😊 EDA Data Acquisition, Merging, Duplicate Values and Outlier Analysis

1. Now we are going to work on some other data sets. So, from the folder that you downloaded in the last lab, you need to upload three files other than data.csv as you can see below.

😄 Data Acquisition

2. So, here we start by importing some important libraries. Now we are going to load our data.

```
In [2]: # Step 1: Data Acquisition
```

```
In [3]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```

3. Below you can see that our data has been loaded for Cloud Bello customers and local Bello sales.

```
In [4]: cloud_df = pd.read_csv('cloud_bello_customers.csv')
        cloud_df.head()
```

Out[4]:

	CustomerID	ProductType	Rating	Total_Ratings	PaymentMode
0	10471	coates	1 start	8736	Creditcard
1	10472	dresses	2 star	8547	Ewallet
2	10473	NaN	2 star	9833	Prepaid Card
3	10474	dresses	2star	7591	Directdeposit
4	10475	pantas	1 start	5131	NaN

```
In [5]: local_df = pd.read_excel('local_bello_sales.xlsx')
        local_df.head()
```

Out[5]:

	CustomerID	Price
0	10471	₹ 675036.2138626401
1	10472	€ 2,315.53
2	10473	₹ 611426.29122378
3	10474	₹ 210869.48083470002
4	10475	₹ 679422.2811434

4. The third file that we have is a database file for it we need to install the SQL Alchemy library. SQL Alchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

```
In [6]: !pip install SQLAlchemy
```

```
Requirement already satisfied: SQLAlchemy in c:\users\pulkit\anaconda3\lib\site-packages (1.4.39)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\pulkit\anaconda3\lib\site-packages (from SQLAlchemy) (2.0.1)
```

5. So, by running this command we were able to read the database file.

   from sqlalchemy import create_engine
   engine=create_engine('sqlite:///db_bello_customers.db')

```
In [7]: from sqlalchemy import create_engine
        engine=create_engine('sqlite:///db_bello_customers.db')
```

6. To connect with this database file, we need to run this command.

   connect = engine.connect()
   type(connect)

```
In [8]: connect = engine.connect()
        type(connect)
Out[8]: sqlalchemy.engine.base.Connection
```

7. Now to know the table name inside this database we can make use of the command given below.

   from sqlalchemy import inspect
   inspector = inspect(engine)
   print(f"table name is {inspector.get_table_names()}")

```
In [9]: from sqlalchemy import inspect
        inspector = inspect(engine)
        print(f"table name is {inspector.get_table_names()}")

        table name is ['Payments']
```

8. As we have the table name, now we can see the data inside the table using the command below.

   on_premise_database = pd.read_sql_table('Payments',connect)
   on_premise_database.head()

```
In [10]: on_premise_database = pd.read_sql_table('Payments',connect)
         on_premise_database.head()
```

Out[10]:

	Date	CustomerID	PaymentMode	ShippingMode	Gender	ShippingTime
0	16/08/20	10471	Creditcard	Economy	Female	17.0
1	2/8/2020 0:00	10472	Ewallet	Mail	Unspecified	12.0
2	8/8/2020 0:00	10473	Prepaid Card	Mail	None	10.0
3	None	10474	Directdeposit	Normal	Female	2133.0
4	24/08/20	10475	None	Economy	Male	2133.0

9. Now that we have the data from all three files, we are going to merge the data. So, it'll be easier for us to work on single data instead of working on three different data sets.
10. By running these commands, we got the columns of all three tables.

😄 Merging

```
In [11]: ## Merging

In [12]: on_premise_database.columns

Out[12]: Index(['Date', 'CustomerID', 'PaymentMode', 'ShippingMode', 'Gender',
                'ShippingTime'],
               dtype='object')

In [13]: local_df.columns

Out[13]: Index(['CustomerID', 'Price'], dtype='object')

In [14]: cloud_df.columns

Out[14]: Index(['CustomerID', 'ProductType', 'Rating', 'Total_Ratings', 'PaymentMode'], dtype='object')
```

11. Now we will merge all these data frames by considering that they are from a single source. But if you know that the data frames are from different sources then first you need to perform EDA and then move ahead.
12. So, by running the below commands we have the concatenated data frame.

```
In [15]: on_premise_database.drop(columns=['CustomerID', 'PaymentMode'], inplace=True)
         local_df.drop(columns=['CustomerID'], inplace=True)

In [16]: raw_data = pd.concat([on_premise_database,local_df,cloud_df], axis=1)
         raw_data.head()
```

Out[16]:

	Date	ShippingMode	Gender	ShippingTime	Price	CustomerID	ProductType	Rating	Total_Ratings	PaymentMode
0	16/08/20	Economy	Female	17.0	₹ 675036.2138626401	10471	coates	1 start	8736	Creditcard
1	2/8/2020 0:00	Mail	Unspecified	12.0	€ 2,315.53	10472	dresses	2 star	8547	Ewallet
2	8/8/2020 0:00	Mail	None	10.0	₹ 611426.29122378	10473	NaN	2 star	9833	Prepaid Card
3	None	Normal	Female	2133.0	₹ 210869.48083470002	10474	dresses	2star	7591	Directdeposit
4	24/08/20	Economy	Male	2133.0	₹ 679422.2811434	10475	pantas	1 start	5131	NaN

13. Run this command to get the raw data based on your preferred columns as you can see below. We'll be using this data and move forward. So, to save this data we run this command.

```
In [17]: raw_data = raw_data[['Date','CustomerID','ProductType','Rating','Total_Ratings','Gender',
                              'PaymentMode','ShippingMode','ShippingTime','Price']]
         raw_data.head()
```

Out[17]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price
0	16/08/20	10471	coates	1 start	8736	Female	Creditcard	Economy	17.0	₹ 675036.2138626401
1	2/8/2020 0:00	10472	dresses	2 star	8547	Unspecified	Ewallet	Mail	12.0	€ 2,315.53
2	8/8/2020 0:00	10473	NaN	2 star	9833	None	Prepaid Card	Mail	10.0	₹ 611426.29122378
3	None	10474	dresses	2star	7591	Female	Directdeposit	Normal	2133.0	₹ 210869.48083470002
4	24/08/20	10475	pantas	1 start	5131	Male	NaN	Economy	2133.0	₹ 679422.2811434

```
In [18]: raw_data.to_csv('uncleaned_data.csv',index=False)
```

14. And if you go to the EDA Lab folder you will see that you have a new CSV file.

😄 Duplicate Entries

15. The command below will return you the PANDAS series filled with Boolean values.

```
In [19]: # Duplicate Entries

In [20]: raw_data.duplicated()

Out[20]: 0     False
         1     False
         2     False
         3     False
         4     False
               ...
         95    False
         96    False
         97    False
         98    False
         99    False
         Length: 100, dtype: bool
```

16. Now we can use these Boolean values, and mention them inside the indexing operator, and that would help us to return the rows, that have been identified as duplicates.
17. Below you can see that we have the duplicate values from the dataset.

```
In [21]: raw_data[raw_data.duplicated()]
```

Out[21]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price
88	31/08/20	10558	coates	5 star	7007	Unspecified	Credit Card	Normal	11.0	₹ 234895.78095268
89	31/08/20	10558	coates	5 star	7007	Unspecified	Credit Card	Normal	11.0	₹ 234895.78095268

18. If we filter out the by Customer ID we can see that one True value and other as a replica of it.

```
In [22]: raw_data[raw_data['CustomerID'] == 10558]
```

Out[22]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price
87	31/08/20	10558	coates	5 star	7007	Unspecified	Credit Card	Normal	11.0	₹ 234895.78095268
88	31/08/20	10558	coates	5 star	7007	Unspecified	Credit Card	Normal	11.0	₹ 234895.78095268
89	31/08/20	10558	coates	5 star	7007	Unspecified	Credit Card	Normal	11.0	₹ 234895.78095268

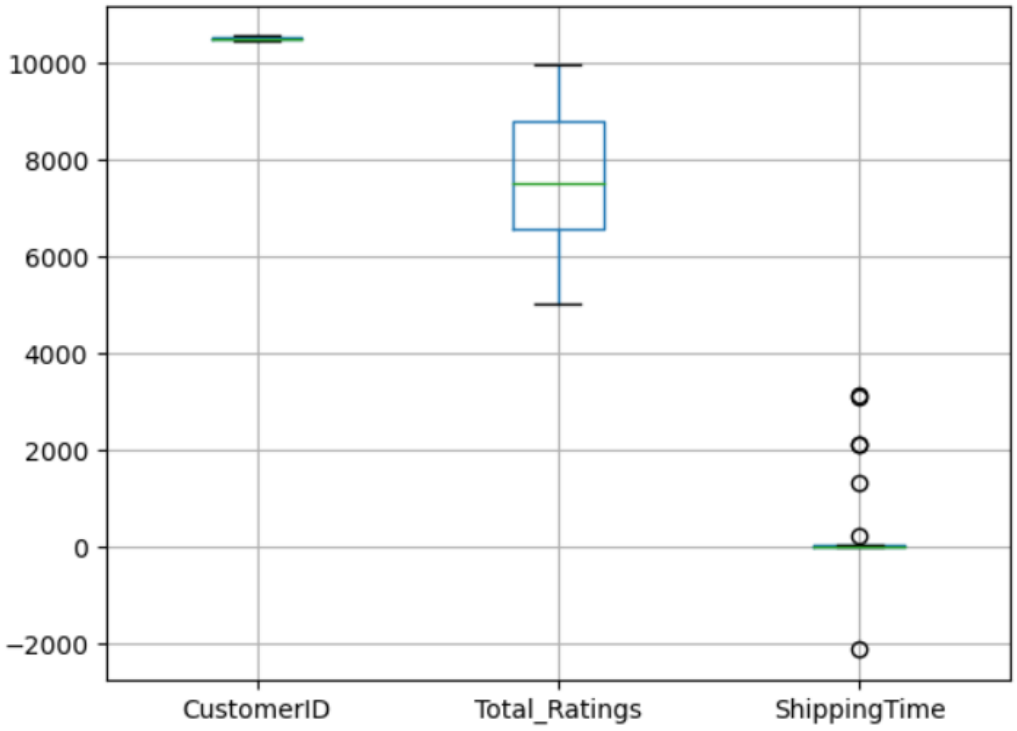
19. By running these, we have dropped the duplicated items from the dataset and we can see that there 0 duplicated items now.

```
In [23]: # drop duplicates
         raw_data.drop_duplicates(inplace=True)

In [24]: raw_data.duplicated().sum()

Out[24]: 0
```

20. Now we will see the analysis of outliers.
21. So, the outlier is someone or some value where we would observe a slightly different value than the existing values.
22. By executing the below command, we can see that on the shipping time, we got the outliers.

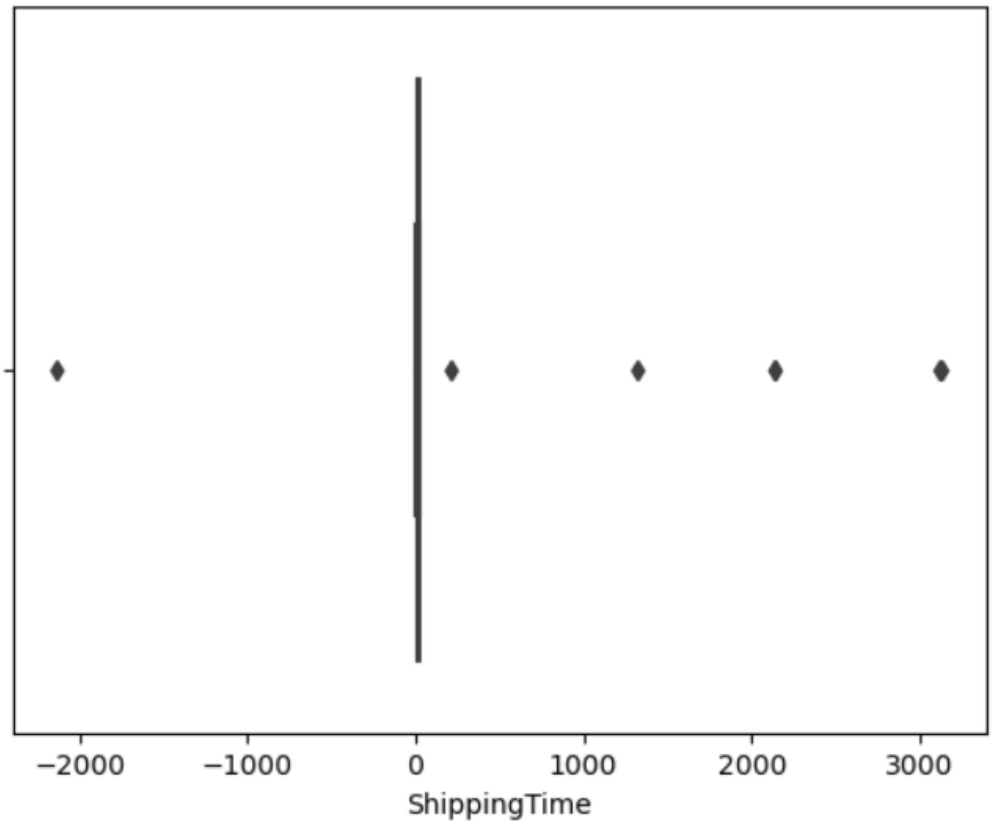
😄 Outlier Analysis

```
In [25]: # Outlier Analysis

In [26]: raw_data.boxplot()
         plt.show()
```



23. Now we created a boxplot just for the outliers and you see them here in the snapshot below.
24. Well, this plot is called a box plot. This is also known as a box and a whisker plot. Now, this box and whisker plot is a graphical method to display the variation in each set of datasets, and that is the usage of this box plot.

```
In [27]: sns.boxplot(raw_data,x='ShippingTime')
         plt.show()
```



25. By using the command below, we get the interquartile value.

- **q1: This computes the first quartile (25th percentile) of the ShippingTime column in the raw_data DataFrame.**
- **q2: This calculates the median (50th percentile) of the ShippingTime.**
- **q3: This finds the third quartile (75th percentile) of the ShippingTime.**

```
In [28]: q1 = raw_data['ShippingTime'].quantile(0.25)
         q2 = raw_data['ShippingTime'].median()
         q3 = raw_data['ShippingTime'].quantile(0.75)
         iqr = q3 - q1
         iqr
Out[28]: 9.0
```

26. Now run the below commands to get the output. And below you can see that IDs which have been identified as the outliers.
    - **These calculations set the thresholds for identifying outliers. Data points below minimum or above maximum are considered outliers. The factor of 1.5 is a common choice in outlier detection.**

- **cond1: This condition identifies data points in ShippingTime that are less than the minimum threshold.**
- **cond2: This condition identifies data points in ShippingTime that are greater than the maximum threshold.**
- **This line uses a logical OR (|) to combine the two conditions. It creates a new DataFrame called outliers, which includes only those rows from raw_data where the ShippingTime is either less than minimum or greater than maximum.**

```
In [29]: q1,q2,q3
Out[29]: (9.0, 13.0, 18.0)

In [30]: minimum = (q1 - (1.5*iqr))
         maximum = (q3 + (1.5*iqr))

In [31]: cond1 = raw_data['ShippingTime'] < minimum
         cond2 = raw_data['ShippingTime'] > maximum
         outliers = raw_data[cond1 | cond2]
         outliers
```

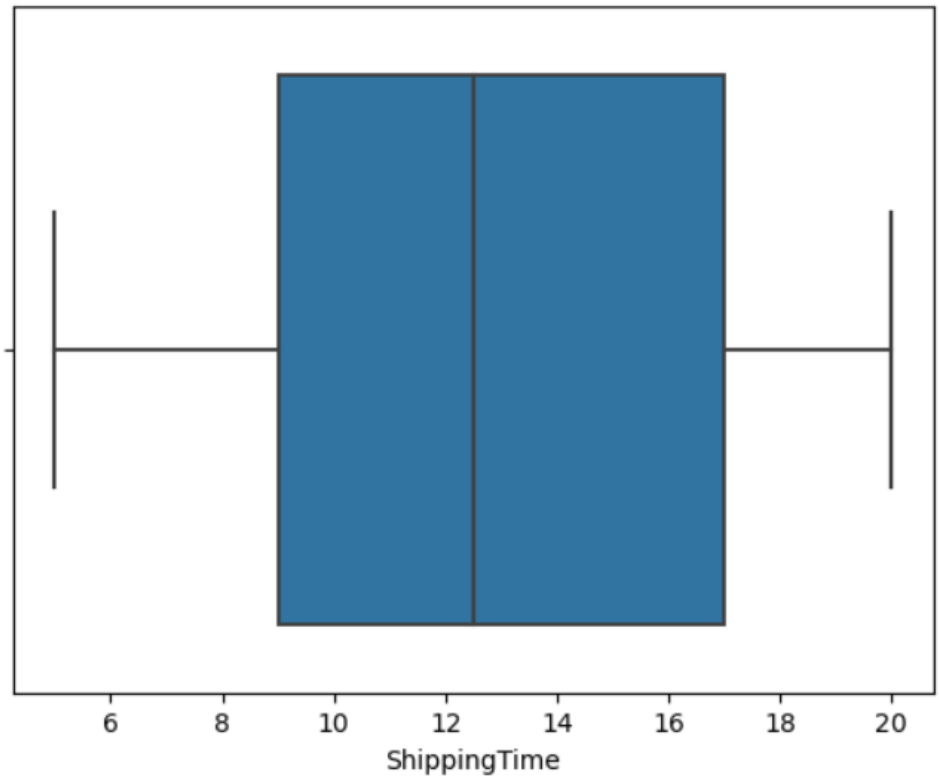
Out[31]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price
3	None	10474	dresses	2star	7591	Female	Directdeposit	Normal	2133.0	₹ 210869.48083470002
4	24/08/20	10475	pantas	1 start	5131	Male	NaN	Economy	2133.0	₹ 679422.2811434
27	19/08/20	10498	Coats/Jackets	3 star	9297	Unspecified	Direct Deposit	Express	213.0	₹ 635118.07344224
43	14/08/20	10514	dresses	5 star	5567	Unspecified	Credit Card	Express	1321.0	€ 7,261.79
59	17/08/20	10530	Cardigan	4 star	6930	Female	Cash	Economy	3113.0	$8,670.69
71	20/08/20	10542	Hats	2star	9157	Male	Cash	Mail	3132.0	₹ 708227.40839738
98	26/08/20	10569	Pants	5 star	5200	Male	Directdeposit	Economy	-2131.0	£ 4118.0053827500005

27. By using the command below first we dropped the outliers and then we ran the boxplot command to see the data and all the outliers have been cleared.

In [32]:
```
raw_data.drop(index=outliers.index, inplace=True) # drop the outliers
```

In [33]:
```
sns.boxplot(raw_data,x='ShippingTime')
plt.show()
```



😄 Missing Values

28. Now we are going to work on the missing values in our dataset for that we use dropping or imputation.
29. So, first we are going to install the missing number library using pip command.

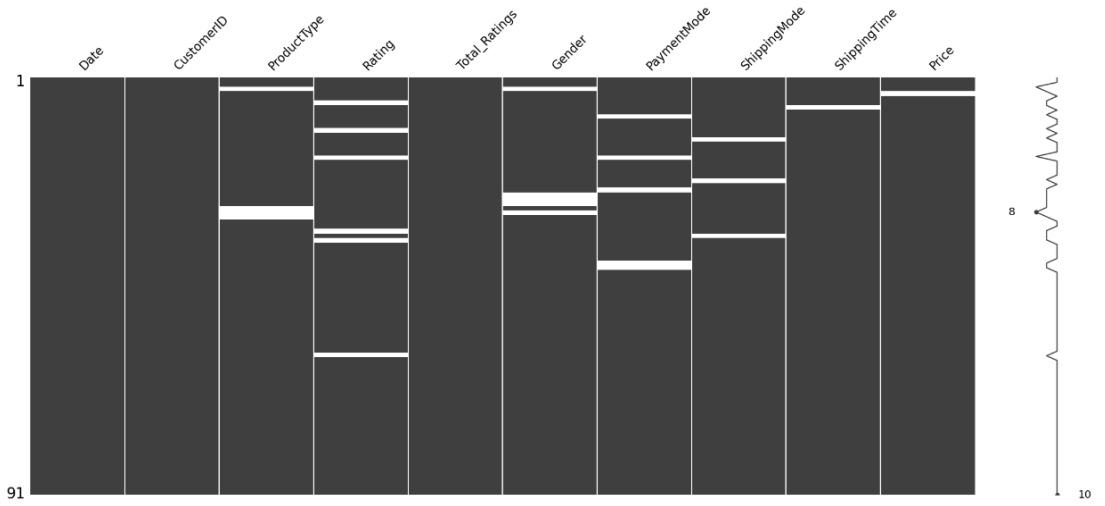
In [34]:
```
# Missing Values
```

In [36]:
```
!pip install missingno
```

```
Collecting missingno
  Obtaining dependency information for missingno from https://files.pythonhosted.org/packages/87/22/cd5cf999af21c2f97486622c551ac3d07361ced8125121e907f588ff5f24/missingno-0.5.2-py3-none-any.whl.metadata
  Downloading missingno-0.5.2-py3-none-any.whl.metadata (639 bytes)
Requirement already satisfied: numpy in c:\users\pulkit\anaconda3\lib\site-packages (from missingno) (1.24.3)
Requirement already satisfied: matplotlib in c:\users\pulkit\anaconda3\lib\site-packages (from missingno) (3.7.2)
Requirement already satisfied: scipy in c:\users\pulkit\anaconda3\lib\site-packages (from missingno) (1.11.1)
Requirement already satisfied: seaborn in c:\users\pulkit\anaconda3\lib\site-packages (from missingno) (0.12.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (10.3.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\pulkit\anaconda3\lib\site-packages (from matplotlib->missingno) (2.8.2)
Requirement already satisfied: pandas>=0.25 in c:\users\pulkit\anaconda3\lib\site-packages (from seaborn->missingno) (2.0.3)
Requirement already satisfied: pytz>=2020.1 in c:\users\pulkit\anaconda3\lib\site-packages (from pandas>=0.25->seaborn->missingno) (2023.3.post1)
```

30. Then we will use the below command to import the missing number library and then have the visual representation of our data.

```
In [37]: import missingno as mn
         mn.matrix(raw_data)
         plt.show()
```



31. Use the below command to look at the missing numbers in the column format.

```
In [38]: raw_data.isnull().sum()

Out[38]: Date            0
         CustomerID      0
         ProductType     4
         Rating          6
         Total_Ratings   0
         Gender          5
         PaymentMode     5
         ShippingMode    3
         ShippingTime    1
         Price           1
         dtype: int64
```

32. Now we will learn about imputation, it is about filling the missing values with a statistically computed value depending on the data type that we are dealing with.
33. So, if we say raw_data.info, this is going to display the information about the data type of the individual column in the data set.

```
In [39]: # imputation
         raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 91 entries, 0 to 99
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Date            91 non-null     object
 1   CustomerID      91 non-null     int64
 2   ProductType     87 non-null     object
 3   Rating          85 non-null     object
 4   Total_Ratings   91 non-null     int64
 5   Gender          86 non-null     object
 6   PaymentMode     86 non-null     object
 7   ShippingMode    88 non-null     object
 8   ShippingTime    90 non-null     float64
 9   Price           90 non-null     object
dtypes: float64(1), int64(2), object(7)
memory usage: 7.8+ KB
```

34. By using the below command, we have the unique values.

```
In [41]: raw_data['Gender'].unique()
```

```
Out[41]: array(['Female', 'Unspecified', None, 'Male'], dtype=object)
```

35. We ran the below command to clear all the missing values but we didn't clear the price one because it is being treated as string and we need to transform it before clearing the missing values.

```
In [42]: raw_data['ProductType'] = raw_data['ProductType'].fillna(raw_data['ProductType'].mode()[0])
         raw_data['Rating'] = raw_data['Rating'].fillna(raw_data['Rating'].mode()[0])
         raw_data['Gender'] = raw_data['Gender'].fillna('Unspecified')
         raw_data['PaymentMode'] = raw_data['PaymentMode'].fillna(raw_data['PaymentMode'].mode()[0])
         raw_data['ShippingMode'] = raw_data['ShippingMode'].fillna(raw_data['ShippingMode'].mode()[0])
         raw_data['ShippingTime'] = raw_data['ShippingTime'].fillna(raw_data['ShippingTime'].median())
```

```
In [43]: raw_data.isnull().sum()
```

```
Out[43]: Date            0
         CustomerID      0
         ProductType     0
         Rating          0
         Total_Ratings   0
         Gender          0
         PaymentMode     0
         ShippingMode    0
         ShippingTime    0
         Price           1
         dtype: int64
```

```
In [44]: raw_data.head()
```

Out[44]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price
0	16/08/20	10471	coates	1 start	8736	Female	Creditcard	Economy	17.0	₹ 675036.2138626401
1	2/8/2020 0:00	10472	dresses	2 star	8547	Unspecified	Ewallet	Mail	12.0	€ 2,315.53
2	8/8/2020 0:00	10473	coates	2 star	9833	Unspecified	Prepaid Card	Mail	10.0	₹ 611426.29122378
5	28/08/20	10476	Coats/Jackets	2 star	5347	Female	Direct Deposit	Mail	20.0	NaN
6	8/8/2020 0:00	10477	coates	5 star	9406	Unspecified	Direct Deposit	Mail	14.0	£ 6610.077565000001

😄 Fixing the Errors

36. So below we have the data in raw and if you look at the unique values in just Product type columns you will see that there are some typos and extra things in this categorical data.

```
In [45]: # Fixing the Errors in the DataSet
```

```
In [46]: raw_data.head()
```

Out[46]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price
0	16/08/20	10471	coates	1 start	8736	Female	Creditcard	Economy	17.0	₹ 675036.2138626401
1	2/8/2020 0:00	10472	dresses	2 star	8547	Unspecified	Ewallet	Mail	12.0	€ 2,315.53
2	8/8/2020 0:00	10473	coates	2 star	9833	Unspecified	Prepaid Card	Mail	10.0	₹ 611426.29122378
5	28/08/20	10476	Coats/Jackets	2 star	5347	Female	Direct Deposit	Mail	20.0	NaN
6	8/8/2020 0:00	10477	coates	5 star	9406	Unspecified	Direct Deposit	Mail	14.0	£ 6610.077565000001

```
In [47]: raw_data['ProductType'].unique()
```

```
Out[47]: array(['coates', 'dresses ', 'Coats/Jackets', 'Jeans', 'pantas', 'Pants',
                'Cardigan', 'Dresses', 'Shirts/Tops', 'Hats'], dtype=object)
```

37. To fix the errors you can use these commands and you can see that in the output 50 all the errors have been fixed.

```
In [48]: def product_type(x):
             if x=='coates':
                 return "Coats/Jackets"
             elif x == 'dresses ':
                 return 'Dresses'
             elif x == 'pantas':
                 return 'Pants'
             else :
                 return x
```

```
In [49]: raw_data['ProductType'] = raw_data['ProductType'].apply(product_type)
```

```
In [50]: raw_data['ProductType'].unique()
```

```
Out[50]: array(['Coats/Jackets', 'Dresses', 'Jeans', 'Pants', 'Cardigan',
                'Shirts/Tops', 'Hats'], dtype=object)
```

38. Now we need to apply the same things on every column.

In [51]:
```
raw_data['Rating'].unique()
```

Out[51]:
```
array(['1 start', '2 star', '5 star', '4 star', '2star', '3 star'],
      dtype=object)
```

In [52]:
```
def rating(x):
    if x=='1 start':
        return '1 star'
    elif x == '2star':
        return '2 star'
    else :
        return x
raw_data['Rating'] = raw_data['Rating'].apply(rating)
raw_data['Rating'].unique()
```

Out[52]:
```
array(['1 star', '2 star', '5 star', '4 star', '3 star'], dtype=object)
```

In [53]:
```
raw_data['PaymentMode'].unique()
```

Out[53]:
```
array(['Creditcard', 'Ewallet', 'Prepaid Card', 'Direct Deposit',
       'Mobile Payment', 'Cash', 'Mobile', 'Credit Card', 'Directdeposit'],
      dtype=object)
```

In [54]:
```
def payment_type(x):
    if x=='Creditcard':
        return 'Credit Card'
    elif x == 'Mobile':
        return 'Mobile Payment'
    elif x == 'Directdeposit':
        return 'Direct Deposit'
    else:
        return x
raw_data['PaymentMode'] = raw_data['PaymentMode'].apply(payment_type)
raw_data['PaymentMode'].unique()
```

Out[54]:
```
array(['Credit Card', 'Ewallet', 'Prepaid Card', 'Direct Deposit',
       'Mobile Payment', 'Cash'], dtype=object)
```

In [55]:
```
raw_data['ShippingMode'].unique()
```

Out[55]:
```
array(['Economy', 'Mail', 'Express', 'Normal'], dtype=object)
```

😄 Data Transformation

39. Now we'll be working on transforming the data. Below you can see that we ran the command to get the information of our dataset. Here you can see that the Data column is of Object data type which is not suitable because whenever we want to work with this column it would be best if it is in Data Time format.

```
In [57]: ## Transformation of Data
```

```
In [58]: raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 91 entries, 0 to 99
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           91 non-null     object
 1   CustomerID     91 non-null     int64
 2   ProductType    91 non-null     object
 3   Rating         91 non-null     object
 4   Total_Ratings  91 non-null     int64
 5   Gender         91 non-null     object
 6   PaymentMode    91 non-null     object
 7   ShippingMode   91 non-null     object
 8   ShippingTime   91 non-null     float64
 9   Price          90 non-null     object
dtypes: float64(1), int64(2), object(7)
memory usage: 7.8+ KB
```

40. Now we will convert this into Data Time format. Using the below commands we have converted it into the suitable format.

```
In [60]: raw_data['Date'] = pd.to_datetime(raw_data['Date'], format='%d/%m/%Y')
```

```
In [61]: raw_data['Date'].head()
```

```
Out[61]: 0   2020-08-16
         1   2020-08-02
         2   2020-08-08
         5   2020-08-28
         6   2020-08-08
         Name: Date, dtype: datetime64[ns]
```

41. If you look at the Price column, you can see that it is also in the string format and we also need to take care of the available currencies; here we will convert the currency into INR.

```
In [62]: raw_data['Price'].head()
```

```
Out[62]: 0    ₹ 675036.2138626401
         1            € 2,315.53
         2      ₹ 611426.29122378
         5                   NaN
         6    £ 6610.077565000001
         Name: Price, dtype: object
```

```
In [63]: #INR
         raw_data['Price'].head(10)
```

```
Out[63]: 0      ₹ 675036.2138626401
         1              € 2,315.53
         2        ₹ 611426.29122378
         5                     NaN
         6      £ 6610.077565000001
         7              $4,287.06
         8           £ 3010.212198
         9     £ 4032878.5368307503
         10             € 3,702.54
         11             € 6,580.20
         Name: Price, dtype: object
```

42. Run the code shown below and you will see that we have created a new column with the name currency description.

    The provided code defines a function currency that checks the Price column in a dataset for specific currency symbols (like $, €, £, ₹) and returns the corresponding currency name (e.g., "American Dollar", "Euro", etc.). The function is applied to each value in the Price column, and the results are stored in a new column called currency_description. Missing or unrecognized currency values are handled by returning an empty string. The updated dataset is then displayed using raw_data.head().

In [64]:
```
def currency(x):
    if pd.isna(x):
        return ''
    elif '$' in x:
        return 'Americal Dollar'
    elif '€' in x:
        return 'Euro'
    elif '£' in x:
        return 'British Pound'
    elif '₹' in x:
        return 'Indian Rupee'
    else:
        return ''
```

In [65]:
```
raw_data['currency_description'] = raw_data['Price'].apply(currency)
raw_data.head()
```

Out[65]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price	currency_description
0	2020-08-16	10471	Coats/Jackets	1 star	8736	Female	Credit Card	Economy	17.0	₹ 675036.2138626401	Indian Rupee
1	2020-08-02	10472	Dresses	2 star	8547	Unspecified	Ewallet	Mail	12.0	€ 2,315.53	Euro
2	2020-08-08	10473	Coats/Jackets	2 star	9833	Unspecified	Prepaid Card	Mail	10.0	₹ 611426.29122378	Indian Rupee
5	2020-08-28	10476	Coats/Jackets	2 star	5347	Female	Direct Deposit	Mail	20.0	NaN	
6	2020-08-08	10477	Coats/Jackets	5 star	9406	Unspecified	Direct Deposit	Mail	14.0	£ 6610.077565000001	British Pound

In [66]:
```
raw_data['currency_description'].unique()
```

Out[66]:
```
array(['Indian Rupee', 'Euro', '', 'British Pound', 'Americal Dollar'],
      dtype=object)
```

43. Here In this code, we are cleaning the Price column by removing currency symbols and commas from the price values. Here's what happens step by step:
    Remove Dollar Sign ($): The line raw_data['Price'] = raw_data['Price'].str.replace('$', "") removes the dollar sign from all price entries in the Price column.
    Remove Pound Sign (£): Similarly, the next line removes the British pound symbol (£).
    Remove Euro Sign (€): The third line removes the euro symbol.
    Remove Rupee Sign (₹): This line removes the Indian rupee symbol.
    Remove Commas (,): The final line removes any commas in the price values, which might be used for thousands of separators (e.g., "1,000" becomes "1000").
44. After this, the Price column will contain clean numeric values, without currency symbols or commas, making it easier to work with the data.

In [67]:
```
c = {'Euro': 92.02,
    'British Pound':109.75 ,
    'Americal Dollar':83.98 }
```

In [68]:
```
raw_data['Price']= raw_data['Price'].str.replace('$',"")
raw_data['Price']= raw_data['Price'].str.replace('£',"")
raw_data['Price']= raw_data['Price'].str.replace('€',"")
raw_data['Price']= raw_data['Price'].str.replace('₹',"")
raw_data['Price']= raw_data['Price'].str.replace(',',"")
```

In [69]:
```
raw_data.head()
```

Out[69]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price	currency_description
0	2020-08-16	10471	Coats/Jackets	1 star	8736	Female	Credit Card	Economy	17.0	675036.2138626401	Indian Rupee
1	2020-08-02	10472	Dresses	2 star	8547	Unspecified	Ewallet	Mail	12.0	2315.53	Euro
2	2020-08-08	10473	Coats/Jackets	2 star	9833	Unspecified	Prepaid Card	Mail	10.0	611426.29122378	Indian Rupee
5	2020-08-28	10476	Coats/Jackets	2 star	5347	Female	Direct Deposit	Mail	20.0	NaN	
6	2020-08-08	10477	Coats/Jackets	5 star	9406	Unspecified	Direct Deposit	Mail	14.0	6610.077565000001	British Pound

45. Below you can see that first we converted the missing values with Zero then we changed the data type into a floating number.

```
In [70]: raw_data['Price'].fillna('0', inplace=True)

In [71]: raw_data['Price'] = raw_data['Price'].astype('float')

In [72]: raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 91 entries, 0 to 99
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   Date                  91 non-null     datetime64[ns]
 1   CustomerID            91 non-null     int64         
 2   ProductType           91 non-null     object        
 3   Rating                91 non-null     object        
 4   Total_Ratings         91 non-null     int64         
 5   Gender                91 non-null     object        
 6   PaymentMode           91 non-null     object        
 7   ShippingMode          91 non-null     object        
 8   ShippingTime          91 non-null     float64       
 9   Price                 91 non-null     float64       
 10  currency_description  91 non-null     object        
dtypes: datetime64[ns](1), float64(2), int64(2), object(6)
memory usage: 8.5+ KB
```

46. We need to check the currency for each row in the dataset, and based on the identified currency, we will append the corresponding value.
47. Here the code converts prices to Indian Rupees using the appropriate exchange rate for each row's currency, and if no currency is found, it keeps the original price.
48. Then we ran the price_inr to check the conversion and you can see that everything has been converted to INR.

In [73]:
```
price_inr = []
for i in range(len(raw_data)):
    if raw_data['currency_description'].iloc[i] in c.keys():
        price_inr.append(raw_data['Price'].iloc[i]*c[raw_data['currency_description'].iloc[i]])
    else:
        price_inr.append(raw_data['Price'].iloc[i])
```

In [74]:
```
price_inr
```

Out[74]:
```
[675036.2138626401,
 213075.0706,
 611426.29122378,
 0.0,
 725456.0127587501,
 360027.29880000005,
 330370.78873050003,
 442608419.4171748,
 340707.73079999996,
 605510.004,
 567447.8212,
 264423.62700000004,
 818226.3778000001,
 240425.0824765625,
 2828677987.442325,
 167219.29640000002,
 160979.788,
 522469.8128,
```

49. Below you can see that we have a new column in which the prices have been converted to INR.

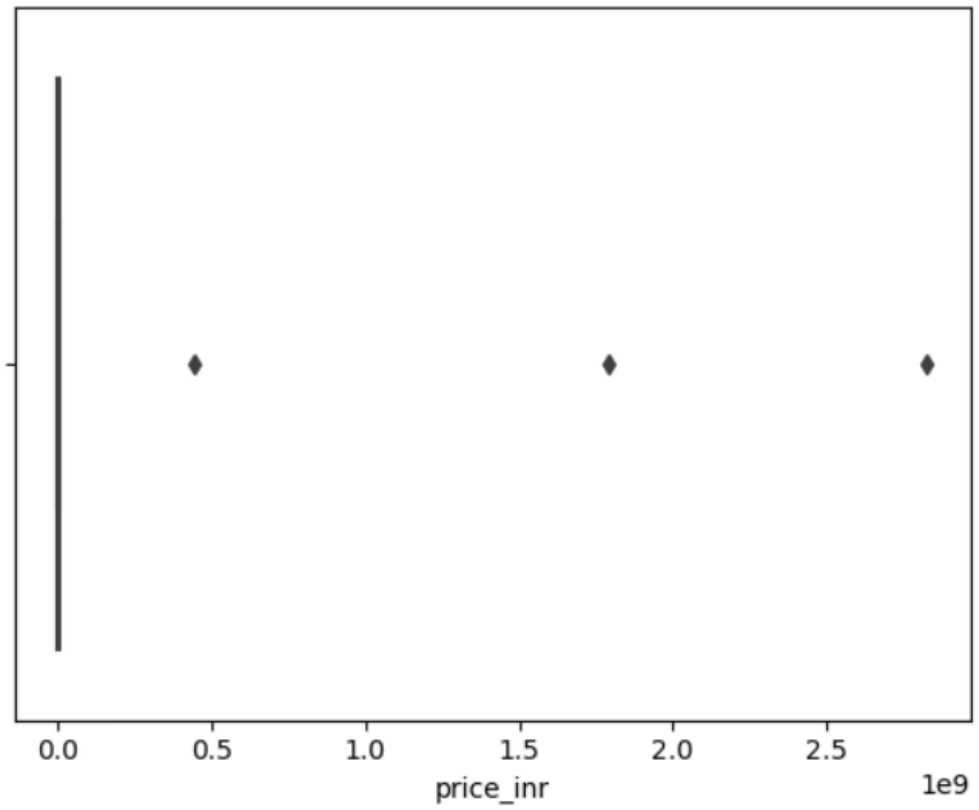
In [75]:
```
raw_data['price_inr'] = price_inr
raw_data.head()
```

Out[75]:

	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price	currency_description	price_inr
	10471	Coats/Jackets	1 star	8736	Female	Credit Card	Economy	17.0	675036.213863	Indian Rupee	675036.213863
	10472	Dresses	2 star	8547	Unspecified	Ewallet	Mail	12.0	2315.530000	Euro	213075.070600
	10473	Coats/Jackets	2 star	9833	Unspecified	Prepaid Card	Mail	10.0	611426.291224	Indian Rupee	611426.291224
	10476	Coats/Jackets	2 star	5347	Female	Direct Deposit	Mail	20.0	0.000000		0.000000
	10477	Coats/Jackets	5 star	9406	Unspecified	Direct Deposit	Mail	14.0	6610.077565	British Pound	725456.012759

50. Now we'll check whether any outliers are present in this data or not. Below you can see that we have some outliers in the given data.

In [76]:
```
sns.boxplot(data=raw_data,x='price_inr')
plt.show()
```



51. So, to remove Outliers we will use the Interquartile range (IQR) and below you can see that we have removed the outliers from our data.

In [77]:
```
q1 = raw_data['price_inr'].quantile(0.25)
q2 = raw_data['price_inr'].quantile(0.50)
q3 = raw_data['price_inr'].quantile(0.75)
iqr = q3-q1
iqr
```

Out[77]: 374927.50366774

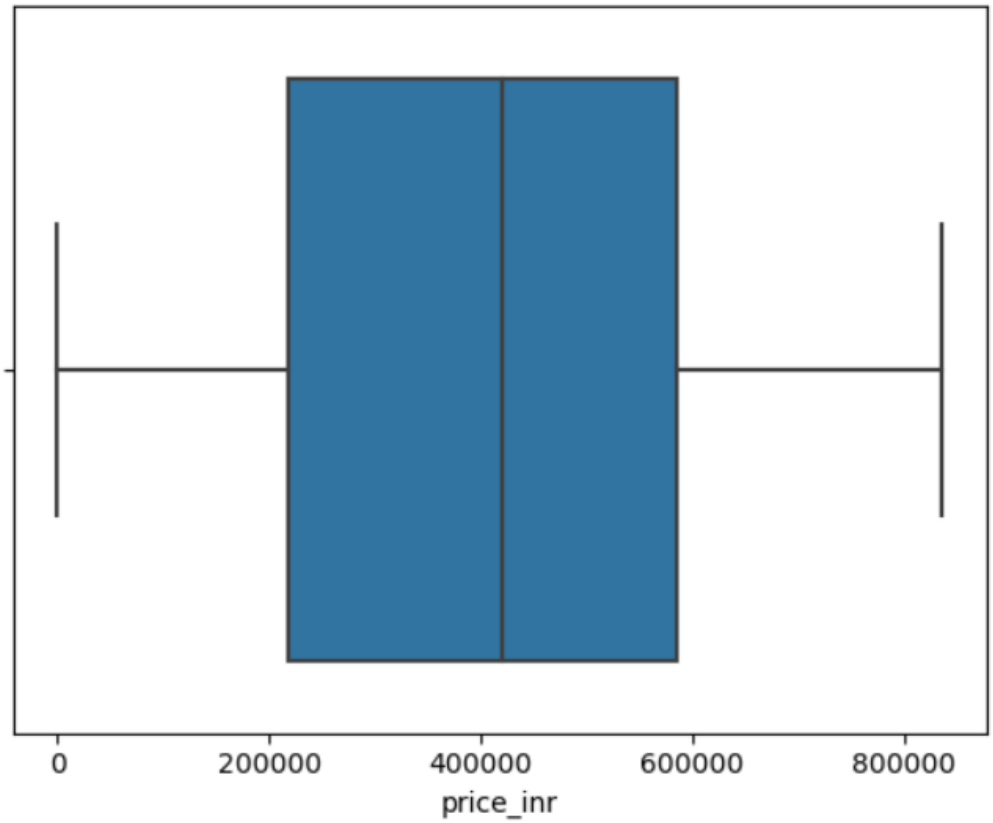
In [78]:
```
minimum = (q1 - (1.5*iqr))
maximum = (q3 + (1.5*iqr))
cond1 = (raw_data['price_inr'] < minimum)
cond2 = (raw_data['price_inr'] > maximum)
outliers = raw_data[cond1|cond2]
outliers
```

Out[78]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingMode	ShippingTime	Price	currency_description	p
9	2020-08-16	10480	Pants	1 star	6914	Female	Prepaid Card	Express	19.0	4.032879e+06	British Pound	4.4260
16	2020-08-17	10487	Pants	2 star	5783	Female	Credit Card	Economy	19.0	2.577383e+07	British Pound	2.8286
29	2020-08-02	10500	Jeans	1 star	9410	Unspecified	Direct Deposit	Mail	5.0	2.131232e+07	Americal Dollar	1.7898

```
In [79]: raw_data.drop(index=outliers.index, inplace=True)
```

```
In [80]: sns.boxplot(data=raw_data,x='price_inr')
         plt.show()
```



😄 Encoding

52. Encoding is nothing but finding a way to represent non-numerical data in a numerical format.
53. The command from sklearn.preprocessing import LabelEncoder imports the LabelEncoder class from the sklearn.preprocessing module in the scikit-learn library.
    What does LabelEncoder do?
    LabelEncoder is used to convert categorical labels (like names or categories) into numerical values (integers). For example, it can transform categories like ['cat', 'dog', 'fish'] into [0, 1, 2], making the data easier to use in machine learning models.
    In summary, this command allows you to access the LabelEncoder tool for converting categorical data into numeric form.
54. Then we get the unique values from the Rating column.

In [81]:
```
# Encoding
```

In [82]:
```
# Dataset Categorical (string)
# Ordinal - Example: Rating of restaurant
# Nominal - Example: Gender
```

In [83]:
```
# Ordinal --> LabelEncoder
from sklearn.preprocessing import LabelEncoder
```

In [84]:
```
raw_data['Rating'].unique()
```

Out[84]:
```
array(['1 star', '2 star', '5 star', '4 star', '3 star'], dtype=object)
```

55. In this code, we use LabelEncoder to convert the values in the Rating column (which are categorical) into numerical values. The transformed ratings are stored in a new column called rating_encoded, so we can see both the original ratings and their corresponding numeric codes side by side.
56. **Okay, so this is a methodology that we use when we have ordinal data.**

In [85]:
```
le = LabelEncoder()
le.fit(raw_data['Rating'])
```

Out[85]:
```
▾ LabelEncoder

LabelEncoder()
```

In [86]:
```
le.classes_
```

Out[86]:
```
array(['1 star', '2 star', '3 star', '4 star', '5 star'], dtype=object)
```

In [87]:
```
raw_data['rating_encoded'] = le.transform(raw_data['Rating'])
raw_data[['Rating','rating_encoded']]
```

Out[87]:

	Rating	rating_encoded
0	1 star	0
1	2 star	1
2	2 star	1
5	2 star	1
6	5 star	4
...	...	...
94	1 star	0
95	2 star	1
96	5 star	4
97	3 star	2
99	1 star	0

88 rows × 2 columns

57. Well, when we have nominal data at that time, we make use of one hot encoder.
58. First, we start by importing one hot encoder then we apply it on the Shipping mode and get the categories.

In [88]:
```
#One Hot Encoder
from sklearn.preprocessing import OneHotEncoder
```

In [89]:
```
ohe = OneHotEncoder()
```

In [90]:
```
raw_data['ShippingMode'].unique()
```

Out[90]:
```
array(['Economy', 'Mail', 'Express', 'Normal'], dtype=object)
```

In [91]:
```
ohe.fit(raw_data["ShippingMode"].values.reshape(-1,1))
```

Out[91]:
```
OneHotEncoder
OneHotEncoder()
```

In [92]:
```
ohe.categories_
```

Out[92]:
```
[array(['Economy', 'Express', 'Mail', 'Normal'], dtype=object)]
```

59. In this code, we use one-hot encoding to convert the categorical values in the ShippingMode column into a numerical format, where each unique category is represented as a separate column. The resulting encoded data is then stored in a new DataFrame (df1), making it easier to analyze and use in machine learning models.

```
In [93]: r = ohe.transform(raw_data['ShippingMode'].values.reshape(-1,1))
         df1 = pd.DataFrame(r.todense(), columns=ohe.categories_)
         df1
```

Out[93]:

	Economy	Express	Mail	Normal
0	1.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	0.0
...	...	...	...	...
83	0.0	0.0	1.0	0.0
84	0.0	1.0	0.0	0.0
85	0.0	0.0	0.0	1.0
86	0.0	0.0	0.0	1.0
87	0.0	0.0	1.0	0.0

88 rows × 4 columns

60. The first command displays the first few entries in the ShippingMode column of the raw_data DataFrame, allowing you to see the different shipping methods. The second command converts the ShippingMode column into multiple columns of binary values (0s and 1s) for each unique shipping method, making it easier to use in data analysis or machine learning models.

```
In [192]: raw_data['ShippingMode'].head()
```

```
Out[192]: 0    Economy
          1       Mail
          2       Mail
          5       Mail
          6       Mail
          Name: ShippingMode, dtype: object
```

```
In [193]: raw_data = pd.get_dummies(raw_data, columns=['ShippingMode'])
```

😄 Scaling of Numerical Values

61. The **Standard Scaler** transforms data into a format where the average (mean) is zero and the standard deviation is one, making it easier to compare different datasets. **Normalizer** adjusts the data to have a length of one, while the **Min-Max Scaler** scales the data to fit within a specific range, typically between 0 and 1, which helps in bringing all features to a similar scale.
62. In this code, we import the StandardScaler from the scikit-learn library to standardize the price_inr data. We then create an instance of StandardScaler and fit it to the price_inr values, which prepares the scaler to transform these prices so that they have a mean of zero and a standard deviation of one.

```
In [194]: # Scaling of Numerical Values
```

```
In [195]: # Strandard Scaler --> transform the value into Z score
          # It will result in a column that has Zero mean and Std dev = 1
          # Normalizer
          # Min Max Scaler
```

```
In [196]: from sklearn.preprocessing import StandardScaler
          ss = StandardScaler()
```

```
In [197]: raw_data.columns
```

```
Out[197]: Index(['Date', 'CustomerID', 'ProductType', 'Rating', 'Total_Ratings',
                 'Gender', 'PaymentMode', 'ShippingTime', 'Price',
                 'currency_description', 'price_inr', 'rating_encoded',
                 'ShippingMode_Economy', 'ShippingMode_Express', 'ShippingMode_Mail',
                 'ShippingMode_Normal'],
                dtype='object')
```

```
In [198]: ss.fit(raw_data['price_inr'].values.reshape(-1,1))
```

```
Out[198]: ▾ StandardScaler
          StandardScaler()
```

63. In this code, ss.mean_ and ss.var_ retrieve the mean and variance of the price_inr data after fitting the StandardScaler. Then, we use ss.transform() to scale the price_inr values, creating a new column called price_scaled in the raw_data DataFrame that contains the standardized prices, making them easier to compare across different datasets.

```
In [199]: ss.mean_ # mean
Out[199]: array([409661.24524315])

In [200]: ss.var_ # variance
Out[200]: array([4.30714871e+10])

In [201]: raw_data['price_scaled'] = ss.transform(raw_data['price_inr'].values.reshape(-1,1))
          raw_data.head()
```

Out[201]:

	Date	CustomerID	ProductType	Rating	Total_Ratings	Gender	PaymentMode	ShippingTime	Price	currency_description	price_inr	rating
0	2020-08-16	10471	Coats/Jackets	1 star	8736	Female	Credit Card	17.0	675036.213863	Indian Rupee	675036.213863	
1	2020-08-02	10472	Dresses	2 star	8547	Unspecified	Ewallet	12.0	2315.530000	Euro	213075.070600	
2	2020-08-08	10473	Coats/Jackets	2 star	9833	Unspecified	Prepaid Card	10.0	611426.291224	Indian Rupee	611426.291224	
5	2020-08-28	10476	Coats/Jackets	2 star	5347	Female	Direct Deposit	20.0	0.000000		0.000000	
6	2020-08-08	10477	Coats/Jackets	5 star	9406	Unspecified	Direct Deposit	14.0	6610.077565	British Pound	725456.012759	

64. The command raw_data[['price_scaled','price_inr']].describe() generates a summary of statistics for the price_scaled and price_inr columns in the raw_data DataFrame. This summary includes key metrics such as the count, mean, standard deviation, minimum, maximum, and quartiles, helping you understand the distribution and characteristics of these two sets of price data.

```
In [202]: raw_data[['price_scaled','price_inr']].describe()
```

Out[202]:

	price_scaled	price_inr
count	8.800000e+01	88.000000
mean	-6.055762e-17	409661.245243
std	1.005731e+00	208726.044513
min	-1.973922e+00	0.000000
25%	-9.254985e-01	217586.322458
50%	4.994365e-02	420026.385771
75%	8.449975e-01	585029.244150
max	2.057392e+00	836645.711200

65. In this code, we import the MinMaxScaler to scale the price_inr values to a specified range between 1 and 2, ensuring that the smallest value is 1 and the largest is 2. We fit the scaler to the price_inr data, then apply it to create a new column called price_minmax_scaled, which contains the scaled prices, and finally, we summarize the statistics for the price_scaled, price_inr, and price_minmax_scaled columns to understand their distributions.
66. And in the end we save data and name the file as final data.csv.

In [203]:
```
# Zscore = StandardScaler
```

In [204]:
```
from sklearn.preprocessing import MinMaxScaler
minmax_scale = MinMaxScaler(feature_range=(1, 2),clip=True)
minmax_scale.fit(raw_data['price_inr'].values.reshape(-1,1),)
raw_data['price_minmax_scaled'] = minmax_scale.transform(raw_data['price_inr'].values.reshape(-1,1))
raw_data[['price_scaled','price_inr', 'price_minmax_scaled']].describe()
```

Out[204]:

	price_scaled	price_inr	price_minmax_scaled
count	8.800000e+01	88.000000	88.000000
mean	-6.055762e-17	409661.245243	1.489647
std	1.005731e+00	208726.044513	0.249480
min	-1.973922e+00	0.000000	1.000000
25%	-9.254985e-01	217586.322458	1.260070
50%	4.994365e-02	420026.385771	1.502036
75%	8.449975e-01	585029.244150	1.699256
max	2.057392e+00	836645.711200	2.000000

In [205]:
```
raw_data.to_csv('final_data.csv',index=False)
```