Product_Category_3'], axis = 1, inplace = True)
```

```
# The missing data in the product category 2 column have been estimated using mean
train['Product_Category_2'] = train['Product_Category_2'].fillna(train['Product_Ca
```

```
train.isnull().sum()
```

```
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
Purchase                      0
dtype: int64
```

```
train
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Curr
0	1000001	-1.028774	F	0-17	10	A	
1	1000001	0.722139	F	0-17	10	A	
2	1000001	-0.845799	F	0-17	10	A	
3	1000001	-0.869157	F	0-17	10	A	
4	1000002	1.077382	M	55+	16	C	
...	...	...	...	...	...	...	
550063	1006033	1.924156	M	51-55	13	B	
550064	1006035	1.953267	F	26-35	1	C	
550065	1006036	1.953267	F	26-35	15	B	
550066	1006038	1.953267	F	55+	1	C	
550067	1006039	1.916360	F	46-50	0	B	

550068 rows × 11 columns

```
# The Label Encoding technique will now replace all the categorical variables to n
category_cols = ['Gender', 'City_Category', 'Age']
LaNcode = LabelEncoder()
for i in category_cols:
    train[i] = LaNcode.fit_transform(train[i])
train.dtypes
```

```
User_ID                          int64
Product_ID                     float64
Gender                           int64
Age                              int64
Occupation                       int64
City_Category                    int64
Stay_In_Current_City_Years      object
Marital_Status                   int64
Product_Category_1               int64
Product_Category_2             float64
Purchase                         int64
dtype: object
```

```
# Values in the Stay_In_Current_City_Years column will be changed from '4+' to '4'
train['Stay_In_Current_City_Years'] = train['Stay_In_Current_City_Years'].replace(
```

```
# The Gender, Age and Stay_In_Current_City_Years values will be changed to int.
train['Gender'] = train['Gender'].astype(int)
train['Age'] = train['Age'].astype(int)
train['Stay_In_Current_City_Years'] = train['Stay_In_Current_City_Years'].astype(i
```

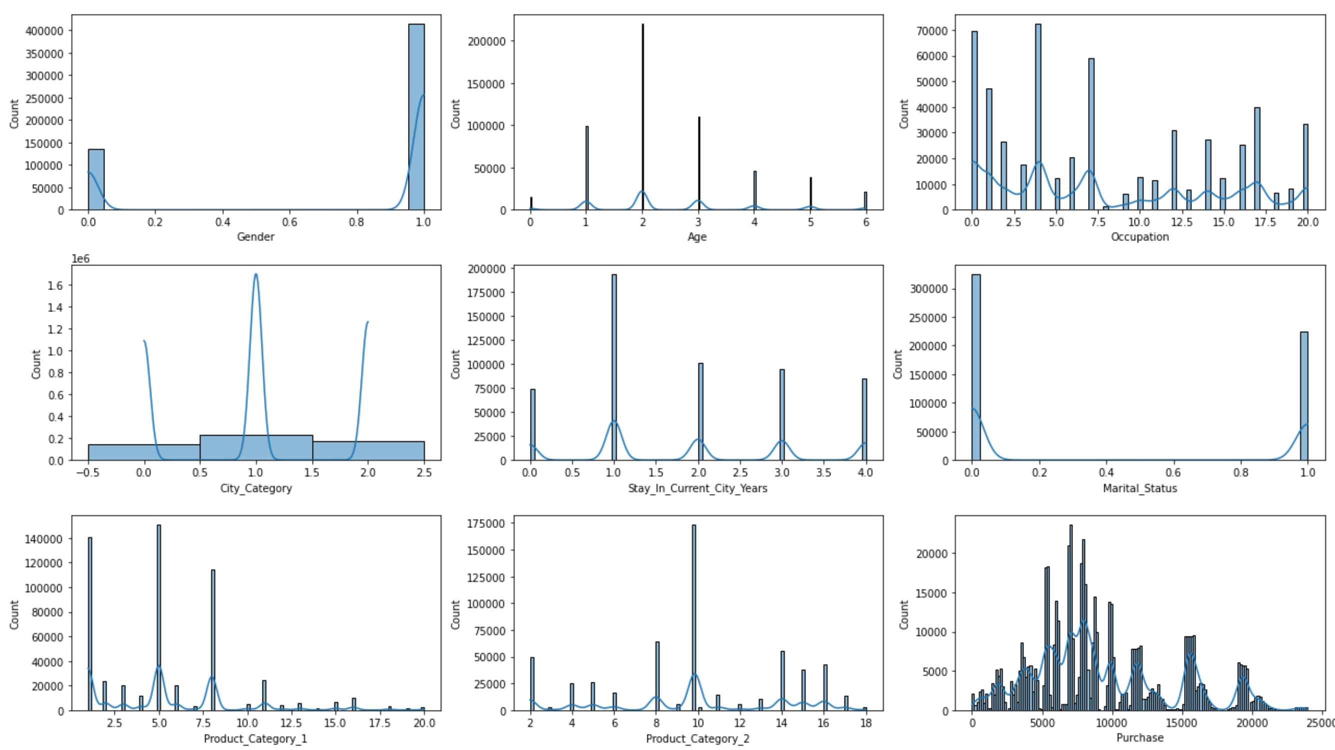
```
# The type of city_category is being changed from int to category.
train['City_Category'] = train['City_Category'].astype('category')
```

```
train
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Curr
0	1000001	1.028774	0	0	10	0	
2	1000001	-0.845799	0	0	10	0	

Distribution plot

```
columns = 3
rows = 3
fig, ax = plt.subplots(nrows = rows, ncols = columns, figsize = (18,10))
col = train.columns
index = 2
for iter in range(rows):
    for jter in range(columns):
        sns.histplot(train[col[index]], ax = ax[iter][jter], kde = True, linewidth
        index = index + 1
plt.tight_layout()
```



Log transformation

```
train['Purchase'] = np.log(train['Purchase'])
# The log transformation will help us transform the data and change the data to no
```

```
train = pd.get_dummies(train)
train.head()
# The get_dummies() function is used to convert categorical variable into dummy/in
```

	User_ID	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Mari
0	1000001	-1.028774	0	0	10	2	
1	1000001	0.722139	0	0	10	2	
2	1000001	-0.845799	0	0	10	2	
3	1000001	-0.869157	0	0	10	2	
4	1000002	1.077382	1	6	16	4	

Train - Test Split

```
# The data has been split into X and Y where independent and dependent variables h
X = train.drop(labels = ['Purchase'], axis = 1)
Y = train['Purchase']
X.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Mari
0	1000001	-1.028774	0	0	10	2	
1	1000001	0.722139	0	0	10	2	
2	1000001	-0.845799	0	0	10	2	
3	1000001	-0.869157	0	0	10	2	
4	1000002	1.077382	1	6	16	4	

```
# Target column.
Y
```

```
0        9.032409
```

```
1         9.629051
2         7.259820
3         6.963190
4         8.983314
            ...
550063    5.908083
550064    5.916202
550065    4.919981
550066    5.899897
550067    6.194405
Name: Purchase, Length: 550068, dtype: float64
```

```
# 80 percent data is used for training purpose and 20 percent is used for testing.
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
# The data has now been split into Train and test.
```

```
(440054, 12) (110014, 12) (440054,) (110014,)
```

Scaling our data

```
ScalD = StandardScaler()
X_train = ScalD.fit_transform(X_train)
X_test = ScalD.transform(X_test)
# StandardScaler standardizes a feature by subtracting the mean and then scaling t
```

Machine Learning.

Linear Regression

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

```
LinearRegression()
```

```
Y_predict = LinMod.predict(X_test)
## Predicting on X_test
```

```
score = r2_score(Y_test, Y_predict)
mae = mean_absolute_error(Y_test, Y_predict)
mse = mean_squared_error(Y_test, Y_predict)
rmse = (np.sqrt(mean_squared_error(Y_test, Y_predict)))
print('r2_score = ', score*100)
```

```
print('mean_absolute_error = ', mae*100)
print('mean_squared_error = ', mse*100)
print('root_mean_squared_error = ', rmse*100)
print('-'*100)
```

```
r2_score =  20.164250230005532
mean_absolute_error =  45.565599521249815
mean_squared_error =  44.379625352132756
root_mean_squared_error =  66.61803460935542
----------------------------------------------------------------------------------------------------
```

Decision Tree Regressor

```
DecTree = DecisionTreeRegressor(max_depth = 9)
DecTree.fit(X_train, Y_train)
```

```
DecisionTreeRegressor(max_depth=9)
```

```
# Prediction on train & test.
train_preds = DecTree.predict(X_train)
test_preds = DecTree.predict(X_test)
```

```
RMSE_train = (np.sqrt(metrics.mean_squared_error(Y_train, train_preds)))
RMSE_test = (np.sqrt(metrics.mean_squared_error(Y_test, test_preds)))
print("RMSE TrainingData = ",str(RMSE_train*100))
print("RMSE TestData = ",str(RMSE_test*100))
print('-'*100)
print('RSquared value on train = ', (DecTree.score(X_train, Y_train)*100))
print('RSquared value on test = ', (DecTree.score(X_test, Y_test)*100))
```

```
RMSE TrainingData =  36.80408214406252
RMSE TestData =  36.89276274553602
----------------------------------------------------------------------------------------------------
RSquared value on train =  75.19510621944242
RSquared value on test =  75.51522403783467
```

XGBoost Regressor

```
XGBoosted = xgb.XGBRegressor(objective ='reg:linear', verbosity = 0, random_state=
XGBoosted.fit(X_train, Y_train)
```

```
XGBRegressor(random_state=42, verbosity=0)
```

```
# Prediction on train & test.
train_predsxg = XGBoosted.predict(X_train)
```

```
test_predsxg = XGBoosted.predict(X_test)
```

```
RMSE_trainxg = (np.sqrt(metrics.mean_squared_error(Y_train, train_predsxg)))
RMSE_testxg = (np.sqrt(metrics.mean_squared_error(Y_test, test_predsxg)))
print("RMSE TrainingData = ",str(RMSE_trainxg*100))
print("RMSE TestData = ",str(RMSE_testxg*100))
print('-'*100)
print('RSquared value on train = ', (XGBoosted.score(X_train, Y_train)*100))
print('RSquared value on test = ', (XGBoosted.score(X_test, Y_test)*100))
```

```
RMSE TrainingData =  37.42268465193167
RMSE TestData =  37.41566498410511
----------------------------------------------------------------------------------------------------
RSquared value on train =  74.35425799964118
RSquared value on test =  74.81623180486669
```

Random Forest Regressor

```
RandFor = RandomForestRegressor()
RandFor.fit(X_train, Y_train)
```

```
RandomForestRegressor()
```

```
# Prediction on train & test.
train_preds1 = RandFor.predict(X_train)
test_preds1 = RandFor.predict(X_test)
```

```
RMSE_train = (np.sqrt(metrics.mean_squared_error(Y_train, train_preds1)))
RMSE_test = (np.sqrt(metrics.mean_squared_error(Y_test, test_preds1)))
print("RMSE TrainingData = ",str(RMSE_train*100))
print("RMSE TestData = ",str(RMSE_test*100))
print('-'*100)
print('RSquared value on train = ', (RandFor.score(X_train, Y_train)*100))
print('RSquared value on test = ', (RandFor.score(X_test, Y_test)*100))
```

```
RMSE TrainingData =  13.145791621373679
RMSE TestData =  34.95857171273171
----------------------------------------------------------------------------------------------------
RSquared value on train =  96.83539961460662
RSquared value on test =  78.01526992845884
```

```
test = pd.read_csv('/content/test.csv')
# Loading the test dataset.
```

```
test
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Curr
0	1000004	P00128942	M	46-50	7	B	
1	1000009	P00113442	M	26-35	17	C	
2	1000010	P00288442	F	36-45	1	B	
3	1000010	P00145342	F	36-45	1	B	
4	1000011	P00053842	F	26-35	1	C	
...	...	...	...	...	...	...	
233594	1006036	P00118942	F	26-35	15	B	
233595	1006036	P00254642	F	26-35	15	B	
233596	1006036	P00031842	F	26-35	15	B	
233597	1006037	P00124742	F	46-50	1	C	
233598	1006039	P00316642	F	46-50	0	B	

233599 rows × 11 columns

```
test.isnull().sum()
# We are currently checking for missing values present in the test dataset.
```

```
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             72344
Product_Category_3            162562
dtype: int64
```

```
# The 'P00' value is being replaced into int and the ProductId column is being sca
```

```
test['Product_ID'] = test['Product_ID'].str.replace('P00', '')
StanScald = StandardScaler()
test['Product_ID'] = StanScald.fit_transform(test['Product_ID'].values.reshape(-1,
```

```
test.drop(['Product_Category_3'], axis = 1, inplace = True)
# As the Product_Category_3 column in the train set has been removed, we'll be doi
```

```
test['Product_Category_2'] = test['Product_Category_2'].fillna(test['Product_Categ
# Product_Category_2 has been estimated and is being filled with the mean.
```

```
test.isnull().sum()
# Finally, we have dealt with all null values in the test dataframe as well.
```

```
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
dtype: int64
```

```
test
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Curr
0	1000004	-0.434752	M	46-50	7	B	
1	1000009	-0.587188	M	26-35	17	C	
2	1000010	1.133865	F	36-45	1	B	
3	1000010	-0.273465	F	36-45	1	B	

Label Encoding our Categorical Data

```
# The label encoding technique replaces all the categorical variables to numeric o
categorie_cols = ['Gender', 'City_Category', 'Age']
LabelNcodr = LabelEncoder()
for i in categorie_cols:
    test[i] = LabelNcodr.fit_transform(test[i])
test.dtypes
```

```
User_ID                         int64
Product_ID                    float64
Gender                          int64
Age                             int64
Occupation                      int64
City_Category                   int64
Stay_In_Current_City_Years     object
Marital_Status                  int64
Product_Category_1              int64
Product_Category_2            float64
dtype: object
```

```
test['Stay_In_Current_City_Years'] = test['Stay_In_Current_City_Years'].replace('4
# The '4+' value in the Stay_In_Current_City_Years is being replaced with '4'.
```

```
# The values in the test set are being converted to integer types same as the trai
test['Gender'] = test['Gender'].astype(int)
test['Age'] = test['Age'].astype(int)
test['Stay_In_Current_City_Years'] = test['Stay_In_Current_City_Years'].astype(int
test['City_Category'] = test['City_Category'].astype('category')
```

```
test = pd.get_dummies(test)
# Dummies have been created for our test set.
```

```
test.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years	Mari
0	1000004	-0.434752	1	4	7	2	
1	1000009	-0.587188	1	2	17	0	
2	1000010	1.133865	0	3	1	4	
3	1000010	-0.273465	0	3	1	4	
4	1000011	-1.173330	0	2	1	1	

```
train.shape
# The train data's shape.
```

```
(550068, 13)
```

```
test.shape
# The test data's shape.
```

```
(233599, 12)
```

```
train
```

	User ID	Product ID	Gender	Age	Occupation	Stay In Current City Years

```
test
```

	User_ID	Product_ID	Gender	Age	Occupation	Stay_In_Current_City_Years
0	1000004	-0.434752	1	4	7	2
1	1000009	-0.587188	1	2	17	0
2	1000010	1.133865	0	3	1	4
3	1000010	-0.273465	0	3	1	4
4	1000011	-1.173330	0	2	1	1
...	...	...	...	...	...	...
233594	1006036	-0.533098	0	2	15	4
233595	1006036	0.801456	0	2	15	4
233596	1006036	-1.389691	0	2	15	4
233597	1006037	-0.476058	0	4	1	4
233598	1006039	1.411200	0	4	0	4

233599 rows × 12 columns

```
test_preds = RandFor.predict(test)
len(test_preds)
```

```
233599
```

```
id_frame = pd.read_csv('/content/test.csv')
```

```
ID_info = id_frame[["User_ID", "Product_ID"]]
ID_info.head()
# We're using the User_Id and Product_Id column from the test set.
```

User_ID	Product_ID

```
Predz = pd.DataFrame(test_preds, columns = ["Purchase"])
Predz["User_ID"] = ID_info["User_ID"]
Predz["Product_ID"] = ID_info["Product_ID"]
Predz.head()
# We'll save all our predictions in a dataframe.
```

	Purchase	User_ID	Product_ID
0	9.532573	1000004	P00128942
1	9.541557	1000009	P00113442
2	4.595913	1000010	P00288442
3	4.595913	1000010	P00145342
4	4.677515	1000011	P00053842

```
Predz.to_csv('Black_Friday_Predictions.csv', index = False)
# Finally, we're converting the prediction into a csv file named 'Black_Friday_Pre
```