DOCUMENTATION OF AMAZON SALES REPORT

EDA – Exploratory Data Analysis

(Pre-Machine Learning)

The four main steps in EDA are:

1. Data Processing
2. Data Cleaning
3. Data Visualization
4. Conclusion

STEP 1 :

This code snippet imports necessary Python libraries for data analysis and visualization.

- **import numpy as np:** Imports the NumPy library, which provides support for working with arrays and matrices of numerical data.
- **import pandas as pd:** Imports the Pandas library, used for data manipulation and analysis, particularly with data structures like DataFrames.
- **import matplotlib.pyplot as plt:** Imports the Matplotlib library's pyplot module, which is widely used for creating visualizations like charts, graphs, and plots.
- **%matplotlib inline:** This is a magic command used in Jupyter Notebook environments to display Matplotlib plots directly in the notebook interface.
- **import seaborn as sns:** Imports the Seaborn library, which is built on top of Matplotlib and provides additional functionality for creating aesthetically pleasing statistical visualizations.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
```

STEP 2:

This command reads a CSV file named 'Amazon Sale Report (1).csv' located at the specified file path ('/Users/abbaskashim/Downloads/') using the Pandas library. The data from the CSV file is loaded into a DataFrame, which is a tabular data structure commonly used in data analysis. The variable name 'df' is assigned to this DataFrame. Lastly, by typing 'df', the code is likely intended to display the contents of the DataFrame in the interactive environment, showing the data and its structure to the user.

In [2]:
```
df= pd.read_csv('/Users/abbaskashim/Downloads/Amazon Sale Report (1).csv')
df
```

Out[2]:

	index	Order ID	Date	Status	Fulfilment	Sales Channel	ship-service-level	Category	Size	Courier Status	...	currency	Amount	ship-city	ship-state	p
0	0	405-8078784-5731545	04-30-22	Cancelled	Merchant	Amazon.in	Standard	T-shirt	S	On the Way	...	INR	647.62	MUMBAI	MAHARASHTRA	40
1	1	171-9198151-1101146	04-30-22	Shipped - Delivered to Buyer	Merchant	Amazon.in	Standard	Shirt	3XL	Shipped	...	INR	406.00	BENGALURU	KARNATAKA	56
2	2	404-0687676-7273146	04-30-22	Shipped	Amazon	Amazon.in	Expedited	Shirt	XL	Shipped	...	INR	329.00	NAVI MUMBAI	MAHARASHTRA	41
3	3	403-9615377-8133951	04-30-22	Cancelled	Merchant	Amazon.in	Standard	Blazzer	L	On the Way	...	INR	753.33	PUDUCHERRY	PUDUCHERRY	60
4	4	407-1069790-7240320	04-30-22	Shipped	Amazon	Amazon.in	Expedited	Trousers	3XL	Shipped	...	INR	574.00	CHENNAI	TAMIL NADU	60
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
128971	128970	406-6001380-7673107	05-31-22	Shipped	Amazon	Amazon.in	Expedited	Shirt	XL	Shipped	...	INR	517.00	HYDERABAD	TELANGANA	50
128972	128971	402-9551604-7544318	05-31-22	Shipped	Amazon	Amazon.in	Expedited	T-shirt	M	Shipped	...	INR	999.00	GURUGRAM	HARYANA	12
128973	128972	407-9547469-3152358	05-31-22	Shipped	Amazon	Amazon.in	Expedited	Blazzer	XXL	Shipped	...	INR	690.00	HYDERABAD	TELANGANA	50
128974	128973	402-6184140-0545956	05-31-22	Shipped	Amazon	Amazon.in	Expedited	T-shirt	XS	Shipped	...	INR	1199.00	Halol	Gujarat	38
128975	128974	408-7436540-8728312	05-31-22	Shipped	Amazon	Amazon.in	Expedited	T-shirt	S	Shipped	...	INR	696.00	Raipur	CHHATTISGARH	49

128976 rows × 21 columns

STEP 3:

The "df.head()" command may be used in a programming environment, especially in Python with the Pandas library. It displays the first few lines of a DataFrame (tabulated data structure) to give a quick overview of what's inside and helps us understand the basic purposes of structure and data.

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

Out[3]:

	index	Order ID	Date	Status	Fulfilment	Sales Channel	ship-service-level	Category	Size	Courier Status	...	currency	Amount	ship-city	ship-state	ship-postal-code
0	0	405-8078784-5731545	04-30-22	Cancelled	Merchant	Amazon.in	Standard	T-shirt	S	On the Way	...	INR	647.62	MUMBAI	MAHARASHTRA	400081.0
1	1	171-9198151-1101146	04-30-22	Shipped - Delivered to Buyer	Merchant	Amazon.in	Standard	Shirt	3XL	Shipped	...	INR	406.00	BENGALURU	KARNATAKA	560085.0
2	2	404-0687676-7273146	04-30-22	Shipped	Amazon	Amazon.in	Expedited	Shirt	XL	Shipped	...	INR	329.00	NAVI MUMBAI	MAHARASHTRA	410210.0
3	3	403-9615377-8133951	04-30-22	Cancelled	Merchant	Amazon.in	Standard	Blazzer	L	On the Way	...	INR	753.33	PUDUCHERRY	PUDUCHERRY	605008.0
4	4	407-1069790-7240320	04-30-22	Shipped	Amazon	Amazon.in	Expedited	Trousers	3XL	Shipped	...	INR	574.00	CHENNAI	TAMIL NADU	600073.0

5 rows × 21 columns

STEP 4:

The "df.tail()" command is used to display the last line of a DataFrame in the Pandas library in Python, so that you can look at the end of the data structure and understand its final value and structure.

In [4]:
```
df.tail()
```

Out[4]:

	index	Order ID	Date	Status	Fulfilment	Sales Channel	ship-service-level	Category	Size	Courier Status	...	currency	Amount	ship-city	ship-state	sh post cc
128971	128970	406-6001380-7673107	05-31-22	Shipped	Amazon	Amazon.in	Expedited	Shirt	XL	Shipped	...	INR	517.0	HYDERABAD	TELANGANA	50001
128972	128971	402-9551604-7544318	05-31-22	Shipped	Amazon	Amazon.in	Expedited	T-shirt	M	Shipped	...	INR	999.0	GURUGRAM	HARYANA	12200
128973	128972	407-9547469-3152358	05-31-22	Shipped	Amazon	Amazon.in	Expedited	Blazzer	XXL	Shipped	...	INR	690.0	HYDERABAD	TELANGANA	50004
128974	128973	402-6184140-0545956	05-31-22	Shipped	Amazon	Amazon.in	Expedited	T-shirt	XS	Shipped	...	INR	1199.0	Halol	Gujarat	38935
128975	128974	408-7436540-8728312	05-31-22	Shipped	Amazon	Amazon.in	Expedited	T-shirt	S	Shipped	...	INR	696.0	Raipur	CHHATTISGARH	49201

5 rows × 21 columns

STEP 5:

The "df.info()" command is used in Python and the Pandas library to get a summary of the DataFrame structure. It provides information about column data types, non-null values, and memory usage, which helps in data analysis and quality assessment.

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128976 entries, 0 to 128975
Data columns (total 21 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   index               128976 non-null  int64  
 1   Order ID            128976 non-null  object 
 2   Date                128976 non-null  object 
 3   Status              128976 non-null  object 
 4   Fulfilment          128976 non-null  object 
 5   Sales Channel       128976 non-null  object 
 6   ship-service-level  128976 non-null  object 
 7   Category            128976 non-null  object 
 8   Size                128976 non-null  object 
 9   Courier Status      128976 non-null  object 
 10  Qty                 128976 non-null  int64  
 11  currency            121176 non-null  object 
 12  Amount              121176 non-null  float64
 13  ship-city           128941 non-null  object 
 14  ship-state          128941 non-null  object 
 15  ship-postal-code    128941 non-null  float64
 16  ship-country        128941 non-null  object 
 17  B2B                 128976 non-null  bool   
 18  fulfilled-by        39263 non-null   object 
 19  New                 0 non-null       float64
 20  PendingS            0 non-null       float64
dtypes: bool(1), float64(4), int64(2), object(14)
memory usage: 19.8+ MB
```

STEP 6:

The provided code uses the drop function from the pandas library to extract the columns named 'New' and 'PendingS' from the DataFrame ('df'). The parameter axis=1 indicates that the operation is column-wise, and replaces the DataFrame instead of inplace=True. This action successfully truncates the specified column from the DataFrame.

```
In [6]: df.drop(['New','PendingS'],axis=1,inplace=True)
```

```
In [7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128976 entries, 0 to 128975
Data columns (total 19 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   index               128976 non-null  int64
 1   Order ID            128976 non-null  object
 2   Date                128976 non-null  object
 3   Status              128976 non-null  object
 4   Fulfilment          128976 non-null  object
 5   Sales Channel       128976 non-null  object
 6   ship-service-level  128976 non-null  object
 7   Category            128976 non-null  object
 8   Size                128976 non-null  object
 9   Courier Status      128976 non-null  object
 10  Qty                 128976 non-null  int64
 11  currency            121176 non-null  object
 12  Amount              121176 non-null  float64
 13  ship-city           128941 non-null  object
 14  ship-state          128941 non-null  object
 15  ship-postal-code    128941 non-null  float64
 16  ship-country        128941 non-null  object
 17  B2B                 128976 non-null  bool
 18  fulfilled-by        39263 non-null   object
dtypes: bool(1), float64(2), int64(2), object(14)
memory usage: 17.8+ MB
```

STEP 7:

The code df.isnull().sum() first determines the number of missing (null) values in each DataFrame 'df' column. This function helps to understand how many values are missing in each column, which can be important for data pre-processing and analysis. The output will show the number of null values in each column of the DataFrame.

```
In [8]: df.isnull().sum()
```

```
Out[8]: index                     0
        Order ID                  0
        Date                      0
        Status                    0
        Fulfilment                0
        Sales Channel             0
        ship-service-level        0
        Category                  0
        Size                      0
        Courier Status            0
        Qty                       0
        currency               7800
        Amount                 7800
        ship-city                35
        ship-state               35
        ship-postal-code         35
        ship-country             35
        B2B                       0
        fulfilled-by          89713
        dtype: int64
```

STEP 8:

The Code df['currency'].unique() Retrieves the unique value of the DataFrame 'df' 'currency' column. This routine provides a list of all unique currency values in that particular column. This is useful for understanding the underlying meanings of currencies in a dataset and can help classify, filter, or analyze data based on currencies.

```
In [9]: df['currency'].unique()
Out[9]: array(['INR', nan], dtype=object)
```

STEP 9:

This line of code uses the mode (maximum value) in that column to fill in the missing values in the 'Currency' column. This is a common method of assigning the most common value to missing categorical values, since it helps to preserve the distribution and quality of the data.

```
In [10]: df['currency']=df['currency'].fillna(df['currency']).mode()[0]

In [11]: df.isnull().sum()
Out[11]: index                     0
         Order ID                  0
         Date                      0
         Status                    0
         Fulfilment                0
         Sales Channel             0
         ship-service-level        0
         Category                  0
         Size                      0
         Courier Status            0
         Qty                       0
         currency                  0
         Amount                 7800
         ship-city                35
         ship-state               35
         ship-postal-code         35
         ship-country             35
         B2B                       0
         fulfilled-by          89713
         dtype: int64
```

STEP 10:

The Code df['Amount'].unique() Retrieves the unique value of the 'Amount' column of the DataFrame 'df'. This routine provides all the unique characters in the 'Amount' column. This is useful for understanding the different quantities in the data set and can help to analyze or further process data based on different quantities.

```
In [12]: df['Amount'].unique()
Out[12]: array([ 647.62,  406.  ,  329.  , ...,  708.58, 1244.  ,  639.  ])
```

STEP 11:

Code df['Amount'].value_counts() Counts and displays the number of times each unique value is in the 'Amount' column of DataFrame 'df'. This routine counts the number of times each unique value appears in a column. It is useful to gain insight into how values are distributed in a column and to see how many are common and which are rare.

```
In [13]: df['Amount'].value_counts()

Out[13]: 399.00    5439
         771.00    2796
         735.00    2436
         0.00      2343
         487.00    2295
                   ...
         558.10       1
         551.42       1
         973.00       1
         727.60       1
         639.00       1
         Name: Amount, Length: 1408, dtype: int64
```

STEP 12:

This line of code fills in the missing value in the ‘Amount’ column with the average value of that column (on average). This method is commonly used for statistical data imputation, as it helps to maintain the overall distribution of the data and the presence of central tendency.

```
In [14]: df['Amount']=df['Amount'].fillna(df['Amount']).mean()

In [15]: df.isnull().sum()

Out[15]: index                     0
         Order ID                  0
         Date                      0
         Status                    0
         Fulfilment                0
         Sales Channel             0
         ship-service-level        0
         Category                  0
         Size                      0
         Courier Status            0
         Qty                       0
         currency                  0
         Amount                    0
         ship-city                35
         ship-state               35
         ship-postal-code         35
         ship-country             35
         B2B                       0
         fulfilled-by          89713
         dtype: int64
```

```
In [16]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128976 entries, 0 to 128975
Data columns (total 19 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   index               128976 non-null  int64
 1   Order ID            128976 non-null  object
 2   Date                128976 non-null  object
 3   Status              128976 non-null  object
 4   Fulfilment          128976 non-null  object
 5   Sales Channel       128976 non-null  object
 6   ship-service-level  128976 non-null  object
 7   Category            128976 non-null  object
 8   Size                128976 non-null  object
 9   Courier Status      128976 non-null  object
 10  Qty                 128976 non-null  int64
 11  currency            128976 non-null  object
 12  Amount              128976 non-null  float64
 13  ship-city           128941 non-null  object
 14  ship-state          128941 non-null  object
 15  ship-postal-code    128941 non-null  float64
 16  ship-country        128941 non-null  object
 17  B2B                 128976 non-null  bool
 18  fulfilled-by        39263 non-null   object
dtypes: bool(1), float64(2), int64(2), object(14)
memory usage: 17.8+ MB
```

STEP 13:

The code df.columns returns a list of column labels in DataFrame 'df'. This provides a quick way to see the names of all the columns in a DataFrame and can be useful for reference when working with data or performing operations on specific columns.

```
In [17]: df.columns
Out[17]: Index(['index', 'Order ID', 'Date', 'Status', 'Fulfilment', 'Sales Channel',
                'ship-service-level', 'Category', 'Size', 'Courier Status', 'Qty',
                'currency', 'Amount', 'ship-city', 'ship-state', 'ship-postal-code',
                'ship-country', 'B2B', 'fulfilled-by'],
               dtype='object')
```

STEP 14:

This line of code uses the mode of that column (the common value) to fill in the missing values in the 'fulfilled-by' column. This can be helpful for categorical data imputation since it replaces missing values with common values to preserve the distribution and quality of the data.

```
In [18]: df['fulfilled-by']=df['fulfilled-by'].fillna(df['fulfilled-by']).mode()[0]

In [19]: df['fulfilled-by'].isnull().sum()
Out[19]: 0

In [20]: df.isnull().sum()
Out[20]: index                  0
         Order ID               0
         Date                   0
         Status                 0
         Fulfilment             0
         Sales Channel          0
         ship-service-level     0
         Category               0
         Size                   0
         Courier Status         0
         Qty                    0
         currency               0
         Amount                 0
         ship-city             35
         ship-state            35
         ship-postal-code      35
         ship-country          35
         B2B                    0
         fulfilled-by           0
         dtype: int64
```

STEP 15:

The code df.shape returns the shapes of a tuple representing the DataFrame 'df'. The tuple has two values: the number of rows (instances) and the number of columns in the DataFrame. It's a convenient way to quickly get an overview of the size of the dataset.

```
In [21]: df.shape
Out[21]: (128976, 19)

In [22]: df.columns
Out[22]: Index(['index', 'Order ID', 'Date', 'Status', 'Fulfilment', 'Sales Channel',
                'ship-service-level', 'Category', 'Size', 'Courier Status', 'Qty',
                'currency', 'Amount', 'ship-city', 'ship-state', 'ship-postal-code',
                'ship-country', 'B2B', 'fulfilled-by'],
               dtype='object')
```

STEP 16:

df['ship-postal-code'].fillna(0): Fills missing values in the 'ship-postal-code' column with the value 0. This is often used for numeric columns to replace missing values a specific value.
df['ship-postal-code'].astype("int"): Convert the data type of the 'ship-postal-code' column to an integer. This is done by filling missing values with 0. It should be noted that converting missing values (now replaced by 0) to absolute numbers may affect the analysis and interpretation of the data.
df['ship-postal-code'].dtype: Retrieves and prints the data type of the 'ship-postal-code' column after conversion to integer.

```
In [23]: df['ship-postal-code']=df['ship-postal-code'].fillna(0)

In [24]: df['ship-postal-code']=df['ship-postal-code'].astype("int")

In [25]: df['ship-postal-code'].dtype
Out[25]: dtype('int64')

In [26]: df.isnull().sum()
Out[26]: index                  0
         Order ID               0
         Date                   0
         Status                 0
         Fulfilment             0
         Sales Channel          0
         ship-service-level     0
         Category               0
         Size                   0
         Courier Status         0
         Qty                    0
         currency               0
         Amount                 0
         ship-city             35
         ship-state            35
         ship-postal-code       0
         ship-country          35
         B2B                    0
         fulfilled-by           0
         dtype: int64
```

STEP 17:

The code df['ship-city'].unique() retrieves the unique value of the DataFrame 'df' 'ship-city' column. This process provides a distinct list of city names in that category. This is useful for understanding the cities in the dataset and can help to cluster, filter, or analyze information based on cities.

```
In [27]: df['ship-city'].unique()
Out[27]: array(['MUMBAI', 'BENGALURU', 'NAVI MUMBAI', ...,
                'GULABPURA, Distt BHILWARA', 'Prayagraj (ALLAHABAD)', 'Halol'],
               dtype=object)
```

STEP 18:

df['ship-city'] = df['ship-city'].fillna('0'): Fill in missing values in the 'ship-city' column with the string '0'. This method is used for categorical data that replaces missing values with a specific placeholder.

```
In [28]: df['ship-city']=df['ship-city'].fillna(0)

In [29]: df.isnull().sum()

Out[29]: index                  0
         Order ID               0
         Date                   0
         Status                 0
         Fulfilment             0
         Sales Channel          0
         ship-service-level     0
         Category               0
         Size                   0
         Courier Status         0
         Qty                    0
         currency               0
         Amount                 0
         ship-city              0
         ship-state            35
         ship-postal-code       0
         ship-country          35
         B2B                    0
         fulfilled-by           0
         dtype: int64
```

STEP 19:

df['ship-state']=df['ship-state'].fillna(0): Fill missing values in the 'ship-state' column with the numeric value 0. This method is useful if the “ship-country.” Numeric data is expected in the ' column, but keep in mind that it may not be appropriate to use 0 for missing class data.

```
In [30]: df['ship-state']=df['ship-state'].fillna(0)

In [31]: df.isnull().sum()

Out[31]: index                  0
         Order ID               0
         Date                   0
         Status                 0
         Fulfilment             0
         Sales Channel          0
         ship-service-level     0
         Category               0
         Size                   0
         Courier Status         0
         Qty                    0
         currency               0
         Amount                 0
         ship-city              0
         ship-state             0
         ship-postal-code       0
         ship-country          35
         B2B                    0
         fulfilled-by           0
         dtype: int64
```

STEP 20:

df['ship-country'].unique(): Gets the unique values of the DataFrame 'df' 'ship-country' column. This event provides a list of different national standards in that particular color.
df['ship-country']=df['ship-country'].fillna('IN'): Fills missing values in the 'ship-country' column with the string value 'IN', which may stop there for India This method is useful when you want to set missing state values with default values based on contextual data.

```
In [32]: df['ship-country'].unique()
Out[32]: array(['IN', nan], dtype=object)

In [33]: df['ship-country']=df['ship-country'].fillna('IN')

In [34]: df.isnull().sum()
Out[34]: index                 0
         Order ID              0
         Date                  0
         Status                0
         Fulfilment            0
         Sales Channel         0
         ship-service-level    0
         Category              0
         Size                  0
         Courier Status        0
         Qty                   0
         currency              0
         Amount                0
         ship-city             0
         ship-state            0
         ship-postal-code      0
         ship-country          0
         B2B                   0
         fulfilled-by          0
         dtype: int64
```

STEP 20:

The df.describe() function provides a summary of descriptive statistics for the statistical characters in DataFrame 'df'. It includes calculations such as counts, averages, standard deviations, minimums, maximums, and percentages (25, 50, and 75).

```
In [36]: df.describe()
Out[36]:
```

	index	Qty	Amount	ship-postal-code
count	128976.000000	128976.000000	1.289760e+05	128976.000000
mean	64486.130427	0.904401	6.485622e+02	463819.777587
std	37232.897832	0.313368	1.288759e-09	191584.970648
min	0.000000	0.000000	6.485622e+02	0.000000
25%	32242.750000	1.000000	6.485622e+02	382421.000000
50%	64486.500000	1.000000	6.485622e+02	500033.000000
75%	96730.250000	1.000000	6.485622e+02	600024.000000
max	128974.000000	15.000000	6.485622e+02	989898.000000

STEP 21:

The code df.describe(include='object') creates a short descriptive statistic for categorical (object) columns in DataFrame 'df'. It includes statistics such as count, peak, surface (common value), and freq (frequency of value).

In [37]:
```
df.describe(include='object')
```

Out[37]:

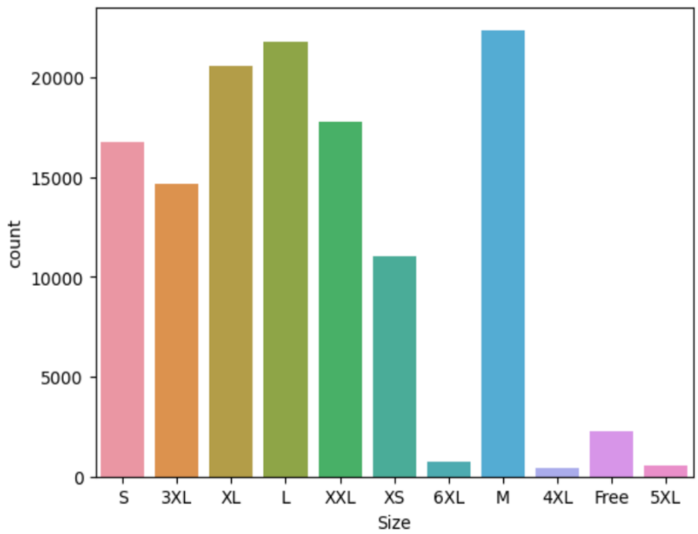
	Order ID	Date	Status	Fulfilment	Sales Channel	ship-service-level	Category	Size	Courier Status	currency	ship-city	ship-state	ship-country	fulfilled-by
count	128976	128976	128976	128976	128976	128976	128976	128976	128976	128976	128976	128976	128976	128976
unique	120229	91	13	2	2	2	9	11	4	1	8949	70	1	1
top	403-4984515-8861958	05-03-2022	Shipped	Amazon	Amazon.in	Expedited	T-shirt	M	Shipped	INR	BENGALURU	MAHARASHTRA	IN	Easy Ship
freq	12	2085	77815	89713	128852	88630	50292	22373	109486	128976	11208	22272	128976	128976

STEP 22:

This line of code uses the Seaborn count plot function to create the plot. This 'Size' column indicates which variables will be sorted along the x-axis, and data=df indicates that the data will be loaded from the DataFrame 'df'. The variable axis contains a reference to the plot, which can be used for further optimization or analysis.

SIZE

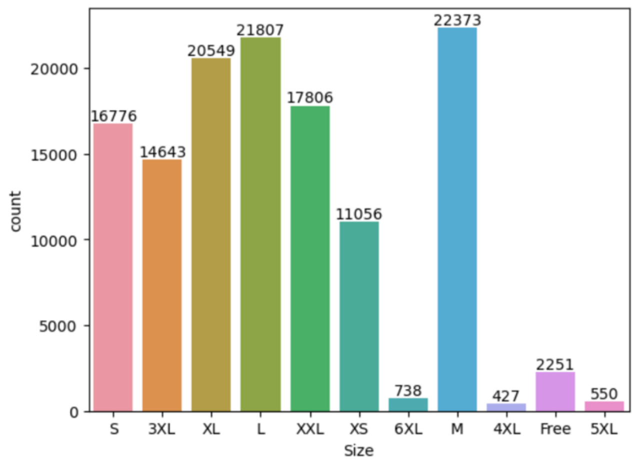
In [38]:
```
ax=sns.countplot(x='Size' ,data=df)
```



STEP 23:

The code first creates a statistical plot using the Seaborn countplot function and plots the statistics for each unique value in the ‘Size’ column of the DataFrame ‘df’. The statistical plot is then repeated on containers (bars) and labels for statistics are added to each bar. This increases visualization by providing real statistical values on each bar, making it easier to define size distributions in the data set.

In [39]:
```
x=sns.countplot(x='Size' ,data=df)

for bars in x.containers:
    x.bar_label(bars)
```



STEP 24:

df.groupby(['Size'], as_index=False)['Qty'].sum(): This groups the DataFrame 'df' by the 'Size' column and calculates the 'Qty' column of each group the whole of the The as_index=False argument ensures that the 'Size' column stays in the results as a constant column instead of being set as an index.

.sort_values(by='Qty', ascending=False): This sorts the collected and grouped DataFrame based on the 'Qty' column in descending order (large streams first). The result is a DataFrame showing the total numbers for each unique ‘Size’, ordered from highest to lowest number.

In [40]:
```
df.groupby(['Size'],as_index=False)['Qty'].sum(). sort_values(by='Qty',ascending=False)
```

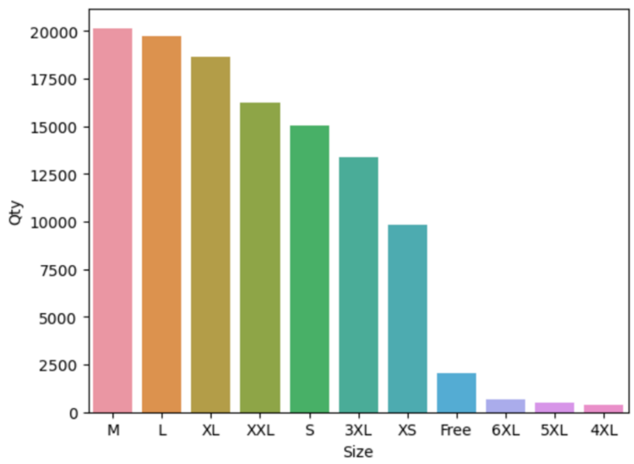
Out[40]:

	Size	Qty
6	M	20138
5	L	19706
8	XL	18636
10	XXL	16246
7	S	15041
0	3XL	13360
9	XS	9850
4	Free	2070
3	6XL	688
2	5XL	513
1	4XL	398

STEP 25:

ax=df.groupby(['Size'],as_index=False)['Qty'].sum().sort_values (by='Qty',ascending=False):
This line groups quantity ('Qty') data mouth Based on different sizes ('Show'), it multiplies the number of each size together, then sorts the result in descending order by total number. The results are stored in variable 'ax', which is a DataFrame with 'Size' and aggregated 'Qty' columns.
sns.barplot(x='Size', y='Qty', data=ax): This line uses the Seaborn barplot function to create a bar plot. It specifies the 'Size' column for the x-axis and the 'Qty' column stacked from the DataFrame 'ax' to the y-axis. This plot shows the sizes on the x-axis and their corresponding aggregate volumes on the y-axis and gives an idea of how the sizes contribute to total volume.

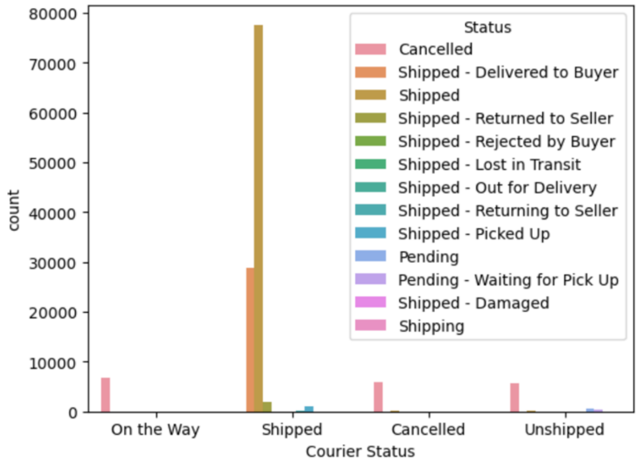
```
In [42]: ax=df.groupby(['Size'], as_index=False)['Qty'].sum().sort_values(by='Qty',ascending=False)

         sns.barplot(x='Size',y='Qty', data=ax)

Out[42]: <Axes: xlabel='Size', ylabel='Qty'>
```



STEP 26:

sns.countplot(data=df, x='Courier Status', hue='Status'): This line of code uses the Seaborn countplot function to create a bar plot. The data argument specifies the DataFrame 'df' containing the data to be formatted. The x argument specifies the categorical variable to be plotted on the x-axis, which in this case is 'Courier Status'. The hue argument specifies a new categorical variable ('Status') that will be used to group and color the bars based on 'Status' values. This results in bands per 'Courier Status' group, and each band is further divided into sections based on 'Status' criteria.

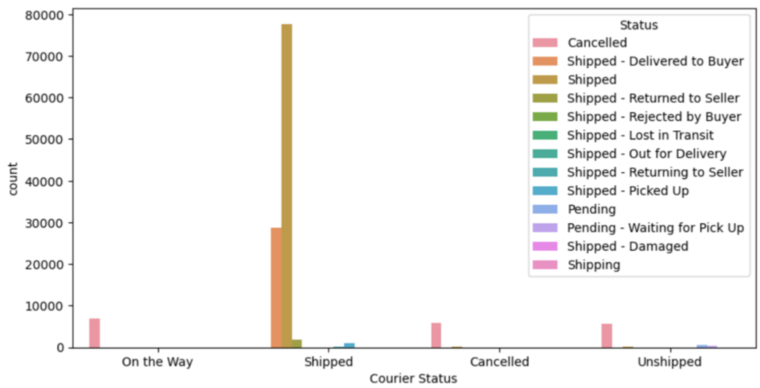
```
In [43]: sns.countplot(data=df, x='Courier Status',hue= 'Status')
Out[43]: <Axes: xlabel='Courier Status', ylabel='count'>
```



STEP 27:

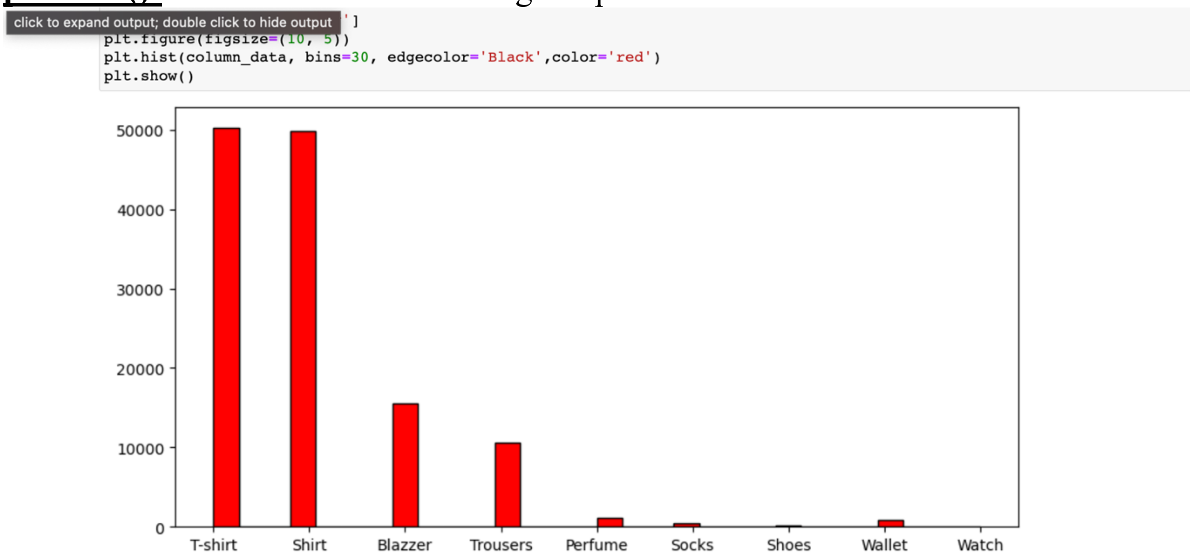
plt.figure(figsize=(10, 5)): This line sets the figure size to (10, 5) inches, which determines the size of the plot.
ax = sns.countplot(data=df, x='Courier Status', hue='Status'): This line creates a count plot using the Seaborn countplot function, as described earlier. The axis variable contains a description of the
plot.plt.show(): This line displays the plot using the show function in Matplotlib.

```
In [44]: plt.figure(figsize=(10,5))
         ax=sns.countplot(data=df, x='Courier Status',hue= 'Status')
         plt.show()
```



STEP 28:

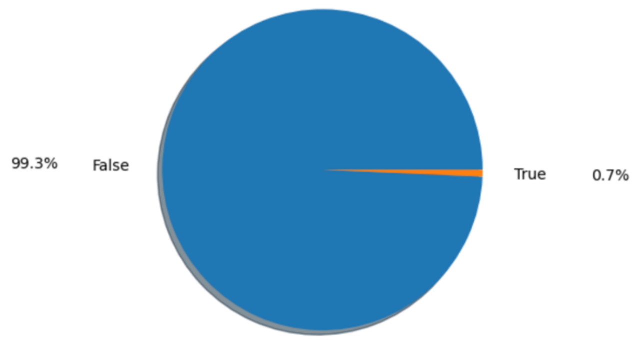
column_data = df['Category']: This line extracts data from the 'Category' column of the DataFrame and assigns it to the variable column_data.
plt.figure(figsize=(10, 5)): This line sets the zoomed figure to (10, 5) inches.
plt.hist(column_data, bins=30, edgecolor='Black', color='red'): This line uses Matplotlib's hist function to create a histogram. Column_data is the data to be plotted. The Bins parameter specifies the number of bins (bars) in the histogram. The edgecolor parameter sets the color of the edges of the bar, and the color parameter sets the color of the edges of the bars.
plt.show(): This line shows the histogram plot.



STEP 29:

B2B_Check = df['B2B'].value_counts(): This line counts the unique value of the 'B2B' column and stores the result in the variable B2B_Check. It creates a Series where the 'B2B' column contains the index of the unique value, and the value of the corresponding figure.
plt.pie(B2B_Check,labels=B2B_Check.index,autopct='%2.1f%%',shadow=True, pctdistance=1.8, labeldistance=1.2): This line uses Matplotlib's pie function to create a pie chart. B2B_Check Series Provides data for charts. The Labels parameter specifies the labels for each slice of the pie, which are unique values from the 'B2B' column. The Autopct parameter adds percentages to each slice. The shadow parameter adds a shadow effect to the chart. The pctdistance and labeldistance parameters adjust the distance between a percentage label and a category label from the center of the pie.
plt.show(): This line shows the histogram plot

```
In [46]: B2B_Check = df['B2B'].value_counts()

         plt.pie(B2B_Check,labels=B2B_Check.index,autopct='%2.1f%%',shadow='True',pctdistance=1.8,labeldistance=1.2)

         plt.show()
```



STEP 30:

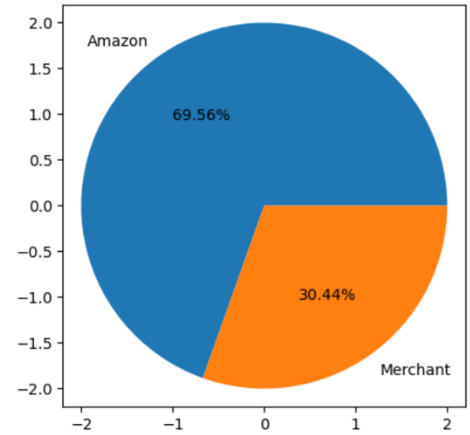
Code df['B2B'].value_counts() Calculates and displays the counts for each unique value in the 'B2B' column of DataFrame 'df'. This feature provides a quick summary of the number of events available for each value in the ‘B2B’ column. It helps to understand the distribution and spread of values in a column.

```
In [47]: df['B2B'].value_counts()

Out[47]: False    128104
         True        872
         Name: B2B, dtype: int64
```

STEP 31:

The code snippet provided is a pie chart based on the calculation of the unique values of the 'Fulfilment' column of the DataFrame 'df'. The chart shows the distribution of the complete elements, adds percentage labels to slices, and expands the radius of the chart. Using frame=True adds a circular frame to the pie chart. The resulting image provides insight into the distribution of each fulfilment in the data set.

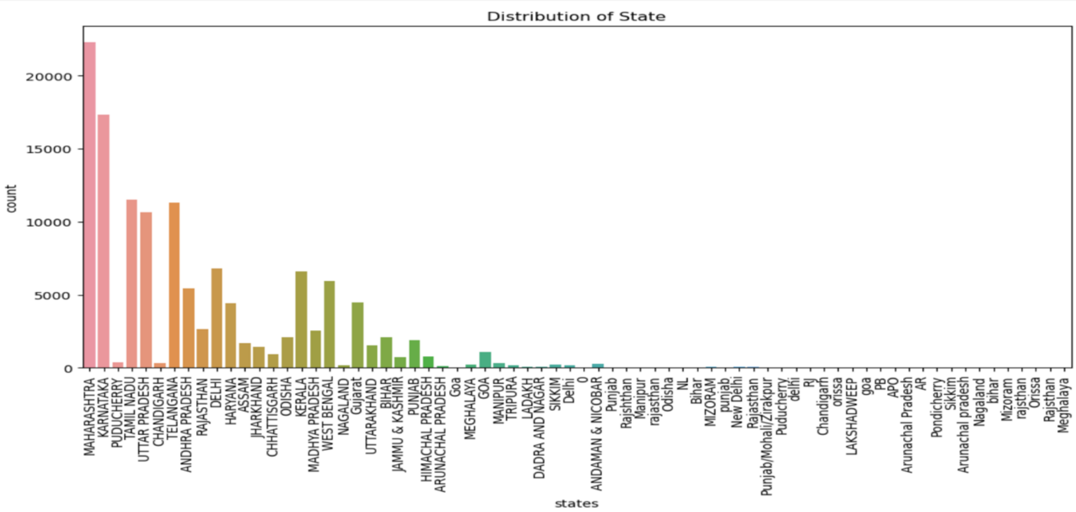
```
In [48]: full =df['Fulfilment'].value_counts()

         plt.pie(full,labels=full.index,autopct="%1.2f%%",radius=2,frame=True)

         plt.show()
```



STEP 32:

The code snippet provided is a horizontal bar plot that uses the Seaborn countplot function to plot the distribution of the state in the 'ship-state' column of the DataFrame 'df' The size of the figure has been adjusted for performance good, and labels and titles have been added to the plot. The x-axis represents the conditions, and the y-axis represents the number of events. The plot provides an overview of the frequency of each country in the data set, which facilitates understanding of the country distribution. The x-axis labels are rotated for better legibility.

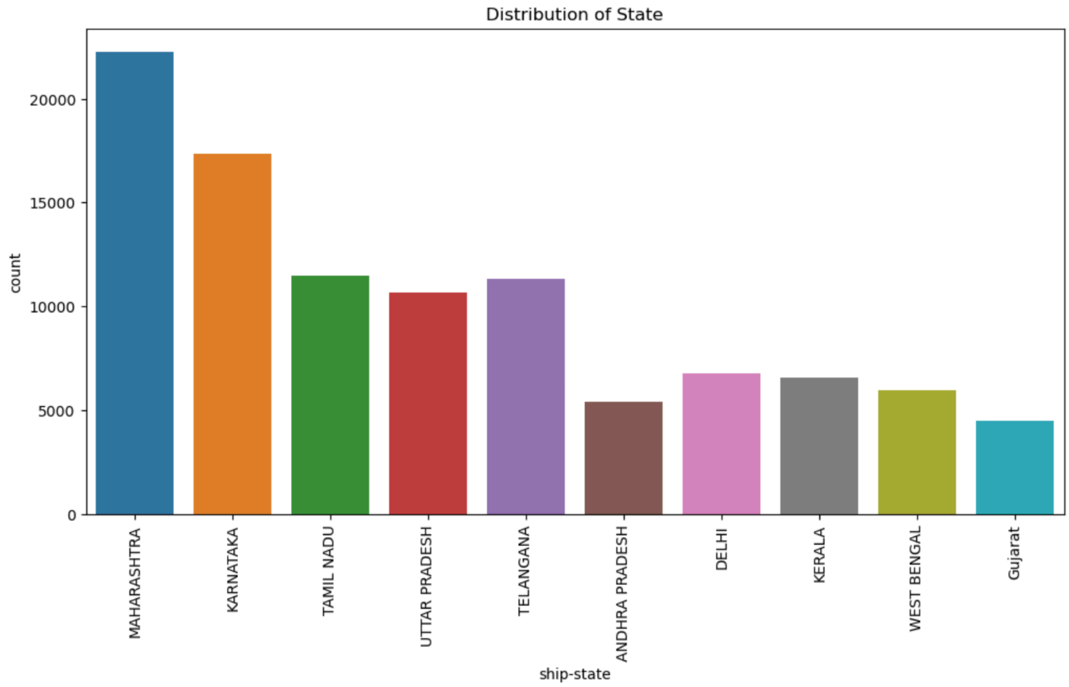
```
In [49]: plt.figure(figsize=(12, 6))
         sns.countplot(data=df, x='ship-state')
         plt.xlabel('states')
         plt.ylabel('count')
         plt.title('Distribution of State')
         plt.xticks(rotation=90)
         plt.show()
```



STEP 33:

top10 = df['ship-state'].value_counts().head(10): This line counts the counts for each unique state in the 'ship-state' column and then selects the top 10 most common states. The results are stored in the 'top10' series.
plt.figure(figsize=(12, 6)): This line sets the size of the figure to be sorted.
sns.countplot(data=df[df['ship-state'].isin(top10.index)], x='ship-state'): This line creates a count plot using the Seaborn countplot function. It uses the 'top10' series to filter the DataFrame to only include rows with ship conditions that are among the top 10 most frequent ones. The x-axis represents ship status, and the y-axis represents the number of events.
plt.xlabel('ship-state'): This line sets the label for the x-axis.
plt.ylabel('count'): This line sets the label for the y-axis.
plt.title('Distribution of State'): This line sets the title of the plot.
plt.xticks(rotation=90): This line rotates the x-axis label 90 degrees for better readability.
plt.show(): This line displays the plot.

In [50]:
```
top10= df['ship-state'].value_counts().head(10)
plt.figure(figsize=(12, 6))
sns.countplot(data=df[df['ship-state'].isin(top10.index)], x='ship-state')
plt.xlabel('ship-state')
plt.ylabel('count')
plt.title('Distribution of State')
plt.xticks(rotation=90)
plt.show()
```



STEP 34:

last10 = df['ship-state'].value_counts().tail(10): This line counts the counts for each unique state in the 'ship-state' column and then selects the minimum 10 states. The results are stored in the 'last10' series.
plt.figure(figsize=(12, 6)): This line determines the size of the figure to be created.
sns.countplot(data=df[df['ship-state'].isin(last10.index)], x='ship-state'): This line creates a count plot using the Seaborn countplot function. It uses the 'last10' series to filter the DataFrame to only include rows with a ship status of at least 10. The x-axis represents ship status, and the y-axis represents text the number of events
plt.xlabel('ship-state'): This line sets the label for the x-axis.
plt.ylabel('count'): This line sets the label for the y-axis.
plt.title('Distribution of State'): This line sets the title of the plot.
plt.xticks(rotation=90): This line rotates the x-axis label 90 degrees for better readability.
plt.show(): This line displays the plot.

In [51]:
```
last10= df['ship-state'].value_counts().tail(10)
plt.figure(figsize=(12, 6))
sns.countplot(data=df[df['ship-state'].isin(last10.index)], x='ship-state')
plt.xlabel('ship-state')
plt.ylabel('count')
plt.title('Distribution of State')
plt.xticks(rotation=90)
plt.show()
```