## You have been given the role of a Business Analyst for an E-Commerce company and have been asked to prepare a basic report on the data. Follow the steps below for preparation of the report.

## Load the necessary libraries. Import and load the dataset with a name ECom_Data.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

pd.options.display.max_rows = 10
```

In [3]:
```
# Get the Data

ECom_Data=pd.read_csv("E-Commerce.csv")
```

We have read the data and stored the data in "ECom_Data" variable.

## Q 1. To get familiar with the data:

### a) Print out the first 5 and the last 5 records of the data.

### b) How many rows and columns are present in the dataset? Use any two different methods to extract this information.

### c) How many object data types are there?

### d) Is there any Boolean data type?

### Ans 1 a)

head method is used to return first n rows in a DataFrame object. It will return first 5 rows (default value) if n is not provided as parameter.

df.head(), df.head(n)

In [4]:
```
ECom_Data.head()
```

Out[4]:

| | Customer_uniq_id | Region | Order_Date | Expected_Delivery_Date | Deliver |
|---|---|---|---|---|---|
| **0** | e71017e224688489edfe856f2308806d | East | 24-10-2021 | 25-10-2021 | 25 |
| **1** | 6286847ee2da18f587503db49511c539 | East | 24-10-2021 | 25-10-2021 | 25 |
| **2** | 0686fec9b70e5039583a38119ca0c835 | West | 24-10-2021 | 25-10-2021 | 25 |
| **3** | ea2406dc597bee2abb6b867fa668501f | West | 24-10-2021 | 25-10-2021 | 25 |
| **4** | 5935ed077915347dc695744df68c565c | East | 03-09-2021 | 04-09-2021 | 04 |

tail method is used to return last n rows in a DataFrame object. It will return last 5 rows (default value) if n is not provided as parameter.

df.tail(), df.tail(n)

In [5]:
```
ECom_Data.tail()
```

Out[5]:

| | Customer_uniq_id | Region | Order_Date | Expected_Delivery_Date | De |
|---|---|---|---|---|---|
| **8901** | 90d30478255e23621e8929ed15c2f6e4 | South | 01-12-2020 | 04-12-2020 | |
| **8902** | 20a73e3f41490a73ceeba5f17658db8f | West | 01-12-2020 | 04-12-2020 | |
| **8903** | 5c1554cd45f9d538c2c6947dbdd59c75 | East | 01-12-2020 | 04-12-2020 | |
| **8904** | 6b737a4deca1ed0e56c179e66036e994 | West | 01-12-2020 | 04-12-2020 | |
| **8905** | a5235ac28d3d5487f54025f9d6b57433 | North | 01-12-2020 | 04-12-2020 | |

## Ans 1 b)

shape attribute of DataFrame object returns tuple containing number of rows and columns. First value in tuple represent number of rows and second value represent number of columns.

In [6]:
```
rows = ECom_Data.shape[0]
cols = ECom_Data.shape[1]
```

In [7]:
```
print("Number of Rows: ", rows, "\nNumber of Columns: ", cols)
```

```
Number of Rows:  8906
Number of Columns:  17
```

len method can be used to compute number of rows and columns in a DataFrame object. axes 0 represent rows and 1 represent columns in a DataFrame object.

In [8]:
```
rows = len(ECom_Data.axes[0])
cols = len(ECom_Data.axes[1])
```

In [9]:
```
print("Number of Rows: ", rows, "\nNumber of Columns: ", cols)
```

```
Number of Rows:  8906
Number of Columns:  17
```

### Ans 1 c)

info method gives concise summary of DataFrame object.

There are 14 object data types in ECom_Data DataFrame object.

In [10]:
```
ECom_Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8906 entries, 0 to 8905
Data columns (total 17 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Customer_uniq_id            8906 non-null   object 
 1   Region                      8906 non-null   object 
 2   Order_Date                  8906 non-null   object 
 3   Expected_Delivery_Date      8906 non-null   object 
 4   Delivered_Date              8906 non-null   object 
 5   product_name                8906 non-null   object 
 6   product_main_category       8906 non-null   object 
 7   product_subcategory         8906 non-null   object 
 8   product_category_filter     8906 non-null   object 
 9   product_category_subfilter  8906 non-null   object 
 10  product_unique ID           8906 non-null   object 
 11  retail_price                8906 non-null   int64  
 12  discounted_price            8906 non-null   int64  
 13  product_rating              8906 non-null   float64
 14  Brand                       8906 non-null   object 
 15  product_specifications      8906 non-null   object 
 16  description                 8906 non-null   object 
dtypes: float64(1), int64(2), object(14)
memory usage: 1.2+ MB
```

### Ans 1 d)

There is no Boolean data type column in ECom_Data DataFrame object.

## Once we are familiar with the data, we may decide that not all features are of use to you and we may want to delete the non-informative features (columns).

## Q 2. Eliminating the non-informative columns:

### a) Drop the columns product_specifications and description.

### b) Which method or function is used to permanently delete the columns mentioned in part (a)?

### Ans 2 a)

drop() method removes specified row or column in a DataFrane object. axis = 0 is specified for rows and 1 is for columns.

In [11]:
```
ECom_Data_Drop = ECom_Data.copy()
```

In [12]:
```
ECom_Data_Drop.drop(['product_specifications','description'], axis=1)
```

Out[12]:

| | Customer_uniq_id | Region | Order_Date | Expected_Delivery_Date | De |
|---|---|---|---|---|---|
| **0** | e71017e224688489edfe856f2308806d | East | 24-10-2021 | 25-10-2021 | |
| **1** | 6286847ee2da18f587503db49511c539 | East | 24-10-2021 | 25-10-2021 | |
| **2** | 0686fec9b70e5039583a38119ca0c835 | West | 24-10-2021 | 25-10-2021 | |
| **3** | ea2406dc597bee2abb6b867fa668501f | West | 24-10-2021 | 25-10-2021 | |
| **4** | 5935ed077915347dc695744df68c565c | East | 03-09-2021 | 04-09-2021 | |
| **...** | ... | ... | ... | ... | |
| **8901** | 90d30478255e23621e8929ed15c2f6e4 | South | 01-12-2020 | 04-12-2020 | |
| **8902** | 20a73e3f41490a73ceeba5f17658db8f | West | 01-12-2020 | 04-12-2020 | |
| **8903** | 5c1554cd45f9d538c2c6947dbdd59c75 | East | 01-12-2020 | 04-12-2020 | |
| **8904** | 6b737a4deca1ed0e56c179e66036e994 | West | 01-12-2020 | 04-12-2020 | |
| **8905** | a5235ac28d3d5487f54025f9d6b57433 | North | 01-12-2020 | 04-12-2020 | |

8906 rows × 15 columns

### Ans 2 b)

If inplace = True is set in drop method than specific row or column is removed permanently from the DataFrame object.

In [13]:
```
ECom_Data_Drop.dtypes
```

Out[13]:
```
Customer_uniq_id           object
Region                     object
Order_Date                 object
Expected_Delivery_Date     object
Delivered_Date             object
                            ...
discounted_price            int64
product_rating            float64
Brand                      object
product_specifications     object
description                object
Length: 17, dtype: object
```

In [14]:
```
ECom_Data_Drop.drop(['product_specifications','description'], axis=1, inplace=True)
```

In [15]:
```
ECom_Data_Drop.dtypes
```

Out[15]:
```
Customer_uniq_id           object
Region                     object
Order_Date                 object
Expected_Delivery_Date     object
Delivered_Date             object
                            ...
product_unique ID          object
retail_price                int64
discounted_price            int64
product_rating            float64
Brand                      object
Length: 15, dtype: object
```

## The next steps in this project involves summarization of data at various levels and visualization. Apparently, such simple steps are very useful to get an overall sense of the data.

## Q 3. Here we summarize the data at Brand level:

### a) How many unique Brand are there?

### b) Show the average product_rating within each Brand.

### Ans 3 a)

nunique method is used to return the number of unique values in a column.

In [16]:
```
tot_unique_brands = ECom_Data['Brand'].nunique()
print('Total unique brands:',tot_unique_brands)
```

```
Total unique brands: 2484
```

### Ans 3 b)

agg() method can be used to apply the aggregation (ex: mean, sum, min, max ..) on a groupby object.

In [17]:
```
ECom_Data.groupby('Brand').agg(avg_product_rating=('product_rating', 'mean'))
```

Out[17]:

| Brand | avg_product_rating |
|---|---|
| **1OAK** | 1.500000 |
| **3A AUTOCARE** | 3.268293 |
| **3D MAT** | 3.000000 |
| **3KFACTORY** | 2.000000 |
| **4D** | 3.600000 |
| **...** | ... |
| **ZORDEN** | 4.000000 |
| **ZOSIGN** | 3.400000 |
| **ZRESTHA** | 1.000000 |
| **ZYXEL** | 3.333333 |
| **TARKAN** | 5.000000 |

2484 rows × 1 columns

## Q 4. Next we study the main categories of the products:

### a) Create an appropriate plot to show the count of items ordered for each product_main_category.

### b) From the plot identify for which two product_main_category(s) maximum and minimum orders were placed.

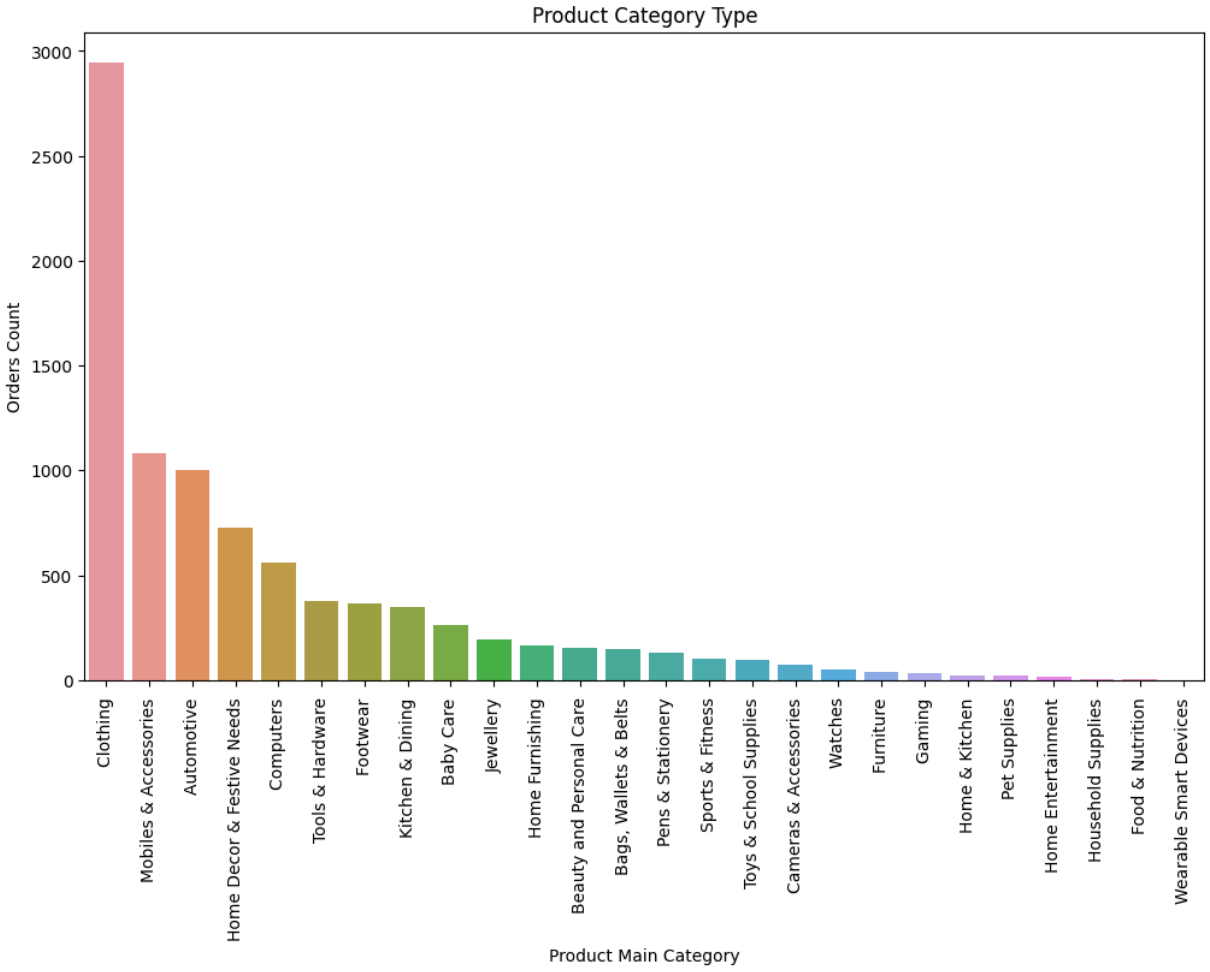
### c) Write code to print out the top 5 product_main_category(s) in descending order.

## Ans 4 a)

Count plot is used to represent the occurrence(counts) of the observation present in the categorical variable. It uses the concept of a bar chart for the visual depiction.

xticks or yticks can be used to rotate column labels on x or y axis in a chart.
title can be used to give header to a chart.
xlabel or ylabel can be used to change column names on x or y axis in a chart.

In [18]:
```
plt.figure(figsize=(12,7))
sns.countplot(data=ECom_Data, x='product_main_category', order = ECom_Data['product
plt.xticks(rotation=90)
plt.title('Product Category Type')
plt.xlabel('Product Main Category')
plt.ylabel('Orders Count')
plt.show()
```




## Ans 4 b)

In [19]:
```
print('Maximum orders were placed for ' + '\033[1m' + 'Clothing' + '\033[0m' + ' pr
print('Minimum orders were placed for ' + '\033[1m' + 'Wearable Smart Devices' + '\
```

Maximum orders were placed for **Clothing** product main category.
Minimum orders were placed for **Wearable Smart Devices** product main category.

### Ans 4 c)

value_counts method returns a Series containing counts of unique values. Default order is descending.

In [20]:
```
ECom_Data['product_main_category'].value_counts().head(5)
```

Out[20]:
```
product_main_category
Clothing                        2943
Mobiles & Accessories           1084
Automotive                      1001
Home Decor & Festive Needs       727
Computers                        558
Name: count, dtype: int64
```

## In E-commerce, both the retailers (here brands) and the company have to make profit to sustain in the business. The E-Commerce company has the following rule for computing their own revenue:

### The company charges:

### (i) 25% on the orders having final price (discounted price) greater than 600

### (ii) 15% on the orders having final price (discounted price) greater than 350 but less than or equal to 600

### (iii) 10% on the orders having final price (discounted price) greater than 100 but less than or equal to 350

### (iv) Otherwise, 5% on the final price (discounted price)

## Q 5. Find the Total Revenue generated by the E-Commerce company over all orders placed.

### Ans 5

map is a method that works as an iterator to return a result after applying a function to every item of an iterable (ex: column in a DataFrame object). It is used when we want to apply a single transformation function to all the iterable elements.

In [21]:
```
def compute_revenue(discounted_price):
    if discounted_price > 600:
        return discounted_price * 0.25
    elif 350 < discounted_price <= 600:
        return discounted_price * 0.15
    elif 100 < discounted_price <= 350:
```

```
        return discounted_price * 0.10
    else:
        return discounted_price * 0.05

ECom_Data['Revenue'] = ECom_Data['discounted_price'].map(compute_revenue)

Total_Revenue = ECom_Data['Revenue'].sum()

print('Total revenue:',Total_Revenue)
```

```
Total revenue: 2217486.85
```

## Now we need to find the revenue for each retailer (Brand).

## Q6. Calculate the total BrandRevenue and list the top 10 Brand having maximum revenue in descending order.

### Ans 6

Brand Revenue is calculated by deducting the Revenue from Discounted Price. Total Brand Revenue is shown as output using SUM method.

In [22]:
```
ECom_Data['brand_revenue'] = ECom_Data['discounted_price'] - ECom_Data['Revenue']
Total_Brand_Revenue = ECom_Data['brand_revenue'].sum()

print('Total brand revenue:',Total_Brand_Revenue)
```

```
Total brand revenue: 7522992.15
```

Data is grouped on Brand column and Brand Revenue is aggregated using SUM method. Result is sorted on aggregated Brand Revenue in descending order and top 10 rows are shown as output using head method.

In [23]:
```
ECom_Data.groupby('Brand').agg({'brand_revenue':'sum'}).sort_values(by='brand_reven
```

Out[23]:

| Brand | brand_revenue |
|---|---|
| **ALLURE AUTO** | 498464.25 |
| **GAGA** | 237390.00 |
| **SLIM** | 212062.60 |
| **DAILYOBJECTS** | 181980.00 |
| **DIVINITI** | 143115.00 |
| **THELOSTPUPPY** | 127287.50 |
| **REGULAR** | 126536.50 |
| **ENTHOPIA** | 123146.25 |
| **ASUS** | 99241.50 |
| **SPRINGWEL** | 88978.50 |

## Let us now investigate multiple features for each product to determine any pattern.

## Q 7. Compare prices for each product.

### a) Draw boxplots of retail_price & discounted_price.

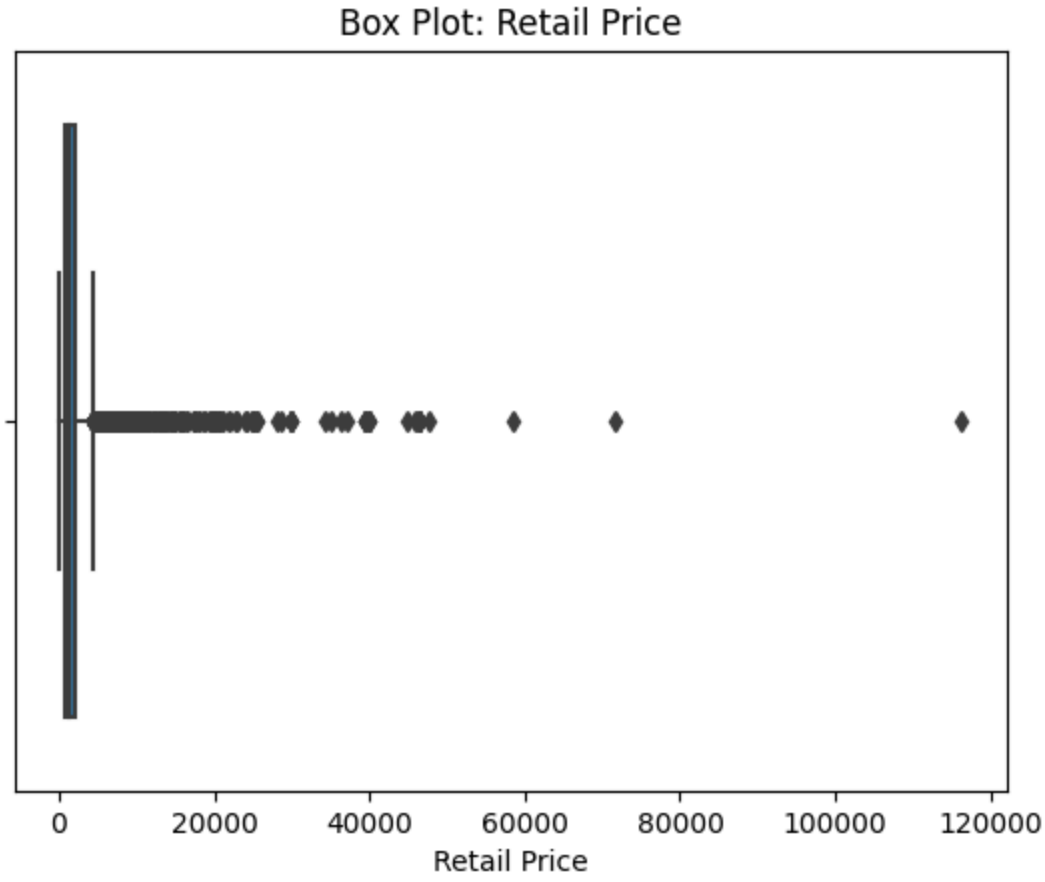
### b) Are there any outliers? (Yes/No)

### c) Create a scatterplot of retail_price (x-axis) and discounted_price (y-axis).

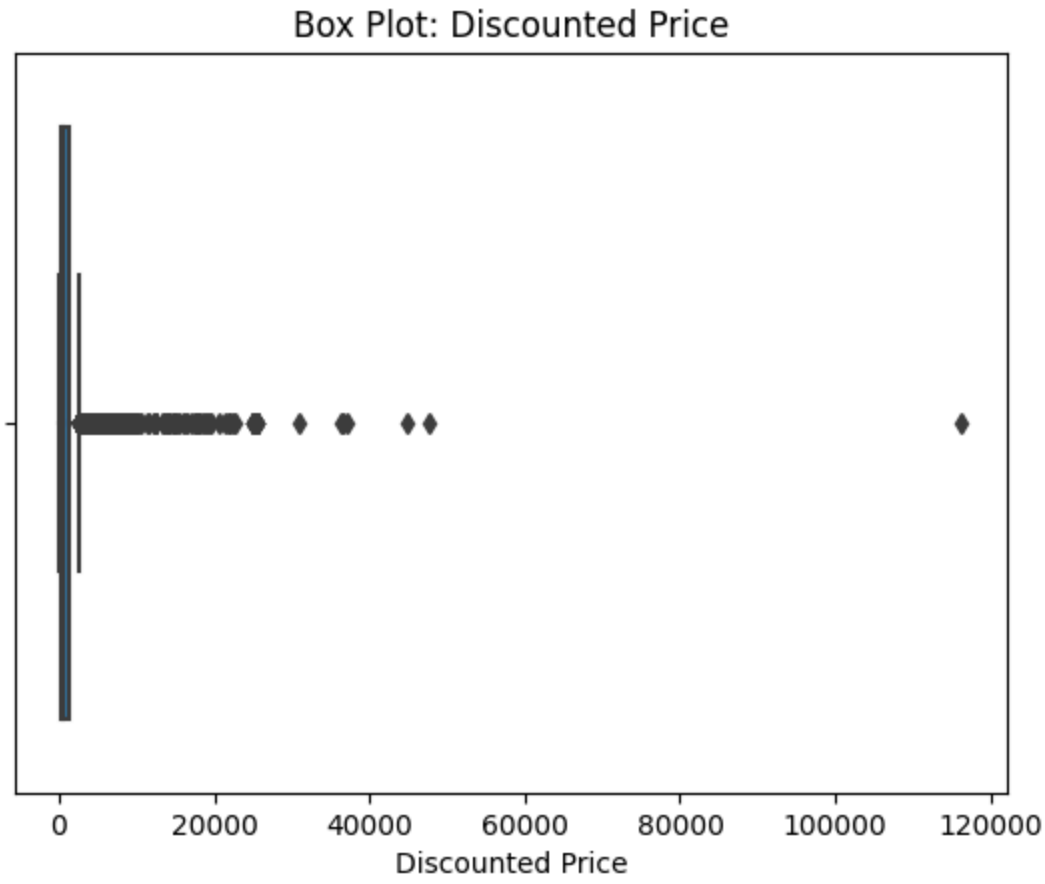
### Ans 7 a)

Box plots provide a quick visual summary of the variability of values in a dataset. They show the median, upper and lower quartiles, minimum and maximum values, and any outliers in the dataset.

In [24]:
```
sns.boxplot(data=ECom_Data, x='retail_price')
plt.title('Box Plot: Retail Price')
plt.xlabel('Retail Price')
plt.show()
```




In [25]:
```
sns.boxplot(data=ECom_Data, x='discounted_price')
plt.title('Box Plot: Discounted Price')
plt.xlabel('Discounted Price')
plt.show()
```




### Ans 7 b)

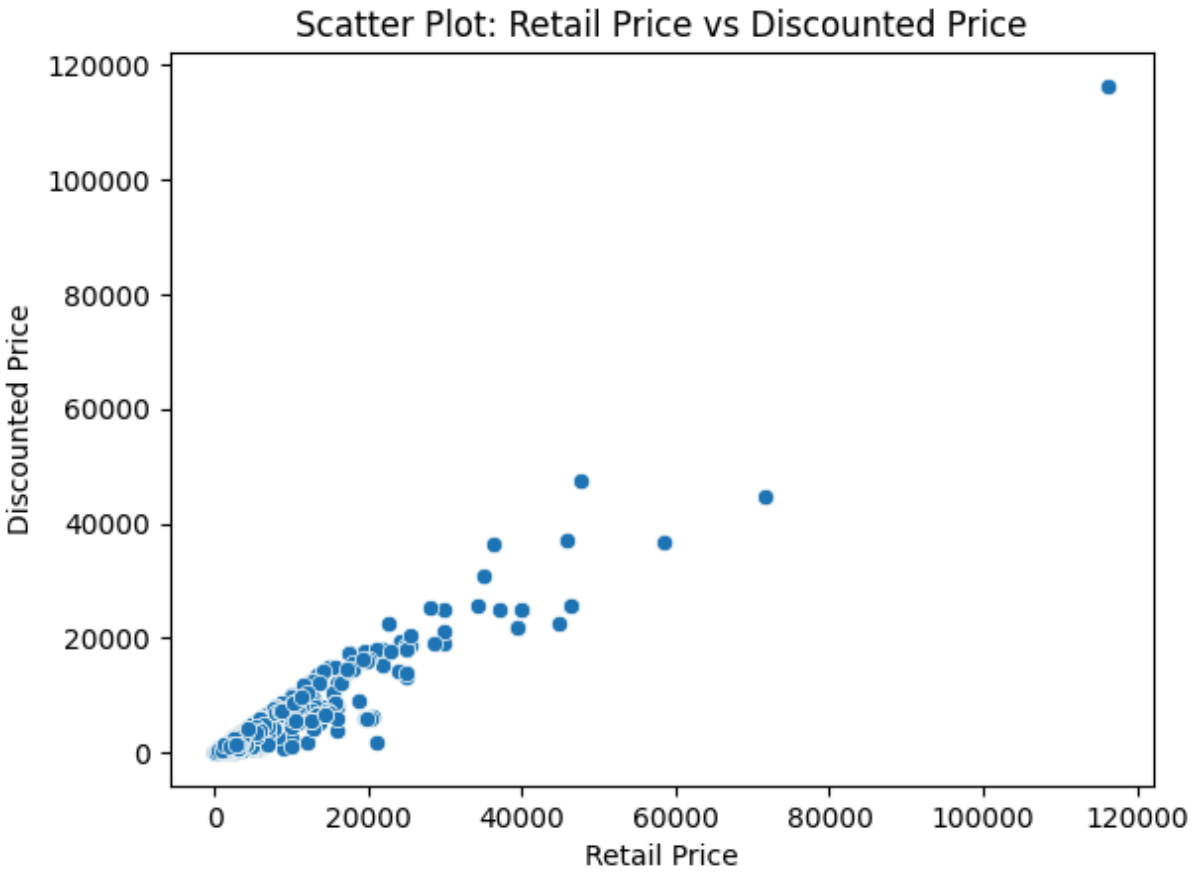
Yes

### Ans 7 c)

A scatter plot is a chart that shows the relationship between two variables in a data set.

The below scatter plot shows positive correlation between Discounted Price and Retail Price.

In [26]:
```
sns.scatterplot(data=ECom_Data, x='retail_price', y='discounted_price')
plt.title('Scatter Plot: Retail Price vs Discounted Price')
plt.xlabel('Retail Price')
plt.ylabel('Discounted Price')
plt.show()
```




## The next steps will enable to study brand-level information.

## Q 8. Create a new dataframe to include the Brand specific information as stated:

### i. total number of orders placed per Brand

### ii. total retail_price per Brand

### iii. total discounted_price per Brand, and

### iv. total BrandRevenue generated per Brand.

### Also, draw a pairplot using these four features.

### Ans 8

DataFrame ECom_Data_Brand is created using concat method to combine Total_Orders, Total_Retail_Price, Total_Discounted_Price and Total_Brand_Revenue datasets.

In [27]:
```
Total_Orders = ECom_Data.groupby('Brand')['Brand'].count()
Total_Retail_Price = ECom_Data.groupby('Brand').agg({'retail_price':'sum'})
Total_Discounted_Price = ECom_Data.groupby('Brand').agg({'discounted_price':'sum'})
Total_Brand_Revenue = ECom_Data.groupby('Brand').agg({'brand_revenue':'sum'})
```

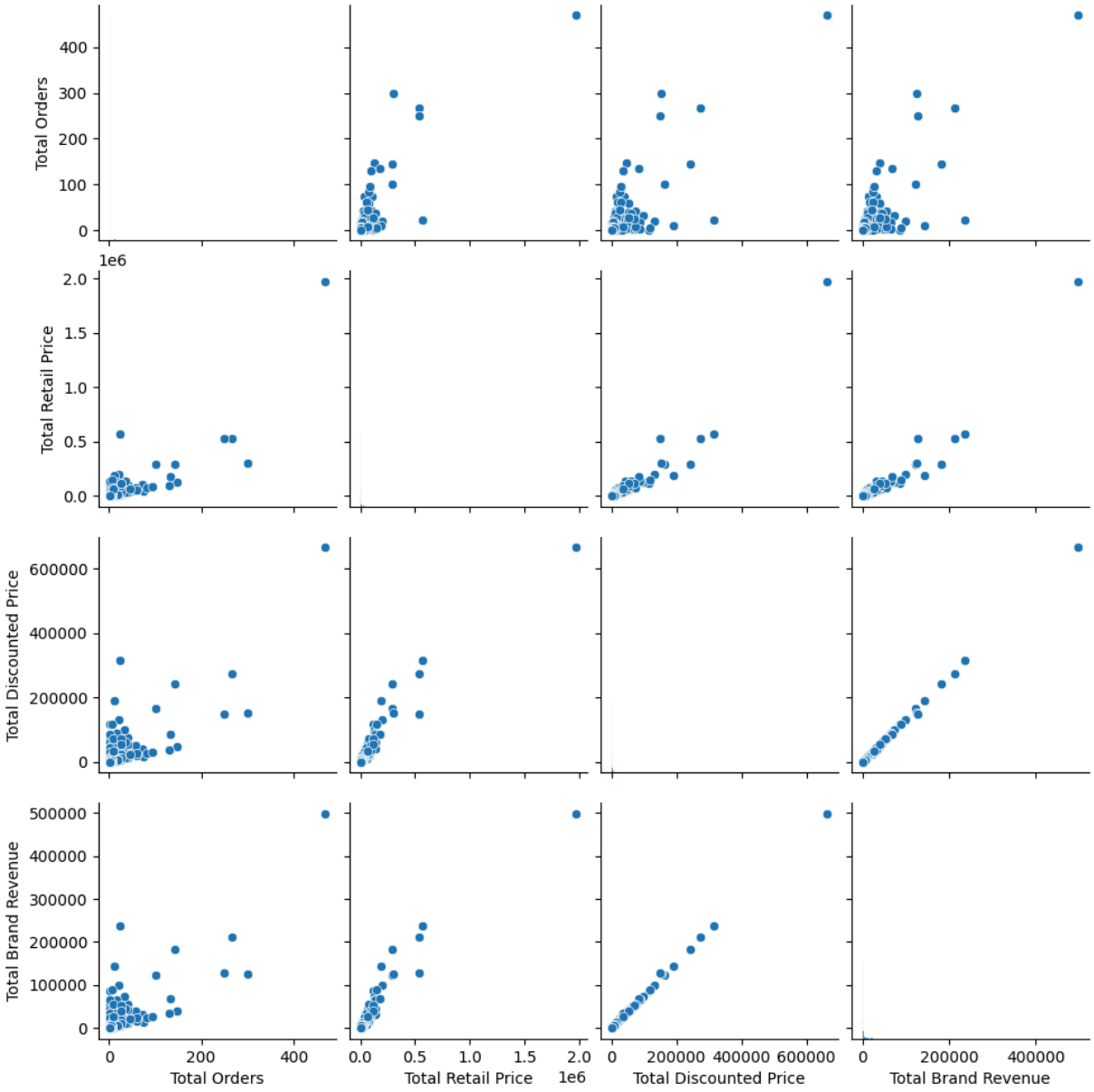
```
ECom_Data_Brand = pd.concat([Total_Orders,Total_Retail_Price,Total_Discounted_Price
ECom_Data_Brand.columns = ['Total Orders','Total Retail Price','Total Discounted Pr
ECom_Data_Brand.head()
```

Out[27]:

| Brand | Total Orders | Total Retail Price | Total Discounted Price | Total Brand Revenue |
|---|---|---|---|---|
| **1OAK** | 2 | 1698 | 1274 | 1015.40 |
| **3A AUTOCARE** | 41 | 107059 | 74134 | 55647.90 |
| **3D MAT** | 1 | 7250 | 6999 | 5249.25 |
| **3KFACTORY** | 1 | 399 | 174 | 156.60 |
| **4D** | 5 | 17500 | 7948 | 5961.00 |

Pair plot pairwise relationships in a dataset. The pairplot creates a grid of Axes such that each variable in data set will be shared in the y-axis across a single row and in the x-axis across a single column.

In [28]:
```
sns.pairplot(data=ECom_Data_Brand, vars=['Total Orders', 'Total Retail Price', 'Tot
plt.show()
```




## The E-Commerce company operate in multiple regions. It is important to understand its performance in each region.

## Q 9. Compare performance regionwise:

### a) Draw a lineplot for the monthly Revenue of E-Commerce Company for each region separately.

### b) Identify the best and the worst performing months for each region.

### Ans 9 a)

Order_Date column data type is changed to datetime64[ns] to extract month and year from it. Month_Year is added as new column in dataset by concatenating the month and year values.

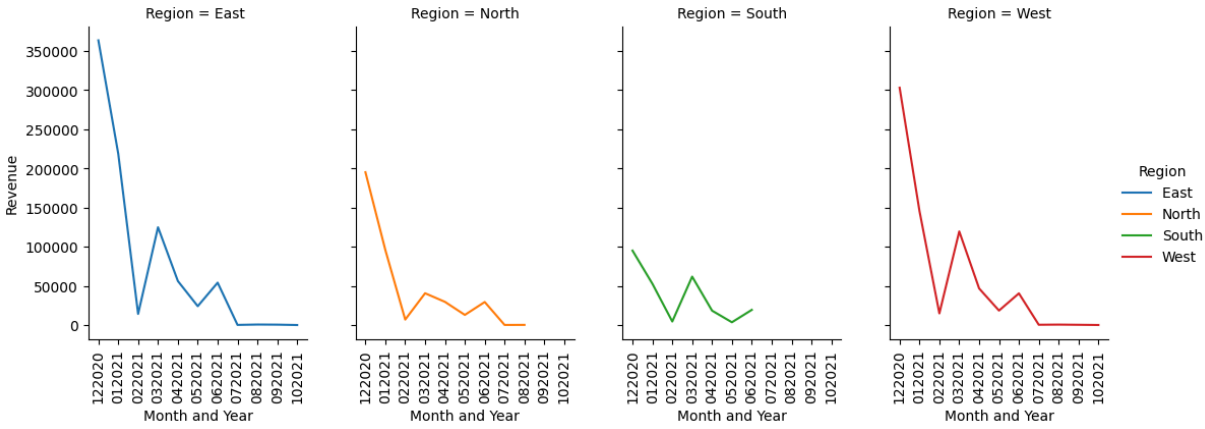
In [29]:
```
ECom_Data['Order_Date'] = ECom_Data['Order_Date'].astype('datetime64[ns]')
ECom_Data['month'] = ECom_Data['Order_Date'].dt.month.map("{:02}".format)
ECom_Data['year'] = ECom_Data['Order_Date'].dt.year
ECom_Data['Month_Year'] = ECom_Data['month'].astype(str) + ECom_Data["year"].astype
```

Group by is performed on Region and Month_Year columns and SUM method is used to aggregate Revenue. ECom_Data_rwr4 dataset is created to keep Month and Year in sorted order by concatenating the ECom_Data_rwr2 and ECom_Data_rwr3 datasets (created using loc method and filter criteria on Month_Year column).

In [30]:
```
ECom_Data_rwr1 = ECom_Data.groupby(['Region','Month_Year']).agg({'Revenue':'sum'}).
ECom_Data_rwr2 = ECom_Data_rwr1.loc[ECom_Data_rwr1['Month_Year'] == '122020']
ECom_Data_rwr3 = ECom_Data_rwr1.loc[ECom_Data_rwr1['Month_Year'] != '122020']
ECom_Data_rwr4 = pd.concat([ECom_Data_rwr2,ECom_Data_rwr3]).reset_index()
```

The Relational Plot (relplot) allows us to visualise how variables within a dataset relate to each other.

In [31]:
```
g = sns.relplot(data=ECom_Data_rwr4, x = 'Month_Year', y = 'Revenue', kind='line',
                height=4, aspect=.7)
g.set_xlabels('Month and Year')
g.set_xticklabels(rotation=90)
plt.show()
```




## Ans 9 b)

East:
Best performing month: Dec 2020
Worst performing month: Oct 2021

North:
Best performing month: Dec 2020
Worst performing month: July 2021

South:
Best performing month: Dec 2020

Worst performing month: May 2021

West:

Best performing month: Dec 2020

Worst performing month: Oct 2021

## Congratulations! We have learnt how to approach a complex data and extract information out of it.