

PROJECT ON
BLACK FRIDAY SALES USING ML
CS-5710
MACHINE LEARNING

SUBMITTED BY
SAI KRISHNA PERUKA

SUBMITTED TO
MUHAMMAD ZUBAIR KHAN

GITHUB Link: <https://github.com/Sai12345644/Project.git>

Importing set of libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
```

Reading the csv file which we have downloaded and placing the path below in order to display the dataset.

```
pd.read_csv("C:\\Users\\makut\\Downloads\\black friday sales\\Black-Friday-Sales-Prediction-master\\Data\\BlackFridaySales.csv")
```

head() displays the first 5 rows of the dataset

```
data.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Prod
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	

shape attribute in pandas is used to get the shape of a DataFrame

```
data.shape
```

```
(550068, 12)
```

info() is used to print the information of a DataFrame

```
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
```

Checking whether if we have any null values in the dataset

```
data.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	173638
Product_Category_3	383247
Purchase	0

dtype: int64

Checking null values in percentage for the dataset

```
data.isnull().sum()/data.shape[0]*100
```

User_ID	0.000000
Product_ID	0.000000
Gender	0.000000
Age	0.000000
Occupation	0.000000
City_Category	0.000000
Stay_In_Current_City_Years	0.000000
Marital_Status	0.000000
Product_Category_1	0.000000
Product_Category_2	31.566643
Product_Category_3	69.672659
Purchase	0.000000

dtype: float64

Checking unique values in the elements

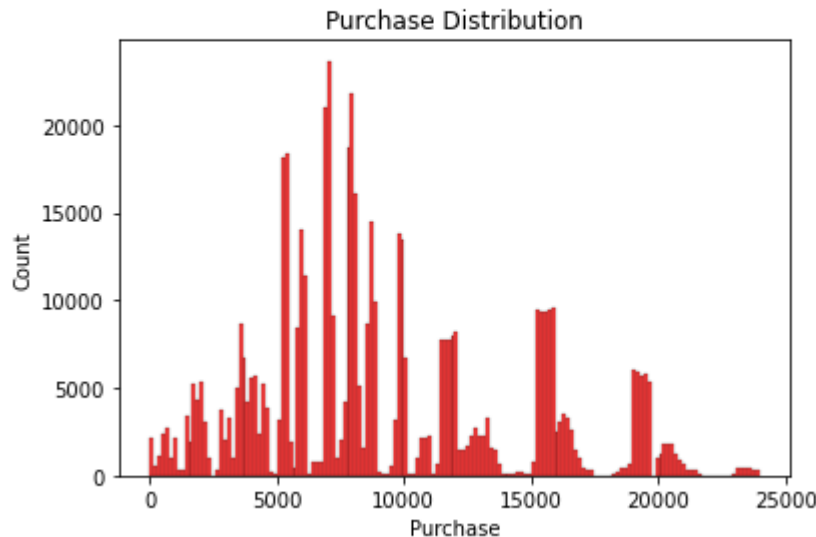
```
data.nunique()
```

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

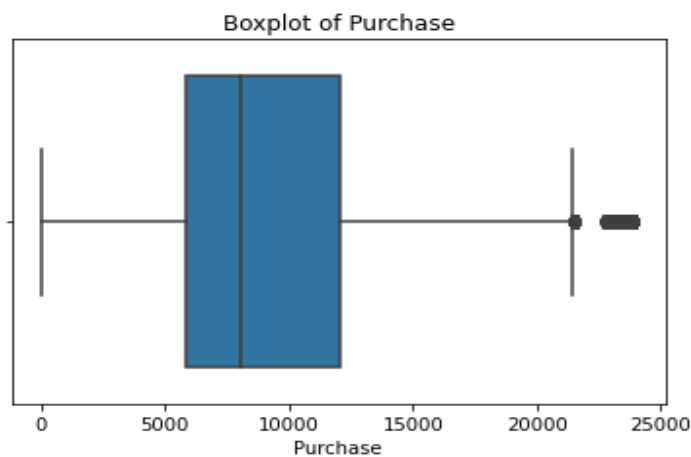
Histogram (histplot()) is used to display the distribution of the data and shows the values of your data in series

```
sns.histplot(data["Purchase"],color='r')
plt.title("Purchase Distribution")
plt.show()
```



(boxplot()) is used to display the groups of the numerical in the form quartiles

```
sns.boxplot(data["Purchase"])
plt.title("Boxplot of Purchase")
plt.show()
```



Skewness of the data which is present in the given axis of the dataframe, it denotes an asymmetric distribution

```
data["Purchase"].skew()
```

0.6001400037087128

Kurtosis is another method which is used when the data are heavy outliers or light – tailed to normal distribution

```
data["Purchase"].kurtosis()
```

-0.3383775655851702

Describe() gives the entire details of the dataset

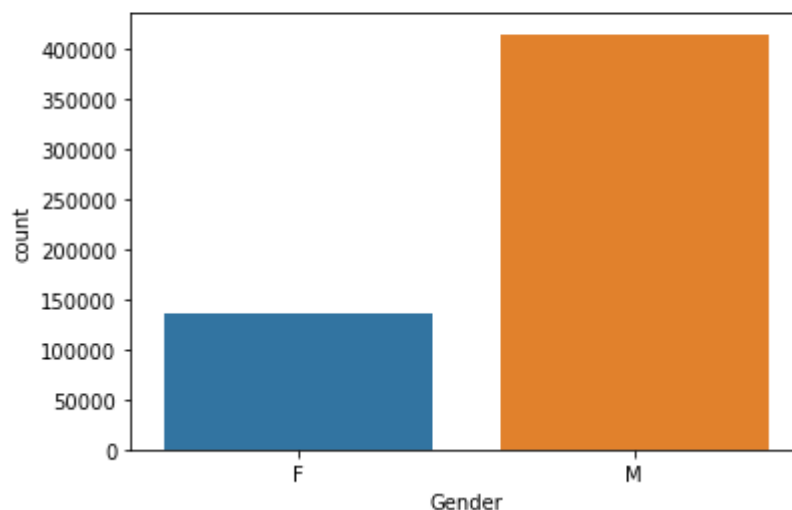
```
data["Purchase"].describe()
```

```
count    550068.000000
mean      9263.968713
std       5023.065394
min        12.000000
25%       5823.000000
50%       8047.000000
75%      12054.000000
max      23961.000000
Name: Purchase, dtype: float64
```

Countplot is used to display the counts of the observations in each category i.e for female and male

Gender

```
sns.countplot(data['Gender'])
plt.show()
```



Displaying the percentage of purchase which were made by both male and female

```
: data['Gender'].value_counts(normalize=True)*100
: M    75.310507
: F    24.689493
: Name: Gender, dtype: float64
```

Groupby is used for gender category so that it can separate the data into groups and mean() is used to get the average of the purchases.

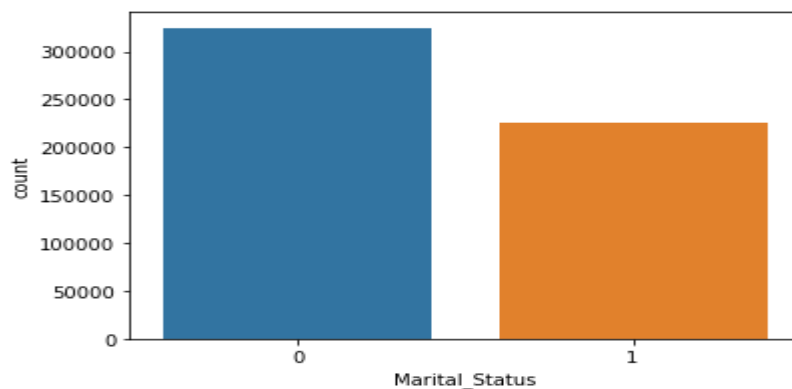
```
data.groupby("Gender").mean()["Purchase"]
```

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

Countplot is used to display the counts of the observations in each category i.e for married and unmarried

Marital Status

```
sns.countplot(data['Marital_Status'])
plt.show()
```



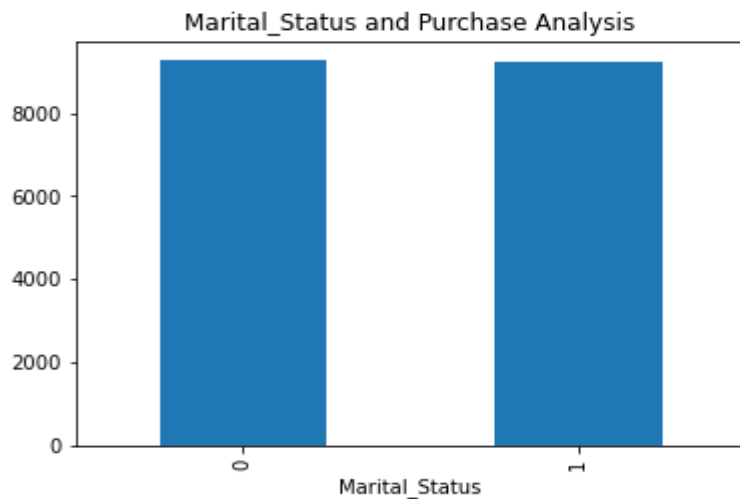
Groupby is used for marital_status so that it can separate the data into groups and mean() is used to get the average of the purchases made by unmarried and married people

```
data.groupby("Marital_Status").mean()["Purchase"]
```

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

Plotting a graph for the grouped data of marital_status and mean of purchases

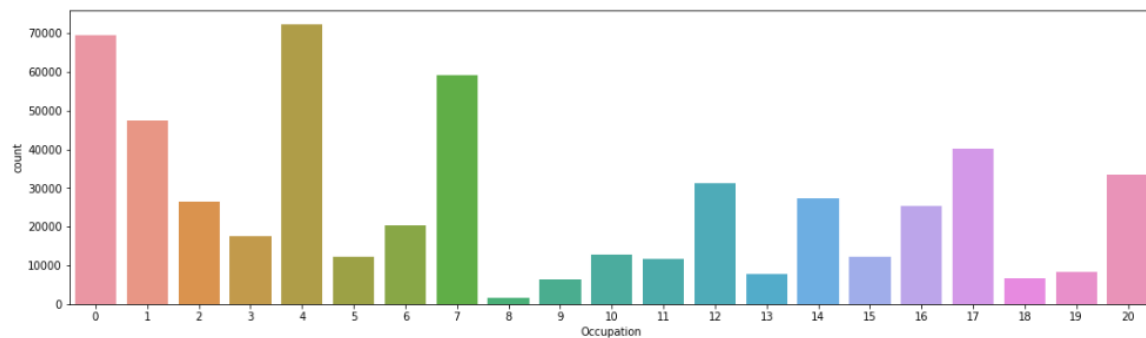
```
data.groupby("Marital_Status").mean()["Purchase"].plot(kind='bar')
plt.title("Marital_Status and Purchase Analysis")
plt.show()
```



Countplot is used to display the counts of the observations in each category i.e for occupation data

Occupation

```
: plt.figure(figsize=(18,5))
sns.countplot(data['Occupation'])
plt.show()
```



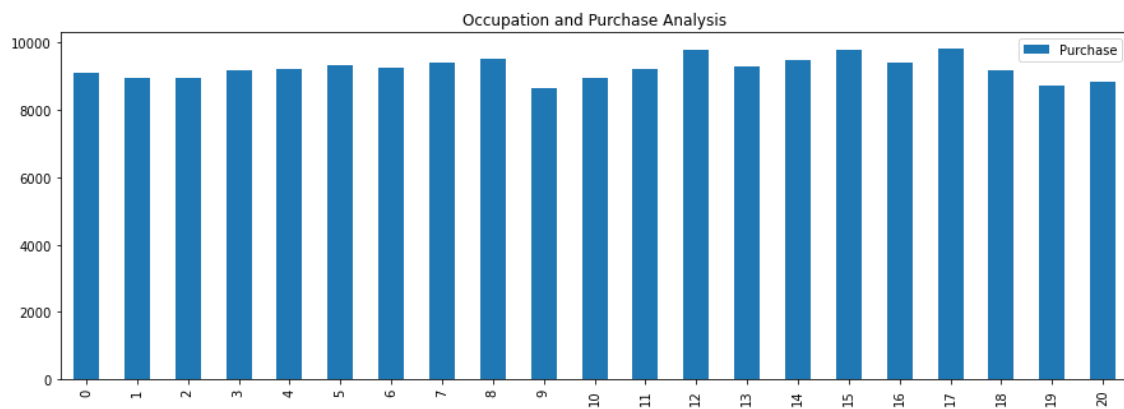
Groupby is used for occupation data so that it can separate the data into groups and mean() is used to get the average of the purchase of each occupation

```
occup = pd.DataFrame(data.groupby("Occupation").mean()["Purchase"])
occup
```

	Purchase
Occupation	
0	9124.428588
1	8953.193270
2	8952.481683
3	9178.593088
4	9213.980251
5	9333.149298
6	9256.535691
7	9425.728223
8	9532.592497
9	8637.743761
10	8959.355375
11	9213.845848
12	9796.640239
13	9306.351061

Plotting a graph for the grouped data of occupation and mean of purchases

```
|: occup.plot(kind='bar',figsize=(15,5))
plt.title("Occupation and Purchase Analysis")
plt.show()
```



Countplot is used to display the counts of the observations in each category i.e for City_Category data

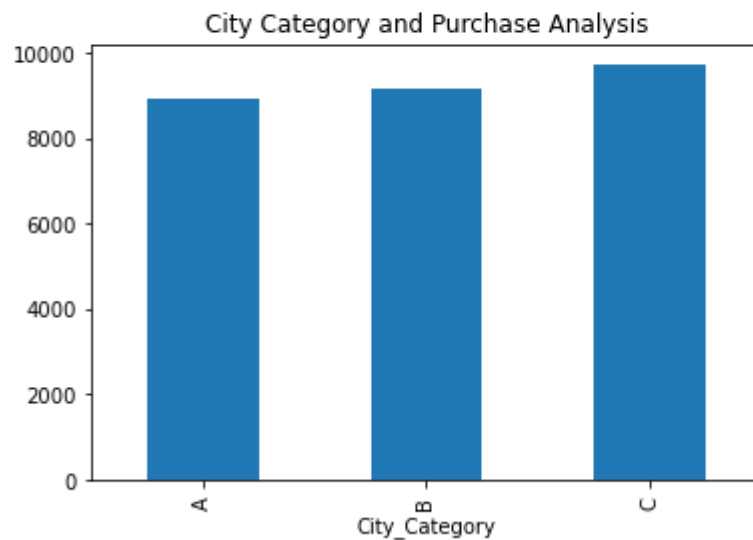
City_Category

```
: sns.countplot(data['City_Category'])  
plt.show()
```



Groupby is used for city_category data so that it can separate the data into groups and mean() is used to get the average of the purchase

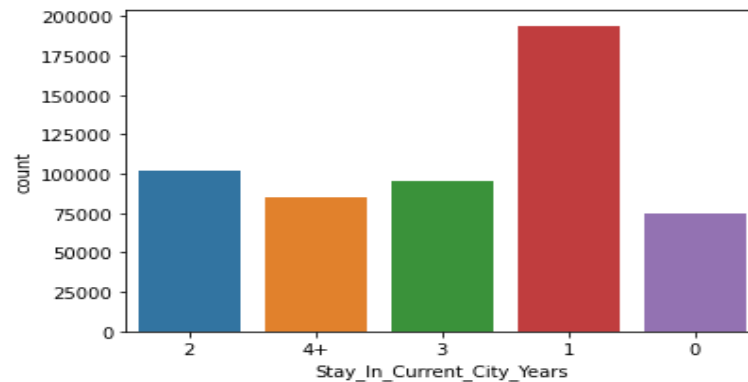
```
data.groupby("City_Category").mean()["Purchase"].plot(kind='bar')  
plt.title("City Category and Purchase Analysis")  
plt.show()
```



Countplot is used to display the counts of the observations in each category i.e for stay_in_current_city_years.

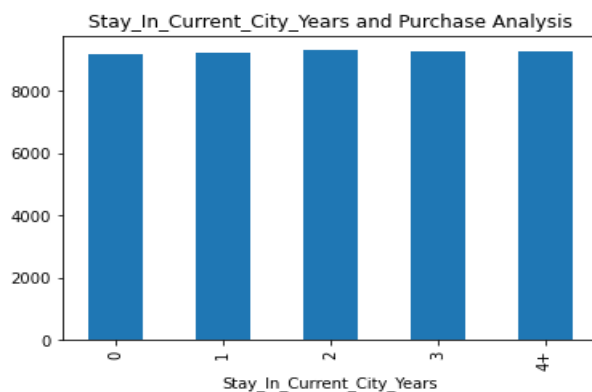
Stay_In_Current_City_Years

```
: sns.countplot(data['Stay_In_Current_City_Years'])  
plt.show()
```



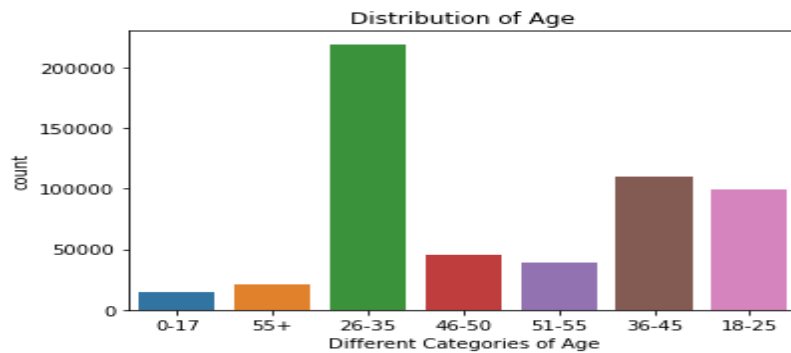
Groupby is used for Stay_In_Current_City_Years data so that it can separate the data into groups and mean() is used to get the average of the purchase

```
data.groupby("Stay_In_Current_City_Years").mean()["Purchase"].plot(kind='bar')  
plt.title("Stay_In_Current_City_Years and Purchase Analysis")  
plt.show()
```



Countplot is used to display the counts of the observations in each category i.e for diff age groups
Age

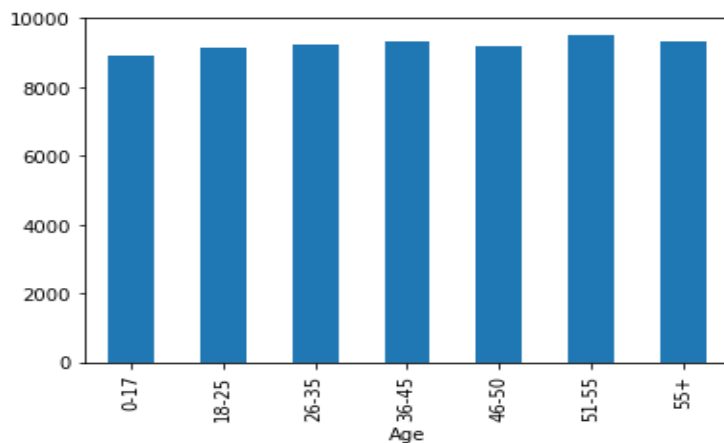
```
sns.countplot(data['Age'])  
plt.title('Distribution of Age')  
plt.xlabel('Different Categories of Age')  
plt.show()
```



Groupby is used for Age data so that it can separate the data into groups and mean() is used to get the average of the purchase

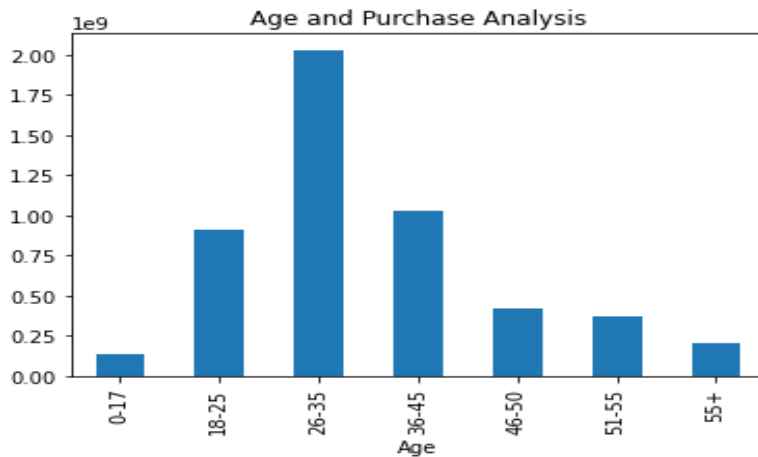
```
data.groupby("Age").mean()["Purchase"].plot(kind='bar')
```

<AxesSubplot:xlabel='Age'>



Groupby is used for Age data so that it can separate the data into groups and sum() is used to get the sum of the purchase of the items

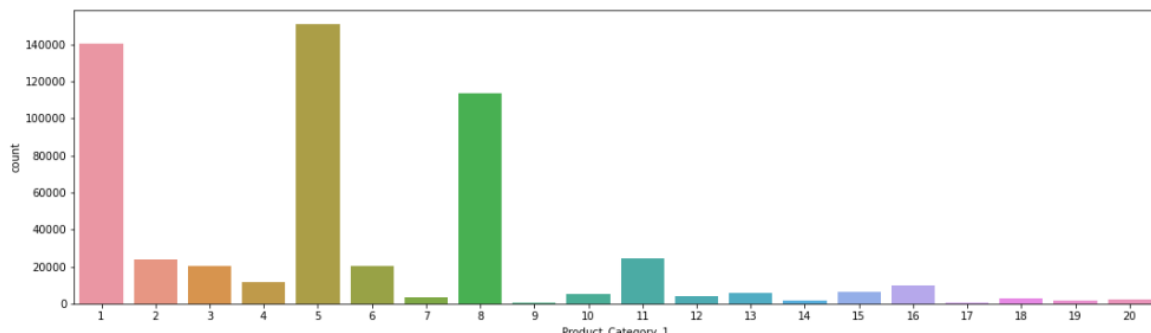
```
data.groupby("Age").sum()['Purchase'].plot(kind="bar")
plt.title("Age and Purchase Analysis")
plt.show()
```



Countplot is used to display the counts of the observations in each category i.e for product_category_1

Product_Category_1

```
plt.figure(figsize=(18,5))
sns.countplot(data['Product_Category_1'])
plt.show()
```



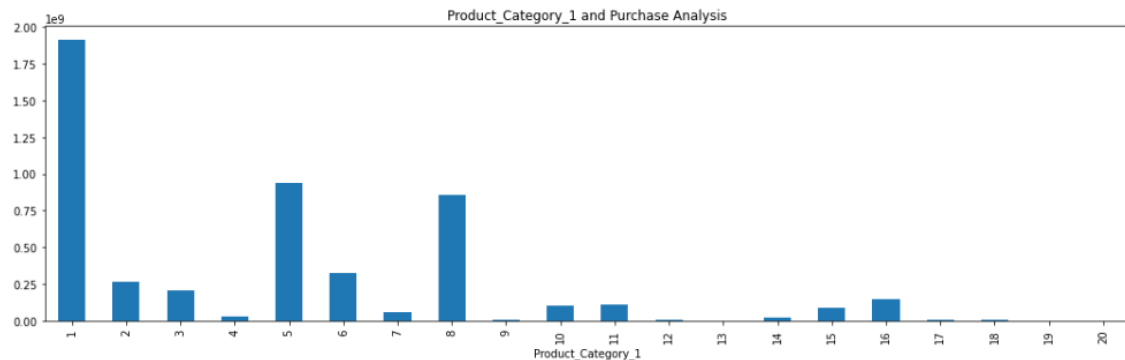
Groupby is used for product_category_1 data so that it can separate the data into groups and mean() is used to get the average of the purchase

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



Groupby is used for product_category_1 data so that it can separate the data into groups and mean() is used to get the sum of the purchase of the item

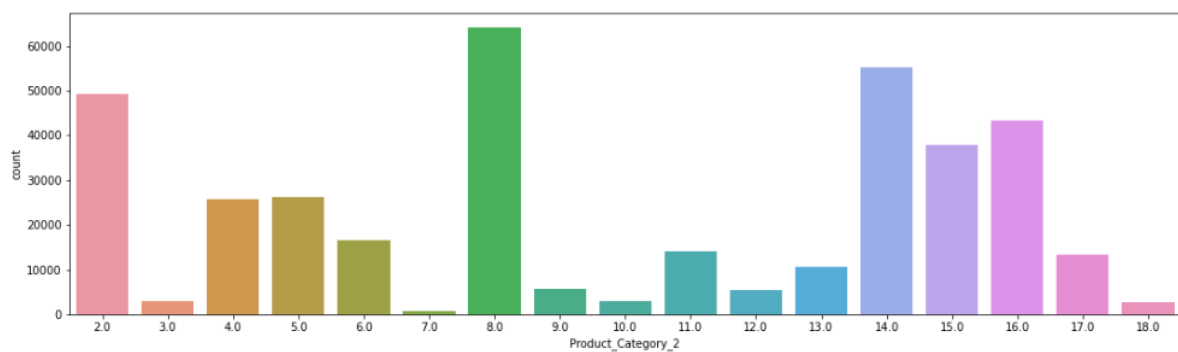
```
data.groupby('Product_Category_1').sum()['Purchase'].plot(kind='bar',figsize=(18,5))
plt.title("Product_Category_1 and Purchase Analysis")
plt.show()
```



Countplot is used to display the counts of the observations in each category i.e for product_category_2

Product_Category_2

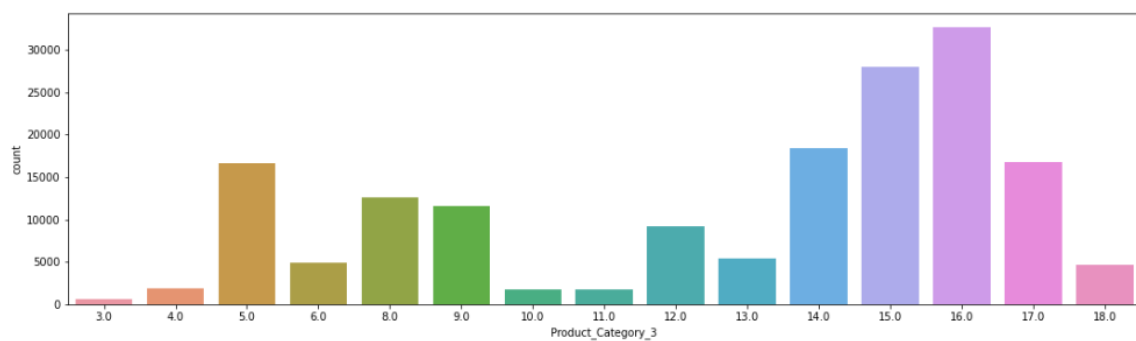
```
plt.figure(figsize=(18,5))
sns.countplot(data['Product_Category_2'])
plt.show()
```



Countplot is used to display the counts of the observations in each category i.e for product_category_3

Product_Category_3

```
1]: plt.figure(figsize=(18,5))
sns.countplot(data['Product_Category_3'])
plt.show()
```



corr() is used to find the correlation of each columns in the DataFrame. Nan values were automatically ignored.

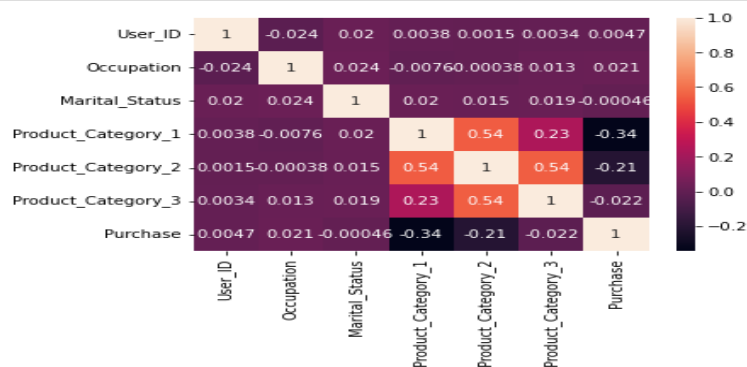
```
data.corr()
```

	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

HeatMap is used to display the correlation between the different variables in a matrix from ranging from -1 to +1. It displays the various shades of colour for each value.

HeatMap

```
: sns.heatmap(data.corr(),annot=True)
plt.show()
```



Displaying the labels of each column below

```
data.columns
```

```
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')
```

copy() is used to create a copies of the list

```
|: df = data.copy()|
```

Head() displays the first 5 rows of the dataset

```
: df.head()
:
```

	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	Stay_In_Current_City_Years_0	Stay_In_Current_City_Years_1	Stay_In_Current_City_Years_2	Stay_In_Current_City_Years_3	Stay_In_Current_City_Years_4+
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	NaN	8370	0	0	0	0	0
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	14.0	15200	0	0	0	0	0
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	NaN	1422	0	0	0	0	0
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	NaN	1057	0	0	0	0	0
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	NaN	7969	0	0	0	0	0

Get_dummies is used for manipulating the data

```
#Dummy Variables:
df = pd.get_dummies(df, columns=['Stay_In_Current_City_Years'])
```

Encoding the variables i.e converting set of labels into numeric form and displaying them using head() so that top five data is displayed

Encoding the categorical variables

```
3]: from sklearn.preprocessing import LabelEncoder
lr = LabelEncoder()

3]: df['Gender'] = lr.fit_transform(df['Gender'])

1]: df['Age'] = lr.fit_transform(df['Age'])

2]: df['City_Category'] = lr.fit_transform(df['City_Category'])

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

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase	Stay
0	1000001	P00069042	0	0	10	0	0	3	NaN	NaN	8370	
1	1000001	P00248942	0	0	10	0	0	1	6.0	14.0	15200	
2	1000001	P00087842	0	0	10	0	0	12	NaN	NaN	1422	
3	1000001	P00085442	0	0	10	0	0	12	14.0	NaN	1057	
4	1000002	P00285442	1	6	16	2	0	8	NaN	NaN	7969	

fillna() is used to replace the values which are NULL with a specified value and isnull() is used to check whether any null values are present or not

```
df['Product_Category_2'] =df['Product_Category_2'].fillna(0).astype('int64')
df['Product_Category_3'] =df['Product_Category_3'].fillna(0).astype('int64')
```

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

```
User_ID      0
Product_ID   0
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
Stay_In_Current_City_Years_0 0
Stay_In_Current_City_Years_1 0
Stay_In_Current_City_Years_2 0
Stay_In_Current_City_Years_3 0
Stay_In_Current_City_Years_4+ 0
dtype: int64
```

info() is used to print the information of a DataFrame

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 16 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  int32
3   Age                                          550068 non-null  int32
4   Occupation                                   550068 non-null  int64
5   City_Category                               550068 non-null  int32
6   Marital_Status                             550068 non-null  int64
7   Product_Category_1                          550068 non-null  int64
8   Product_Category_2                          550068 non-null  int64
9   Product_Category_3                          550068 non-null  int64
10  Purchase                                     550068 non-null  int64
11  Stay_In_Current_City_Years_0               550068 non-null  uint8
12  Stay_In_Current_City_Years_1               550068 non-null  uint8
13  Stay_In_Current_City_Years_2               550068 non-null  uint8
14  Stay_In_Current_City_Years_3               550068 non-null  uint8
15  Stay_In_Current_City_Years_4+              550068 non-null  uint8
dtypes: int32(3), int64(7), object(1), uint8(5)
memory usage: 42.5+ MB
```

Dropping the irrelevant columns

```
df = df.drop(["User_ID", "Product_ID"], axis=1)
```

Splitting data into dependent and independent variables i.e train and test

```
X = df.drop("Purchase", axis=1)
```

```
y=df['Purchase']
```

```
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=123)
```

A random forest regressor that fits a set of classification decision trees to different samples of data set and used to improve the accuracy of the predictions.

Creating and regressor object called RFRegressor.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# create a regressor object
RFRegressor = RandomForestRegressor(random_state = 0)
```

Fit() is used to take the training data as set of parameters and perform set of calculations on the input data

```
RFRegressor.fit(X_train, y_train)
```

```
RandomForestRegressor(random_state=0)
```


Declaring a variable as rf_y_pred and using predict() is to predict the values of the data.

MAE(mean_absolute_error) is used to calculate the summation of the absolute difference between the predicted value and the true value

```
: rf_y_pred = RFRegressor.predict(X_test)

: mean_absolute_error(y_test, rf_y_pred)

: 2222.049109204734
```

MSE(mean_squared_error) is used to calculate the average of the square difference between the predicted value and the true value

```
mean_squared_error(y_test, rf_y_pred)

9310769.87311957
```

R2_score in regression means the proportion of variance of true variable which is explained by estimated variable

```
r2_score(y_test, rf_y_pred)

0.6309821516972987
```

RMSE means square rooting the value of MSE

```
from math import sqrt
print("RMSE of Random Forest Model is ",sqrt(mean_squared_error(y_test, rf_y_pred)))

RMSE of Random Forest Model is 3051.35541573242
```

Installing xgboost in order to perform xgboost regressor

```
!j: !pip install xgboost

Collecting xgboost
  Downloading xgboost-1.7.1-py3-none-win_amd64.whl (89.1 MB)
Requirement already satisfied: numpy in c:\users\makut\anaconda3\lib\site-packages (from xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\users\makut\anaconda3\lib\site-packages (from xgboost) (1.7.3)
Installing collected packages: xgboost
Successfully installed xgboost-1.7.1
```

XGBOOST means eXtreme Gradient Boosting which layout parallel tree boosting.

Declaring a variable and initialising the variables with some values and Fit() is used to take the training data as set of parameters and perform set of calculations on the input data

```
] : from xgboost.sklearn import XGBRegressor

]: xgb_reg = XGBRegressor(learning_rate=1.0, max_depth=6, min_child_weight=40, seed=0)
xgb_reg.fit(X_train, y_train)

]: XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
               colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
               early_stopping_rounds=None, enable_categorical=False,
               eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
               grow_policy='depthwise', importance_type=None,
               interaction_constraints='', learning_rate=1.0, max_bin=256,
               max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
               max_depth=6, max_leaves=0, min_child_weight=40, missing=nan,
               monotone_constraints='()', n_estimators=100, n_jobs=0,
               num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

Declaring a variable as xgb_y_pred and using predict() is to predict the values of the data.

MAE(mean_absolute_error) is used to calculate the summation of the absolute difference between the predicted value and the true value

```
xgb_y_pred = xgb_reg.predict(X_test)
```

```
mean_absolute_error(y_test, xgb_y_pred)
```

2144.8588298827412

MSE(mean_squared_error) is used to calculate the average of the square difference between the predicted value and the true value

```
mean_squared_error(y_test, xgb_y_pred)
```

8268802.184348016

R2_score in regression means the proportion of variance of true variable which is explained by estimated variable

```
r2_score(y_test, xgb_y_pred)
```

0.67227891659979

RMSE means square rooting the value of MSE

```
: from math import sqrt
print("RMSE of XGBoost Model is ",sqrt(mean_squared_error(y_test, xgb_y_pred)))
```

RMSE of XGBoost Model is 2875.5525007114747