# SHOPPING BEHAVIOR TRENDS

Name: Kota Ruchitha
Person Id: 50560500

Name: Thupakula Pallavi
Person Id: 50560598

***Abstract*—**

In the project "Shopping Behavior Trends" holds immense potential for addressing critical issues in retail and e-commerce shopping. By calculating the data analytics, businesses can increase the customer/consumer experiences, streamline decision-making processes, and catalyze innovation and growth in the retail and e-commerce industries.

## *Introduction*—

In this project we made analysis on retail and e-commerce websites, the focus mainly revolves around improving the marketing strategies both in offline and online, balancing the inventory, understanding customer preference, and finally improving the overall effiency.

In summary, the study of shopping behaviour trends address the strategy for critical issues to resolve and increasing the market.

## I. PHASE 1

### PROBLEM STATEMENT:

Our project is titled as **"SHOPPING BEHAVIOR TRENDS".**
Discussion of retail and e-commerce shopping analytics, the main point often revolves around increasing of marketing strategies, optimizing the inventory, knowing costumer habits of purchasing, and increasing overall operational efficiency.

Because of huge transactional data both in ecommerce and digital payments and the quick expansion of this shopping trends, companies have increased directed their attention on their retail and e-commerce shopping.

By examining shopping data analysis, companies can achieve the Enhance Customer Experience, Optimize Operations, Foster Growth and Development, Enhance Competitive Edge, Business Innovation and Growth.

### DATA SOURCES:

The dataset for shopping behaviour trends project, we explored data from many sources. The following are some sources:

- Datasets in the **Kaggle platform** are used for e-commerce and retail.
- E-commerce Sites: A lot of e-commerce sites use APIs to provide users access to their product catalogue and sales information. If I wanted to gather data from websites like eBay, Walmart and others, I would think about using APIs.

**Dataset source link:** **https://www.kaggle.com/datasets/jacksondivakarr/online-shopping-dataset/data**

There are 21 columns and 52954 rows in the selected data set.
It contains the columns
{Customer ID, Gender, Location, Tenure_Months, Transaction_ID, Transaction_Date, Product_SKU, Product_Description, Product_Category, Quantity, Avg_Price, Delivery_Charges, Coupon_Status, GST, Date, Offline_Spend, online_Spend, Month, Coupon_Code, Discount_pct}.

### DATA CLEANING:

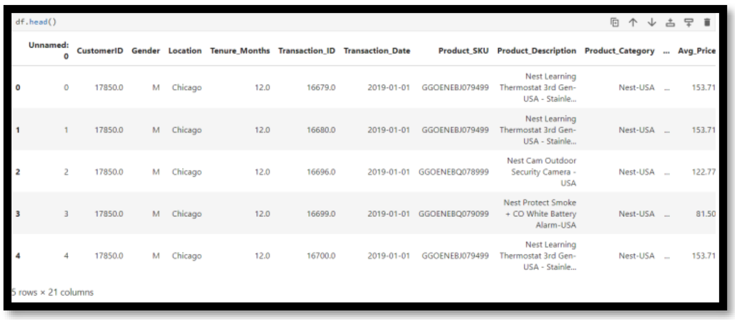
1. **df = pd.read_csv("online_sales.csv")**

Importing "online_sales.csv" as a CSV file into a panda Data Frame.

```
df = pd.read_csv("online_sales.csv")
```

2. **df.head()**

df.head() will show the Data Frame's initial few rows.This might help you rapidly examine the data and comprehend its structure.



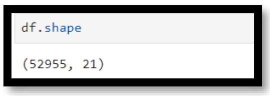
```
df.head()
```

| | Unnamed: 0 | CustomerID | Gender | Location | Tenure_Months | Transaction_ID | Transaction_Date | Product_SKU | Product_Description | Product_Category | ... | Avg_Price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 17850.0 | M | Chicago | 12.0 | 16679.0 | 2019-01-01 | GGOENEBJ079499 | Nest Learning Thermostat 3rd Gen-USA - Stainle... | Nest-USA | ... | 153.71 |
| 1 | 1 | 17850.0 | M | Chicago | 12.0 | 16680.0 | 2019-01-01 | GGOENEBJ079499 | Nest Learning Thermostat 3rd Gen-USA - Stainle... | Nest-USA | ... | 153.71 |
| 2 | 2 | 17850.0 | M | Chicago | 12.0 | 16696.0 | 2019-01-01 | GGOENEBQ078999 | Nest Cam Outdoor Security Camera - USA | Nest-USA | ... | 122.77 |
| 3 | 3 | 17850.0 | M | Chicago | 12.0 | 16699.0 | 2019-01-01 | GGOENEBQ079099 | Nest Protect Smoke + CO White Battery Alarm-USA | Nest-USA | ... | 81.50 |
| 4 | 4 | 17850.0 | M | Chicago | 12.0 | 16700.0 | 2019-01-01 | GGOENEBJ079499 | Nest Learning Thermostat 3rd Gen-USA - Stainle... | Nest-USA | ... | 153.71 |

5 rows × 21 columns

3. **df.shape**

You can determine the dimensions of your Data Frame using the df.shape property. As mentioned in the Data sources there are 52955 rows and 21 columns which is represented in the format (52955,21).

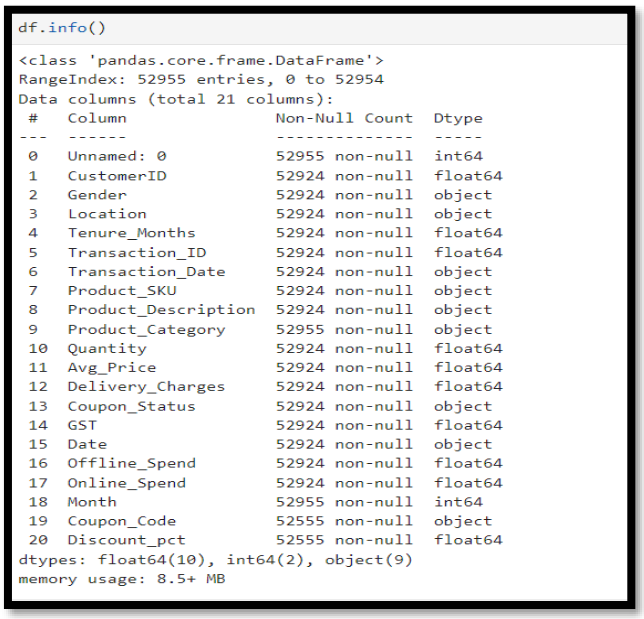


```
df.shape
```

```
(52955, 21)
```

## 4. df.info()

A brief overview of your Data Frame df is given by df.info(), which also includes information on the data type, memory utilization, and the number of non-null entries in each column. This might help you comprehend your Data Frame's general layout and contents.

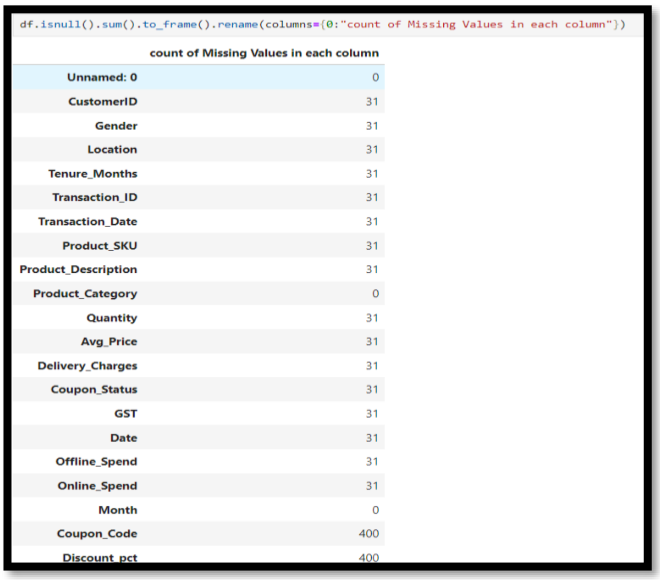


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52955 entries, 0 to 52954
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Unnamed: 0           52955 non-null  int64
 1   CustomerID           52924 non-null  float64
 2   Gender               52924 non-null  object
 3   Location             52924 non-null  object
 4   Tenure_Months        52924 non-null  float64
 5   Transaction_ID       52924 non-null  float64
 6   Transaction_Date     52924 non-null  object
 7   Product_SKU          52924 non-null  object
 8   Product_Description  52924 non-null  object
 9   Product_Category     52955 non-null  object
 10  Quantity             52924 non-null  float64
 11  Avg_Price            52924 non-null  float64
 12  Delivery_Charges     52924 non-null  float64
 13  Coupon_Status        52924 non-null  object
 14  GST                  52924 non-null  float64
 15  Date                 52924 non-null  object
 16  Offline_Spend        52924 non-null  float64
 17  Online_Spend         52924 non-null  float64
 18  Month                52955 non-null  int64
 19  Coupon_Code          52555 non-null  object
 20  Discount_pct         52555 non-null  float64
dtypes: float64(10), int64(2), object(9)
memory usage: 8.5+ MB
```

## 5. df.isnull().sum().to_frame().rename(columns={0 :"count of Missing Values in each column"})

The following line of code makes use of isnull().The process involves using sum() to count the number of missing values in each column of the Data Frame df, converting the result to a Data Frame using to_frame(), and then renaming the column to "count of Missing Values in each column".

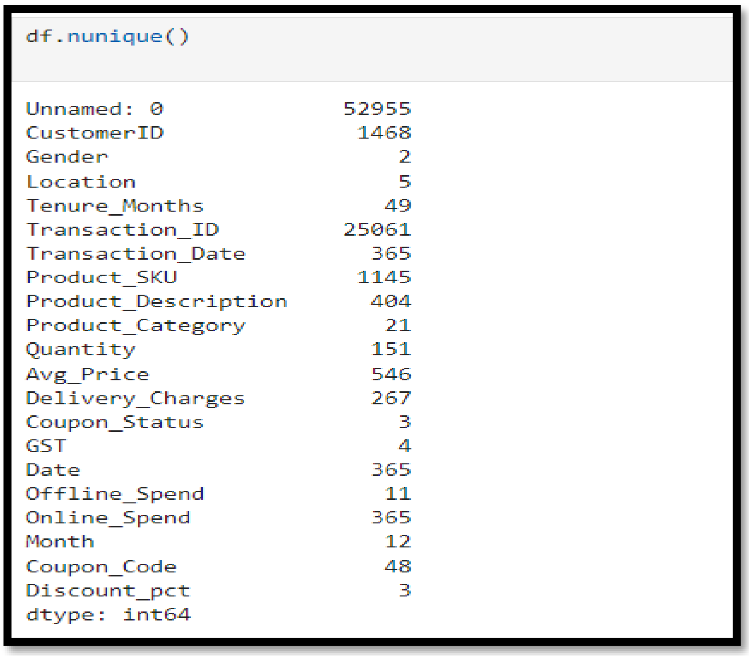


```
df.isnull().sum().to_frame().rename(columns={0:"count of Missing Values in each column"})
```

| | count of Missing Values in each column |
|---|---|
| Unnamed: 0 | 0 |
| CustomerID | 31 |
| Gender | 31 |
| Location | 31 |
| Tenure_Months | 31 |
| Transaction_ID | 31 |
| Transaction_Date | 31 |
| Product_SKU | 31 |
| Product_Description | 31 |
| Product_Category | 0 |
| Quantity | 31 |
| Avg_Price | 31 |
| Delivery_Charges | 31 |
| Coupon_Status | 31 |
| GST | 31 |
| Date | 31 |
| Offline_Spend | 31 |
| Online_Spend | 31 |
| Month | 0 |
| Coupon_Code | 400 |
| Discount_pct | 400 |

## 6. df.nunique()

The number of unique values in each column of the Data Frame df is returned by the df.nunique() function. Understanding the variety of values in each column may be aided by this.



```
df.nunique()
```

```
Unnamed: 0             52955
CustomerID              1468
Gender                     2
Location                   5
Tenure_Months             49
Transaction_ID         25061
Transaction_Date         365
Product_SKU             1145
Product_Description      404
Product_Category          21
Quantity                 151
Avg_Price                546
Delivery_Charges         267
Coupon_Status              3
GST                        4
Date                     365
Offline_Spend             11
Online_Spend             365
Month                     12
Coupon_Code               48
Discount_pct               3
dtype: int64
```

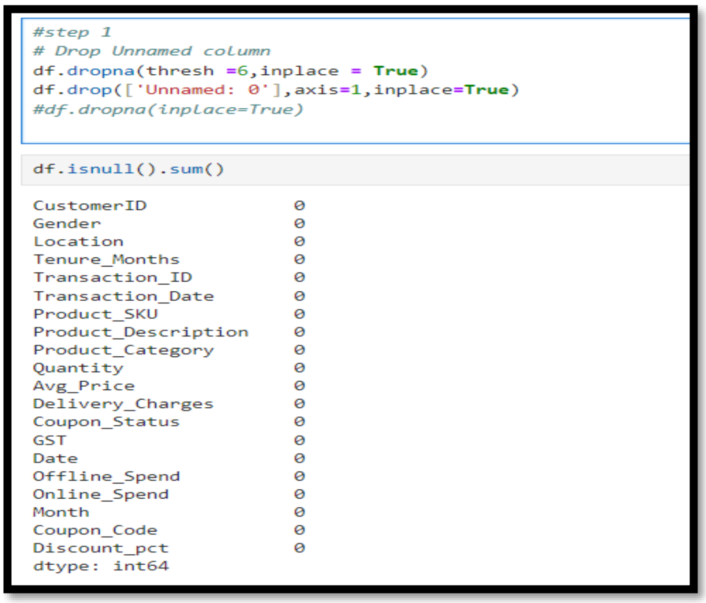
**<u>Step1:</u> Dropping Missing Values**

df.dropna(thresh =6,inplace = True)

df.drop(['Unnamed: 0'],axis=1,inplace=True)

#df.dropna(inplace=True)

To get the result of data analytics clearly,we do this dropping of missing value. By setting this thresh=6, we are saying that a row must have at least 6 non-missing values in a row that to be consider for training and testing.



```
#step 1
# Drop Unnamed column
df.dropna(thresh =6,inplace = True)
df.drop(['Unnamed: 0'],axis=1,inplace=True)
#df.dropna(inplace=True)
```

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

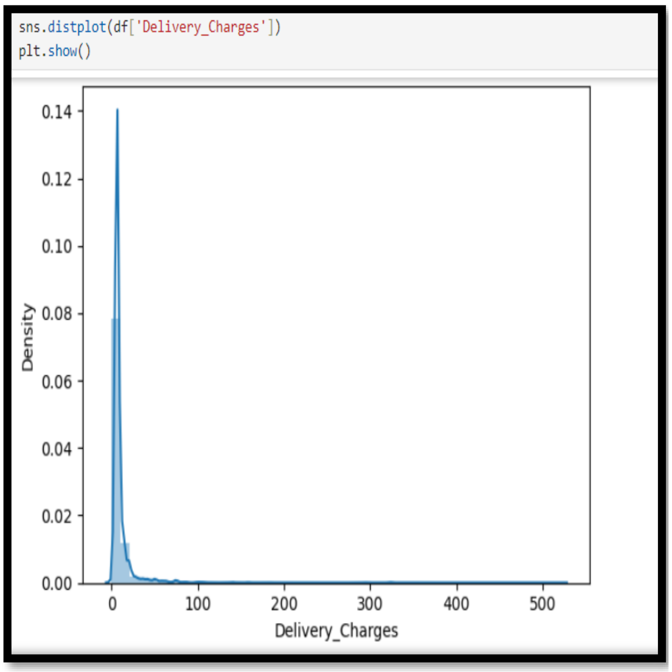
```
CustomerID             0
Gender                 0
Location               0
Tenure_Months          0
Transaction_ID         0
Transaction_Date       0
Product_SKU            0
Product_Description    0
Product_Category       0
Quantity               0
Avg_Price              0
Delivery_Charges       0
Coupon_Status          0
GST                    0
Date                   0
Offline_Spend          0
Online_Spend           0
Month                  0
Coupon_Code            0
Discount_pct           0
dtype: int64
```

## **Step2:** Dropping Unwanted columns:

drop_columns =
['Product_Description','Transaction_ID','Product_SKU']
df.drop(columns=drop_columns, inplace=True)

In the above we are dropping the unwanted columns like 'Product_Description','Transaction_ID','Product_SKU' because they don't get used in predicting total price.

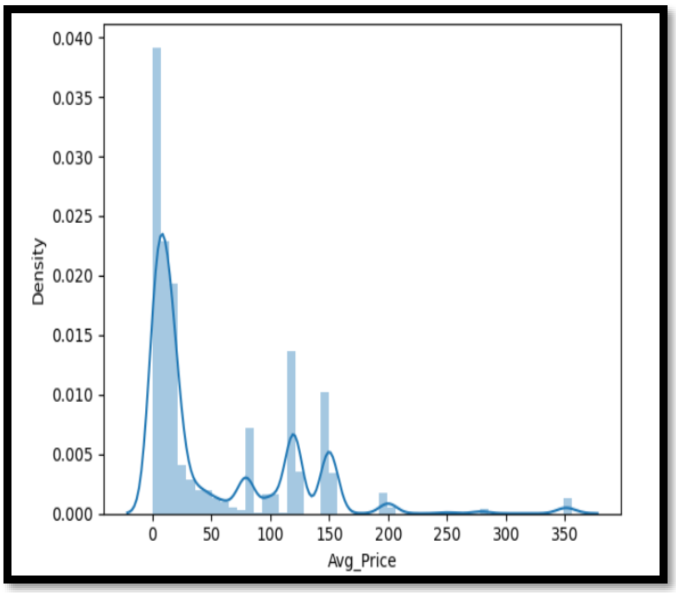
## **Step3:** Data Imputation



```
sns.distplot(df['Delivery_Charges'])
plt.show()
```

The above plot provides a few information about delivery charges in our retail and e-commerce shopping data analysis project. There may be more charges on the upper end and fewer on the lower end of the delivery charge distribution and that is the reason it is left-skewed. There might be a number of reasons for this skewness, including the package's weight, the delivery distance, or the particular delivery services that the consumers have selected.

```
sns.distplot(df['Avg_Price'])
plt.show()
```

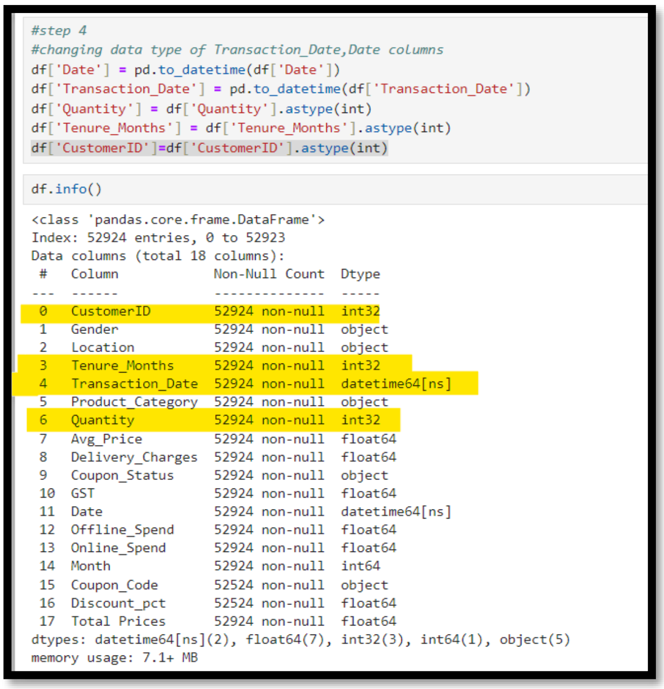


The above plot of Average price is also left skewed.

Replacing Avg price and delivery charges by mean since both are left skewed.
**Firstly, calculate mean value for Avg Price, Delivery charges(impute).**
mean= df['Avg_Price'].mean()
mean_d= df['Delivery_Charges'].mean()
**Replace null values with the mean.**
df['Avg_Price'] = df['Avg_Price'].fillna(mean)
df['Delivery_Charges'] =
df['Delivery_Charges'].fillna(mean_d)

## **Step4:** Changing data type of Transaction_Date, Date, Quantity, Tenure_Months columns.

df['Date'] = pd.to_datetime(df['Date'])
df['Transaction_Date'] =
pd.to_datetime(df['Transaction_Date'])
df['Quantity'] = df['Quantity'].astype(int)
df['Tenure_Months'] = df['Tenure_Months'].astype(int)
df['CustomerID']=df['CustomerID'].astype(int)



```
#step 4
#changing data type of Transaction_Date,Date columns
df['Date'] = pd.to_datetime(df['Date'])
df['Transaction_Date'] = pd.to_datetime(df['Transaction_Date'])
df['Quantity'] = df['Quantity'].astype(int)
df['Tenure_Months'] = df['Tenure_Months'].astype(int)
df['CustomerID']=df['CustomerID'].astype(int)
```

```
df.info()
```

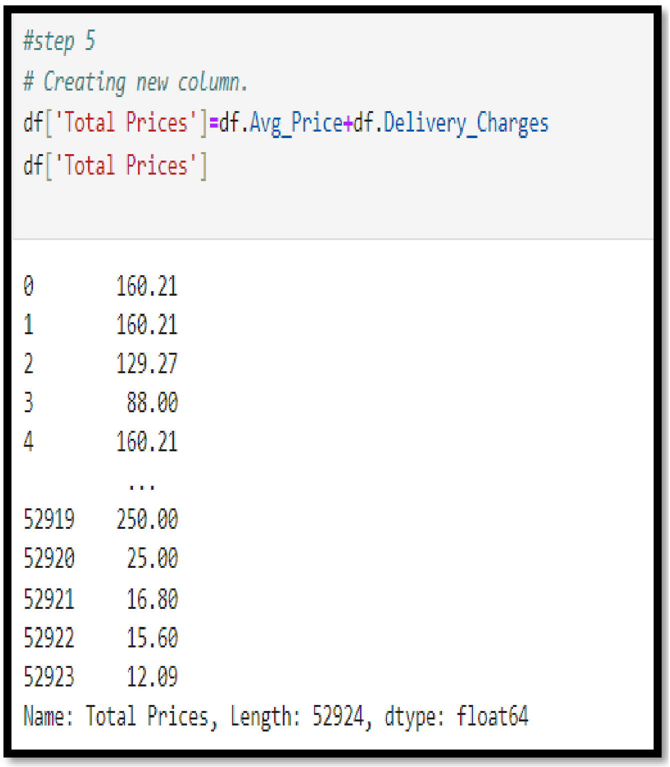
```
<class 'pandas.core.frame.DataFrame'>
Index: 52924 entries, 0 to 52923
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CustomerID        52924 non-null  int32
 1   Gender            52924 non-null  object
 2   Location          52924 non-null  object
 3   Tenure_Months     52924 non-null  int32
 4   Transaction_Date  52924 non-null  datetime64[ns]
 5   Product_Category  52924 non-null  object
 6   Quantity          52924 non-null  int32
 7   Avg_Price         52924 non-null  float64
 8   Delivery_Charges  52924 non-null  float64
 9   Coupon_Status     52924 non-null  object
 10  GST               52924 non-null  float64
 11  Date              52924 non-null  datetime64[ns]
 12  Offline_Spend     52924 non-null  float64
 13  Online_Spend      52924 non-null  float64
 14  Month             52924 non-null  int64
 15  Coupon_Code       52524 non-null  object
 16  Discount_pct      52524 non-null  float64
 17  Total Prices      52924 non-null  float64
dtypes: datetime64[ns](2), float64(7), int32(3), int64(1), object(5)
memory usage: 7.1+ MB
```

Here we are changing the data types of customerid,tenure_months,transaction _date,quantity from string to int and datetime data types to use them in prediction.

## **Step5:** Creating new column.

df['Total Prices']=df.Avg_Price+df.Delivery_Charges
df['Total Prices']

As we want to prediction shopping behaviour of customers in one packed pack.Total Price is created by adding avg_price and delivery_charges column.



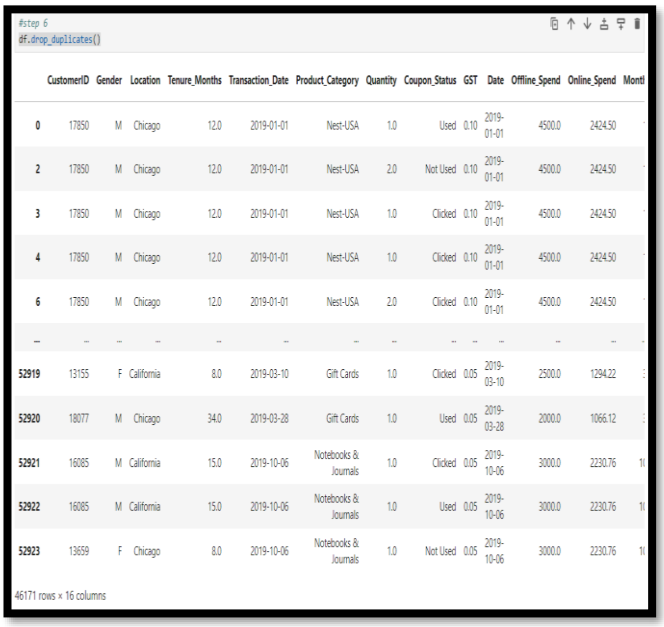
```
#step 5
# Creating new column.
df['Total Prices']=df.Avg_Price+df.Delivery_Charges
df['Total Prices']
```

```
0        160.21
1        160.21
2        129.27
3         88.00
4        160.21
          ...
52919    250.00
52920     25.00
52921     16.80
52922     15.60
52923     12.09
Name: Total Prices, Length: 52924, dtype: float64
```

## **Step6:** Finding duplicate values

df[df.duplicated()]
df.drop_duplicates()

To do the analysis accurately , because of duplicates can skew away the results which  lead to inaccurate conclusions, and affect the performance of ML models.



```
#step 6
df.drop_duplicates()
```

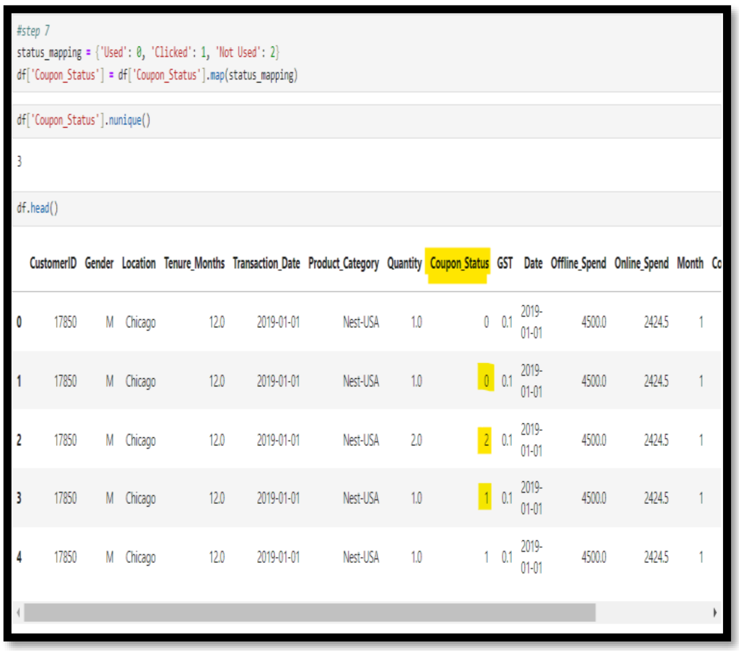
| | CustomerID | Gender | Location | Tenure_Months | Transaction_Date | Product_Category | Quantity | Coupon_Status | GST | Date | Offline_Spend | Online_Spend |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 1.0 | Used | 0.10 | 2019-01-01 | 4500.0 | 2424.50 |
| **2** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 2.0 | Not Used | 0.10 | 2019-01-01 | 4500.0 | 2424.50 |
| **3** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 1.0 | Clicked | 0.10 | 2019-01-01 | 4500.0 | 2424.50 |
| **4** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 1.0 | Clicked | 0.10 | 2019-01-01 | 4500.0 | 2424.50 |
| **6** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 2.0 | Clicked | 0.10 | 2019-01-01 | 4500.0 | 2424.50 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **52919** | 13155 | F | California | 8.0 | 2019-03-10 | Gift Cards | 1.0 | Clicked | 0.05 | 2019-03-10 | 2500.0 | 1294.22 |
| **52920** | 18077 | M | Chicago | 34.0 | 2019-03-28 | Gift Cards | 1.0 | Used | 0.05 | 2019-03-28 | 2000.0 | 1066.12 |
| **52921** | 16085 | M | California | 15.0 | 2019-10-06 | Notebooks & Journals | 1.0 | Clicked | 0.05 | 2019-10-06 | 3000.0 | 2230.76 |
| **52922** | 16085 | M | California | 15.0 | 2019-10-06 | Notebooks & Journals | 1.0 | Used | 0.05 | 2019-10-06 | 3000.0 | 2230.76 |
| **52923** | 13659 | F | Chicago | 8.0 | 2019-10-06 | Notebooks & Journals | 1.0 | Not Used | 0.05 | 2019-10-06 | 3000.0 | 2230.76 |

46171 rows × 16 columns

## **Step7:** Label Encoding

status_mapping = {'Used': 0, 'Clicked': 1, 'Not Used': 2}
df['Coupon_Status'] = df['Coupon_Status'].map(status_mapping)

ML models will work on numerical data and donot handle the categorical data so that's why we encode the coupon _status as 0 for used ,1 for clicked and 3 for not used.



```
#step 7
status_mapping = {'Used': 0, 'Clicked': 1, 'Not Used': 2}
df['Coupon_Status'] = df['Coupon_Status'].map(status_mapping)
```

```
df['Coupon_Status'].nunique()
```

```
3
```

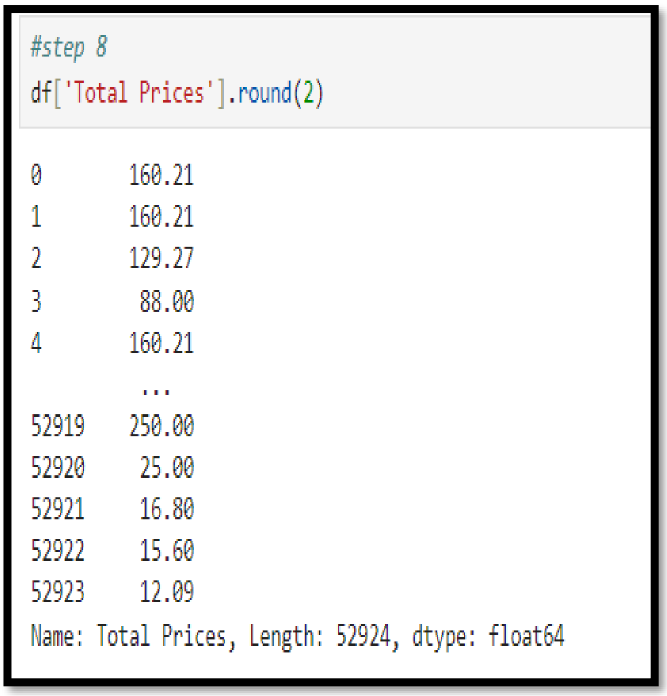
```
df.head()
```

| | CustomerID | Gender | Location | Tenure_Months | Transaction_Date | Product_Category | Quantity | Coupon_Status | GST | Date | Offline_Spend | Online_Spend | Month |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 1.0 | 0 | 0.1 | 2019-01-01 | 4500.0 | 2424.5 | 1 |
| **1** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 1.0 | 0 | 0.1 | 2019-01-01 | 4500.0 | 2424.5 | 1 |
| **2** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 2.0 | 2 | 0.1 | 2019-01-01 | 4500.0 | 2424.5 | 1 |
| **3** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 1.0 | 1 | 0.1 | 2019-01-01 | 4500.0 | 2424.5 | 1 |
| **4** | 17850 | M | Chicago | 12.0 | 2019-01-01 | Nest-USA | 1.0 | 1 | 0.1 | 2019-01-01 | 4500.0 | 2424.5 | 1 |

**<u>Step8:</u> Data Rounding Off**

df['Total Prices'].round(2)

Total prices are rounded off to 2 decimals because it will be easy for us to train and do the prediction in future for data.



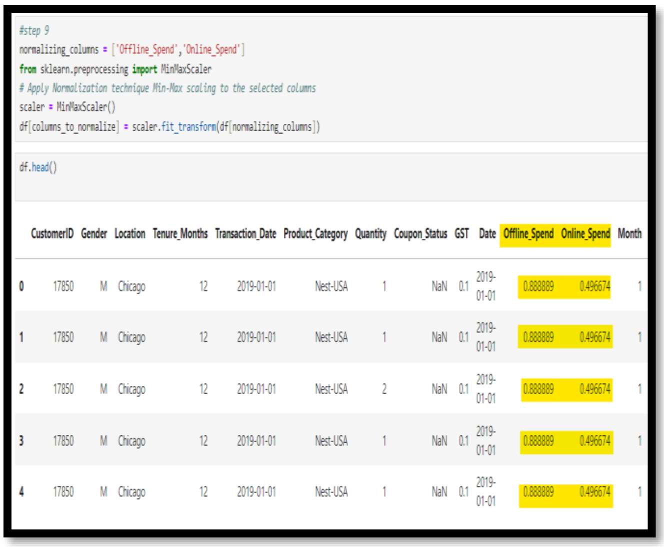
```
#step 8
df['Total Prices'].round(2)
```

```
0        160.21
1        160.21
2        129.27
3         88.00
4        160.21
          ...  
52919    250.00
52920     25.00
52921     16.80
52922     15.60
52923     12.09
Name: Total Prices, Length: 52924, dtype: float64
```

**<u>Step9:</u>**
**Data Normalization**

normalizing_columns = ['Offline_Spend','Online_Spend']
from sklearn.preprocessing import MinMaxScaler
# Apply Normalization technique Min-Max scaling to the selected columns
scaler = MinMaxScaler()
df[columns_to_normalize] =
scaler.fit_transform(df[normalizing_columns])

We have applied normalization to offline_spend and online_spend because to scale the data for these columns to a uniform range.



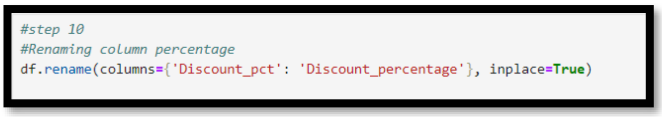
```
#step 9
normalizing_columns = ['Offline_Spend','Online_Spend']
from sklearn.preprocessing import MinMaxScaler
# Apply Normalization technique Min-Max scaling to the selected columns
scaler = MinMaxScaler()
df[columns_to_normalize] = scaler.fit_transform(df[normalizing_columns])
```

```
df.head()
```

| | CustomerID | Gender | Location | Tenure_Months | Transaction_Date | Product_Category | Quantity | Coupon_Status | GST | Date | Offline_Spend | Online_Spend | Month |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 17850 | M | Chicago | 12 | 2019-01-01 | Nest-USA | 1 | NaN | 0.1 | 2019-01-01 | 0.888889 | 0.496674 | 1 |
| **1** | 17850 | M | Chicago | 12 | 2019-01-01 | Nest-USA | 1 | NaN | 0.1 | 2019-01-01 | 0.888889 | 0.496674 | 1 |
| **2** | 17850 | M | Chicago | 12 | 2019-01-01 | Nest-USA | 2 | NaN | 0.1 | 2019-01-01 | 0.888889 | 0.496674 | 1 |
| **3** | 17850 | M | Chicago | 12 | 2019-01-01 | Nest-USA | 1 | NaN | 0.1 | 2019-01-01 | 0.888889 | 0.496674 | 1 |
| **4** | 17850 | M | Chicago | 12 | 2019-01-01 | Nest-USA | 1 | NaN | 0.1 | 2019-01-01 | 0.888889 | 0.496674 | 1 |

**<u>Step10:</u> Renaming column percentage.**

df.rename(columns={'Discount_pct': 'Discount_percentage'}, inplace=True)

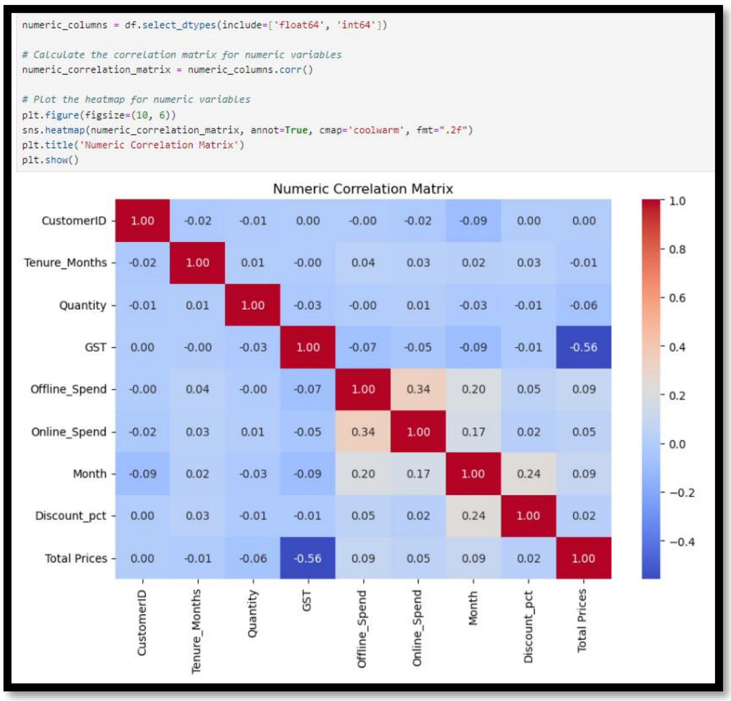


```
#step 10
#Renaming column percentage
df.rename(columns={'Discount_pct': 'Discount_percentage'}, inplace=True)
```

## <u>EXPLORATORY DATA ANALYSIS</u>

**<u>Step1:</u> Multivariate Analysis**

This involves in analysing the relationships between multiple variables in a dataset. The correlation matrix specifically helps in understanding the linear relationship between pairs of variables, which is valuable for various purposes such as feature selection, identifying redundant variables.
numeric_columns=df.select_dtypes(include=['float64', 'int64'])
# Calculate the correlation matrix for numeric variables
numeric_correlation_matrix = numeric_columns.corr()
# Plot the heatmap for numeric variables
plt.figure(figsize=(10, 6))
sns.heatmap(numeric_correlation_matrix,annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Numeric Correlation Matrix')
plt.show()

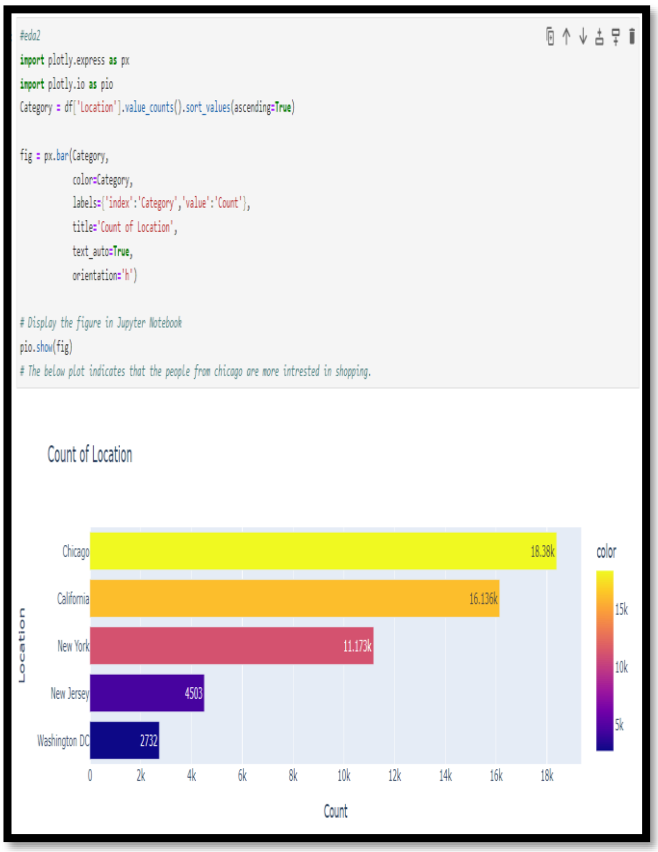


```
numeric_columns = df.select_dtypes(include=['float64', 'int64'])

# Calculate the correlation matrix for numeric variables
numeric_correlation_matrix = numeric_columns.corr()

# Plot the heatmap for numeric variables
plt.figure(figsize=(10, 6))
sns.heatmap(numeric_correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Numeric Correlation Matrix')
plt.show()
```

Numeric Correlation Matrix

| | CustomerID | Tenure_Months | Quantity | GST | Offline_Spend | Online_Spend | Month | Discount_pct | Total Prices |
|---|---|---|---|---|---|---|---|---|---|
| CustomerID | 1.00 | -0.02 | -0.01 | 0.00 | -0.00 | -0.02 | -0.09 | 0.00 | 0.00 |
| Tenure_Months | -0.02 | 1.00 | 0.01 | -0.00 | 0.04 | 0.03 | 0.02 | 0.03 | -0.01 |
| Quantity | -0.01 | 0.01 | 1.00 | -0.03 | -0.00 | 0.01 | -0.03 | -0.01 | -0.06 |
| GST | 0.00 | -0.00 | -0.03 | 1.00 | -0.07 | -0.05 | -0.09 | -0.01 | -0.56 |
| Offline_Spend | -0.00 | 0.04 | -0.00 | -0.07 | 1.00 | 0.34 | 0.20 | 0.05 | 0.09 |
| Online_Spend | -0.02 | 0.03 | 0.01 | -0.05 | 0.34 | 1.00 | 0.17 | 0.02 | 0.05 |
| Month | -0.09 | 0.02 | -0.03 | -0.09 | 0.20 | 0.17 | 1.00 | 0.24 | 0.09 |
| Discount_pct | 0.00 | 0.03 | -0.01 | -0.01 | 0.05 | 0.02 | 0.24 | 1.00 | 0.02 |
| Total Prices | 0.00 | -0.01 | -0.06 | -0.56 | 0.09 | 0.05 | 0.09 | 0.02 | 1.00 |

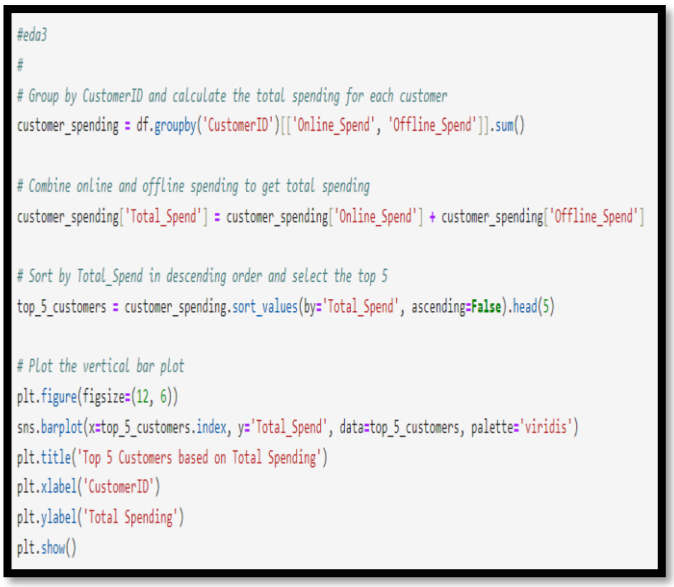
**<u>Step2:</u>**
Category= df['Location'].value_counts().sort_values(ascending=True)
fig = px.bar(Category, color=Category, labels={'index':'Category','value':'Count'},title='Count of Location',text_auto=True, orientation='h')
**Display the figure in Jupyter Notebook**
pio.show(fig)



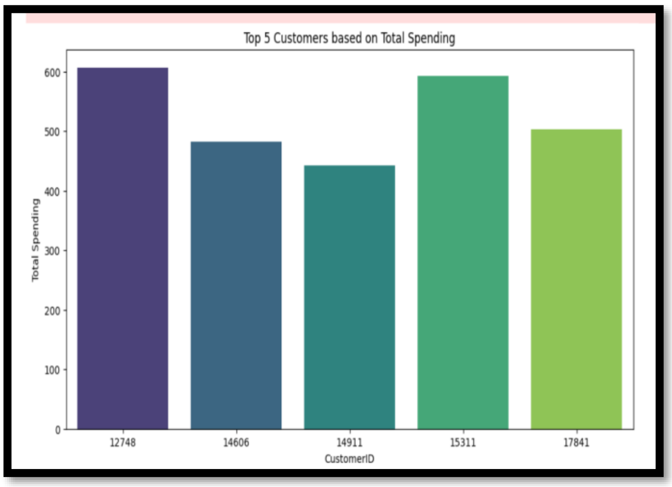
```
#eda2
import plotly.express as px
import plotly.io as pio
Category = df['Location'].value_counts().sort_values(ascending=True)

fig = px.bar(Category,
         color=Category,
         labels={'index':'Category','value':'Count'},
         title='Count of Location',
         text_auto=True,
         orientation='h')

# Display the figure in Jupyter Notebook
pio.show(fig)
# The below plot indicates that the people from chicago are more intrested in shopping.
```

**Observation:** The above plot indicates that the people from Chicago are more interested in shopping than California, New York, New jersey, Washington DC.

**<u>Step3:</u> Group by CustomerID and calculate the total spending for each customer**
customer_spending = df.groupby('CustomerID')[['Online_Spend', 'Offline_Spend']].sum()
**Combine online and offline spending to get total spending**
customer_spending['Total_Spend'] = customer_spending['Online_Spend'] + customer_spending['Offline_Spend']
**Sort by Total_Spend in descending order and select the top 5**
top_5_customers = customer_spending.sort_values(by='Total_Spend', ascending=False).head(5)
**Plot the vertical bar plot**
plt.figure(figsize=(12, 6))
sns.barplot(x=top_5_customers.index, y='Total_Spend', data=top_5_customers, palette='viridis')
plt.title('Top 5 Customers based on Total Spending')
plt.xlabel('CustomerID')
plt.ylabel('Total Spending')
plt.show()



```
#eda3
#
# Group by CustomerID and calculate the total spending for each customer
customer_spending = df.groupby('CustomerID')[['Online_Spend', 'Offline_Spend']].sum()

# Combine online and offline spending to get total spending
customer_spending['Total_Spend'] = customer_spending['Online_Spend'] + customer_spending['Offline_Spend']

# Sort by Total_Spend in descending order and select the top 5
top_5_customers = customer_spending.sort_values(by='Total_Spend', ascending=False).head(5)

# Plot the vertical bar plot
plt.figure(figsize=(12, 6))
sns.barplot(x=top_5_customers.index, y='Total_Spend', data=top_5_customers, palette='viridis')
plt.title('Top 5 Customers based on Total Spending')
plt.xlabel('CustomerID')
plt.ylabel('Total Spending')
plt.show()
```

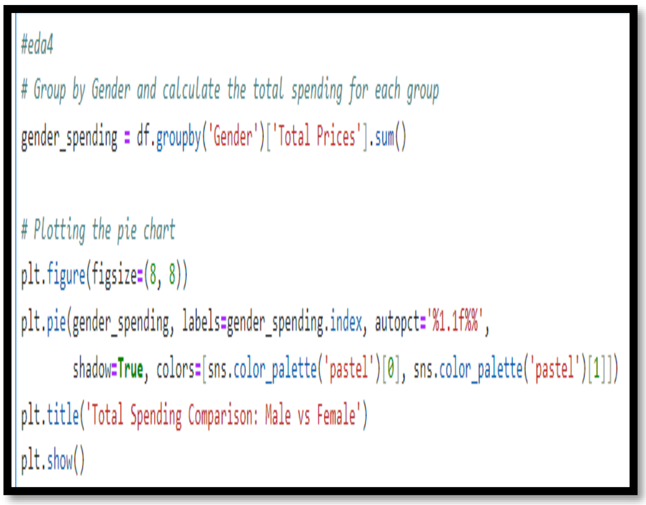


**<u>Observation:</u>** This plot denotes the customer id vs total spend in the shopping of various product_category.

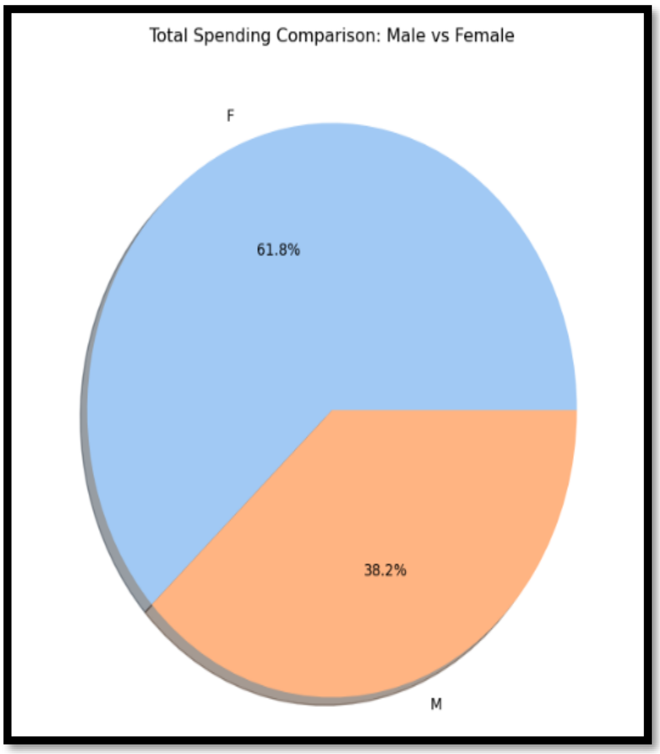
1. Based on their purchase habits, it might help identify different customer types. Take high, medium, and low spenders as an example.

Taking everything into account, the story may provide useful data on consumer behaviour and assist businesses in making choices that will boost customer satisfaction and loyalty.

**<u>Step4:</u> Group by Gender and calculate the total spending for each group**
gender_spending = df.groupby('Gender')['Total Prices'].sum()
**Plotting the pie chart**
plt.figure(figsize=(8, 8))
plt.pie(gender_spending, labels=gender_spending.index, autopct='%1.1f%%', shadow=True, colors=[sns.color_palette('pastel')[0], sns.color_palette('pastel')[1]])
plt.title('Total Spending Comparison: Male vs Female')
plt.show()

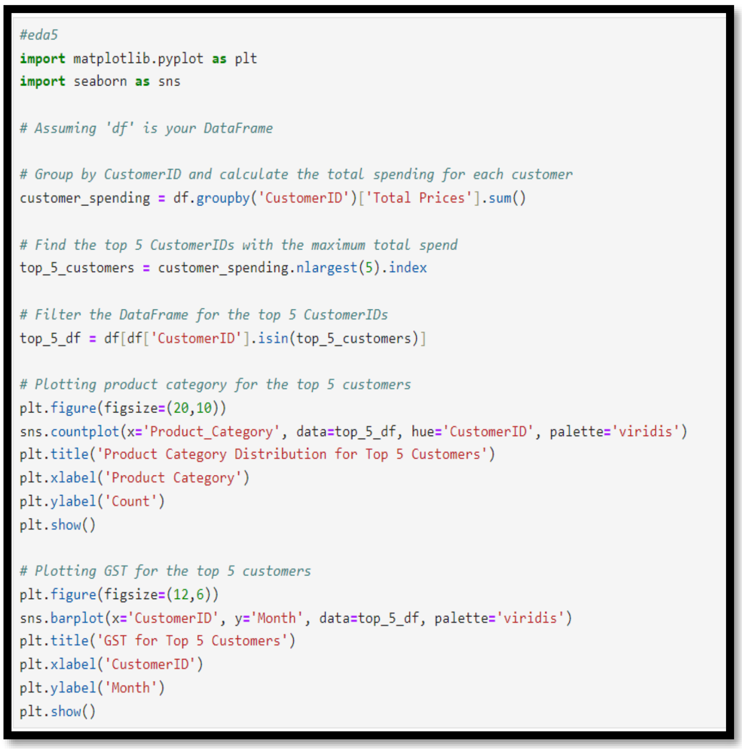


```python
#eda4
# Group by Gender and calculate the total spending for each group
gender_spending = df.groupby('Gender')['Total Prices'].sum()


# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(gender_spending, labels=gender_spending.index, autopct='%1.1f%%',
        shadow=True, colors=[sns.color_palette('pastel')[0], sns.color_palette('pastel')[1]])
plt.title('Total Spending Comparison: Male vs Female')
plt.show()
```

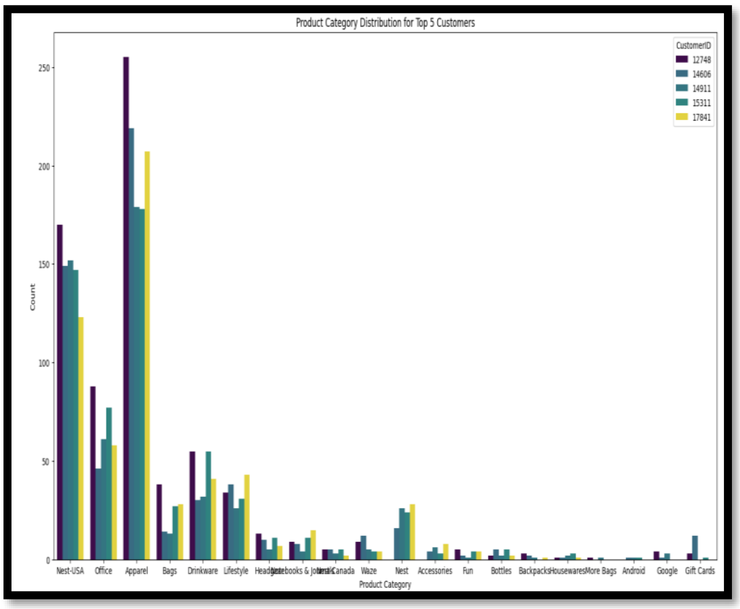


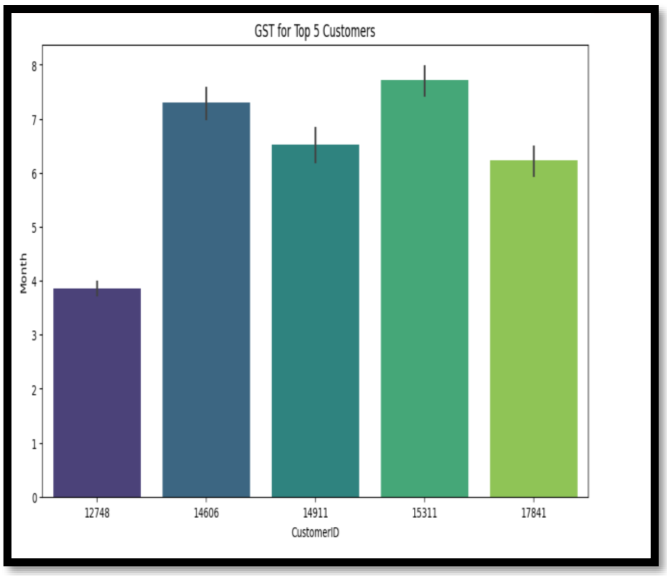
**Observation:** This plot denotes the gender vs Total prices. It emphasis which gender tends to spend more overall. This could be useful for marketing or product targeting purposes for companies.

**Step5:**
Assuming 'df' is your DataFrame
**Group by CustomerID and calculate the total spending for each customer**
customer_spending = df.groupby('CustomerID')['Total Prices'].sum()
**Find the top 5 CustomerIDs with the maximum total spend**
top_5_customers = customer_spending.nlargest(5).index
**Filter the DataFrame for the top 5 CustomerIDs**
top_5_df = df[df['CustomerID'].isin(top_5_customers)]
**Plotting product category for the top 5 customers**
plt.figure(figsize=(20,10))
sns.countplot(x='Product_Category', data=top_5_df, hue='CustomerID', palette='viridis')
plt.title('Product Category Distribution for Top 5 Customers')
plt.xlabel('Product Category')
plt.ylabel('Count')
plt.show()
**Plotting GST for the top 5 customers**
plt.figure(figsize=(12,6))
sns.barplot(x='CustomerID', y='Month', data=top_5_df, palette='viridis')
plt.title('GST for Top 5 Customers')
plt.xlabel('CustomerID')
plt.ylabel('Month')
plt.show()



```python
#eda5
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame

# Group by CustomerID and calculate the total spending for each customer
customer_spending = df.groupby('CustomerID')['Total Prices'].sum()

# Find the top 5 CustomerIDs with the maximum total spend
top_5_customers = customer_spending.nlargest(5).index

# Filter the DataFrame for the top 5 CustomerIDs
top_5_df = df[df['CustomerID'].isin(top_5_customers)]

# Plotting product category for the top 5 customers
plt.figure(figsize=(20,10))
sns.countplot(x='Product_Category', data=top_5_df, hue='CustomerID', palette='viridis')
plt.title('Product Category Distribution for Top 5 Customers')
plt.xlabel('Product Category')
plt.ylabel('Count')
plt.show()

# Plotting GST for the top 5 customers
plt.figure(figsize=(12,6))
sns.barplot(x='CustomerID', y='Month', data=top_5_df, palette='viridis')
plt.title('GST for Top 5 Customers')
plt.xlabel('CustomerID')
plt.ylabel('Month')
plt.show()
```



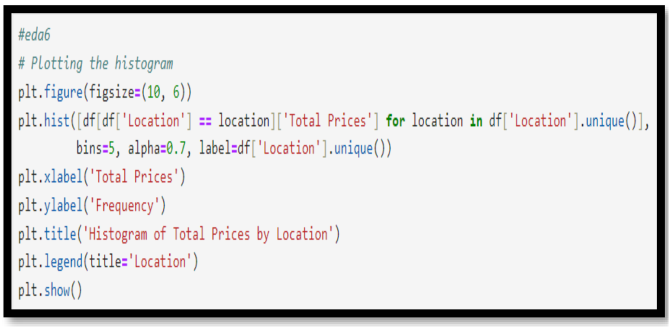


**Observation:**

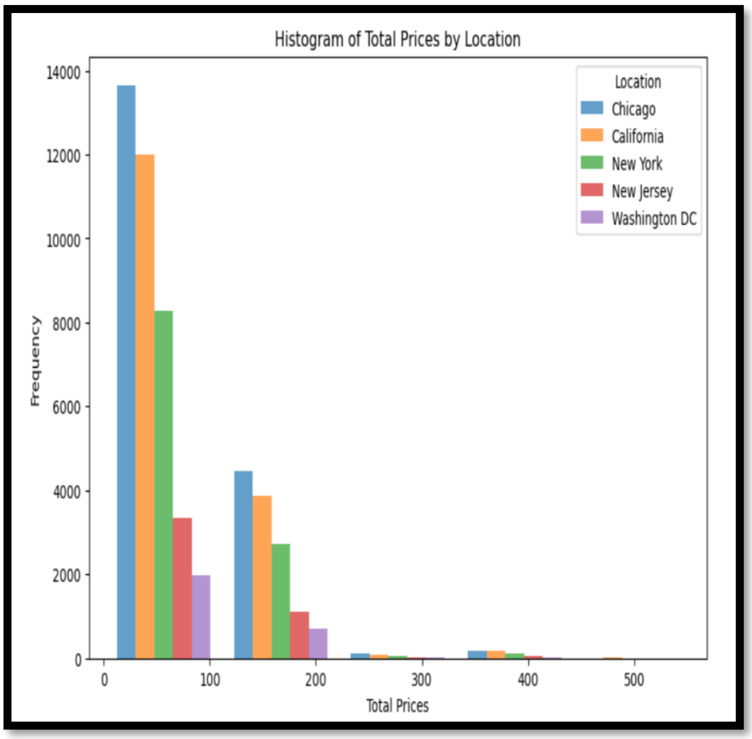
1. Plotting the amount of GST that each client contributes across several product categories can be useful for tax analysis and comprehending the total tax burden.
2. Determine the most popular product categories with consumers based on their purchasing patterns.

**Step6: Plotting the histogram**

plt.figure(figsize=(10, 6))
plt.hist([df[df['Location'] == location]['Total Prices'] for location in df['Location'].unique()],bins=5, alpha=0.7, label=df['Location'].unique())
plt.xlabel('Total Prices')
plt.ylabel('Frequency')
plt.title('Histogram of Total Prices by Location')
plt.legend(title='Location')
plt.show()



```
#eda6
# Plotting the histogram
plt.figure(figsize=(10, 6))
plt.hist([df[df['Location'] == location]['Total Prices'] for location in df['Location'].unique()],
        bins=5, alpha=0.7, label=df['Location'].unique())
plt.xlabel('Total Prices')
plt.ylabel('Frequency')
plt.title('Histogram of Total Prices by Location')
plt.legend(title='Location')
plt.show()
```

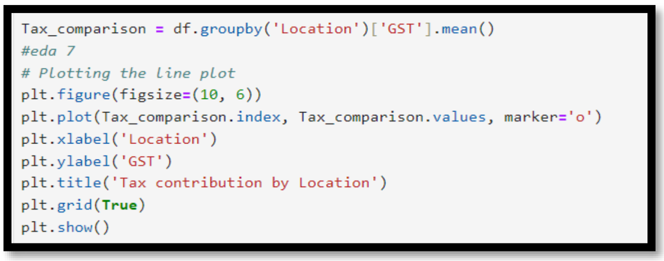


**Observation:**

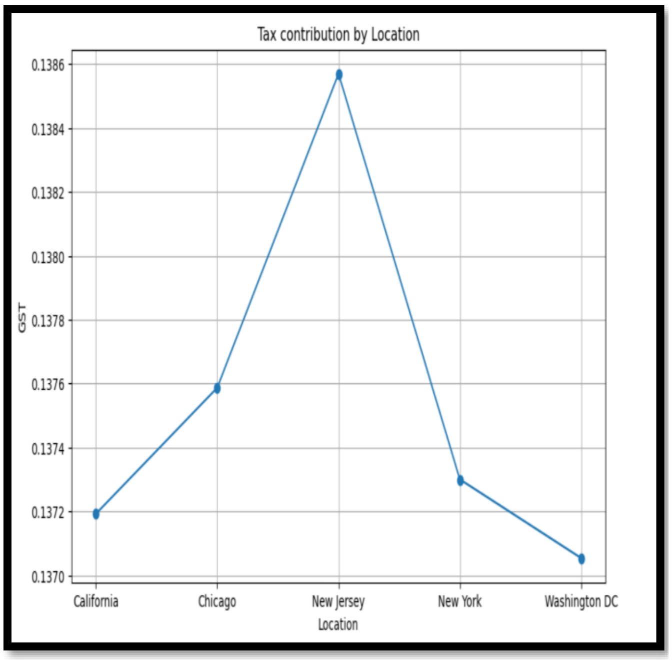
1. It displays the distribution of total prices among various places, which may be used to determine whether areas have higher or lower average costs.
2. The plot can emphasize locations with prices that are noticeably higher or lower than others in order to illustrate regional pricing disparities.

**Step7:**

Tax_comparison = df.groupby('Location')['GST'].mean()
# Plotting the line plot
plt.figure(figsize=(10, 6))
plt.plot(Tax_comparison.index, Tax_comparison.values, marker='o')
plt.xlabel('Location')
plt.ylabel('GST')
plt.title('Tax contribution by Location')
plt.grid(True)
plt.show()



```
Tax_comparison = df.groupby('Location')['GST'].mean()
#eda 7
# Plotting the line plot
plt.figure(figsize=(10, 6))
plt.plot(Tax_comparison.index, Tax_comparison.values, marker='o')
plt.xlabel('Location')
plt.ylabel('GST')
plt.title('Tax contribution by Location')
plt.grid(True)
plt.show()
```



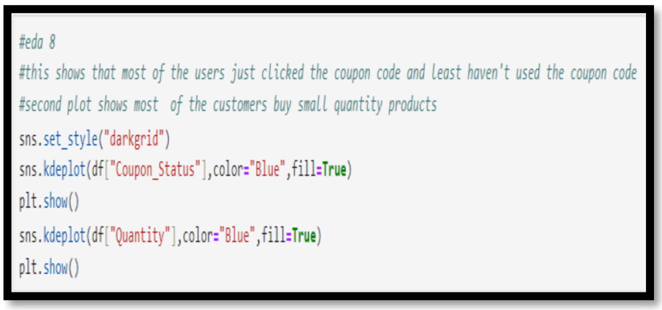
**Observation:**

1. You may notice that different places or areas have varied GST rates. Because of regional laws or other economic considerations, the GST rate may be greater or lower in some areas.
2. The layout may make it easier for you to comprehend how various places' businesses are impacted by GST rates. distinct locations may have distinct effects from higher GST rates on business operations and customer behaviour.

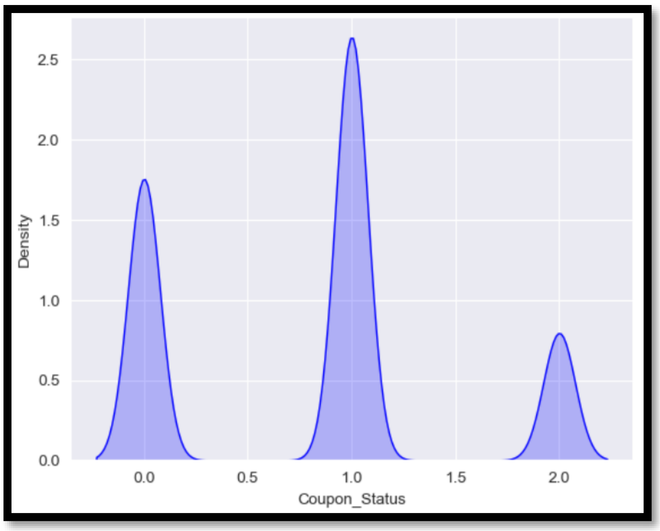
**Step8:**

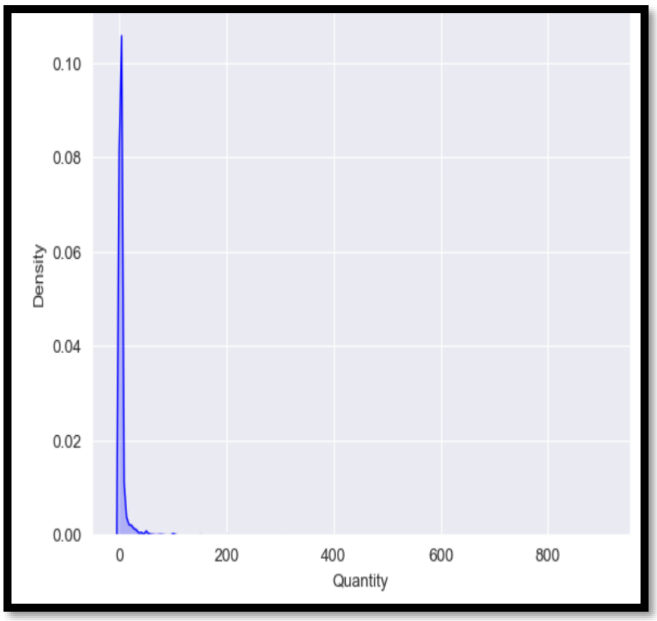
**This shows that most of the users just clicked the coupon code and least haven't used the coupon code.**
**second plot shows most of the customers buy small quantity products**

sns.set_style("darkgrid")
sns.kdeplot(df["Coupon_Status"],color="Blue",fill=True)
plt.show()
sns.kdeplot(df["Quantity"],color="Blue",fill=True)
plt.show()



```
#eda 8
#this shows that most of the users just clicked the coupon code and least haven't used the coupon code
#second plot shows most  of the customers buy small quantity products
sns.set_style("darkgrid")
sns.kdeplot(df["Coupon_Status"],color="Blue",fill=True)
plt.show()
sns.kdeplot(df["Quantity"],color="Blue",fill=True)
plt.show()
```



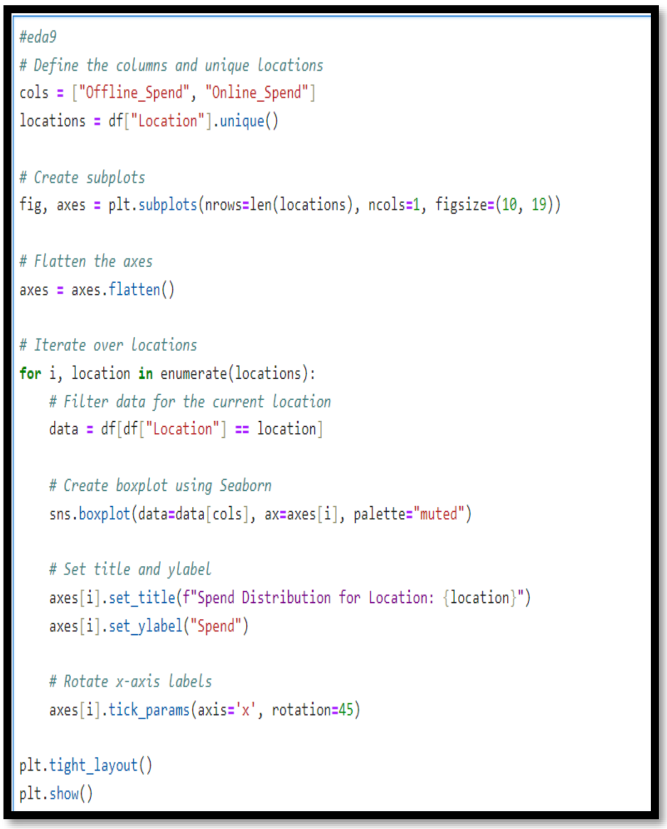


**Observation:**
This plot denotes the status of coupon which are used/ clicked/ not used by customers and the quantity that was frequently ordered.

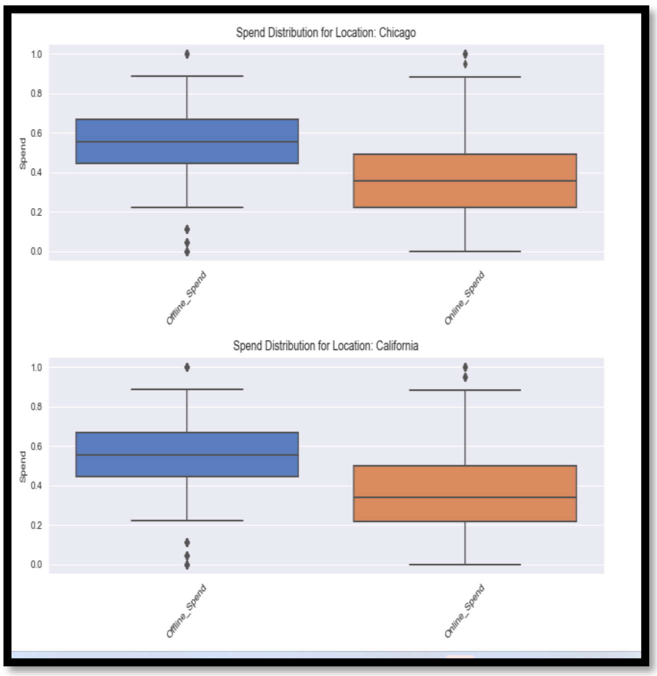
**Step9: Define the columns and unique locations**
cols = ["Offline_Spend", "Online_Spend"]
locations = df["Location"].unique()
**Create subplots**
fig, axes = plt.subplots(nrows=len(locations), ncols=1, figsize=(10, 19))
**Flatten the axes**
axes = axes.flatten()
**Iterate over locations**
for i, location in enumerate(locations):
 **Filter data for the current location**
 data = df[df["Location"] == location]
 **Create boxplot using Seaborn**
 sns.boxplot(data=data[cols], ax=axes[i], palette="muted")

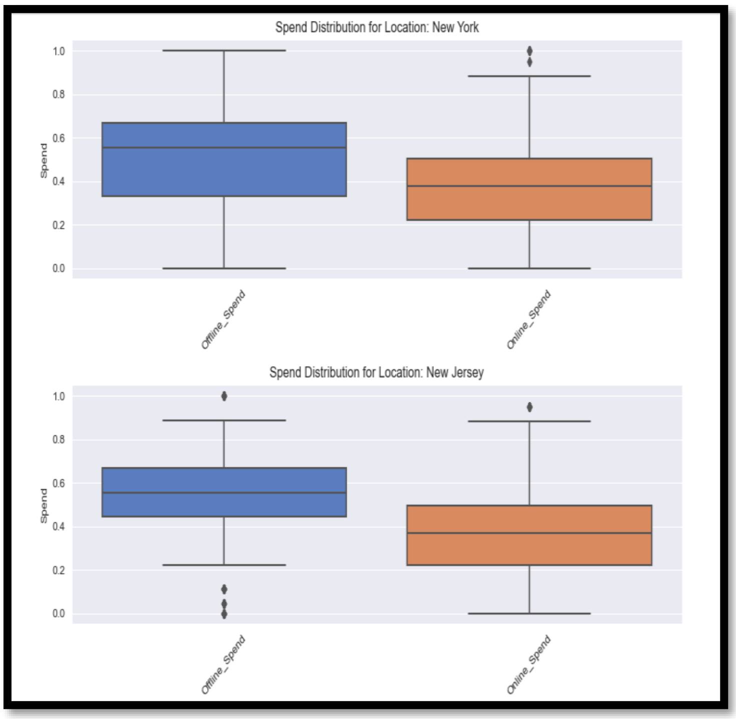
**Set title and ylabel**

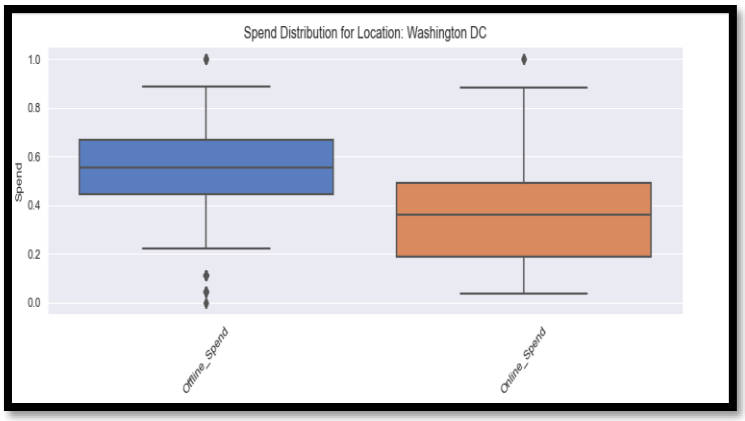
axes[i].set_title(f"Spend Distribution for Location: {location}")
axes[i].set_ylabel("Spend")
**Rotate x-axis labels**
axes[i].tick_params(axis='x', rotation=45)
plt.tight_layout()
plt.show()



```
#eda9
# Define the columns and unique locations
cols = ["Offline_Spend", "Online_Spend"]
locations = df["Location"].unique()

# Create subplots
fig, axes = plt.subplots(nrows=len(locations), ncols=1, figsize=(10, 19))

# Flatten the axes
axes = axes.flatten()

# Iterate over locations
for i, location in enumerate(locations):
    # Filter data for the current location
    data = df[df["Location"] == location]

    # Create boxplot using Seaborn
    sns.boxplot(data=data[cols], ax=axes[i], palette="muted")

    # Set title and ylabel
    axes[i].set_title(f"Spend Distribution for Location: {location}")
    axes[i].set_ylabel("Spend")

    # Rotate x-axis labels
    axes[i].tick_params(axis='x', rotation=45)

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

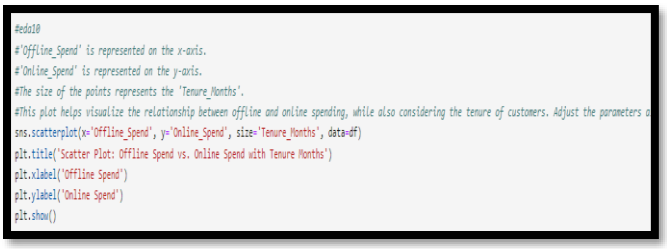




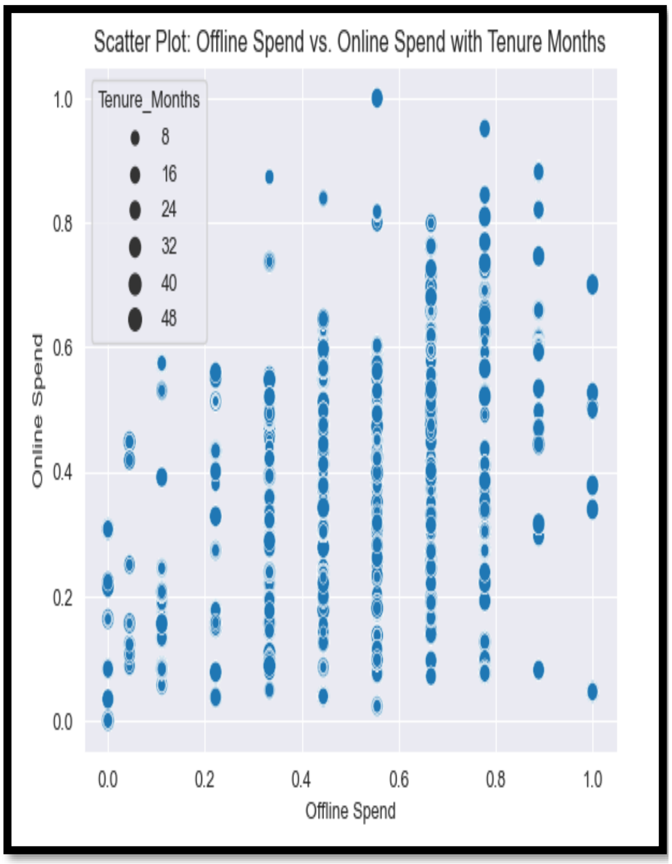


**Observation:**
This plot denotes area-wise expenditure in online and offline.

**Step10:**
**'Offline_Spend' is represented on the x-axis.**
**'Online_Spend' is represented on the y-axis.**
The size of the points represents the 'Tenure_Months'.
sns.scatterplot(x='Offline_Spend', y='Online_Spend', size='Tenure_Months', data=df)
plt.title('Scatter Plot: Offline Spend vs. Online Spend with Tenure Months')
plt.xlabel('Offline Spend')
plt.ylabel('Online Spend')
plt.show()

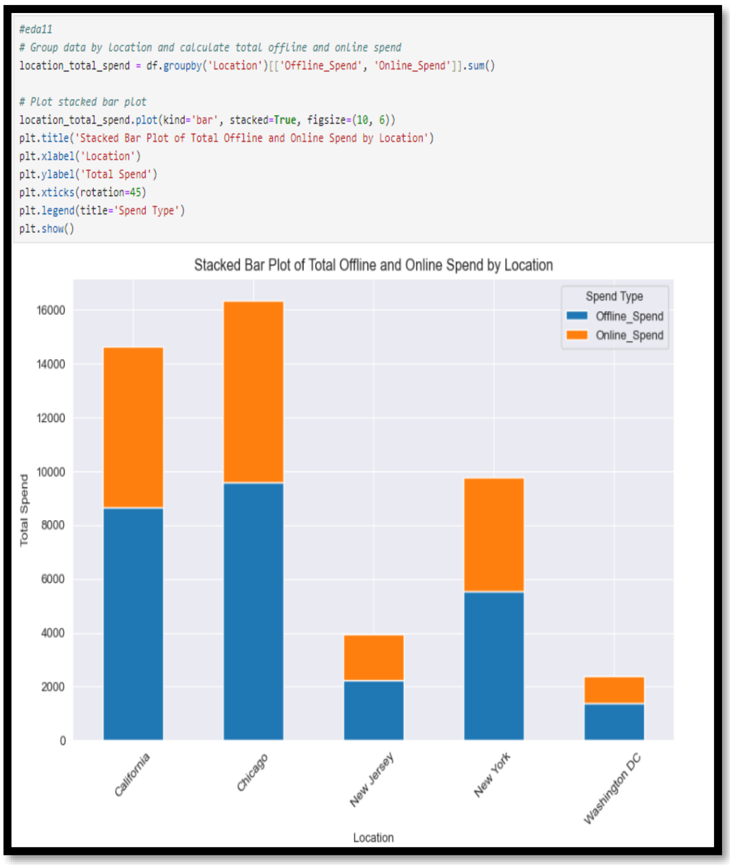


```
#eda10
#'Offline_Spend' is represented on the x-axis.
#'Online_Spend' is represented on the y-axis.
#The size of the points represents the 'Tenure_Months'.
#This plot helps visualize the relationship between offline and online spending, while also considering the tenure of customers. Adjust the parameters a
sns.scatterplot(x='Offline_Spend', y='Online_Spend', size='Tenure_Months', data=df)
plt.title('Scatter Plot: Offline Spend vs. Online Spend with Tenure Months')
plt.xlabel('Offline Spend')
plt.ylabel('Online Spend')
plt.show()
```



**Observation:** This plot helps visualize the relationship between offline and online spending, while also considering the tenure of customers. Adjust the parameters as needed based on your specific dataset and analysis goals.

**<u>Step11:</u> Group data by location and calculate total offline and online spend**
location_total_spend = df.groupby('Location')[['Offline_Spend', 'Online_Spend']].sum()
**Plot stacked bar plot**
location_total_spend.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Stacked Bar Plot of Total Offline and Online Spend by Location')
plt.xlabel('Location')
plt.ylabel('Total Spend')
plt.xticks(rotation=45)
plt.legend(title='Spend Type')
plt.show()



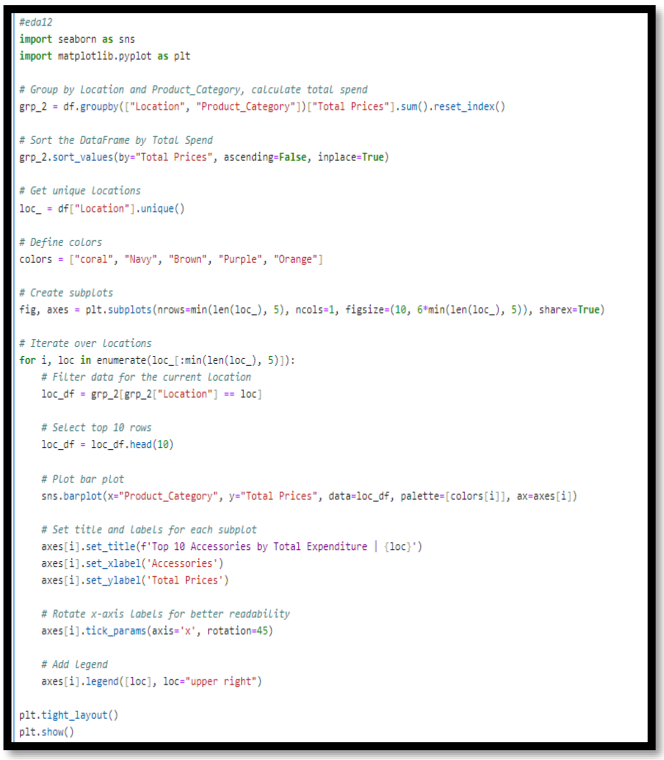
```
#eda11
# Group data by location and calculate total offline and online spend
location_total_spend = df.groupby('Location')[['Offline_Spend', 'Online_Spend']].sum()

# Plot stacked bar plot
location_total_spend.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Stacked Bar Plot of Total Offline and Online Spend by Location')
plt.xlabel('Location')
plt.ylabel('Total Spend')
plt.xticks(rotation=45)
plt.legend(title='Spend Type')
plt.show()
```

**<u>Observation:</u>**
This plot denotes the Location wise online and offline spend separately.
**<u>Step12:</u>**
**Group by Location and Product_Category, calculate total spend**
grp_2 = df.groupby(["Location", "Product_Category"])["Total Prices"].sum().reset_index()
**Sort the DataFrame by Total Spend**
grp_2.sort_values(by="Total Prices", ascending=False, inplace=True)
**Get unique locations**
loc_ = df["Location"].unique()
**Define colors**
colors = ["coral", "Navy", "Brown", "Purple", "Orange"]
**Create subplots**
fig, axes = plt.subplots(nrows=min(len(loc_), 5), ncols=1, figsize=(10, 6*min(len(loc_), 5)), sharex=True)
**Iterate over locations**
for i, loc in enumerate(loc_[:min(len(loc_), 5)]):
    # Filter data for the current location
    loc_df = grp_2[grp_2["Location"] == loc]
    **Select top 10 rows**
    loc_df = loc_df.head(10)
    **Plot bar plot**
    sns.barplot(x="Product_Category", y="Total Prices", data=loc_df, palette=[colors[i]], ax=axes[i])
**Set title and labels for each subplot**
    axes[i].set_title(f'Top 10 Accessories by Total Expenditure | {loc}')
    axes[i].set_xlabel('Accessories')
    axes[i].set_ylabel('Total Prices')
**Rotate x-axis labels for better readability**

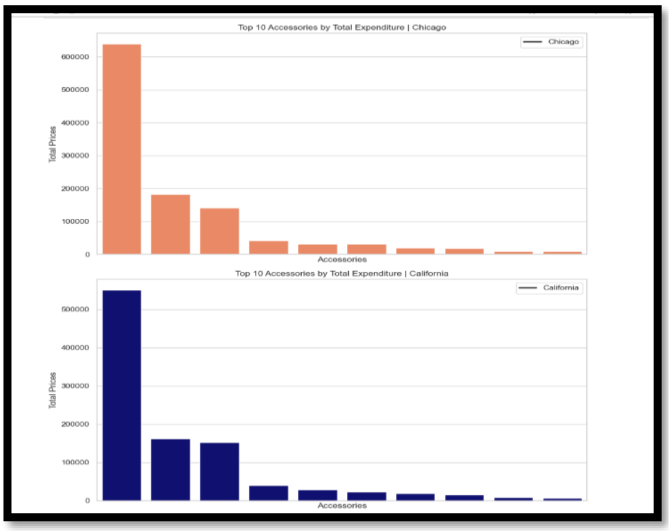
axes[i].tick_params(axis='x', rotation=45)
Add legend
axes[i].legend([loc], loc="upper right")
plt.tight_layout()
plt.show()

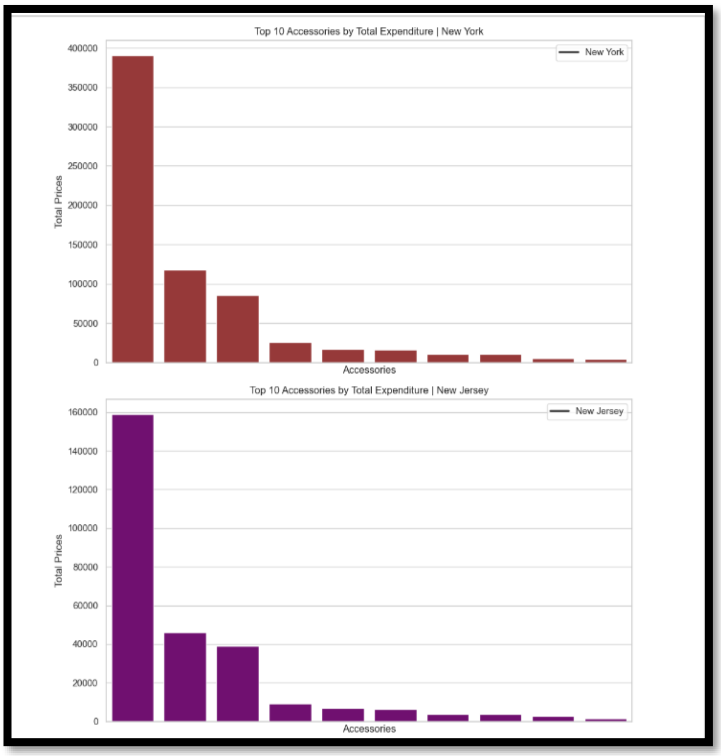
**Observation:**

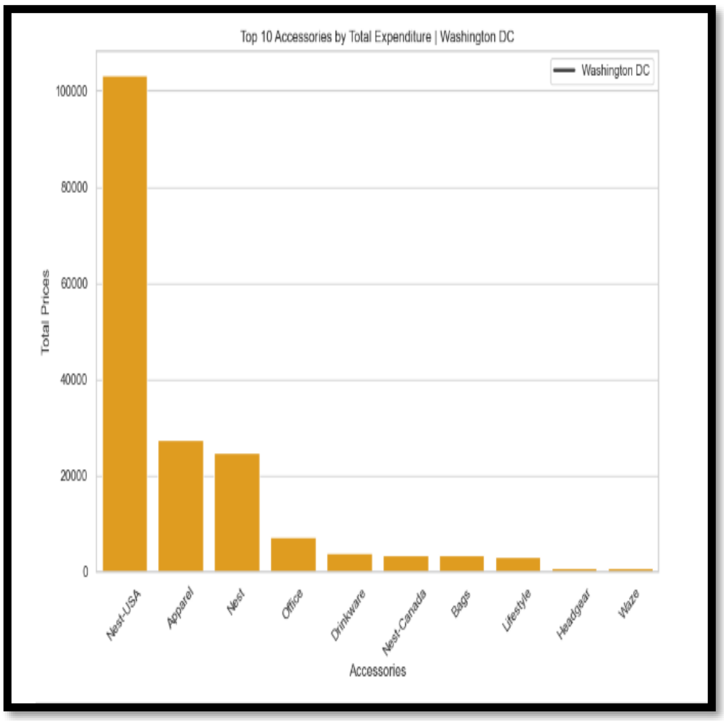
The graph comprises multiple subplots, each representing a location, showcasing the top 10 accessories by total expenditure. In each subplot, accessories are plotted on the x-axis, while total expenditure is represented on the y-axis. The height of each bar indicates the amount spent on a particular accessory category. Each subplot is distinguished by a unique color, with a legend providing location identification. This visualization offers a comparative insight into spending patterns across different locations.



```
#eda12
import seaborn as sns
import matplotlib.pyplot as plt

# Group by Location and Product_Category, calculate total spend
grp_2 = df.groupby(["Location", "Product_Category"])["Total Prices"].sum().reset_index()

# Sort the DataFrame by Total Spend
grp_2.sort_values(by="Total Prices", ascending=False, inplace=True)

# Get unique locations
loc_ = df["Location"].unique()

# Define colors
colors = ["coral", "Navy", "Brown", "Purple", "Orange"]

# Create subplots
fig, axes = plt.subplots(nrows=min(len(loc_), 5), ncols=1, figsize=(10, 6*min(len(loc_), 5)), sharex=True)

# Iterate over locations
for i, loc in enumerate(loc_[:min(len(loc_), 5)]):
    # Filter data for the current location
    loc_df = grp_2[grp_2["Location"] == loc]

    # Select top 10 rows
    loc_df = loc_df.head(10)

    # Plot bar plot
    sns.barplot(x="Product_Category", y="Total Prices", data=loc_df, palette=[colors[i]], ax=axes[i])

    # Set title and labels for each subplot
    axes[i].set_title(f'Top 10 Accessories by Total Expenditure | {loc}')
    axes[i].set_xlabel('Accessories')
    axes[i].set_ylabel('Total Prices')

    # Rotate x-axis labels for better readability
    axes[i].tick_params(axis='x', rotation=45)

    # Add legend
    axes[i].legend([loc], loc="upper right")

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







## II. PHASE 2

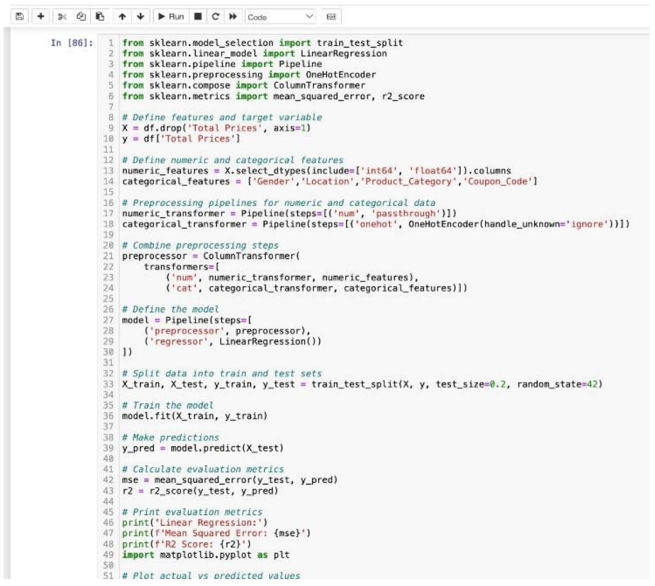
### 1. LINEAR REGRESSION ALGORITHM:

Linear regression is a statistical model used to predict the relationship between independent and dependent variables.

The simplest linear regression equation with one dependent variable and one independent variable is:

$$y = f(x) = \beta 0 + \beta 1 \ x$$

**code:**

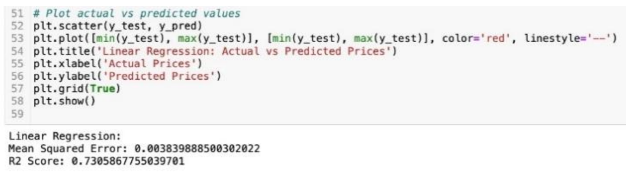


```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error, r2_score

# Define features and target variable
X = df.drop('Total Prices', axis=1)
y = df['Total Prices']

# Define numeric and categorical features
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = ['Gender','Location','Product_Category','Coupon_Code']

# Preprocessing pipelines for numeric and categorical data
numeric_transformer = Pipeline(steps=[('num', 'passthrough')])
categorical_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Define the model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print('Linear Regression:')
print(f'Mean Squared Error: {mse}')
print(f'R2 Score: {r2}')
import matplotlib.pyplot as plt

# Plot actual vs predicted values
```

In this code we import the Linear Regression method from sklearn.linear_model.

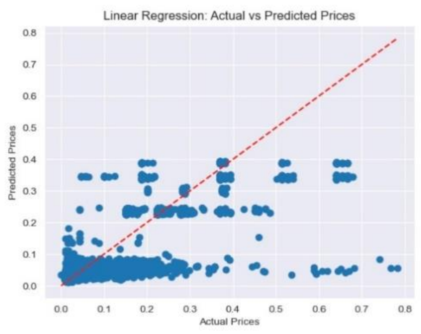
Test data size=0.2 and train data size= 0.8.

**Plot of linear Regression:**



```python
# Plot actual vs predicted values
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.title('Linear Regression: Actual vs Predicted Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.grid(True)
plt.show()
```

```
Linear Regression:
Mean Squared Error: 0.003839888500302022
R2 Score: 0.7305867755039701
```



**Justification for using this Linear Regression:**

1. This is the simplest algorithm which is used for continuous variable.
2. It provides linear relationship between independent and dependent variables.
3. Even after having many assumptions like linear dependency etc. still it can perform better if it’s some assumptions are reasonably met. We can see the accuracy from the above plot figure which is 73.05%.
4. It doesn't require extra computational resources for calculation which makes its suitable for large size datasets.
5. It's interpretability tells us how individual feature affecting target variable i.e Total price by keeping remaining input features constant.
6. This model is also computationally efficient,so choosen in case when i need immediate prediction.

**Work done for train/tune this model:**

1. Defining the target variable/independent variable, dependent variable(numerical/categorical features)
2. Initializing the pipeline construction for both numerical and categorical features.
3. Fitting the model into linear regression model
4. Calculating mean square error and R2
5. Plotting the linear regression graph.
6. We handled missing values of some columns namely offline_spend and online_spend by filling with mean values.
7. Categorical variables like location,gender,product category etc were encoded using one-hot encoding to convert them into a numerical format suitable for Linear Regression.
8. We normalized the features to ensure that all variables contribute equally to the model training.
9. We split our training and test data using train_test_split function by considering 80% of the data for training and 20% for testing.

**Model Effectiveness:**

The higher value of R^2 which is 73.05% and lower value of mean square error which is 0.003839 which indicate that this model performed very well.

With a MSE of approximately 0.00384 and an r2 score of of about 0.73, my model seems to perform reasonably well in terms of how closely the predictions match the actual values (as indicated by the MSE) and in terms of the proportion of variance explained by the model (as indicated by the r2 score).

The R2 score indicates likelihood of coefficients in this case R2 score is about 0.73 which indicates a better value in prediction.

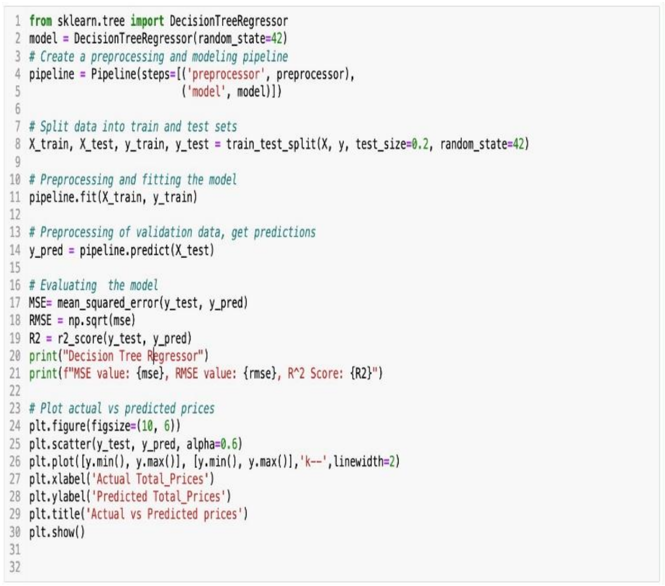
**Intelligence:**

Our Linear Regression model achieved an R-squared value of 0.73, indicating a strong linear relationship between the features and the target variable.

## 2. DECISION TREE REGRESSION ALGORITHM:

It is one of those machine learning algorithms under supervised learning which are used for both regression and classification types.

The model is trained and tested on a set of dataset that contains the categorization input(numerical/categorical).



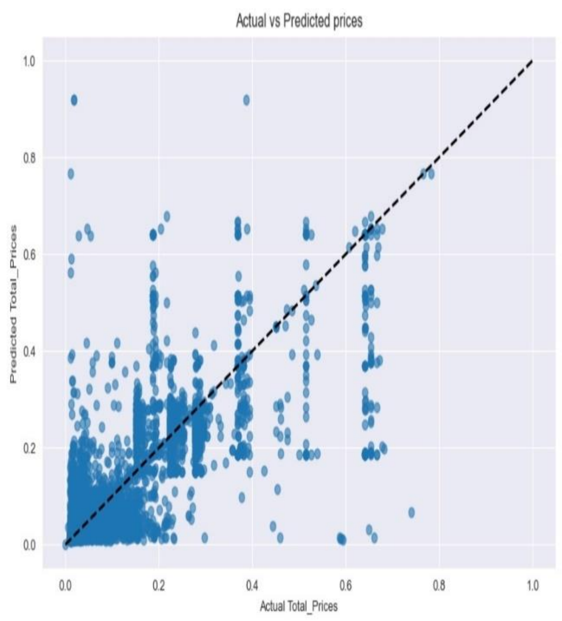
```python
from sklearn.tree import DecisionTreeRegressor
model = DecisionTreeRegressor(random_state=42)
# Create a preprocessing and modeling pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('model', model)])

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Preprocessing and fitting the model
pipeline.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
y_pred = pipeline.predict(X_test)

# Evaluating  the model
MSE= mean_squared_error(y_test, y_pred)
RMSE = np.sqrt(mse)
R2 = r2_score(y_test, y_pred)
print("Decision Tree Regressor")
print(f"MSE value: {mse}, RMSE value: {rmse}, R^2 Score: {R2}")

# Plot actual vs predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()],'k--',linewidth=2)
plt.xlabel('Actual Total_Prices')
plt.ylabel('Predicted Total_Prices')
plt.title('Actual vs Predicted prices')
plt.show()
```

In this code we import the Decision Tree Regression method from sklearn.tree.

Test data size=0.2 and train data size= 0.8 .

**Plot of Decision Tree Algorithm:**

```
Decision Tree Regressor
MSE value: 0.014338229570181625, RMSE value: 0.07194539119992807, R^2 Score: 0.6603178978006177
```



**Justification for using this decision tree:**

1. This model is mostly useful to find non-linear relationships between the model. In my dataset there are some input features whose relationship with target variable(Total Prices)is non linear.so i choose this model.
2. My dataset includes a mix of categorical (gender, product category, coupon status) and numerical (possibly quantities, prices, discount percentage) features. Decision trees naturally handle this variety of data types.
3. Robust towards outliers.
4. It is more efficient and scalable for large datasets.
5. The flexibility of decision tree allows to capture complex patterns in the datasets without being the specific functional forms.
6. This model is easy to understand.

**Work done for train/tune this model:**

1. Defining the target variable/independent variable, dependent variable (numerical/categorical features).
2. Initializing the pipeline construction for both numerical and categorical features.
3. Splitting the data into training data and testing data.
4. Since decision tree works well with categorical data One hot encoding is done before hand.
5. Analysed feature importances because of this model so that it is useful for deteremining which features are most useful in predicting total price.
6. Fitting the model into Decision Tree algorithm.
7. Preprocessing of validation data and get predictions.
8. Calculating mean square error and R2.
9. Plotting the Decision Tree graph.

**Model Effectiveness:**

The higher value of R^2 which is 66.031% and lower value of mean square error which is 0.041 which indicate that this model performed very well.

The MSE value indicates the Decision tree Regressor makes small errors in predicting the Total Prices. The R2 Score shows that the model has a moderate fit to the data.

Overall we can say that model performs moderatley well.

**Intelligence**:

With this Decision tree Regressor feature importance is known. Careful Tuning should be done in order to apply this model. It is useful for dynamic decision making.

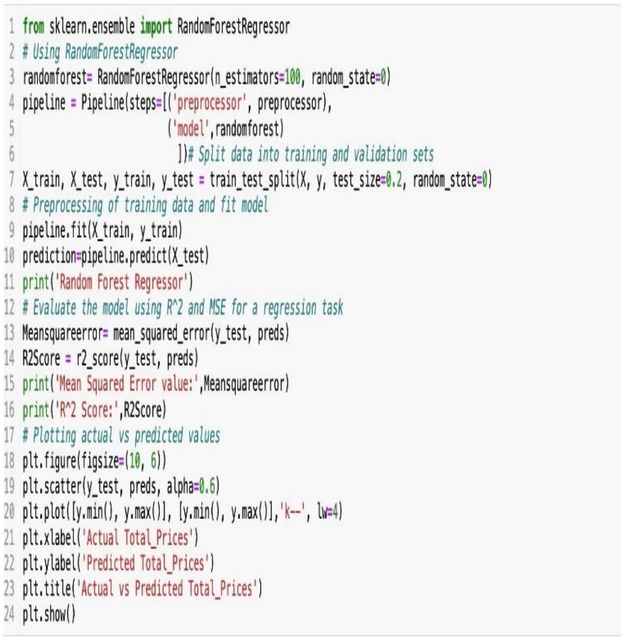
## 3. RANDOM FOREST REGRESSION ALGORITHM:

It is one of those machine learning algorithms under supervised learning which are used for both regression and classification types.

Random forest contains more than 1 decision tree on different subsets of the given dataset and predicts the average accuracy over all.

In this code we import the Random forest Regression method from sklearn.ensemble.

Test data size=0.2 and train data size= 0.8 .

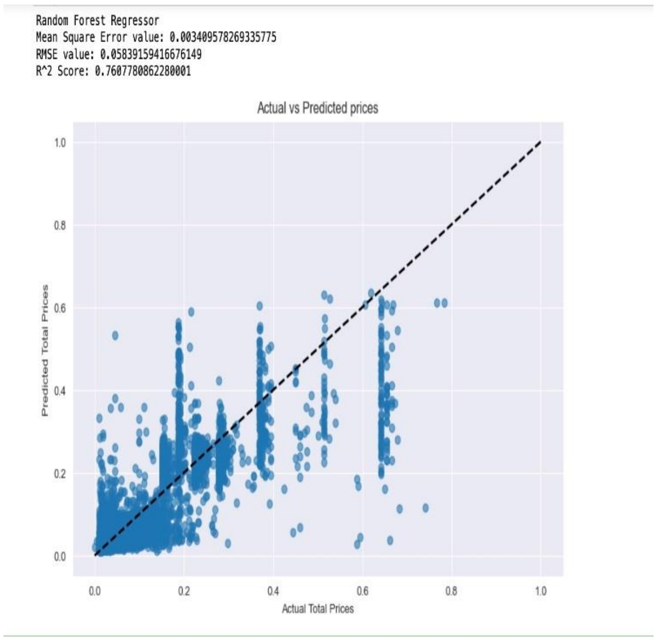


```python
from sklearn.ensemble import RandomForestRegressor
# Using RandomForestRegressor
randomforest= RandomForestRegressor(n_estimators=100, random_state=0)
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                          ('model',randomforest)
                           ])# Split data into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Preprocessing of training data and fit model
pipeline.fit(X_train, y_train)
prediction=pipeline.predict(X_test)
print('Random Forest Regressor')
# Evaluate the model using R^2 and MSE for a regression task
Meansquareerror= mean_squared_error(y_test, preds)
R2Score = r2_score(y_test, preds)
print('Mean Squared Error value:',Meansquareerror)
print('R^2 Score:',R2Score)
# Plotting actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, preds, alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()],'k--', lw=4)
plt.xlabel('Actual Total_Prices')
plt.ylabel('Predicted Total_Prices')
plt.title('Actual vs Predicted Total_Prices')
plt.show()
```

In this code we import the Random Forest Regression method from sklearn.ensemble.

Test data size=0.2 and train data size= 0.8 .

**Plot of Random Forest Algorithm:**



```
Random Forest Regressor
Mean Square Error value: 0.003409578269335775
RMSE value: 0.05839159416676149
R^2 Score: 0.7607780862280001
```

**Justification for using this Random Forest Algorithm:**

1. This algorithm can be used for non-linear relationships between dependent and the independent variable.
2. Random Forest can be applied to classification and regression problems.
3. Robust towards outliers.
4. It is more efficient and scalable for large datasets.
5. The flexibility of Random forest algorithm allows to capture complex patterns in the datasets without being the specific functional forms.
6. Random Forests generally will provide high accuracy due to their ensemble nature.
7. This model is mostly useful for capturing complex interactions between features.
8. This models includes Bagging, Boosting, Stacking which makes the model more efficient.
9. This model also gave how different features impact Total Price. Individual decision trees can overfit sometimes so this model makes more efficient prediction than Decision Tree Regressor and overall it reduces overfitting of the data.
10. Similar to decision trees, Random Forest can handle a mix of numerical and categorical data directly, which is good in predicting online sales.(Total Price).This random forest regressor inherently performs feature selection which makes the best choice to choose this algorithm.

**Work done for train/tune this model:**

1. Defining the target variable/independent variable, dependent variable (numerical/categorical features).
2. Initializing the pipeline construction for both numerical and categorical features.
3. Splitting the data into training data and testing data.
4. Fitting the model into Random Forest algorithm.
5. Preprocessing of validation data and get predictions.
6. One hot encoding is done so that Random Forest can handle categorical features when using certain implementations.
7. Feature importance is also known using this algorithm.

**Model Effectiveness:**

The higher value of R^2 which is 76.07% and lower value of mean square error which is 0.003400 which indicate that this model performed very well.

Random Forest Regression model demonstrates a good level of predictive accuracy and model fit, as evidenced by the low MSE and reasonably high R2 Score.

**Intelligence:**

This model performs well compared to Decision Tree Regressor.

Decision Tree Regressor

MSE value: 0.014338229570181625, RMSE value: 0.07194539119992807, R^2 Score: 0.6603178978006177.

Random Forest Regressor

Mean Square Error value: 0.003409578269335775

RMSE value: 0.05839159416676149

The comparison indicates that the Random Forest Regression model works better than the Decision Tree Regressor in my dataset. The Random Forest's lower MSE and higher R2 Score suggest it not only makes more accurate predictions on average but also has a slightly better fit to the data.

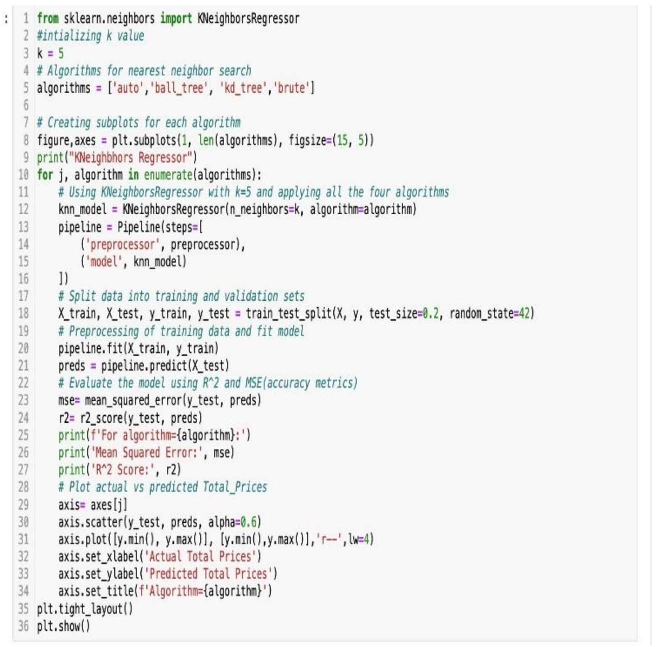
## 4. K- NEAREST NEIGHBHOUR ALGORITHM:

K-nearest neighbours (KNN) is a type of supervised learning algorithm used for both regression and classification types.

This process based on similar data points tends to keep in one similar class.

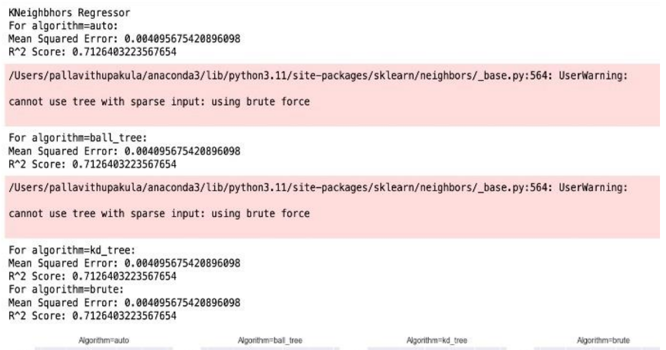
In this code we import the K-Nearest Neighbours method from sklearn.neighbors.

Test data size=0.2 and train data size= 0.8 .

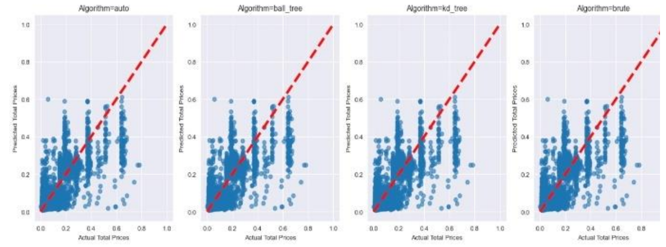


```
from sklearn.neighbors import KNeighborsRegressor
#intializing k value
k = 5
# Algorithms for nearest neighbor search
algorithms = ['auto','ball_tree', 'kd_tree','brute']

# Creating subplots for each algorithm
figure,axes = plt.subplots(1, len(algorithms), figsize=(15, 5))
print("KNeighbhors Regressor")
for j, algorithm in enumerate(algorithms):
    # Using KNeighborsRegressor with k=5 and applying all the four algorithms
    knn_model = KNeighborsRegressor(n_neighbors=k, algorithm=algorithm)
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', knn_model)
    ])
    # Split data into training and validation sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    # Preprocessing of training data and fit model
    pipeline.fit(X_train, y_train)
    preds = pipeline.predict(X_test)
    # Evaluate the model using R^2 and MSE(accuracy metrics)
    mse= mean_squared_error(y_test, preds)
    r2= r2_score(y_test, preds)
    print(f'For algorithm={algorithm}:')
    print('Mean Squared Error:', mse)
    print('R^2 Score:', r2)
    # Plot actual vs predicted Total_Prices
    axis= axes[j]
    axis.scatter(y_test, preds, alpha=0.6)
    axis.plot([y.min(), y.max()], [y.min(),y.max()],'r--',lw=4)
    axis.set_xlabel('Actual Total Prices')
    axis.set_ylabel('Predicted Total Prices')
    axis.set_title(f'Algorithm={algorithm}')
plt.tight_layout()
plt.show()
```

### Plot of K-Nearest Neighbours:



```
KNeighbhors Regressor
For algorithm=auto:
Mean Squared Error: 0.004095675420896098
R^2 Score: 0.7126403223567654

/Users/pallavithupakula/anaconda3/lib/python3.11/site-packages/sklearn/neighbors/_base.py:564: UserWarning:

cannot use tree with sparse input: using brute force

For algorithm=ball_tree:
Mean Squared Error: 0.004095675420896098
R^2 Score: 0.7126403223567654

/Users/pallavithupakula/anaconda3/lib/python3.11/site-packages/sklearn/neighbors/_base.py:564: UserWarning:

cannot use tree with sparse input: using brute force

For algorithm=kd_tree:
Mean Squared Error: 0.004095675420896098
R^2 Score: 0.7126403223567654
For algorithm=brute:
Mean Squared Error: 0.004095675420896098
R^2 Score: 0.7126403223567654
```



### Justification for using this K-Nearest Neighbour:

1. This is the simplest algorithm which is used for continuous variable.
2. K-NN can be used for both regression and classification types for mixed datatypes (numerical and categorical features).
3. Robust towards outliers.
4. This method requires no training generally. It will only store the training dataset to get high efficient, especially for large datasets.
5. K-value in the KNN algorithm involves in the cross-validation to find the optimal value for the specific dataset.
6. KNN is a versatile and flexible algorithm.

### Work done for train/tune this model:

1. Initializing the k value.
2. Creating the sub-plots for each algorithm.
3. Initializing the pipeline construction for all algorithms.
4. Splitting the data into training data and testing data.
5. Fitting the model into K-Nearest Neighbour algorithm.
6. Preprocessing of validation data and get predictions.
7. Calculating mean square error and R2.
8. Plotting the K-Nearest Neighbour Algorithm.

### Model Effectiveness:

The higher value of R^2 which is 71.27% and lower value of mean square error which is 0.00409 which indicate that this model performed very well.

## 5. BAYESIAN RIDGE REGRESSION:

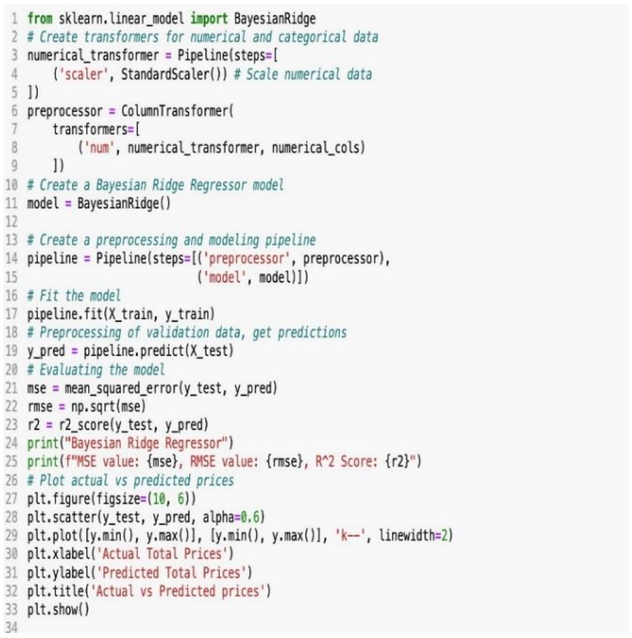
Bayesian Ridge regression which estimates a solution by a probabilistic model.

Prior for the coefficient w is given by spherical Gaussian:

$$p(w/\lambda)=N(w|0,\lambda-1Ip)$$

In this code we import the Bayesian ridge Regression method from sklearn.linear_model.

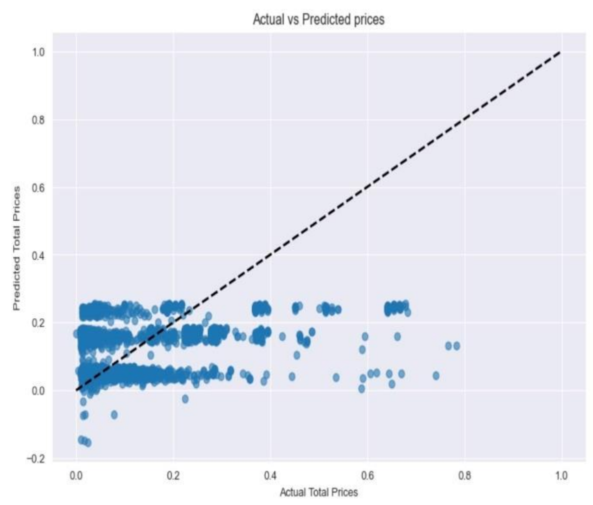
Test data size=0.2 and train data size= 0.8



```python
from sklearn.linear_model import BayesianRidge
# Create transformers for numerical and categorical data
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler()) # Scale numerical data
])
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols)
    ])
# Create a Bayesian Ridge Regressor model
model = BayesianRidge()

# Create a preprocessing and modeling pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('model', model)])
# Fit the model
pipeline.fit(X_train, y_train)
# Preprocessing of validation data, get predictions
y_pred = pipeline.predict(X_test)
# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("Bayesian Ridge Regressor")
print(f"MSE value: {mse}, RMSE value: {rmse}, R^2 Score: {r2}")
# Plot actual vs predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', linewidth=2)
plt.xlabel('Actual Total Prices')
plt.ylabel('Predicted Total Prices')
plt.title('Actual vs Predicted prices')
plt.show()
```

**Plot of Bayesian Ridge:**

```
Bayesian Ridge Regressor
MSE value: 0.009683475970922877, RMSE value: 0.09840465421372546, R^2 Score: 0.32059056260332564
```



**Justification for using this Bayesian Ridge Algorithm:**

1. It provides a probabilistic framework for input parameters which will make more stable estimation/target variables.
2. Robust towards outliers.
3. It is overcomes the overfitting of the data due to having the capability of regularization.
4. It is scalable to large data sets.
5. The flexibility of Bayesian ridge algorithm allows to capture complex patterns in the datasets without being the specific functional forms which is same like random forest.

**Work done for train/tune this model:**

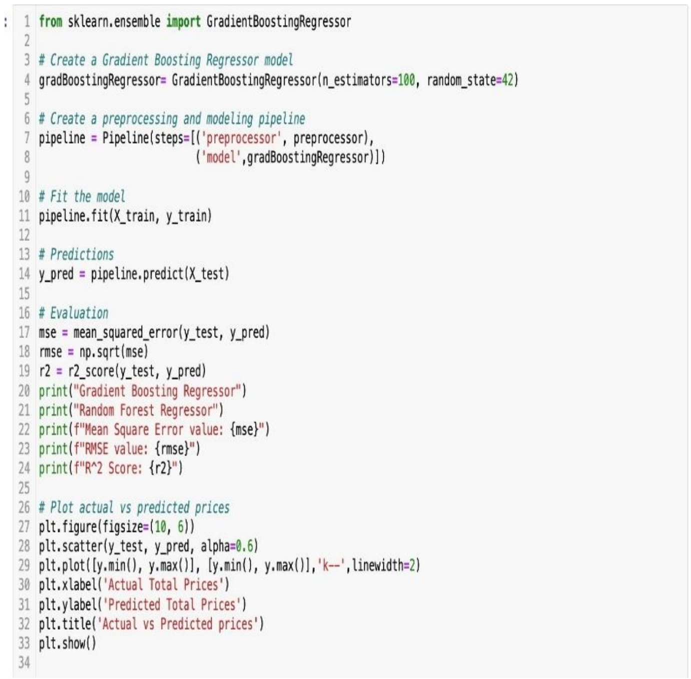
1. Defining the target variable/independent variable, dependent variable (numerical/categorical features).
2. Initializing the pipeline construction for both numerical and categorical features.
3. Splitting the data into training data and testing data.
4. Fitting the model into Bayesian ridge algorithm.
5. Calculating mean square error and R2.
6. Plotting the Bayesian ridge Algorithm.

**Model Effectiveness:**

The low value of R^2 which is 32.5% and lower value of mean square error which is 0.0096,root mean square error is 0.05 which indicate that this model performed not very well.

## 6. GRADIENT BOOSTING ALGORITHM:

Gradient boosting is an algorithm whose efficiency or accuracy will increase gradually.



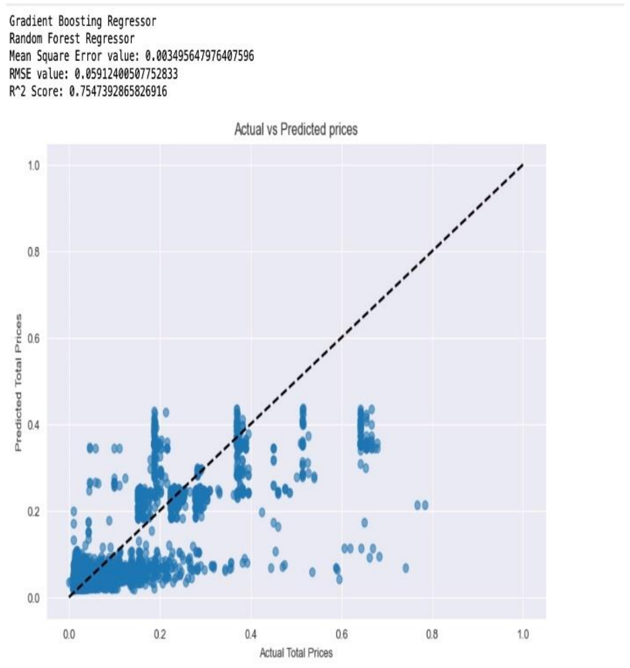
```python
from sklearn.ensemble import GradientBoostingRegressor

# Create a Gradient Boosting Regressor model
gradBoostingRegressor= GradientBoostingRegressor(n_estimators=100, random_state=42)

# Create a preprocessing and modeling pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('model',gradBoostingRegressor)])

# Fit the model
pipeline.fit(X_train, y_train)

# Predictions
y_pred = pipeline.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("Gradient Boosting Regressor")
print("Random Forest Regressor")
print(f"Mean Square Error value: {mse}")
print(f"RMSE value: {rmse}")
print(f"R^2 Score: {r2}")

# Plot actual vs predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()],'k--',linewidth=2)
plt.xlabel('Actual Total Prices')
plt.ylabel('Predicted Total Prices')
plt.title('Actual vs Predicted prices')
plt.show()
```

To start the process, we need an initial guess or prediction. The initial guess is always the average of the target.

In this code we import the Gradient Boosting Regressor method from sklearn.linear_ensemble.

Test data size=0.2 and train data size= 0.8

**Plot of Gradient Boosting:**



```
Gradient Boosting Regressor
Random Forest Regressor
Mean Square Error value: 0.003495647976407596
RMSE value: 0.05912400507752833
R^2 Score: 0.7547392865826916
```

**Justification for using this Gradient Boosting Algorithm:**

1. Gradient Boosting works well for regression problem. Since my target variable is a continuous variable i choose this algorithm.
2. It works well with huge dataset and my dataset contains about 52k rows.
3. It is also robust to outliers.
4. It performs well even without hyperparameter tuning. Work done to tune/train model. In this model tuning hyper parameter is done that is i have set the number of trees i.e n_estimators to 100.
5. The model is trained on a training dataset, and the hyperparameter(n_estimators) is tuned.
6. After tuning, the final model is evaluated on dataset to assess its performance on unseen data

**Work done for train/tune this model:**

1. Defining the target variable/independent variable, dependent variable (numerical/categorical features).
2. Initializing the pipeline construction for both numerical and categorical features.
3. Splitting the data into training data and testing data
4. Fitting the model into gradient boosting algorithm.
5. Calculating mean square error and R2.
6. Plotting the gradient boosting Algorithm.

**Effectiveness of Gradient Boosting Regressor:**

The higher value of R^2 which is 75.47% and lower value of mean square error which is 0.0034,root mean square error is 0.059 which indicate that this model performed very well. This Shows that algorithm performs well and able to predict unknown value with good accuracy.

MSE and RMSE measure the average squared difference between the predicted and actual values, with lower values indicating better performance.

R2 score measures the proportion of the variance in the target variable that is predictable from the features. A higher R2 score indicates better model fit. Here the values of RMSE,MSE,R2 SCORE are better than Linear Regression.

The plot also shows that model mostly predicted correctly for predicted values. The Random Forest Regressor has the lowest MSE (0.003409578269335775) and the highest R2 Score (0.7607780862280001), indicating better performance in terms of both minimizing prediction errors and explaining variance in the target variable.

The Gradient Boosting Regressor also performed well with a low MSE (0.003495647976407596) and a high R2 Score (0.7547392865826916). --->This model also minimizes errors.

R^2 score measures the proportion of the variance in the target variable that is predictable from the features. A higher R2

score indicates better model fit. Here the values of RMSE,MSE,R2 SCORE are better than Linear Regression.

**Intelligence :**

The MSE of the Gradient Boosting Regressor (0.003495647976407596) is lower than that of Linear Regression (0.0038396333905515233).This indicates Gradient Boosting Regressor performed well than Linear Regression in calculating difference between actual and predicted values.

R2 score of the Gradient Boosting Regressor (0.7547392865826916) is higher than that of Linear Regression (0.7306046744456911). A higher R2 score indicates that a larger proportion of the variance in the Total Prices. This indicates that again Gradient Boosting Regressor predicts well and generalizes well compared to that of Linear Regression.

Therefore, We can say Gradient Boosting Regressor performs well than Linear Regression model.

## III. PHASE3

**1b) why we choose random forest regressor for my data product ?**

At first we collected data from kaggle i.e online shopping data.Then we have done data cleaning where all the unnecessary columns,rows,outliers everything has been removed and also label encoding is done for some of the columns like Coupon status.

After that EDA(Exploratory Data Analysis) is done which shows relationship among the variables and how they affect

the target variable.Like in my problem statement the total price prediction is influenced by many factors like coupon code,discount percentage,location,product,amount spend in online and offline,GST and how long they have been customers for that particular store.

We have also have plotted covariance matrix which shows how much each variable affecting other variables.

After cleaning and EDA,Dataset is trained with models like Linear Regression,Random Forest Regression,Decision Tree Regressor,Kneighbors regressor,bayesian ridge regressor. Since my target variable is numerical value i mostly applied regression algorithms to my dataset.

Out of all algorithms Random Forest Regressor model fits well for my model and the R2 score,RMSE,MSE values of Random Forest Regressor are as follows:

Mean Square Error value: 0.0033982972864618423

RMSE value: 0.05829491647186607

R^2 Score: 0.761569579515191

The higher the R2 score and lower the mse value shows that model fit well.Out of all the six algorithms applied to my dataset i came to know that this Random Forest Regressor works well.Now we have applied Random Forest Regressor model to predict Total Prices since the models fits the data well.

**Working Instructions:**

For this project,I have used flask framework and used python programming language.Firstly we need to install necessary libraries,modules like Flask. Installing of necessary packages as follows:

**pip install flask flask-cors pandas scikit-learn**

We need to import Pickle module to serialize and deserialize python objects.

Firstly **pickle.dump(pipeline,open('RandomForestRegressorModel.pkl','wb'))**

is done inorder to save the pipeline (including the Random Forest Regressor model it contains) to a file named **'RandomForestRegressorModel.pkl'**. After that pickle.load() function is used to reconstruct the pipeline object and use it for making predictions on new data without needing to retrain the model.

To navigate to the predict page i have created header where we can navigate to the predict web page and give all the necessary inputs and give the model all the necessary inputs to predict the Total Price.

The predict function in python helps to take input from the user and give it to the model to predict the output and again renders predict.html file to display the output.

Get and post methods are used for the following uses:

GET method is used to display the prediction form initially.

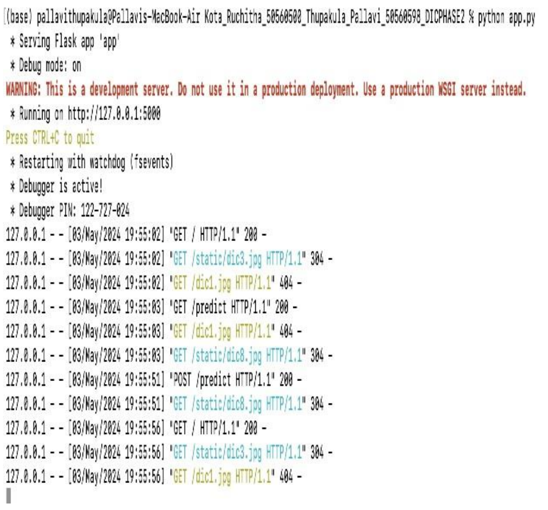
POST method is used to to handle form submissions and provide predictions based on the input data.

Html and css is used to design the web page.

Running the program after necessary installation:

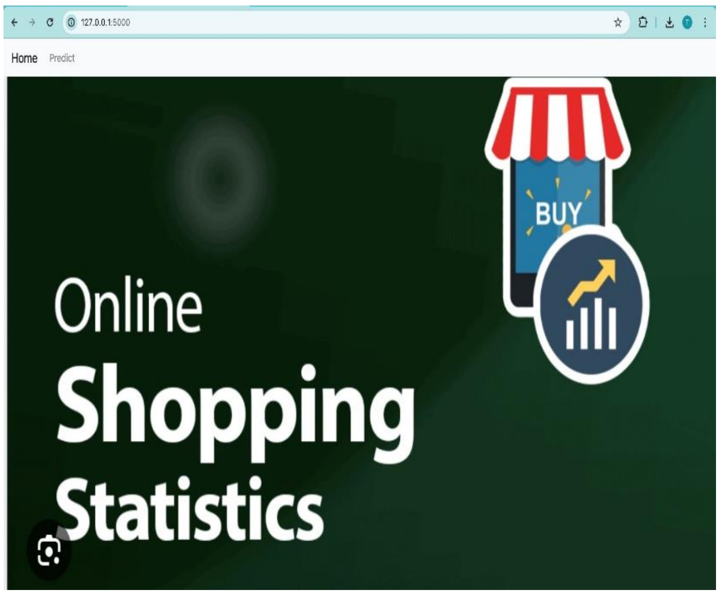
The HTML files are stored in templates folder and images that are required for web application are stored in static folder.The random forest regressor model .pkl file is stored in same path as app.py.Let's run the application using command

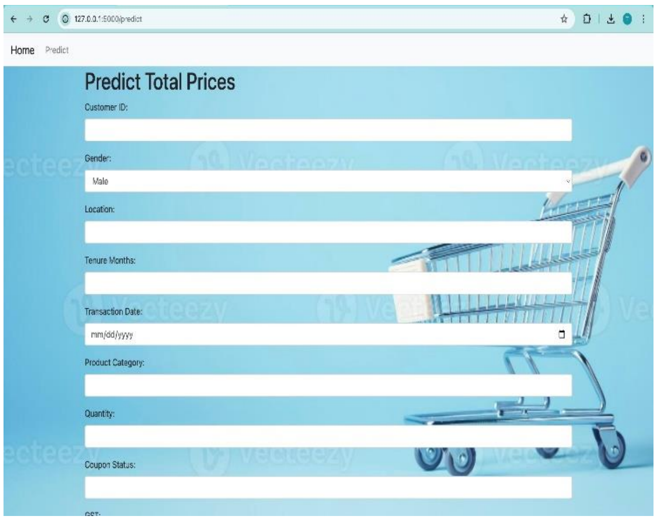
**python app.py**



```
(base) pallavithupakula@Pallavis-MacBook-Air Kota_Ruchitha_50560500_Thupakula_Pallavi_50560598_DICPHASE2 % python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 122-727-024
127.0.0.1 - - [03/May/2024 19:55:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2024 19:55:02] "GET /static/dic3.jpg HTTP/1.1" 304 -
127.0.0.1 - - [03/May/2024 19:55:02] "GET /dic1.jpg HTTP/1.1" 404 -
127.0.0.1 - - [03/May/2024 19:55:03] "GET /predict HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2024 19:55:03] "GET /dic1.jpg HTTP/1.1" 404 -
127.0.0.1 - - [03/May/2024 19:55:03] "GET /static/dic8.jpg HTTP/1.1" 304 -
127.0.0.1 - - [03/May/2024 19:55:51] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2024 19:55:51] "GET /static/dic8.jpg HTTP/1.1" 304 -
127.0.0.1 - - [03/May/2024 19:55:56] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/May/2024 19:55:56] "GET /static/dic3.jpg HTTP/1.1" 304 -
127.0.0.1 - - [03/May/2024 19:55:56] "GET /dic1.jpg HTTP/1.1" 404 -
```

The home page of our web application looks like this where we can navigate to the predict page to predict output.





Predict Total Prices

Customer ID:

Gender:

Male

Location:

Tenure Months:

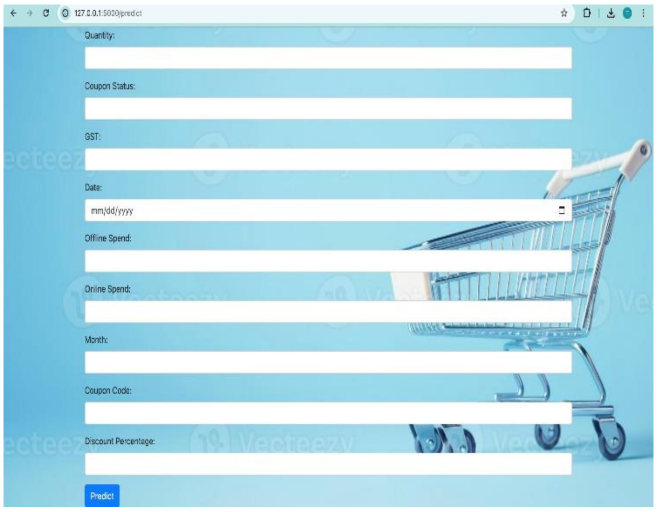
Transaction Date:

mm/dd/yyyy

Product Category:

Quantity:

Coupon Status:



127.0.0.1:5000/predict

Quantity:

Coupon Status:

GST:

Date: mm/dd/yyyy

Offline Spend:

Online Spend:

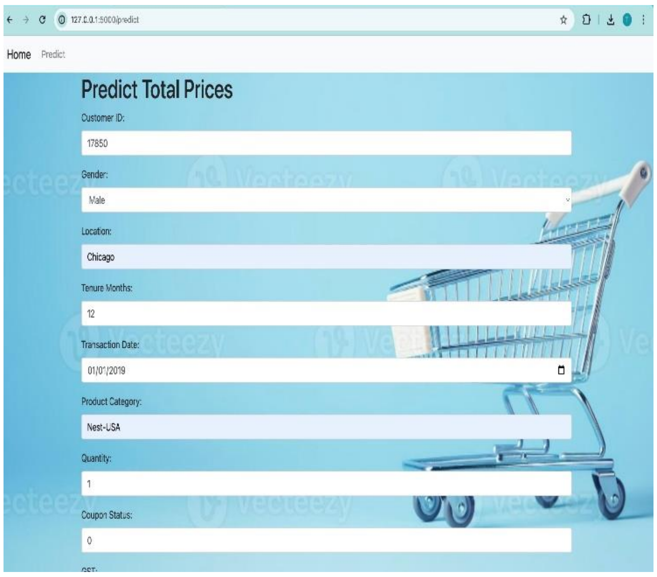
Month:

Coupon Code:

Discount Percentage:

Predict

Predicting our data product with some random values



127.0.0.1:5000/predict

Home Predict

Predict Total Prices

Customer ID: 17850

Gender: Male

Location: Chicago

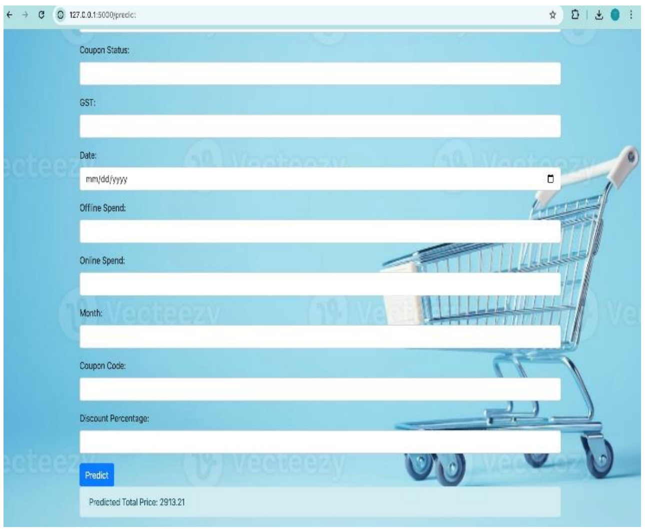
Tenure Months: 12

Transaction Date: 01/01/2019

Product Category: Nest-USA

Quantity: 1

Coupon Status: 0



127.0.0.1:5000/predic:

Coupon Status:

GST:

Date: mm/dd/yyyy

Offline Spend:

Online Spend:

Month:

Coupon Code:

Discount Percentage:

Predict

Predicted Total Price: 2913.21

The above is predicted value for the random inputs given.

**<u>Recommendation of Data Product to target users:</u>**

1. This data product will help us know how different factors like GST,Coupon status,Discount percentage,Product Category,Product quantity,Online and offline amount spend affects in maximizing sales or revenue and also for giving out dioscounts or offers based on the purchase.
2. Customer Segmentation: Analyzing total prices of different customers helps to know customer behavior.This is also useful for retailers for their marketing strategies.
3. With analyzing total prices personalized recommendations can also be possible like which customer needs what and thereby providing them with the additional products that they need.
4. It will also help the retailers to know the quality of products thereby the retailer takes care of that next time in quality issue and also plan inventory accordingly.
5. It will also useful to identify the fraud customer based on total price variations and by predicting it.
6. Finally it useful to identify customer behavior and how various factors affecting this total price.

## References

I. [1] C. O’Neill and R. Schutt. Doing Data Science., O’Reilly. 2013.
II. [2] J. VanderPlas, Python Data Science Handbook., O'Reilly. 2016.
III. https://seaborn.pydata.org/tutorial.html
IV. https://scikit-learn.org/stable/
V. https://plotly.com/python/
VI. https://getbootstrap.com/