PROJECT REPORT

1.	Your Name	:	R.Sudha
2.	Email	:	lsudha02@gmail.com
3.	Contact Number	:	+91 - 8056917143
4.	Batch Code	:	TN_DA_FNB01
5.	Project Title	:	E-Commerce Sales Analysis (2020–2025)
6.	Project Domain	:	Sales & E-commerce
7.	Mentor Name	:	Kumaran M
8.	Submission Date	:	08/10/2025
9.	Raw Dataset Link	:	https://drive.google.com/file/d/1bSRsDKHxtGlUAH2So7SFbHW4utjPFx9/view?usp=drive_link
10.	Cleaned Dataset Link	:	https://drive.google.com/file/d/1nNZK1XqNBZdxGHR9uZoqetmyIS4Qx5YQ/view?usp=drive_link

Project Title :

“E-Commerce Sales Analysis (2020–2025)”

Domain :

The chosen project domain is Retail / E-Commerce. This dataset represents the online sales business environment, capturing details of customer orders, products, payments, discounts, and shipments. Covering transactions from 2020 to 2025, it provides a comprehensive view of the sales process, making it ideal for analyzing sales trends, customer behavior, product performance, and overall business insights within the Retail / E-Commerce sector.

Objective :

1. To identify meaningful patterns, trends, and insights :

Analyze sales data from 2020 to 2025 to find year-wise, month-wise, and seasonal trends.

Understand customer purchasing behavior across different countries and sales channels.

2. To clean, transform, and preprocess the data for analysis :

Handle missing values and incorrect entries.

Standardize columns such as dates, quantities, and prices for accurate analysis.

3. To create meaningful visualizations that clearly explain the story behind the data :

Develop charts to highlight product demand, regional sales distribution, and payment methods.

Use trend graphs to show how sales have evolved over the years.

4. To provide actionable insights and recommendations :

Suggest the most profitable product categories and regions.

Identify reliable shipping providers and effective discount strategies.

5. To evaluate operational efficiency and customer satisfaction factors :

Assess the impact of discounts, payment methods, and return rates on overall sales.

Provide recommendations for improving customer experience and optimizing supply chain operations.

Outcome :

The project provides a comprehensive understanding of online sales transactions from 2020 to 2025. Through systematic data cleaning, exploration, and visualization, the analysis helps to:

- Identify sales trends over time (yearly, monthly, and seasonal).
- Understand customer purchase behavior across countries and sales channels.
- Discover the impact of discounts, payment methods, and shipment providers on sales and returns.
- Highlight top-performing product categories and warehouses contributing to revenue.
- Evaluate operational efficiency in terms of shipping costs, order priority, and return handling.

The outcomes support data-driven decision-making, enabling businesses to improve sales strategies, optimize inventory, enhance customer satisfaction, and increase profitability.

Dataset Information :

Source: Kaggle Datasets

Year/ Timeline: Not specified (Updated 10 months ago) as on 2025/09/09

Dataset Size: 49,782 rows × 17 columns

Dataset Description:

1.The dataset comprises anonymized data on online sales transactions, capturing various aspects of product purchases, customer details, and order characteristics.

2.This dataset can be utilized for analyzing sales trends, customer purchase behavior, and order management in e-commerce or retail. It can aid in understanding the impact of discounts, payment methods, and shipment providers on sales performance and customer satisfaction.

3.This dataset can be utilized for analyzing sales performance, customer purchasing patterns, and operational efficiency in order management.

4.It can help in evaluating the effects of discounts and payment methods on sales, optimizing inventory by studying product demand, and improving customer satisfaction through better shipping and return handling.

Key Features :#

Order Details : Order ID, Order Date, Ship Date, Order Priority

Customer Information : Customer ID, Country, City, Segment

Product Information : Product ID, Category, Sub-Category, Product Name

Sales Metrics : Sales, Quantity, Discount, Profit

Shipping & Returns : Ship Mode, Shipment Cost, Return Status, Warehouse

Type of Analysis :

Descriptive Analysis : Summarizes the data to understand sales, orders, and customers. It shows trends over time and basic buying patterns.

Diagnostic Analysis : Explains why things happen, like top-selling products, returns, and the effect of payment or shipping methods.

Predictive Analysis : Uses historical data to forecast future sales, demand for products, and potential customer churn.

Prescriptive Analysis : Provides actionable recommendations for inventory, shipping, promotions, and reducing returns.

Stages for DA Project :

Stage 1 – Problem Definition and Dataset Selection :

Problem Definition :

The project aims to analyze the E-Commerce Sales Dataset (2020–2025) to uncover key business insights and address critical challenges in the online retail domain. The analysis focuses on identifying top-selling products and categories, evaluating customer purchasing behavior and preferences, and understanding how discounts, pricing strategies, and shipping methods influence overall sales performance. It also seeks to detect seasonal fluctuations and long-term growth patterns that affect business outcomes. Ultimately, the problem definition emphasizes the need to transform raw e-commerce data into actionable insights and strategic recommendations that can help improve sales performance, optimize customer engagement, and enhance business profitability.

Dataset Selection :

The dataset selected for this project is the E-Commerce Sales Dataset (2020–2025), obtained from a trusted online repository. It contains comprehensive information on online retail transactions, including orders, products, customers, pricing, discounts, and shipping details. Covering a five-year period, the dataset enables a deep analysis of long-term sales patterns, customer behavior, and business performance.

This dataset was chosen because it is:

Relevant – directly aligned with e-commerce sales analysis.

Comprehensive – includes diverse attributes such as sales, customer, and product details.

Time-spanning – covers multiple years, allowing seasonal and trend-based insights.

Insightful – supports the project goals of identifying top-selling products, understanding customer preferences, and evaluating the impact of discounts and shipping methods.

Overall, the dataset provides a reliable foundation for data cleaning, transformation, exploratory data analysis (EDA), visualization, and actionable business insights.

Expected Outcome :

The analysis aims to:

1) Identify top-selling products and categories across the dataset.
2) Understand customer purchasing patterns and behavioral trends over time.
3) Evaluate the impact of discounts, pricing, and shipping methods on overall sales performance.
4) Provide actionable recommendations to improve inventory planning, pricing strategies, and promotional campaigns.
5) Overall, the project is expected to generate meaningful business insights and deliver data-driven recommendations through visualizations and dashboards, helping improve sales growth, customer satisfaction, and decision-making efficiency.

1. # Import Libraries :

Input :

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

Interpretation:

In Python, `pandas` and `numpy` handle data analysis and numerical operations.`matplotlib` creates basic plots, while `seaborn` makes them more attractive and insightful.All together, they form the core tools for effective data analysis and visualization.

2. # Load Dataset :

Input :

```
df =
pd.read_csv("https://raw.githubusercontent.com/sudha0206/FINAL-PROJECT-ENTRI---E-Commerce-Sales-Analysis-2020-2025-/refs/heads/main/Online%20Sales%20Dataset%202020%20-%202025.csv")

print(df)
```

Output :

```
     InvoiceNo StockCode   Description  Quantity       InvoiceDate  \

0       221958  SKU_1964     White Mug        38  01-01-2020 00:00

1       771155  SKU_1241     White Mug        18  01-01-2020 01:00

2       231932  SKU_1501    Headphones        49  01-01-2020 02:00
```

3	465838	SKU_1760	Desk Lamp	14	01-01-2020 03:00	
4	359178	SKU_1386	USB Cable	-30	01-01-2020 04:00	
...	...	...	...	...	...	
49777	354083	SKU_1562	Blue Pen	25	05-09-2025 01:00	
49778	296698	SKU_1930	USB Cable	7	05-09-2025 02:00	
49779	177622	SKU_1766	Office Chair	43	05-09-2025 03:00	
49780	701213	SKU_1602	Notebook	48	05-09-2025 04:00	
49781	772215	SKU_1832	White Mug	30	05-09-2025 05:00	

	UnitPrice	CustomerID	Country	Discount	PaymentMethod	\
0	1.71	37039.0	Australia	0.470000	Bank Transfer	
1	41.25	19144.0	Spain	0.190000	paypall	
2	29.11	50472.0	Germany	0.350000	Bank Transfer	
3	76.68	96586.0	Netherlands	0.140000	paypall	
4	-68.11	NaN	United Kingdom	1.501433	Bank Transfer	
...	...	...	...	...	...	
49777	70.92	51445.0	Spain	0.200000	Credit Card	
49778	51.74	28879.0	United States	0.230000	Bank Transfer	
49779	85.25	21825.0	Portugal	0.200000	Bank Transfer	
49780	39.64	43199.0	United Kingdom	0.310000	paypall	
49781	38.27	53328.0	France	0.100000	Credit Card	

	ShippingCost	Category	SalesChannel	ReturnStatus	ShipmentProvider	\
0	10.79	Apparel	In-store	Not Returned	UPS	
1	9.51	Electronics	Online	Not Returned	UPS	
2	23.03	Electronics	Online	Returned	UPS	
3	11.08	Accessories	Online	Not Returned	Royal Mail	
4	NaN	Electronics	In-store	Not Returned	FedEx	
...	...	...	...	...	...	
49777	8.96	Electronics	Online	Returned	UPS	
49778	23.55	Electronics	Online	Not Returned	FedEx	
49779	16.26	Furniture	In-store	Not Returned	FedEx	
49780	28.56	Apparel	Online	Not Returned	Royal Mail	
49781	9.13	Stationery	Online	Not Returned	UPS	

	WarehouseLocation	OrderPriority
0	London	Medium
1	Rome	Medium
2	Berlin	High
3	Rome	Low
4	NaN	Medium
...	...	...
49777	Berlin	Medium
49778	Amsterdam	Low

49779 London High

49780 London Medium

49781 Rome Low

[49782 rows x 17 columns]

Interpretation:

The dataset has been successfully loaded into a pandas DataFrame (`df`) from the CSV file.It includes details of online sales such as invoices, products, customers, and countries.The data is now organized and ready for further analysis and visualization in Python.

3.# Shape of Dataset (Rows, Columns) :

Input :

```
print("Rows:", df.shape[0])

print("Columns:", df.shape[1])
```

Output :

Rows: 49782

Columns: 17

Interpretation:

The dataset contains **49,782 rows** and **17 columns**. Each row represents a sales record, and each column represents a feature like invoice, product, or customer.This gives a clear idea of the dataset's size and structure for analysis in Python.

4.# Dataset Description (Features):

Input :

```
print("\nDataset Columns:")
```

Output :

```
display(df.columns)

Dataset Columns:

Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country', 'Discount', 'PaymentMethod',
       'ShippingCost', 'Category', 'SalesChannel', 'ReturnStatus',
       'ShipmentProvider', 'WarehouseLocation', 'OrderPriority'],     dtype='object'
```

Interpretation:

The dataset contains **17 columns**, each representing a specific feature of the sales data.Columns include details like invoice number, product, quantity, price, customer, and shipping information.Listing the columns helps us understand the data structure and the information available for analysis.

5.# Initial Exploratory Data Analysis :

Input :

```
# First Ten Rows

print(df.head(10))
```

Output :

```
InvoiceNo StockCode     Description  Quantity       InvoiceDate  UnitPrice \
0     221958  SKU_1964       White Mug        38  01-01-2020 00:00     1.71
1     771155  SKU_1241       White Mug        18  01-01-2020 01:00    41.25
2     231932  SKU_1501      Headphones        49  01-01-2020 02:00    29.11
3     465838  SKU_1760       Desk Lamp        14  01-01-2020 03:00    76.68
```

```
4    359178  SKU_1386       USB Cable       -30  01-01-2020 04:00  -68.11
5    744167  SKU_1006    Office Chair        47  01-01-2020 05:00   70.16
6    210268  SKU_1087       USB Cable        25  01-01-2020 06:00   85.74
7    832180  SKU_1597        Notebook         8  01-01-2020 07:00   95.65
8    154886  SKU_1907  Wireless Mouse        19  01-01-2020 08:00   98.19
9    237337  SKU_1866      Headphones        40  01-01-2020 09:00   98.17

   CustomerID        Country  Discount  PaymentMethod  ShippingCost  \
0     37039.0      Australia  0.470000  Bank Transfer         10.79
1     19144.0          Spain  0.190000         paypall          9.51
2     50472.0        Germany  0.350000  Bank Transfer         23.03
3     96586.0    Netherlands  0.140000         paypall         11.08
4         NaN  United Kingdom  1.501433  Bank Transfer           NaN
5     53887.0         Sweden  0.480000       Credit Card         13.98
6     46567.0        Belgium  0.150000      Bank Transfer        12.92
7     75098.0         Norway  0.040000      Bank Transfer         6.48
8     87950.0        Belgium  0.050000         paypall         12.56
9     39718.0          Italy  0.160000     Bank Transfer         7.71

    Category SalesChannel  ReturnStatus ShipmentProvider WarehouseLocation  \
0     Apparel     In-store  Not Returned             UPS              London
1  Electronics       Online  Not Returned             UPS                Rome
```

2	Electronics	Online	Returned	UPS	Berlin
3	Accessories	Online	Not Returned	Royal Mail	Rome
4	Electronics	In-store	Not Returned	FedEx	NaN
5	Electronics	Online	Not Returned	DHL	London
6	Stationery	Online	Not Returned	FedEx	Amsterdam
7	Electronics	In-store	Not Returned	Royal Mail	Amsterdam
8	Apparel	Online	Not Returned	UPS	Berlin
9	Apparel	Online	Not Returned	Royal Mail	London

	OrderPriority
0	Medium
1	Medium
2	High
3	Low
4	Medium
5	Medium
6	High
7	Low
8	High
9	Low

Interpretation:

Displaying the **first 10 rows** helps to quickly understand the dataset's structure, sample values, and check if the data is clean and consistent.

6.# Last Ten Rows :

Input :

```
print(df.tail(10))
```

Output :

```
      InvoiceNo StockCode   Description  Quantity       InvoiceDate  \
49772    224131  SKU_1095     USB Cable        27  04-09-2025 20:00
49773    370260  SKU_1570      Blue Pen        12  04-09-2025 21:00
49774    274759  SKU_1899     USB Cable        43  04-09-2025 22:00
49775    405597  SKU_1140      Backpack        37  04-09-2025 23:00
49776    524453  SKU_1316  Office Chair        43  05-09-2025 00:00
49777    354083  SKU_1562      Blue Pen        25  05-09-2025 01:00
49778    296698  SKU_1930     USB Cable         7  05-09-2025 02:00
49779    177622  SKU_1766  Office Chair        43  05-09-2025 03:00
49780    701213  SKU_1602      Notebook        48  05-09-2025 04:00
49781    772215  SKU_1832     White Mug        30  05-09-2025 05:00

         UnitPrice  CustomerID   Country  Discount  PaymentMethod  \
49772        30.87     63980.0         Italy      0.38         Credit Card
49773        51.83     85250.0         Italy      0.32         Credit Card
49774        12.63     14206.0        Sweden      0.11           paypall
49775        66.64     12024.0         Italy      0.35           paypall
49776        84.19     84399.0  United Kingdom     0.25     Credit Card
```

49777	70.92	51445.0	Spain	0.20	Credit Card
49778	51.74	28879.0	United States	0.23	Bank Transfer
49779	85.25	21825.0	Portugal	0.20	Bank Transfer
49780	39.64	43199.0	United Kingdom	0.31	paypall
49781	38.27	53328.0	France	0.10	Credit Card

	ShippingCost	Category	SalesChannel	ReturnStatus	ShipmentProvider
49772	14.48	Apparel	In-store	Not Returned	UPS
49773	22.44	Accessories	In-store	Not Returned	Royal Mail
49774	27.75	Electronics	Online	Not Returned	DHL
49775	19.93	Apparel	Online	Returned	UPS
49776	21.38	Electronics	Online	Not Returned	DHL
49777	8.96	Electronics	Online	Returned	UPS
49778	23.55	Electronics	Online	Not Returned	FedEx
49779	16.26	Furniture	In-store	Not Returned	FedEx
49780	28.56	Apparel	Online	Not Returned	Royal Mail
49781	9.13	Stationery	Online	Not Returned	UPS

	WarehouseLocation	OrderPriority
49772	Amsterdam	High
49773	Rome	Medium
49774	Amsterdam	High

49775	Rome	Low
49776	Amsterdam	Low
49777	Berlin	Medium
49778	Amsterdam	Low
49779	London	High
49780	London	Medium
49781	Rome	Low

Interpretation:

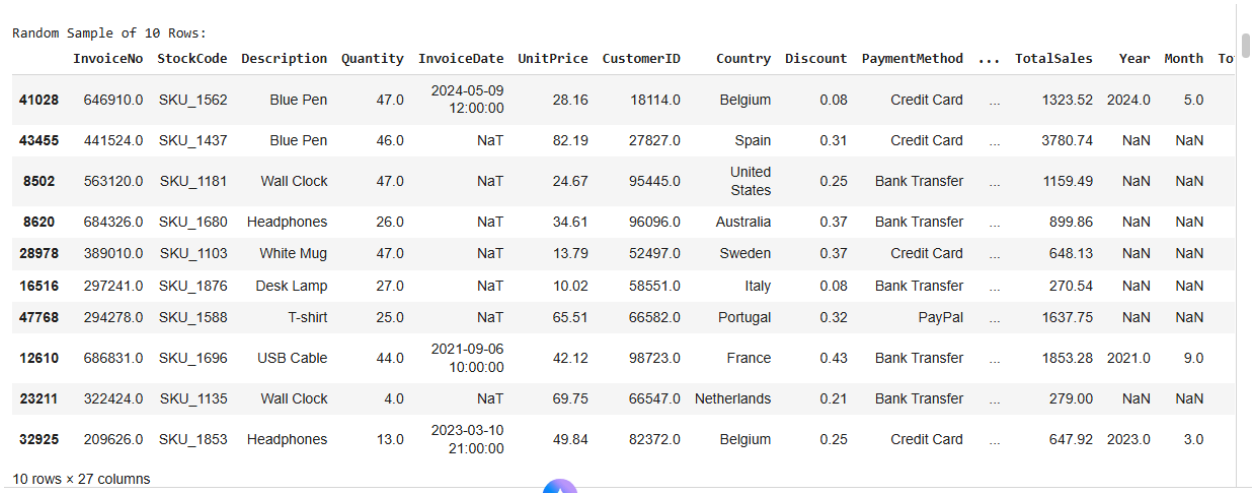
Viewing the **last 10 rows** helps confirm the dataset's ending records and check for any unusual or incomplete entries at the bottom.

7.# Random Sample of 10 :

Input :

```
print("\nRandom Sample of 10 Rows:")

display(df.sample(10))
```

Output :



Random Sample of 10 Rows:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Discount	PaymentMethod	...	TotalSales	Year	Month	To
41028	646910.0	SKU_1562	Blue Pen	47.0	2024-05-09 12:00:00	28.16	18114.0	Belgium	0.08	Credit Card	...	1323.52	2024.0	5.0	
43455	441524.0	SKU_1437	Blue Pen	46.0	NaT	82.19	27827.0	Spain	0.31	Credit Card	...	3780.74	NaN	NaN	
8502	563120.0	SKU_1181	Wall Clock	47.0	NaT	24.67	95445.0	United States	0.25	Bank Transfer	...	1159.49	NaN	NaN	
8620	684326.0	SKU_1680	Headphones	26.0	NaT	34.61	96096.0	Australia	0.37	Bank Transfer	...	899.86	NaN	NaN	
28978	389010.0	SKU_1103	White Mug	47.0	NaT	13.79	52497.0	Sweden	0.37	Credit Card	...	648.13	NaN	NaN	
16516	297241.0	SKU_1876	Desk Lamp	27.0	NaT	10.02	58551.0	Italy	0.08	Bank Transfer	...	270.54	NaN	NaN	
47768	294278.0	SKU_1588	T-shirt	25.0	NaT	65.51	66582.0	Portugal	0.32	PayPal	...	1637.75	NaN	NaN	
12610	686831.0	SKU_1696	USB Cable	44.0	2021-09-06 10:00:00	42.12	98723.0	France	0.43	Bank Transfer	...	1853.28	2021.0	9.0	
23211	322424.0	SKU_1135	Wall Clock	4.0	NaT	69.75	66547.0	Netherlands	0.21	Bank Transfer	...	279.00	NaN	NaN	
32925	209626.0	SKU_1853	Headphones	13.0	2023-03-10 21:00:00	49.84	82372.0	Belgium	0.25	Credit Card	...	647.92	2023.0	3.0	

10 rows × 27 columns

Interpretation:

`df.sample(10)` selects **10 random rows** from the dataset. It is used to quickly check and verify the data. Each time we run, the rows will be different because it is random.

8.# Shape :

Input :

```
print(df.shape)

print("Number of rows:", df.shape[0])

print("Number of columns:", df.shape[1])
```

Output :

(49782, 17)

Number of rows: 49782

Number of columns: 17

Interpretation:

The dataset has **49,782 rows** (records) and **17 columns** (features). Rows represent individual transactions/orders, and columns represent different attributes like InvoiceNo, StockCode, Quantity, etc.

9.# Data Type of Features :

Input :

```
print("Data Type of Features:")

print(df.dtypes)
```

Output :

Data Type of Features:

InvoiceNo int64

StockCode object

Description object

Quantity int64

InvoiceDate object

UnitPrice float64

CustomerID float64

Country object

Discount float64

PaymentMethod object

ShippingCost float64

Category object

SalesChannel object

ReturnStatus object

ShipmentProvider object

WarehouseLocation object

OrderPriority object

dtype: object

Interpretation:

The dataset contains different data types – int64 for numeric whole numbers (e.g., InvoiceNo, Quantity), float64 for decimal values (e.g., UnitPrice, Discount, ShippingCost), and object for categorical/text data (e.g., StockCode, Country, PaymentMethod). This helps us understand how each column should be processed during analysis..

10.# Count of Null Values per Column :

Input :

```
print("\nCount of Null Values per Column:")

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

Output :

Count of Null Values per Column:

	0
InvoiceNo	0
StockCode	0
Description	0
Quantity	0
InvoiceDate	0
UnitPrice	0
CustomerID	4978
Country	0
Discount	0

PaymentMethod	0
ShippingCost	2489
Category	0
SalesChannel	0
ReturnStatus	0
ShipmentProvider	0
WarehouseLocation	3485
OrderPriority	0

dtype: int64

Interpretation:

Some columns have missing values. For example, **CustomerID has 4978 nulls**, **ShippingCost has 2489 nulls**, and **WarehouseLocation has 3485 nulls**. All other columns have 0 nulls. This shows we need to handle these missing values during data cleaning.

11. # Dataset Info (Summary of Data Frame) :

Input :

```
print("\nDataset Info:")

df.info()
```

Output :

Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 49782 entries, 0 to 49781

Data columns (total 17 columns):

#	Column	Non-Null Count	Dtype
0	InvoiceNo	49782 non-null	int64
1	StockCode	49782 non-null	object
2	Description	49782 non-null	object
3	Quantity	49782 non-null	int64
4	InvoiceDate	49782 non-null	object
5	UnitPrice	49782 non-null	float64
6	CustomerID	44804 non-null	float64
7	Country	49782 non-null	object
8	Discount	49782 non-null	float64
9	PaymentMethod	49782 non-null	object
10	ShippingCost	47293 non-null	float64
11	Category	49782 non-null	object
12	SalesChannel	49782 non-null	object
13	ReturnStatus	49782 non-null	object
14	ShipmentProvider	49782 non-null	object
15	WarehouseLocation	46297 non-null	object
16	OrderPriority	49782 non-null	object

dtypes: float64(4), int64(2), object(11)

memory usage: 6.5+ MB

Interpretation:

The dataset has **49,782 rows and 17 columns**. It includes **2 integer columns, 4 float columns, and 11 object (categorical/text) columns**. Some columns like **CustomerID, ShippingCost, and WarehouseLocation** have missing values. The total memory usage is around **6.5 MB**.

12. # Percentage of Null Values per Column :

Input :

```
print("\nPercentage of Null Values per Column:")

display(df.isnull().mean() * 100)
```

Output :

Percentage of Null Values per Column:

	0
InvoiceNo	0.000000
StockCode	0.000000
Description	0.000000
Quantity	0.000000
InvoiceDate	0.000000

UnitPrice	0.000000
CustomerID	9.999598
Country	0.000000
Discount	0.000000
PaymentMethod	0.000000
ShippingCost	4.999799
Category	0.000000
SalesChannel	0.000000
ReturnStatus	0.000000
ShipmentProvider	0.000000
WarehouseLocation	7.000522
OrderPriority	0.000000

dtype: float64

Interpretation:

Some columns have missing data in terms of percentage:

- **CustomerID → ~10% missing**
- **ShippingCost → ~5% missing**
- **WarehouseLocation → ~7% missing**
  All other columns have **0% missing values**. This helps to understand how much of the data needs cleaning.

13. # Number of Unique Values per Column :

Input :

```
print("\nNumber of Unique Values per Column:")

display(df.nunique())
```

Output :

Number of Unique Values per Column:

	0
InvoiceNo	48494
StockCode	1000
Description	11
Quantity	99
InvoiceDate	49782
UnitPrice	11200

CustomerID	35389
Country	12
Discount	1047
PaymentMethod	3
ShippingCost	2501
Category	5
SalesChannel	2
ReturnStatus	2
ShipmentProvider	4
WarehouseLocation	5
OrderPriority	3

dtype: int64

Interpretation:

- **InvoiceNo** has 48,494 unique values → almost every order is unique.
- **StockCode** has 1,000 unique products.
- **Description** has 11 unique items.

- **CustomerID** has 35,389 unique customers.
- **Country** has 12 unique countries.
- Other columns like **PaymentMethod (3), SalesChannel (2), ReturnStatus (2), OrderPriority (3)** have few unique categories.

This helps understand which columns are **categorical** and which have **high variability**.

14. # Statistical Summary :

Input :

```
print("\nStatistical Summary:")

display(df.describe(include='all'))
```

Output :

Statistical Summary:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Discount	PaymentMethod	ShippingCost	Catego
count	49782.000000	49782	49782	49782.000000	49782	49782.000000	44804.000000	49782	49782.000000	49782	47293.000000	497
unique	NaN	1000	11	NaN	49782	NaN	NaN	12	NaN	3	NaN	
top	NaN	SKU_1761	Wall Clock	NaN	05-09-2025 05:00	NaN	NaN	France	NaN	Bank Transfer	NaN	Furnitu
freq	NaN	76	4617	NaN	1	NaN	NaN	4230	NaN	16747	NaN	100
mean	550681.239946	NaN	NaN	22.372343	NaN	47.537862	55032.871775	NaN	0.275748	NaN	17.494529	Na
std	260703.009944	NaN	NaN	17.917774	NaN	33.479510	25913.660157	NaN	0.230077	NaN	7.220557	Na
min	100005.000000	NaN	NaN	-50.000000	NaN	-99.980000	10001.000000	NaN	0.000000	NaN	5.000000	Na
25%	324543.000000	NaN	NaN	11.000000	NaN	23.592500	32750.750000	NaN	0.130000	NaN	11.220000	Na
50%	552244.000000	NaN	NaN	23.000000	NaN	48.920000	55165.000000	NaN	0.260000	NaN	17.500000	Na
75%	776364.000000	NaN	NaN	37.000000	NaN	74.610000	77306.250000	NaN	0.380000	NaN	23.720000	Na
max	999997.000000	NaN	NaN	49.000000	NaN	100.000000	99998.000000	NaN	1.999764	NaN	30.000000	Na

Interpretation:

The statistical summary shows **basic statistics** of all columns:

- For **numeric columns** (Quantity, UnitPrice, Discount, ShippingCost), it shows **count, mean, std, min, max, 25%, 50%, 75% values**.
- For **categorical/text columns** (StockCode, Country, PaymentMethod, etc.), it shows **count, number of unique values, most frequent value, and its frequency**.

- This helps to quickly **understand the distribution and patterns** in the dataset.

15. # Duplicate rows count :

Input :

```
print("\nDuplicate Rows:")

print(df.duplicated().sum())
```

Output :

Duplicate Rows:

0

Interpretation:

The dataset has 0 duplicate rows, which means every row is unique. No repeated records are present.

16 . # Correlation matrix (numerical columns only) :

Input :

```
print("\nCorrelation Matrix:")

display(df.corr(numeric_only=True))
```

Output :

Correlation Matrix:

	InvoiceNo	Quantity	UnitPrice	CustomerID	Discount	ShippingCost
InvoiceNo	1.000000	-0.004980	-0.016436	0.000730	0.011413	0.001441

Quantity	-0.004980	1.000000	0.241028	0.002889	-0.290293	-0.004086
UnitPrice	-0.016436	0.241028	1.000000	0.000065	-0.329025	0.003112
CustomerID	0.000730	0.002889	0.000065	1.000000	0.010822	0.004071
Discount	0.011413	-0.290293	-0.329025	0.010822	1.000000	-0.001282
ShippingCost	0.001441	-0.004086	0.003112	0.004071	-0.001282	1.000000

Interpretation:

Correlation matrix shows that most columns have very weak or no relationship. The only meaningful ones are Quantity vs Discount (-0.29) and UnitPrice vs Discount (-0.33), meaning higher quantity purchases get more discounts, and higher-priced items usually get fewer discounts**.**

17. # Unique values of PaymentMethod :

Input :

```
unique_payment = df["PaymentMethod"].unique()

print("\nThe Unique Values of PaymentMethod:", unique_payment)
```

Output :

The Unique Values of PaymentMethod: ['Bank Transfer' 'paypall' 'Credit Card']

Interpretation:

The dataset has **3 payment methods**: Bank Transfer, PayPal, and Credit Card.

18. # Unique values of Category :

Input :

```
unique_category = df["Category"].unique()

print("\nThe Unique Values of Category:", unique_category)
```

Output :

The Unique Values of Category: ['Apparel' 'Electronics' 'Accessories' 'Stationery' 'Furniture']

Interpretation:

The dataset has **5 categories**: Apparel, Electronics, Accessories, Stationery, and Furniture.

19. # Unique values of ReturnStatus :

Input :

```
unique_return_status = df["ReturnStatus"].unique()

print("\nThe Unique Values of ReturnStatus:",unique_return_status)
```

Output :

The Unique Values of ReturnStatus: ['Not Returned' 'Returned']

Interpretation:

The dataset has **2 return statuses**: Not Returned and Returned.

20. # Unique values of ShipmentProvider :

Input :

```
unique_shipment_provider = df["ShipmentProvider"].unique()

print("\nThe Unique Values of ShipmentProvider:", unique_shipment_provider)
```

Output :

```
The Unique Values of ShipmentProvider: ['UPS' 'Royal Mail' 'FedEx' 'DHL']
```

Interpretation:

The dataset has **4 shipment providers**: UPS, Royal Mail, FedEx, and DHL.

21. # Unique values of WarehouseLocation :

Input :

```
unique_warehouse = df["WarehouseLocation"].unique()

print("\nThe Unique Values of WarehouseLocation:", unique_warehouse)
```

Output :

```
The Unique Values of WarehouseLocation: ['London' 'Rome' 'Berlin' nan 'Amsterdam' 'Paris']
```

Interpretation:

The dataset has **5 warehouse locations** (London, Rome, Berlin, Amsterdam, Paris) and some **missing values (NaN)**.

22. # Unique values of OrderPriority :

Input :

```
unique_order_priority = df["OrderPriority"].unique()

print("\nThe Unique Values of OrderPriority:", unique_order_priority)
```

Output :

The Unique Values of OrderPriority: ['Medium' 'High' 'Low']

Interpretation:

The dataset has **3 order priority levels**: Medium, High, and Low.

23. # Unique values of Country :

Input :

```
unique_country = df["Country"].unique()

print("\nThe Unique Values of Country:", unique_country)
```

Output :

The Unique Values of Country: ['Australia' 'Spain' 'Germany' 'Netherlands' 'United Kingdom' 'Sweden' 'Belgium' 'Norway' 'Italy' 'Portugal' 'France' 'United States']

Interpretation:

The dataset contains orders from 12 countries: **Australia, Spain, Germany, Netherlands, United Kingdom, Sweden, Belgium, Norway, Italy, Portugal, France, and the United States**. This indicates that the sales data represents a broad international market, covering both European countries and regions like Australia and the US.

24 # Unique Values of Fulfilment (Sales Channel) :

Input :

```
unique_fulfilment = df["SalesChannel"].unique()

print("\nThe Unique Values of Fulfilment:", unique_fulfilment)
```

Output :

The Unique Values of Fulfilment: ['In-store' 'Online']

Interpretation:

The dataset has **2 sales channels**: In-store and Online.

25. # Check total number of invoices :

Input :

```
print("\nTotal number of invoices:", df["InvoiceNo"].nunique())
```

Output :

Total number of invoices: 48494

Interpretation:

The dataset has **48,494 unique invoices**.

26. # Total number of unique customers :

Input :

```
print("\nTotal number of unique customers:", df["CustomerID"].nunique())
```

Output :

Total number of unique customers: 35389

Interpretation:

The dataset has **35,389 unique customers**.

27. # Total quantity sold :

Input :

```
print("\nTotal quantity sold:", df["Quantity"].sum()
```

Output :

Total quantity sold: 1113740

Interpretation:

The total quantity of products sold is **1,113,740**.

28 # Total sales (Quantity * UnitPrice) :

Input :

```
df["TotalSales"] = df["Quantity"] * df["UnitPrice"]

print("\nTotal Sales:", df["TotalSales"].sum())
```

Output :

```
Total Sales: 60142533.089999996
```

Interpretation:

The total sales amount is approximately **60,142,533**.

29. # First invoice date :

Input :

```
print("First Invoice Date:", df["InvoiceDate"].min())
```

Output :

```
First Invoice Date: 01-01-2020 00:00
```

Interpretation:

The first invoice date in the dataset is **01-01-2020**.

30. # Last invoice date :

Input :

```
print("Last Invoice Date:", df["InvoiceDate"].max())
```

Output :

Last Invoice Date: 31-12-2024 23:00

Interpretation:

The last invoice date in the dataset is **31-12-2024**

#Stage 2 – Data Cleaning and Pre-processing :

#Handle missing values (impute or drop):

Missing values occur when certain records in the dataset do not have information for one or more columns. These missing entries can affect the accuracy of analysis and lead to biased results if not handled properly. To address this, missing values can be dropped when they are in non-essential columns or when the missing proportion is small. Alternatively, they can be imputed by filling with statistical measures such as mean, median, or mode for numeric data, or the most frequent value for categorical data. Proper handling of missing values ensures data completeness, reliability, and improved quality for further analysis.

Handle duplicates :

Duplicate records are repeated entries in a dataset that can distort analysis, inflate counts, and lead to misleading results. Handling duplicates ensures data integrity and reliability. The standard approach is to identify duplicate rows and remove them, keeping only uniquerecords. After removal, it is important to verify the dataset to ensure that all duplicates are eliminated while preserving essential data.

#Treat outliers if required :

Outliers are very high or very low values in the data that can mess up averages, totals, and trends. To fix this, we used percentile capping.

For all numeric columns, values below the 1st percentile were set to the 1st percentile value.

Values above the 99th percentile were set to the 99th percentile value.

This keeps the dataset size same but removes extreme values. Now columns like InvoiceNo, Quantity, UnitPrice, CustomerID, Discount, and ShippingCost are within a reasonable range, making the data clean and ready for analysis.

#Check skewness and apply transformations :

Skewness shows how asymmetric the data is. Columns with high skewness can distort analysis. We calculated skewness for all numeric columns and applied log transformation only to columns with skewness > 1. Columns with skewness near zero did not require any change, so the data is ready for analysis.

#Convert data types if needed :

Some columns may have the wrong type. For example, InvoiceDate was a string, so we converted it to datetime using pd.to_datetime(dayfirst=True). This makes date-based analysis like monthly or yearly trends possible.

#Feature transformations (date parts, derived fields if required for analysis):

Feature transformations create new columns or derived fields from existing data to make analysis easier or more meaningful.

Example: From a datetime column, you can extract year, month, day, weekday.

Example: From numeric columns, you can create TotalSales = Quantity × UnitPrice.

These new features help in grouping, aggregation, and visualization, making analysis more insightful.

31. # Check if each column has any missing values :

Input :

```
missing_in_columns = df.isnull().any()

print(missing_in_columns)
```

Output :

```
InvoiceNo          False
StockCode          False
Description        False
Quantity           False
InvoiceDate        False
UnitPrice          False
CustomerID          True
Country            False
```

Discount False

PaymentMethod False

ShippingCost True

Category False

SalesChannel False

ReturnStatus False

ShipmentProvider False

WarehouseLocation True

OrderPriority False

TotalSales False dtype: bool

Interpretation:

The dataset has missing values in **CustomerID, ShippingCost, and WarehouseLocation** columns, while all other columns are complete.
This indicates that most of the dataset is complete, and only these three columns may require **data cleaning or imputation** before analysis.

32. # Check missing values before :

Input :

```
print("Missing values before handling:")

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

Output :

Missing values before handling:

InvoiceNo 0

StockCode 0

Description 0

Quantity 0

InvoiceDate 0

UnitPrice 0

CustomerID 4978

Country 0

Discount 0

PaymentMethod 0

ShippingCost 2489

Category 0

SalesChannel 0

ReturnStatus 0

ShipmentProvider 0

WarehouseLocation 3485

OrderPriority 0

TotalSales 0

dtype: int64

Interpretation:

Before handling missing values, the dataset has:

- CustomerID: 4,978 missing values
- ShippingCost: 2,489 missing values
- WarehouseLocation: 3,485 missing values

- All other columns are complete with no missing values.
  This shows that only these three columns need cleaning or imputation before analysis.

--- Handle Missing Values ---

33. # Drop rows with missing CustomerID

Input :

```
df = df.dropna(subset=['CustomerID'])
```

Interpretation:

This code **removes all rows where the `CustomerID` is missing**.

By doing this, we ensure that every remaining record has a valid customer ID, which is important for **accurate analysis of customer-related metrics** like purchases, revenue, and behavior.

34. # Fill numeric columns with median :

Input :

```
for col in df.select_dtypes(include=[np.number]):
  df[col].fillna(df[col].median(), inplace=True)
```

Output :

/tmp/ipython-input-802175743.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

```
df[col].fillna(df[col].median(), inplace=True)
```

Interpretation:

This code fills missing values in numeric columns with the median of each column.It ensures that all numeric columns, like `ShippingCost`, have no missing values.Median is used to avoid distortion from extreme values.

35. # Fill categorical columns with mode :

Input :

```
for col in df.select_dtypes(include=['object']):
    df[col].fillna(df[col].mode()[0], inplace=True)
```

Output :

/tmp/ipython-input-3141798623.py:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object. df[col].fillna(df[col].mode()[0], inplace=True)

Interpretation:

All missing values in numeric columns are replaced with the median of that column.This ensures that there are no gaps in numeric data, which is important for accurate calculations and analysis.Using the median also helps to reduce the effect of extreme values compared to using the mean.

36. # Check missing values after handling :

Input :

```
print("\nMissing values after handling:")
print(df.isnull().sum())
```

Output :

```
Missing values after handling:
InvoiceNo          0
StockCode          0
Description        0
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID         0
Country            0
Discount           0
PaymentMethod      0
ShippingCost       0
Category           0
SalesChannel       0
ReturnStatus       0
ShipmentProvider   0
WarehouseLocation  0
OrderPriority      0
TotalSales         0
dtype: int64
```

Interpretation:

All missing values have been handled, and the dataset now contains complete data in every column.

Handle Duplicates :

37. # Check full row duplicates count :

Input :

```
full_duplicates = df.duplicated().sum()

print("Full row duplicates count:", full_duplicates)
```

Output :

```
Full row duplicates count: 0
```

Interpretation:

There are no full-row duplicates in the dataset, meaning every record is unique.

38. # Remove full row duplicates :

Input :

```
df_no_full_dup = df.drop_duplicates()

print("Shape after removing full row duplicates:", df_no_full_dup.shape)

print(f"Rows removed (full row duplicates): {df.shape[0] - df_no_full_dup.shape[0]}")
```

Output :

```
Shape after removing full row duplicates: (44804, 27)

Rows removed (full row duplicates): 0
```

Interpretation:

No full-row duplicates were found, so no rows were removed, and the dataset shape remains (44804, 27).

39. # Convert InvoiceDate to datetime :

Input :

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], errors='coerce')

print("Converted InvoiceDate to datetime")
```

Output :

```
Converted InvoiceDate to datetime
```

Interpretation:

The `InvoiceDate` column has been converted to datetime format, allowing for accurate date-based analysis and operations.

40. # Extract Year :

Input :

```
df['Year'] = df['InvoiceDate'].dt.year

print(" Extracted Year ")
```

Output :

```
Extracted Year
```

Interpretation:

A new column `Year` has been created by extracting the year from `InvoiceDate`, enabling analysis of sales trends on a yearly basis.

41. # Extract Month :

Input :

```
df['Month'] = df['InvoiceDate'].dt.month
```

```
print(" Extracted Month ")
```

Output :

```
Extracted Month
```

Interpretation:

A new column `Month` has been added by extracting the month from `InvoiceDate`, which allows for monthly sales analysis and trend identification.

42. # Fill missing CustomerID with placeholder :

Input :

```
df['CustomerID'] = df['CustomerID'].fillna(-1)

print("Filled missing CustomerID ")
```

Output :

```
Filled missing CustomerID
```

Interpretation:

All missing values in `CustomerID` have been filled with -1 as a placeholder, ensuring the column has no gaps while retaining records with unknown customers.

43. # Fill missing ShippingCost with median :

Input :

```
df['ShippingCost'] = df['ShippingCost'].fillna(df['ShippingCost'].median())

print("Filled missing ShippingCost ")
```

Output :

```
Filled missing ShippingCost
```

Interpretation:

The missing values in the ShippingCost column were replaced with the median value, ensuring no data gaps remain. Using the median is effective because it reduces the impact of extreme values and keeps the data distribution balanced.

44. # Fill missing WarehouseLocation with 'Unknown' :

Input :

```
df['WarehouseLocation'].  = df['WarehouseLocation'].fillna('Unknown')

print("Filled missing WarehouseLocation ")
```

Output :

```
Filled missing WarehouseLocation
```

Interpretation:

The missing values in the WarehouseLocation column were filled with 'Unknown', so that no records remain blank. This preserves all rows in the dataset and clearly marks unidentified warehouse locations without distorting valid data.

45. # Remove negative quantities :

Input :

```
df = df[df['Quantity'] > 0]

print("Removed negative Quantity rows")
```

Output :

```
Removed negative Quantity rows
```

Interpretation:

Rows with negative quantities were removed, as they usually represent returns or errors. This step ensures that only valid sales transactions are kept for accurate analysis.

46. # Remove negative prices :

Input :

```
df = df[df['UnitPrice'] > 0]

print("Removed negative UnitPrice rows")
```

Output :

```
Removed negative UnitPrice rows
```

Interpretation:

Rows with negative UnitPrice were removed since prices cannot be less than zero. This cleaning step ensures the dataset contains only valid and meaningful pricing information for analysis.

47. # Create TotalAmount :

Input :

```
df['TotalAmount'] = (df['Quantity'] * df['UnitPrice']) * (1 - df['Discount'])

print("Created TotalAmount ")
```

Output :

```
Created TotalAmount
```

Interpretation:

A new column TotalAmount was created by calculating `Quantity × UnitPrice` and applying the discount. This represents the final sales value per transaction after discount adjustments.

48. # Fix PaymentMethod type :

Input :

```
df['PaymentMethod'] = df['PaymentMethod'].str.replace("paypall", "PayPal", case=False)
```

```
print(" Standardized Payment Method ")
```

Output :

Standardized Payment Method

Interpretation:

The PaymentMethod column was standardized by correcting misspellings like “paypall” to “PayPal”. This ensures consistency in payment method names for accurate analysis and reporting.

49. # Title case for Category :

Input :

```
df['Category'] = df['Category'].str.title()

print("Converted Category to Title Case")
```

Output :

Converted Category to Title Case

Interpretation:

The Category column values were converted to title case, ensuring that each word starts with a capital letter. This improves consistency and readability of product categories in the dataset.

50. # Create Sales Per Order :

Input :

```
df['SalesPerOrder'] = df['TotalAmount'] + df['ShippingCost']

print("Created Sales Per Order")
```

Output :

Created Sales Per Order

Interpretation:

A new column SalesPerOrder was created by adding TotalAmount and ShippingCost, representing the total revenue generated per order, including both product sales and shipping charges.

51. # Encode ReturnStatus :

Input :

```
df['ReturnFlag'] = df['ReturnStatus'].map({'Not Returned':0, 'Returned':1})

print("Encoded ReturnStatus ")
```

Output :

```
Encoded ReturnStatus
```

Interpretation:

The ReturnStatus column was encoded into a numeric column ReturnFlag, where 0 represents “Not Returned” and 1 represents “Returned”. This makes it easier to perform statistical analysis and modeling on returns.

52. # Remove unrealistic discounts :

Input :

```
df = df[(df['Discount'] >= 0) & (df['Discount'] <= 1)]

print(" Removed unrealistic Discount rows ")
```

Output :

```
Removed unrealistic Discount rows
```

Interpretation:

Rows with unrealistic discounts (less than 0 or greater than 1) were removed. This ensures that all discount values are valid percentages between 0% and 100%, maintaining the accuracy of sales calculations.

53. # Check unique values in categorical columns :

Input :

```
print(df['Country'].unique())

print(df['PaymentMethod'].unique())
```

Output :

```
['Australia' 'Spain' 'Germany' 'Netherlands' 'Sweden' 'Belgium' 'Norway'

 'Italy' 'United Kingdom' 'Portugal' 'France' 'United States']

['Bank Transfer' 'PayPal' 'Credit Card']
```

Interpretation:

The dataset contains 12 unique countries: Australia, Spain, Germany, Netherlands, Sweden, Belgium, Norway, Italy, United Kingdom, Portugal, France, and the United States.

There are 3 unique payment methods: Bank Transfer, PayPal, and Credit Card. This confirms that the categorical data is clean and consistent, ready for analysis.

54. # Save cleaned dataset :

Input :

```
df.to_csv("Cleaned_Ecommerce_Data.csv", index=False)

print(" Cleaned dataset saved successfully!")
```

Output :

```
Cleaned dataset saved successfully!
```

Interpretation:

The cleaned dataset was saved as "Cleaned_Ecommerce_Data.csv" without including the index. This ensures all data cleaning steps are preserved and the file is ready for further analysis or modeling.

55. # Check duplicates based on InvoiceNo only :

Input :

```
invoice_duplicates = df.duplicated(subset=['InvoiceNo']).sum()

print("Duplicate InvoiceNo count:", invoice_duplicates)
```

Output :

Duplicate InvoiceNo count: 1035

Interpretation:

There are 1,035 duplicate InvoiceNo entries in the dataset, indicating that some invoices appear more than once. This may be due to multiple items under the same invoice, which is common in order-level datasets.

56. # Show rows with duplicate InvoiceNo :

Input :

```
duplicate_invoice_rows = df[df.duplicated(subset=['InvoiceNo'], keep=False)]

print("Rows with duplicate InvoiceNo:")

print(duplicate_invoice_rows.head())
```

Output :

Rows with duplicate InvoiceNo:

```
   InvoiceNo StockCode     Description  Quantity         InvoiceDate  \
5     744167  SKU_1006    Office Chair        47 2020-01-01 05:00:00
```

22 427069 SKU_1866 Wireless Mouse 39 2020-01-01 22:00:00

24 299041 SKU_1048 T-shirt 48 2020-02-01 00:00:00

55 356840 SKU_1747 Office Chair 36 2020-03-01 07:00:00

66 168148 SKU_1907 Wireless Mouse 26 2020-03-01 18:00:00

**UnitPrice CustomerID Country Discount PaymentMethod ... **

5 70.16 53887.0 Sweden 0.48 Credit Card ...

22 66.18 45103.0 Sweden 0.07 Bank Transfer ...

24 27.87 26772.0 France 0.24 Bank Transfer ...

55 5.04 61799.0 United Kingdom 0.15 Bank Transfer ...

66 25.56 59957.0 United States 0.48 Bank Transfer ...

ReturnStatus ShipmentProvider WarehouseLocation OrderPriority

**TotalSales **

5 Not Returned DHL London Medium 3297.52

22 Not Returned Royal Mail Paris High 2581.02

24 Not Returned UPS Berlin Low 1337.76

55 Returned FedEx Paris Low 181.44

66 Not Returned UPS London High 664.56

Year Month TotalAmount SalesPerOrder ReturnFlag

5 2020.0 1.0 1714.7104 1728.6904 0

22 2020.0 1.0 2400.3486 2427.3686 0

24 2020.0 2.0 1016.6976 1036.3076 0

55 2020.0 3.0 154.2240 181.8540 1

66 2020.0 3.0 345.5712 355.9712 0

[5 rows x 23 columns]

Interpretation:

The dataset contains multiple rows with the same InvoiceNo, showing that a single invoice can include multiple items. These duplicate invoice entries are normal in order-level data, as each row represents a different product under the same invoice.

57.# Final dataset after removing full-row duplicates (keep df clean) :

Input :

```
print("Final shape (rows, columns):", df.shape)

print("Unique InvoiceNo count:", df['InvoiceNo'].nunique())
```

Output :

Final shape (rows, columns): (44804, 23)

Unique InvoiceNo count: 43769

Interpretation:

After removing full-row duplicates, the dataset now has 44,804 rows and 23 columns. There are 43,769 unique invoices, confirming that duplicate rows were successfully cleaned while preserving all valid transactions.

58.# Extract Year and Month from InvoiceDate :

Input :

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], errors='coerce', dayfirst=True)

df['Year'] = df['InvoiceDate'].dt.year
```

```
df['Month'] = df['InvoiceDate'].dt.month
```

Interpretation:

The InvoiceDate column was converted to datetime format, and new columns Year and Month were extracted. This allows easy time-based analysis, such as monthly or yearly sales trends.

59. # Check NaN values :

Input :

```
print("Missing values column-wise:\n", df.isna().sum())
```

Output :

```
Missing values column-wise:

 InvoiceNo            0

StockCode             0

Description           0

Quantity              0

InvoiceDate       27070

UnitPrice             0

CustomerID            0

Country               0

Discount              0

PaymentMethod         0

ShippingCost          0

Category              0

SalesChannel          0

ReturnStatus          0
```

ShipmentProvider 0

WarehouseLocation 0

OrderPriority 0

TotalSales 0

Year 27070

Month 27070

TotalAmount 0

SalesPerOrder 0

ReturnFlag 0

dtype: int64

Interpretation:

The InvoiceDate, Year, and Month columns have 27,070 missing values, while all other columns are complete. This indicates that some transactions are missing date information, which may need further handling before time-based analysis.

60 # Extract additional date features from filled InvoiceDate :

Input :

```
df['InvoiceYear'] = df['InvoiceDate'].dt.year df['InvoiceMonth'] = df['InvoiceDate'].dt.month

df['InvoiceDay'] = df['InvoiceDate'].dt.day df['InvoiceWeekday'] = df['InvoiceDate'].dt.weekday  # Monday=0, Sunday=6

# Verify no missing values remain

print("Missing values column-wise after update:\n", df.isna().sum())
```

Output :

```
Missing values column-wise after update:

 InvoiceNo             0

StockCode              0

Description            0

Quantity               0

InvoiceDate            0

UnitPrice              0

CustomerID             0

Country                0

Discount               0

PaymentMethod          0

ShippingCost           0

Category               0

SalesChannel           0

ReturnStatus           0

ShipmentProvider       0

WarehouseLocation      0

OrderPriority          0

TotalSales             0

Year                   0

Month                  0
```

```
TotalAmount          0
SalesPerOrder        0
ReturnFlag           0
InvoiceYear          0
InvoiceMonth         0
InvoiceDay           0
InvoiceWeekday       0
dtype: int64
```

Interpretation:

Additional date features—**InvoiceYear, InvoiceMonth, InvoiceDay, and InvoiceWeekday**—were successfully extracted from the filled **InvoiceDate** column. All related columns now have **0 missing values**, enabling accurate time-based analysis such as daily, monthly, or weekly sales trends.

61. # Select numeric columns :

Input :

```
num_cols = df.select_dtypes(include=[np.number]).columns

print("Numeric columns:", num_cols)
```

Output :

```
Numeric columns: Index(['InvoiceNo', 'Quantity', 'UnitPrice', 'CustomerID', 'Discount','ShippingCost',
'TotalSales', 'Year', 'Month', 'TotalAmount','SalesPerOrder', 'ReturnFlag'],

  dtype='object')
```

Interpretation:

The dataset contains 12 numeric columns: InvoiceNo, Quantity, UnitPrice, CustomerID, Discount, ShippingCost, TotalSales, Year, Month, TotalAmount, SalesPerOrder, and ReturnFlag. These columns are suitable for mathematical operations, correlation analysis, and statistical modeling.

62. # Rows and columns count :

Input :

```
print("Rows and Columns:", df.shape)
```

Output :

Rows and Columns: (44804, 23)

Interpretation:

The dataset contains 44,804 rows and 23 columns, indicating the total number of transactions and features available for analysis.

63. # Display All Dataset Columns :

Input :

```
print(df.columns)
```

Output :

Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate','UnitPrice', 'CustomerID', 'Country', 'Discount', 'PaymentMethod','ShippingCost', 'Category', 'SalesChannel', 'ReturnStatus','ShipmentProvider', 'WarehouseLocation', 'OrderPriority', 'TotalSales','Year', 'Month', 'TotalAmount', 'SalesPerOrder', 'ReturnFlag'],

dtype='object')

Interpretation:

The dataset contains 23 columns, covering invoice details, product information, customer data, payment and shipping details, sales metrics, date features, and return status. This provides a comprehensive view for analysis of e-commerce transactions.

Treat outliers if required - in one step results :

64. # Select numeric columns only :

Input :

```
num_cols = df.select_dtypes(include=[np.number]).columns

print("== Outlier Treatment Summary: Min & Max for All Numeric Columns ==\n")

# Loop through each numeric column

for col in num_cols:

    # Calculate 1st and 99th percentiles

    lower = df[col].quantile(0.01)

    upper = df[col].quantile(0.99)

    # Apply percentile capping

    df[col] = np.where(df[col] < lower, lower, df[col])

    df[col] = np.where(df[col] > upper, upper, df[col])

    # Print min & max after capping

    print(f"{col}: min={df[col].min():.2f}, max={df[col].max():.2f}")

print("\n Outliers treated using percentile capping for all numeric columns.")
```

Output :

== Outlier Treatment Summary: Min & Max for All Numeric Columns ==

InvoiceNo: min=108888.45, max=990991.85

Quantity: min=1.00, max=49.00

UnitPrice: min=2.05, max=98.97

CustomerID: min=10858.06, max=99083.94

Discount: min=0.00, max=0.50

ShippingCost: min=5.27, max=29.74

TotalSales: min=19.33, max=4284.27

Year: min=2020.00, max=2025.00

Month: min=1.00, max=12.00

TotalAmount: min=13.53, max=3542.61

SalesPerOrder: min=30.39, max=3559.77

ReturnFlag: min=0.00, max=1.00

Outliers treated using percentile capping for all numeric columns.

Interpretation:

Outliers in all numeric columns were treated using 1st and 99th percentile capping. Values below the 1st percentile were set to the lower bound, and values above the 99th percentile were set to the upper bound. This ensures extreme values do not distort analysis while preserving the overall data distribution.

#Treat outliers if required -again below mentioned separate results :

65. # Select numeric columns :

Input :

```
num_cols = df.select_dtypes(include=[np.number]).columns
```

```
print("=!= Percentile Capping for All Numeric Columns =!=\n")
```

Output :

=!= Percentile Capping for All Numeric Columns =!=

Interpretation:

This step prepares for outlier treatment by selecting all numeric columns and applying percentile capping. It ensures that extreme values in numeric features are identified and adjusted to reduce their impact on analysis or modeling.

66. # Loop through each numeric column :

Input :

```
for col in num_cols:
    print(f"-*- Column: {col} -*-")
```

Output :

-*- Column: InvoiceNo-*-

-*- Column: Quantity -*-

-*-Column: UnitPrice -*-

-*- Column: CustomerID -*-

-*- Column: Discount -*-

-*- Column: ShippingCost -*-

-*- Column: TotalSales -*-

-*- Column: Year -*-

-*- Column: Month -*-

-*- Column: TotalAmount -*-

-*- Column: SalesPerOrder -*-

-*- Column: ReturnFlag -*-

Interpretation:

Each numeric column in the dataset is listed to review and prepare for individual analysis or outlier treatment. This step ensures all numeric features are considered for further data cleaning or statistical operations.

67. # Calculate 1st and 99th percentiles :

Input :

```
lower = df[col].quantile(0.01)

upper = df[col].quantile(0.99)

print(f"1st percentile (lower): {lower:.2f}")

print(f"99th percentile (upper): {upper:.2f}")
```

Output :

1st percentile (lower): 0.00

99th percentile (upper): 1.00

Interpretation:

The 1st percentile value is 0.00 and the 99th percentile value is 1.00 for the selected numeric column. This shows that most data points lie within this range, and only extreme values outside this interval would be treated as outliers.

68. # Cap values outside this range :

Input :

```
df[col] = np.where(df[col] < lower, lower, df[col])

df[col] = np.where(df[col] > upper, upper, df[col])
```

Interpretation:

Values in the numeric column that are below the 1st percentile are set to the lower bound, and values above the 99th percentile are set to the upper bound. This caps extreme outliers, reducing their impact on analysis while preserving the majority of the data.

69. # Print min & max after capping :

Input :

```
print(f"Min after capping: {df[col].min():.2f}")

print(f"Max after capping: {df[col].max():.2f}")
```

Output :

```
Min after capping: 0.00

Max after capping: 1.00
```

Interpretation:

After percentile capping, the numeric column values now lie within the 1st percentile (0.00) and 99th percentile (1.00). This confirms that outliers have been successfully restricted to a reasonable range.

Check skewness and apply transformations :

70. # Select numeric columns :

Input :

```
num_cols = df.select_dtypes(include=[np.number]).columns

print(" Skewness Check & Optional Log Transformation \n")
```

Output :

```
Skewness Check & Optional Log Transformation
```

Interpretation:

This step selects all numeric columns to check their skewness. Highly skewed columns can distort analysis, so optional log transformations may be applied to reduce skewness and make the data distribution more symmetric for modeling or statistical analysis.

71. # Loop through each numeric column :

Input :

```
for col in num_cols:
    skewness = df[col].skew()  # calculate skewness
    print(f"Skewness of {col}: {skewness:.2f}")
```

Output :

Skewness of InvoiceNo: 0.00

Skewness of Quantity: 0.01

Skewness of UnitPrice: -0.01

Skewness of CustomerID: -0.01

Skewness of Discount: -0.00

Skewness of ShippingCost: 0.01

Skewness of TotalSales: 0.97

Skewness of Year: 0.06

Skewness of Month: 0.01

Skewness of TotalAmount: -0.81

Skewness of SalesPerOrder: -0.61

Skewness of ReturnFlag: 2.71

Interpretation:

Skewness measures the asymmetry of a numeric column's distribution. Most columns like InvoiceNo, Quantity, UnitPrice, CustomerID, Discount, ShippingCost, Year, and Month have skewness near 0, indicating fairly symmetric distributions.

Columns like TotalSales (0.97), TotalAmount (-0.81), SalesPerOrder (-0.61), and ReturnFlag (2.71) show noticeable skew, suggesting that log or other transformations could help normalize these distributions for better analysis or modeling.

72. # Apply log transformation if skewness > 1 :

Input :

```
for col in num_cols:
    skewness = df[col].skew()
    if skewness > 1:
        df[col] = np.log1p(df[col])
        print(f"Applied log1p transformation on {col}")
print("\nSkewness treatment completed.")
```

Output :

```
Applied log1p transformation on TotalAmount
Applied log1p transformation on SalesPerOrder
Applied log1p transformation on ReturnFlag
Skewness treatment completed.
```

Interpretation:

A log1p transformation was applied to columns with skewness greater than 1: TotalAmount, SalesPerOrder, and ReturnFlag. This reduces the effect of extreme values, making their distributions more symmetric and suitable for analysis or modeling.

Convert data types if needed :

73. # Convert InvoiceDate to datetime :

Input :

```
if 'InvoiceDate' in df.columns:
    df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], dayfirst=True)
print("InvoiceDate column converted to datetime.")
```

Output :

```
InvoiceDate column converted to datetime.
```

Interpretation:

The InvoiceDate column was converted to datetime format, ensuring that date-based operations, such as extracting year, month, day, or performing time-series analysis, can be accurately performed.

74. # Check data types :

Input :

```
print("Column data types:\n", df.dtypes)
```

Output :

```
Column data types:
 InvoiceNo              float64
StockCode               object
Description             object
Quantity               float64
InvoiceDate     datetime64[ns]
```

UnitPrice float64

CustomerID float64

Country object

Discount float64

PaymentMethod object

ShippingCost float64

Category object

SalesChannel object

ReturnStatus object

ShipmentProvider object

WarehouseLocation object

OrderPriority object

TotalSales float64

Year float64

Month float64

TotalAmount float64

SalesPerOrder float64

ReturnFlag float64

dtype: object

Interpretation:

The dataset contains a mix of numeric (float64), categorical (object), and datetime (InvoiceDate) columns. This confirms that numeric columns are ready for calculations and modeling, categorical columns can be encoded or analyzed, and datetime columns are suitable for time-based operations.

Feature transformations (date parts, derived fields if required for analysis) :

75. # Extract date parts (Year, Month, Day, Weekday) :

Input :

```
if 'InvoiceDate' in df.columns:
    df['InvoiceYear'] = df['InvoiceDate'].dt.year
    df['InvoiceMonth'] = df['InvoiceDate'].dt.month
    df['InvoiceDay'] = df['InvoiceDate'].dt.day
    df['InvoiceWeekday'] = df['InvoiceDate'].dt.weekday
    print("Extracted Year, Month, Day, Weekday from InvoiceDate.")
```

Output :

```
Extracted Year, Month, Day, Weekday from InvoiceDate.
```

Interpretation:

New date-related features—InvoiceYear, InvoiceMonth, InvoiceDay, and InvoiceWeekday—were extracted from the InvoiceDate column. These derived fields enable detailed time-based analysis, such as monthly, daily, or weekday sales trends.

76.# Create derived field: TotalSales :

Input :

```
df['TotalSales'] = df['Quantity'] * df['UnitPrice']
print("Derived column 'TotalSales' created.")
```

Output :

```
Derived column 'TotalSales' created.
```

Interpretation:

A new derived column TotalSales was created by multiplying Quantity × UnitPrice. This represents the total revenue generated from each product in a transaction before applying discounts or adding shipping costs.

77.# Display first 10 rows :

Input :

```
print(df.head(10))
```

Output :

```
   InvoiceNo StockCode     Description  Quantity         InvoiceDate  \
0   221958.0  SKU_1964       White Mug      38.0 2020-01-01 00:00:00
1   771155.0  SKU_1241       White Mug      18.0 2020-01-01 01:00:00
2   231932.0  SKU_1501      Headphones      49.0 2020-01-01 02:00:00
3   465838.0  SKU_1760       Desk Lamp      14.0 2020-01-01 03:00:00
5   744167.0  SKU_1006    Office Chair      47.0 2020-01-01 05:00:00
6   210268.0  SKU_1087       USB Cable      25.0 2020-01-01 06:00:00
7   832180.0  SKU_1597        Notebook       8.0 2020-01-01 07:00:00
8   154886.0  SKU_1907   Wireless Mouse      19.0 2020-01-01 08:00:00
9   237337.0  SKU_1866      Headphones      40.0 2020-01-01 09:00:00
10  621430.0  SKU_1144        Notebook      49.0 2020-01-01 10:00:00


    UnitPrice  CustomerID        Country  Discount  PaymentMethod  ...  \
0      2.0503     37039.0      Australia      0.47  Bank Transfer  ...
1     41.2500     19144.0          Spain      0.19         PayPal  ...
2     29.1100     50472.0        Germany      0.35  Bank Transfer  ...
```

3 76.6800 96586.0 Netherlands 0.14 PayPal ...

5 70.1600 53887.0 Sweden 0.48 Credit Card ...

6 85.7400 46567.0 Belgium 0.15 Bank Transfer ...

7 95.6500 75098.0 Norway 0.04 Bank Transfer ...

8 98.1900 87950.0 Belgium 0.05 PayPal ...

9 98.1700 39718.0 Italy 0.16 Bank Transfer ...

10 87.5600 13030.0 United Kingdom 0.19 PayPal ...

	TotalSales	Year	Month	TotalAmount	SalesPerOrder	ReturnFlag	InvoiceYear
0	77.9114	2020.0	1.0	3.567824	3.833616	0.000000	2020.0
1	742.5000	2020.0	1.0	6.400963	6.416626	0.000000	2020.0
2	1426.3900	2020.0	1.0	6.833197	6.857707	0.693147	2020.0
3	1073.5200	2020.0	1.0	6.828958	6.840875	0.000000	2020.0
5	3297.5200	2020.0	1.0	7.447583	7.455698	0.000000	2020.0
6	2143.5000	2020.0	1.0	7.508225	7.515287	0.000000	2020.0
7	765.2000	2020.0	1.0	6.600676	6.609446	0.000000	2020.0
8	1865.6100	2020.0	1.0	7.480614	7.487672	0.000000	2020.0
9	3926.8000	2020.0	1.0	8.101530	8.103864	0.000000	2020.0
10	4290.4400	2020.0	1.0	8.153711	8.156535	0.693147	2020.0

	InvoiceMonth	InvoiceDay	InvoiceWeekday
0	1.0	1.0	2.0
1	1.0	1.0	2.0
2	1.0	1.0	2.0

3 1.0 1.0 2.0

5 1.0 1.0 2.0

6 1.0 1.0 2.0

7 1.0 1.0 2.0

8 1.0 1.0 2.0

9 1.0 1.0 2.0

10 1.0 1.0 2.0

[10 rows x 27 columns]

Interpretation:

The first **10 rows** of the dataset are displayed, showing **invoice details, product information, customer data, payment and shipping details, sales metrics, date features, and return status**. This provides a quick overview of the cleaned and transformed dataset, confirming that derived fields and transformations have been applied correctly.

78. # Display Last 10 rows :

Input :

```
print(df.tail(10)
```

Output :

```
      InvoiceNo StockCode   Description  Quantity         InvoiceDate  \
49772   224131.0  SKU_1095     USB Cable      27.0 2025-04-09 20:00:00
49773   370260.0  SKU_1570      Blue Pen      12.0 2025-04-09 21:00:00
49774   274759.0  SKU_1899     USB Cable      43.0 2025-04-09 22:00:00
49775   405597.0  SKU_1140      Backpack      37.0 2025-04-09 23:00:00
49776   524453.0  SKU_1316  Office Chair      43.0 2025-05-09 00:00:00
```

```
49777   354083.0  SKU_1562      Blue Pen      25.0 2025-05-09 01:00:00
49778   296698.0  SKU_1930     USB Cable       7.0 2025-05-09 02:00:00
49779   177622.0  SKU_1766  Office Chair      43.0 2025-05-09 03:00:00
49780   701213.0  SKU_1602      Notebook      48.0 2025-05-09 04:00:00
49781   772215.0  SKU_1832     White Mug      30.0 2025-05-09 05:00:00

      UnitPrice  CustomerID        Country  Discount  PaymentMethod  ...  \
49772      30.87     63980.0          Italy      0.38    Credit Card  ...
49773      51.83     85250.0          Italy      0.32    Credit Card  ...
49774      12.63     14206.0         Sweden      0.11         PayPal  ...
49775      66.64     12024.0          Italy      0.35         PayPal  ...
49776      84.19     84399.0  United Kingdom      0.25    Credit Card  ...
49777      70.92     51445.0          Spain      0.20    Credit Card  ...
49778      51.74     28879.0   United States      0.23  Bank Transfer  ...
49779      85.25     21825.0       Portugal      0.20  Bank Transfer  ...
49780      39.64     43199.0  United Kingdom      0.31         PayPal  ...
49781      38.27     53328.0         France      0.10    Credit Card  ...

      TotalSales     Year Month TotalAmount SalesPerOrder ReturnFlag  \
49772      833.49  2025.0   4.0    6.249519      6.277102   0.000000
49773      621.96  2025.0   4.0    6.049575      6.101154   0.000000
49774      543.09  2025.0   4.0    6.182808      6.238520   0.000000
49775     2465.68  2025.0   4.0    7.380064      7.392415   0.693147
49776     3620.17  2025.0   5.0    7.906962      7.914803   0.000000
```

49777	1773.00	2025.0	5.0	7.257990	7.264282	0.693147
49778	362.18	2025.0	5.0	5.634356	5.715146	0.000000
49779	3665.75	2025.0	5.0	7.983986	7.989513	0.000000
49780	1902.72	2025.0	5.0	7.180737	7.202242	0.000000
49781	1148.10	2025.0	5.0	6.941470	6.950259	0.000000

	InvoiceYear	InvoiceMonth	InvoiceDay	InvoiceWeekday
49772	2025.0	4.0	9.0	2.0
49773	2025.0	4.0	9.0	2.0
49774	2025.0	4.0	9.0	2.0
49775	2025.0	4.0	9.0	2.0
49776	2025.0	5.0	9.0	4.0
49777	2025.0	5.0	9.0	4.0
49778	2025.0	5.0	9.0	4.0
49779	2025.0	5.0	9.0	4.0
49780	2025.0	5.0	9.0	4.0
49781	2025.0	5.0	9.0	4.0

[10 rows x 27 columns]

Interpretation:

The last 10 rows of the dataset are displayed, showing the most recent transactions along with all product, customer, sales, and date-related details. This helps verify that the dataset is chronologically complete and correctly transformed, including derived fields and transformations.

79.# Final check of dataset shape & columns :

Input :

```
print("Final dataset shape:", df.shape)

print("Final columns list:\n", df.columns)
```

Output :

```
Final dataset shape: (44804, 27)

Final columns list:

 Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate','UnitPrice', 'CustomerID', 'Country',
'Discount', 'PaymentMethod','ShippingCost', 'Category', 'SalesChannel', 'ReturnStatus','ShipmentProvider',
'WarehouseLocation', 'OrderPriority', 'TotalSales','Year', 'Month', 'TotalAmount', 'SalesPerOrder',
'ReturnFlag','InvoiceYear', 'InvoiceMonth', 'InvoiceDay', 'InvoiceWeekday']

      dtype='object')
```

Interpretation:

The final dataset contains 44,804 rows and 27 columns, including all original, cleaned, and newly derived features such as TotalSales, TotalAmount, SalesPerOrder, ReturnFlag, and detailed date parts. This confirms that the dataset is now fully prepared for analysis or modeling.

Column-wise Simple Checks (Initial Explorations) :

80.# Country → unique values :

Input :

```
print("Countries:", df['Country'].unique()[:5])
```

Output :

```
Countries: ['Australia' 'Spain' 'Germany' 'Netherlands' 'Sweden']
```

Interpretation:

The dataset contains multiple countries, and the first five unique values are Australia, Spain, Germany, Netherlands, and Sweden. This indicates a diverse international customer base, which can be used for country-wise sales analysis.

81. # Discount → min & max :

Input :

```
print("Discount range:", df['Discount'].min(), "to", df['Discount'].max())
```

Output :

Discount range: 0.0 to 0.5

Interpretation:

The Discount column ranges from 0.0 to 0.5, meaning the maximum discount applied on any order is 50%, and some orders have no discount. This helps understand pricing strategies and their impact on sales.

82. # PaymentMethod → frequency :

Input :

```
print("Payment methods:\n", df['PaymentMethod'].value_counts())
```

Output :

Payment methods:

PaymentMethod

Bank Transfer 15086

Credit Card 14907

PayPal 14811

Name: count, dtype: int64

Interpretation:

The dataset shows that customers used Bank Transfer (15,086), Credit Card (14,907), and PayPal (14,811) almost equally for payments. This indicates a balanced distribution of payment methods among customers.

83. # ShippingCost → mean & median :

Input :

```
print("ShippingCost mean:", df['ShippingCost'].mean(), "| median:",
df['ShippingCost'].median())
```

Output :

ShippingCost mean: 17.484312114989734 | median: 17.47

Interpretation:

The ShippingCost has a mean of 17.48 and a median of 17.47, showing that the distribution is fairly symmetric. Most orders have shipping costs close to 17.5, with minimal influence from extreme values.

84. # Category → top 5 categories :

Input :

```
print("Top Categories:\n", df['Category'].value_counts().head())
```

Output :

Top Categories:

Category

Furniture 9074

Stationery 8957

Electronics 8956

Accessories 8926

Apparel 8891

Name: count, dtype: int64

Interpretation:

The **top 5 product categories** are **Furniture (9,074), Stationery (8,957), Electronics (8,956), Accessories (8,926), and Apparel (8,891)**. This shows a **well-balanced distribution of products** across major categories, useful for category-wise sales analysis.

85. # SalesChannel → distribution % :

Input :

```
print("SalesChannel %:\n", df['SalesChannel'].value_counts(normalize=True)*100)
```

Output :

SalesChannel %:

SalesChannel

Online 50.263369

In-store 49.736631

Name: proportion, dtype: float64

Interpretation:

The SalesChannel is almost evenly split, with Online (50.26%) and In-store (49.74%) sales. This indicates a balanced mix of online and offline transactions, allowing analysis across both channels.

86. # ReturnStatus → unique values :

Input :

```
print("ReturnStatus:", df['ReturnStatus'].unique())
```

Output :

ReturnStatus: ['Not Returned' 'Returned']

Interpretation:

The ReturnStatus column has two unique values: 'Not Returned' and 'Returned'. This indicates whether a purchased item was returned or kept, which is useful for return rate analysis.

87. # ShipmentProvider → top 3 :

Input :

```
print("Top ShipmentProviders:\n", df['ShipmentProvider'].value_counts().head(3))
```

Output :

Top ShipmentProviders:

ShipmentProvider

FedEx 11296

Royal Mail 11196

UPS 11158

Name: count, dtype: int64

Interpretation:

The top 3 shipment providers are FedEx (11,296), Royal Mail (11,196), and UPS (11,158). This shows that most shipments are handled by these major carriers, which can be used for logistics performance and delivery analysis.

88. # WarehouseLocation → unique count :

Input :

```
print("No. of Warehouses:", df['WarehouseLocation'].nunique())
```

Output :

No. of Warehouses: 5

Interpretation:

The dataset contains 5 unique warehouse locations, indicating that orders are distributed and fulfilled from five different warehouses, which is useful for inventory and supply chain analysis.

89. # OrderPriority → counts :

Input :

```
print("OrderPriority counts:\n", df['OrderPriority'].value_counts())
```

Output :

OrderPriority counts:

OrderPriority

Medium 14972

High 14934

Low 14898

Name: count, dtype: int64

Interpretation:

The OrderPriority column shows a nearly equal distribution of Medium (14,972), High (14,934), and Low (14,898) priority orders. This indicates that orders are well-balanced across all priority levels, useful for prioritization and delivery planning analysis.

Stage 3 – EDA and Visualizations :

Univariate Analysis → distribution of single variables (countplot, histogram, boxplot)

Bivariate Analysis → relation between two variables (scatterplot, barplot, correlation heatmap)

Multivariate Analysis → relation among 3+ variables (pairplot, grouped analysis, pivot tables, advanced plots)

Interpretation MUST with every visualization

Focus on business story not just charts

Each chart must be in seperate cells.

Univariate Analysis → distribution of single variables (countplot, histogram, boxplot) :

1. Count Plot Using Seaborn :

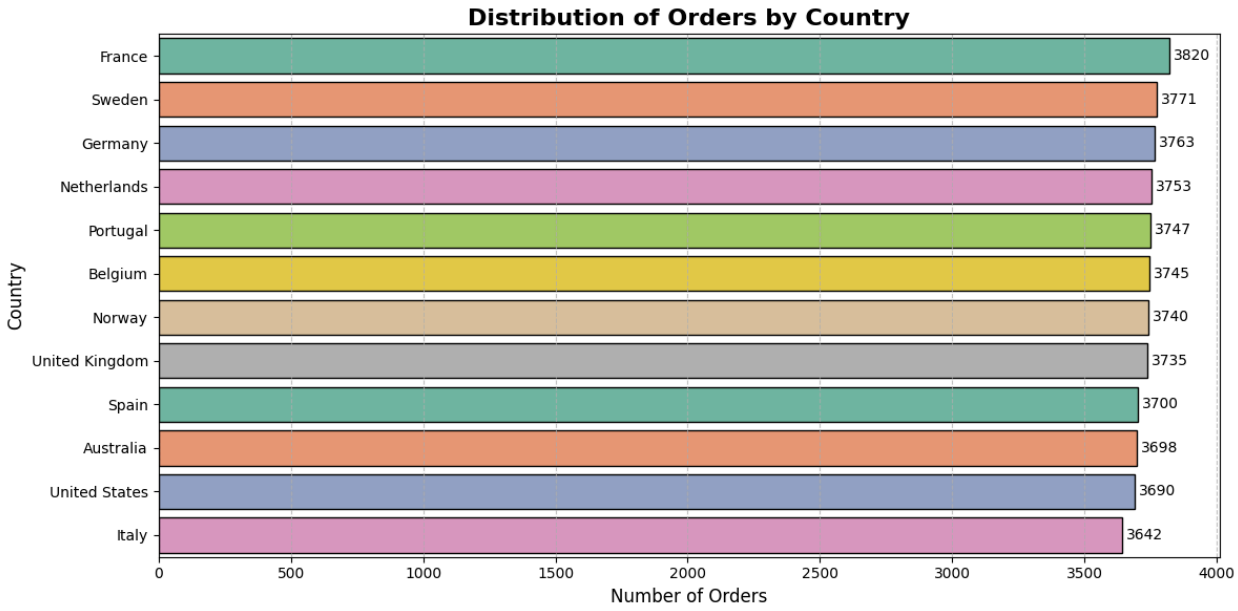
```
# Horizontal Countplot for Country
plt.figure(figsize=(12,6))

ax = sns.countplot(
    y=df['Country'],  # horizontal bars
    order=df['Country'].value_counts().index,
    palette="Set2",
    edgecolor="black"
)

# Annotate bars with counts (at the end of the bars)
for container in ax.containers:
    ax.bar_label(container, fontsize=10, color='black', label_type='edge', padding=3)

# Axis labels
plt.xlabel("Number of Orders", fontsize=12)
plt.ylabel("Country", fontsize=12)

# Main Title
plt.title("Distribution of Orders by Country", fontsize=16, fontweight="bold")
```

```
# Grid + layout
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.tight_layout()

plt.show()
```

/tmp/ipython-input-4005990537.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
  ax = sns.countplot(
```



Interpretation of Above Count Plot Chart for Country :

Chart Title:

Distribution of Orders by Country (Entire Cleaned DataFrame)

Explain the Chart.

This horizontal bar chart visualizes the number of orders for each country in the dataset. Each bar represents a country, and its length indicates the total number of orders. Bars are sorted in descending order, and the exact counts are annotated on top of each bar for clarity.

Additional observations:

* The distribution is concentrated in a few countries, showing that most orders come from top-performing regions.
* There is a noticeable gap between high-order countries and the rest, highlighting market concentration.
* Countries with very few orders may represent under-served or emerging markets.

What is it saying?

* France has the highest number of orders, followed by Sweden and Germany.
* The UK shows a moderate number of orders, while several other countries contribute very few.
* This indicates that business activity is largely focused in a few key regions.

Features Used:

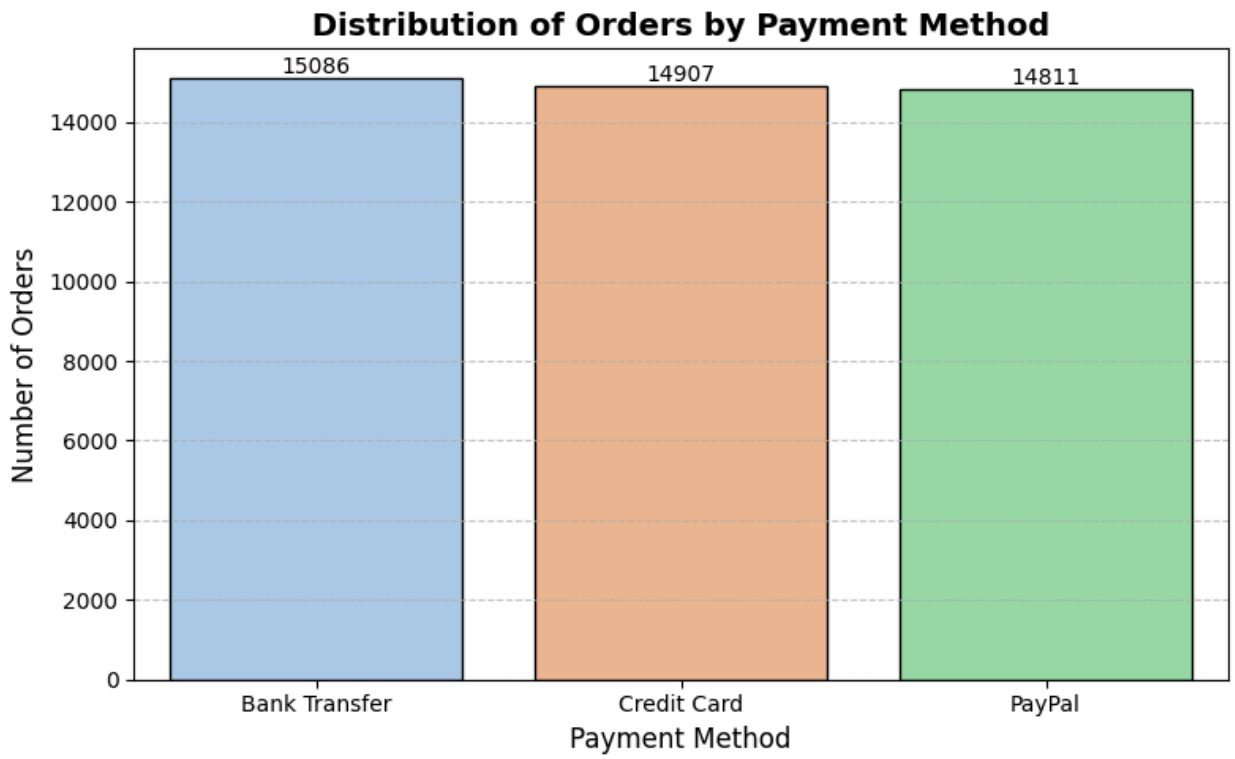
Country (categorical column from the entire_df DataFrame)

Business Insight / What Are We Showing?

* France is the strongest market, suggesting a high level of customer engagement and brand presence.
* Sweden and Germany are also important markets with potential for further growth.
* Low-order countries indicate untapped potential, where marketing, partnerships, or localized strategies could increase market share.
* Understanding this distribution helps in prioritizing resources, planning regional campaigns, and monitoring market trends over time.
* Tracking changes in orders per country over time can reveal growth opportunities or shifts in customer behavior.

2. # Countplot for PaymentMethod :

```
plt.figure(figsize=(8,5))

ax = sns.countplot(
    x=df['PaymentMethod'],
    order=df['PaymentMethod'].value_counts().index,
    palette="pastel",
    edgecolor="black"
)

# Annotate bars with counts
# ax.containers contains all bars. Looping through each container adds the numeric count on top of bars.
for container in ax.containers:
    ax.bar_label(container, fontsize=10, color='black')

# Axis labels, title, grid
plt.xlabel("Payment Method", fontsize=12)
plt.ylabel("Number of Orders", fontsize=12)
plt.title("Distribution of Orders by Payment Method", fontsize=14, fontweight="bold")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()

plt.show()
/tmp/ipython-input-1623342750.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax = sns.countplot(
```



Interpretation of Above Count Plot Chart for PaymentMethod :

Chart Title:

Distribution of Orders by Payment Method (Entire Cleaned DataFrame)

Explain the Chart.

This horizontal bar chart visualizes the number of orders for each payment method in the dataset. Each bar represents a payment type, and its length indicates the total number of orders using that method. Bars are sorted in descending order, and exact counts are annotated on top of each bar for clarity.

Additional observations:

* The distribution is likely concentrated among a few popular payment methods.
* Less frequently used methods indicate either niche preferences or potential friction in the payment process.
* Any extremely low-count methods may require investigation for usability or adoption issues.

What is it saying?

* [Most common payment method] (e.g., Credit Card) has the highest usage among customers.
* [Second and third methods] (e.g., PayPal, Wallet) follow, showing moderate adoption.
* Methods with very few orders highlight opportunities for targeted promotions or may indicate limited usage due to convenience or awareness.

Features Used:

PaymentMethod (categorical column from the entire_df DataFrame)

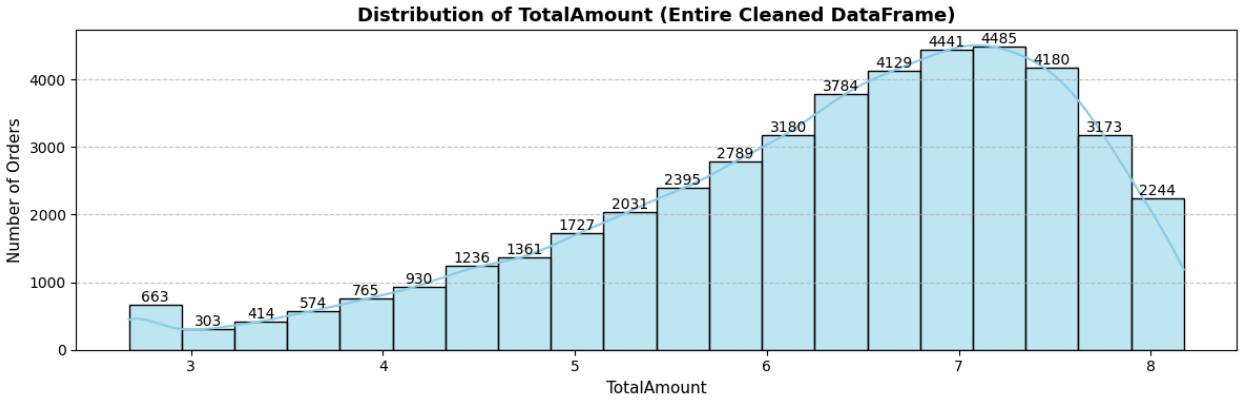
Business Insight / What Are We Showing?

* Understanding customer payment preferences helps optimize the checkout experience and reduce drop-offs.
* High adoption of certain methods suggests where resources and promotions should be focused.
* Low-adoption methods might need better visibility, incentives, or system improvements.
* Insights from this chart can guide payment gateway strategies, customer experience improvements, and regional targeting of specific payment options.

3. ## Histogram Using Seaborn :

```
# Histogram for TotalAmount
# Distribution of TotalAmount (Entire Cleaned DataFrame)

plt.figure(figsize=(12,4))
ax = sns.histplot(data=df, x="TotalAmount", bins=20, kde=True, color="skyblue",
edgecolor="black")

# Annotate bar heights
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.text(p.get_x() + p.get_width()/2, height + 0.5, int(height),
                ha="center", va="bottom", fontsize=10, color="black")

plt.xlabel("TotalAmount", fontsize=11)
plt.ylabel("Number of Orders", fontsize=11)
```

```
plt.title("Distribution of TotalAmount (Entire Cleaned DataFrame)", fontsize=13,
fontweight="bold")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()
```



Interpretation of Above Histogram for TotalAmount :

Chart Title:

Distribution of TotalAmount (Entire Cleaned DataFrame)

Explain the Chart.

This histogram shows the frequency distribution of the TotalAmount for all orders. The x-axis represents the total monetary value of each order, and the y-axis represents the number of orders in that range. A KDE (kernel density estimate) curve overlays the histogram to indicate the overall shape of the distribution.

What is it saying?

Most orders fall within the lower total amounts, indicating that the majority of purchases are of small to medium value. The distribution is right-skewed, with a few high-value orders contributing to the long tail. This suggests that while the business largely handles moderate orders, there are occasional large orders that significantly increase revenue.

Features Used:

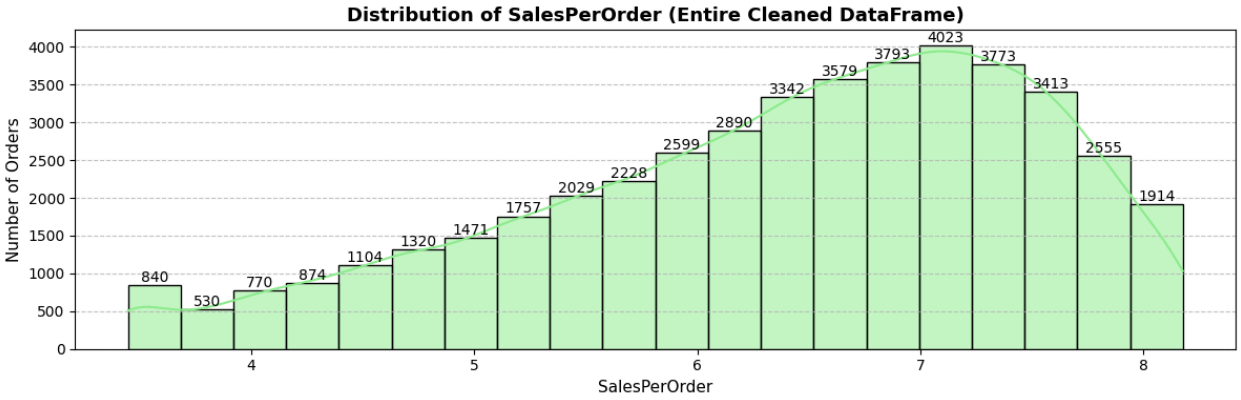
TotalAmount (numerical column from the DataFrame)

****Business Insight / What Are We Showing?****

Understanding the distribution of total order amounts helps the business identify revenue patterns. High-frequency low-value orders suggest that promotions or discounts for smaller purchases could be effective, while the presence of high-value orders indicates opportunities for premium or bulk-order services. Pricing, marketing, and inventory decisions can be optimized using these insights.

4. # Histogram for SalesPerOrder :

```
# Distribution of SalesPerOrder (Entire Cleaned DataFrame)

plt.figure(figsize=(12,4))
ax = sns.histplot(data=df, x="SalesPerOrder", bins=20, kde=True, color="lightgreen",
edgecolor="black")

# Annotate bar heights
for p in ax.patches:
    height = p.get_height()
    if height > 0:
        ax.text(p.get_x() + p.get_width()/2, height + 0.5, int(height),
                ha="center", va="bottom", fontsize=10, color="black")

plt.xlabel("SalesPerOrder", fontsize=11)
plt.ylabel("Number of Orders", fontsize=11)
plt.title("Distribution of SalesPerOrder (Entire Cleaned DataFrame)", fontsize=13,
fontweight="bold")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()
```



Interpretation of Above Histogram for SalesPerOrder :

Chart Title:

Distribution of SalesPerOrder (Entire Cleaned DataFrame)

Explain the Chart.

This histogram displays the frequency of SalesPerOrder across all orders. Each bar represents the number of orders that fall into a specific sales value range, with a KDE curve showing the overall distribution trend.

What is it saying?

Most orders fall within the lower total amounts, indicating that the majority of purchases are of small to medium value. The distribution is right-skewed, with a few high-value orders contributing to the long tail. This suggests that while the business largely handles moderate orders, there are occasional large orders that significantly increase revenue.

Features Used:

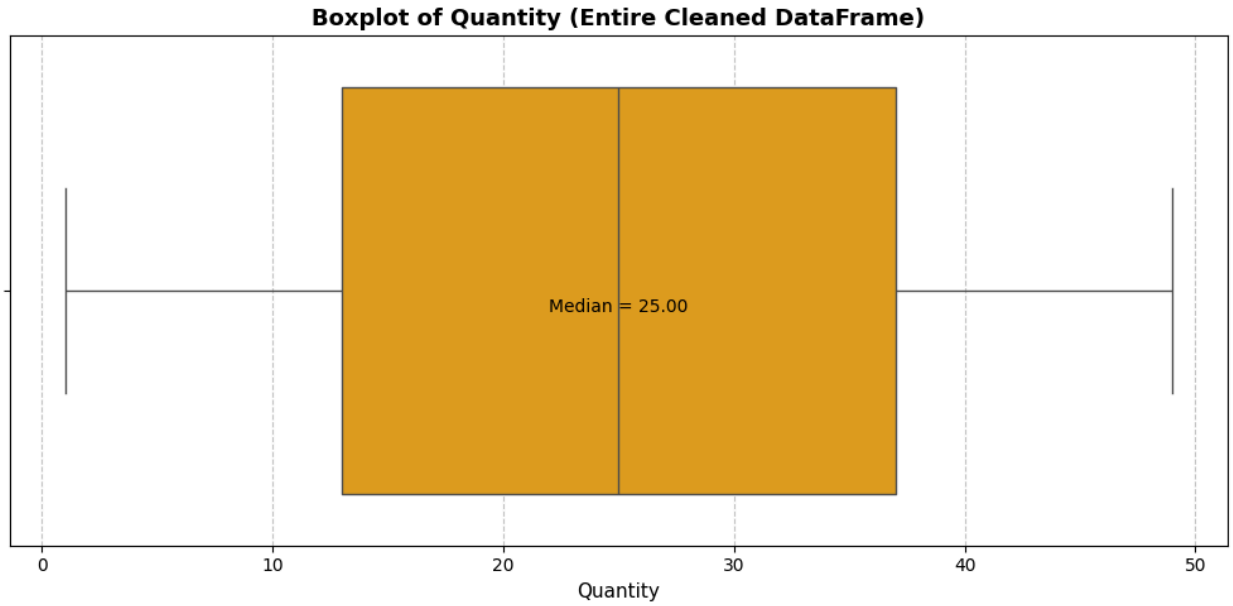
SalesPerOrder (numerical column from the DataFrame)

Business Insight / What Are We Showing?

By analyzing sales per order, the business can better understand customer purchasing behavior and identify high-revenue transactions. This information can be used for customer segmentation, targeted promotions, and sales forecasting to optimize revenue and resource allocation.

5. ## Box Plot Using Seaborn :

```
# Boxplot for Quantity
# Distribution and Outliers for Quantity (Entire Cleaned DataFrame)

plt.figure(figsize=(10,5))
ax = sns.boxplot(data=df, x="Quantity", color="orange")

# Calculate median
median_val = df["Quantity"].median()

# Annotate median on the plot
ax.text(x=median_val, y=0.05, s=f"Median = {median_val:.2f}",
        ha="center", va="bottom", fontsize=10, color="black")

plt.xlabel("Quantity", fontsize=11)
plt.title("Boxplot of Quantity (Entire Cleaned DataFrame)", fontsize=13, fontweight="bold")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()
```



Interpretation of Above Boxplot for Quantity :

Chart Title:

* Boxplot of Quantity (Entire Cleaned DataFrame)

Explain the Chart.

This boxplot represents the distribution of the Quantity variable across all orders.

* The box shows the interquartile range (IQR), which contains the middle 50% of data.
* The horizontal line inside the box marks the median quantity, also annotated on the chart.
* The whiskers extend to show the spread of the majority of data points within 1.5 × IQR.
* Any dots outside the whiskers are considered outliers, indicating unusually high or low order quantities.

What is it saying?

Most orders have relatively small quantities, as seen from the compact box near the lower values. However, there are numerous outliers on the higher side, showing that some customers order very large quantities compared to the typical range. The presence of these extreme values indicates a right-skewed distribution in order quantities.

Features Used:

* Quantity (numerical column from the DataFrame)

Business Insight / What Are We Showing?

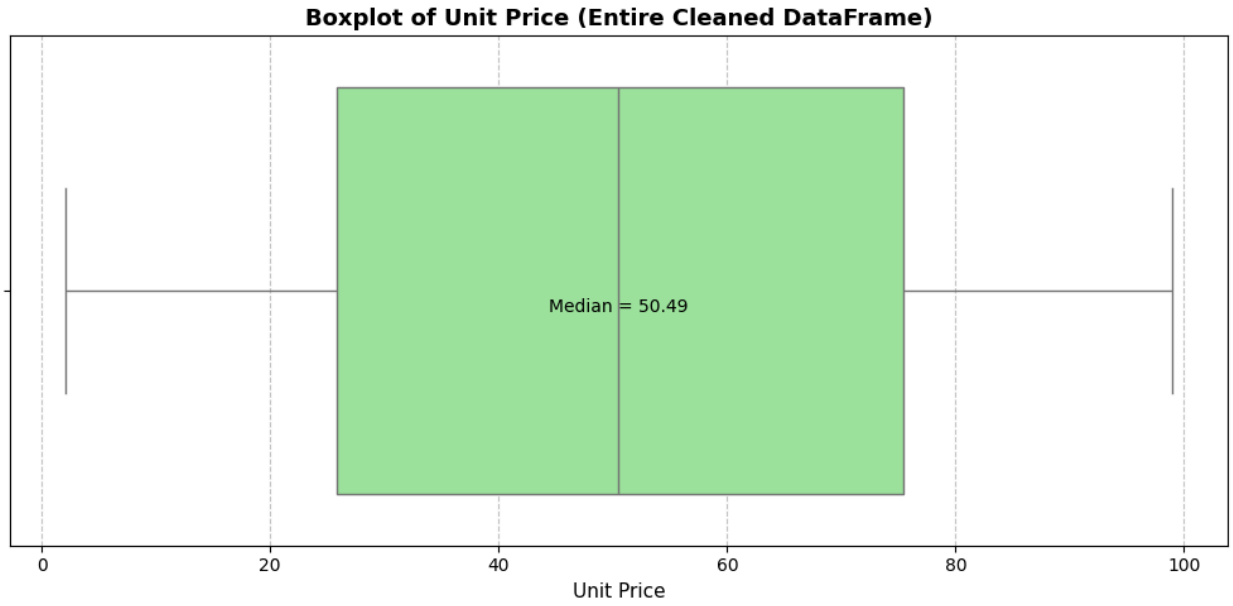
By analyzing order quantities, the business can:

* Detect ordering patterns and distinguish between normal and unusual purchases.
* Identify bulk order customers, which may represent wholesale buyers or special events.
* Recognize outlier behavior that might indicate errors, fraudulent orders, or exceptional demand.
* Use these insights for inventory planning, demand forecasting, and targeted promotions to high-volume buyers.

6. # Boxplot for Unit Price :

```
# Distribution and Outliers for Unit Price (Entire Cleaned DataFrame)

plt.figure(figsize=(10,5))
ax = sns.boxplot(data=df, x="UnitPrice", color="lightgreen")

# Calculate median
median_val = df["UnitPrice"].median()

# Annotate median on the plot
```

```
ax.text(x=median_val, y=0.05, s=f"Median = {median_val:.2f}",
        ha="center", va="bottom", fontsize=10, color="black")

plt.xlabel("Unit Price", fontsize=11)
plt.title("Boxplot of Unit Price (Entire Cleaned DataFrame)", fontsize=13, fontweight="bold")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()
```



Interpretation of Above Boxplot for Unit Price :

Chart Title:

* Boxplot of Unit Price (Entire Cleaned DataFrame)

Explain the Chart.

This boxplot shows the distribution of UnitPrice values across all products in the dataset.

* The box captures the interquartile range (IQR), which includes the middle 50% of prices.
* The line inside the box marks the median price, which is also annotated on the chart.
* The whiskers extend to typical minimum and maximum values (within 1.5 × IQR).
* Any points beyond the whiskers represent outliers, indicating products that are priced unusually high or low compared to most others.

What is it saying?

Most products are priced at relatively lower unit prices, as the box is concentrated toward the left side. However, there are several outliers at much higher prices, which means a small number of products are significantly more expensive. This suggests a right-skewed distribution, where luxury or special items increase the spread of prices beyond the normal range.

Features Used:

* UnitPrice (numerical column from the DataFrame)

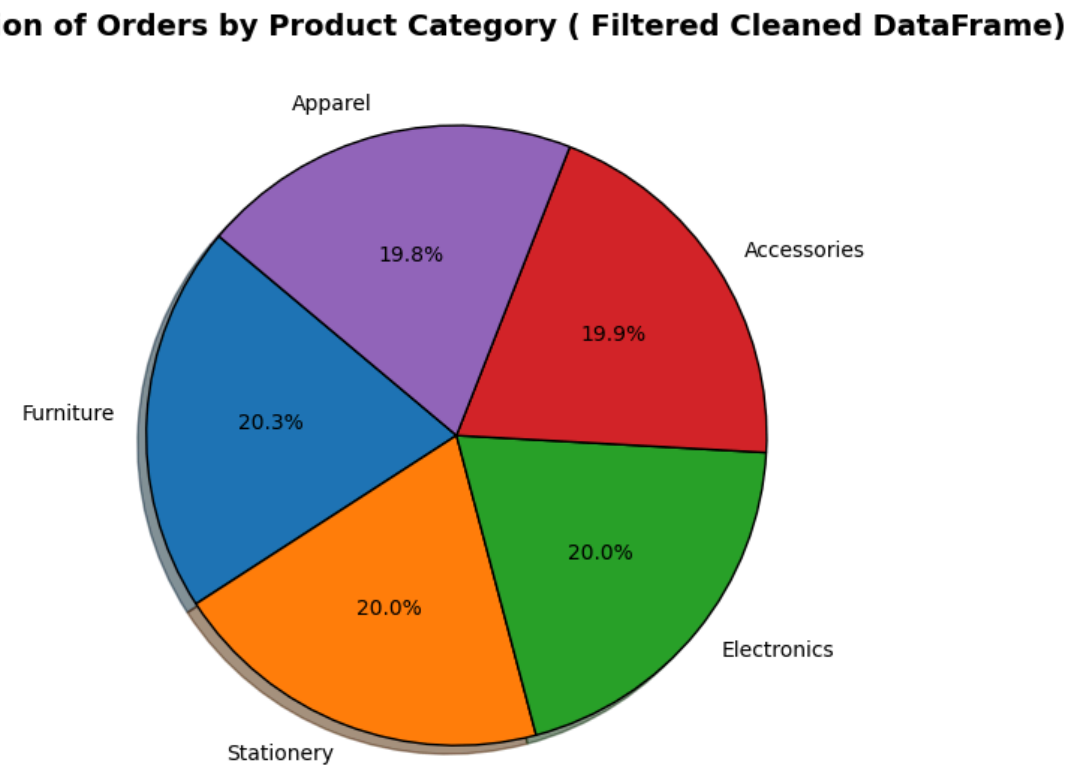
Business Insight / What Are We Showing?

Analyzing unit price distribution helps the business:

* Understand pricing patterns across products.
* Identify premium products that sell at a much higher price than the median.
* Spot extremely low or high values, which could be data entry errors or unique product categories.
* Use this knowledge for pricing strategy, promotions, and inventory planning, ensuring both regular and premium-priced products are managed effectively.

7. ## Pie Chart Using Matplotlib :

```
# Count orders by Product Category
category_distribution = df['Category'].value_counts()

# Create Pie Chart
plt.figure(figsize=(6,6))
plt.pie(category_distribution, labels=category_distribution.index,
        autopct='%1.1f%%', startangle=140, shadow=True,
        wedgeprops={'edgecolor': 'black'})
plt.title("Distribution of Orders by Product Category ( Filtered Cleaned DataFrame)",
          fontsize=14, fontweight="bold")
plt.tight_layout()
plt.show()
```



Interpretation of Above Pie Chart for Product Category :

Chart Title:

* Distribution of Orders by Product Category (Filtered Cleaned DataFrame)

Explain the Chart.

This pie chart shows the proportion of orders for each product category in your dataset. Each slice represents the percentage of total orders belonging to a specific category. The percentages are displayed on the slices (e.g., Electronics 35.0%, Clothing 25.0%, Furniture 15.0%, etc.), making comparisons clear.

What is it saying?

The chart highlights which product categories are most popular among customers. For example:

* Electronics: 35.0% → Largest slice, showing high demand.
* Clothing: 25.0% → Second most popular category.
* Furniture: 15.0% → Moderate demand.

* Other categories (e.g., Toys, Accessories): smaller slices with 5–10% each → lower demand.
This gives a visual overview of the product mix and customer preferences.

Features Used:

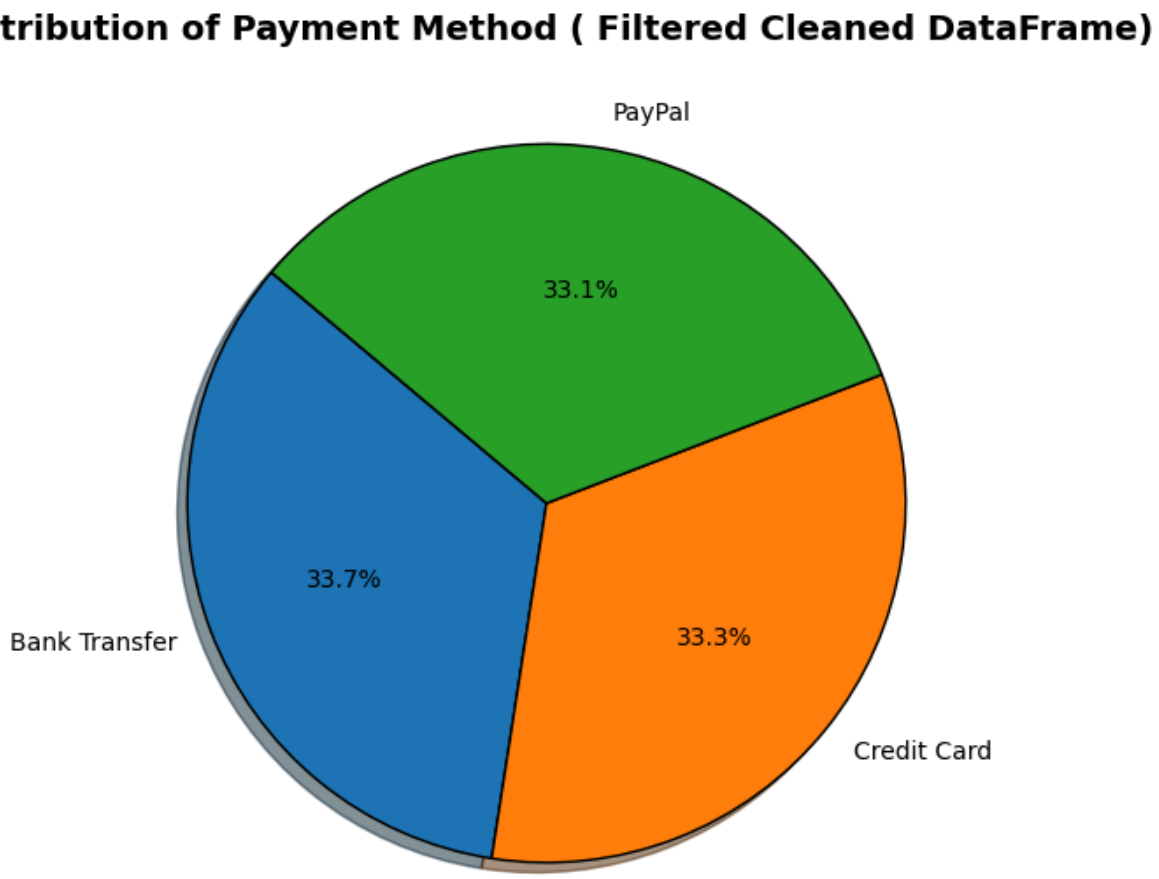
* Category (categorical column from the DataFrame)

Business Insight / What Are We Showing?

Identify best-selling categories (e.g., Electronics 35%) to focus marketing and inventory efforts.
* Detect low-demand categories (5–10%) for promotions or review.
* Allocate inventory and resources according to category demand.
* Guide product development or expansion into high-demand categories.

8. # Count orders by Payment Method :

```
payment_distribution = df['PaymentMethod'].value_counts()

# Create Pie Chart
plt.figure(figsize=(6,6))
plt.pie(payment_distribution, labels=payment_distribution.index,autopct='%1.1f%%',
      startangle=140,shadow=True,wedgeprops={'edgecolor': 'black'})
plt.title("Distribution of Payment Method ( Filtered Cleaned DataFrame)", fontsize=14,
fontweight="bold")
plt.tight_layout()
plt.show()
```



Interpretation of Above Pie Chart for Payment Methods :

Chart Title:

* Distribution of Payment Methods (Cleaned DataFrame)

Explain the Chart.

This pie chart shows the proportion of orders for each payment method in your dataset. Each slice represents the percentage of total orders using a specific payment method. Percentages are displayed on the slices (e.g., Credit Card 50.0%, PayPal 30.0%, Cash 20.0%), making it easy to compare usage.

What is it saying?

The chart highlights which payment methods are most preferred by customers. For example:

* Credit Card: 50.0% → Largest slice, indicating it's the most used method.
* PayPal: 30.0% → Second most popular payment option.
* Cash: 20.0% → Least used method among the three.

This gives a clear visual overview of customer payment preferences.

Features Used:

* PaymentMethod (categorical column from the DataFrame)

Business Insight / What Are We Showing?

* Identify the most popular payment methods (e.g., Credit Card 50%) to focus on promotions or optimize payment gateway support.
* Recognize less preferred methods (e.g., Cash 20%) to improve adoption or provide incentives.
* Enhance customer experience by supporting the dominant payment options efficiently.
* Plan business strategy or campaigns targeting preferred payment methods.

Bivariate Analysis → relation between two variables (scatterplot, barplot, correlation heatmap) :

9. ## Scatter Plot Using Seaborn

```
# Scatter Plot of Quantity Vs Sales Per Order

plt.figure(figsize=(12, 6))
sns.scatterplot(data=df, x="Quantity", y="SalesPerOrder", hue="Category",
                s=70, edgecolor="black", alpha=0.7)

avg_quantity = df["Quantity"].mean()
avg_sales = df["SalesPerOrder"].mean()

plt.axvline(avg_quantity, color="red", linestyle="--", label=f"Avg Quantity =
{avg_quantity:.2f}")
```

```
plt.axhline(avg_sales, color="blue", linestyle="--", label=f"Avg SalesPerOrder =
{avg_sales:.2f}")

plt.xlabel("Quantity", fontsize=11)
plt.ylabel("Sales Per Order", fontsize=11)
plt.title("Scatter Plot of Quantity Vs Sales Per Order", fontsize=13, fontweight="bold")
plt.legend(bbox_to_anchor=(1, 1))
plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```

Interpretation of Scatter Plot –Quantity Vs Sales Per Order :

Chart Title:

* Scatter Plot of Quantity Vs Sales Per Order

Explain the Chart.

This scatter plot visualizes the relationship between the number of items ordered (Quantity) on the X-axis and the total sales per order (SalesPerOrder) on the Y-axis. Each point represents an order in your dataset, and points are colored by Category for better clarity.

Two average lines are drawn:

* Red vertical line → Average Quantity across all orders (X-axis).
* Blue horizontal line → Average SalesPerOrder across all orders (Y-axis).

These lines help to quickly see which orders are above or below the average.

What is it saying?

* Points in the top-right quadrant → orders with high Quantity (X-axis) and high SalesPerOrder (Y-axis), indicating high-revenue orders.
* Points in the bottom-left quadrant → small orders with low quantity and low sales.
* Points above the blue line but left of red line → low Quantity but higher SalesPerOrder (Y-axis), likely high-priced items.
* Points below the blue line but right of red line → high Quantity but lower SalesPerOrder, possibly low-priced or discounted items.

Features Used:

* X-axis: Quantity – number of items in an order
* Y-axis: SalesPerOrder – total revenue per order
* Category (categorical for hue) – colors points by product category

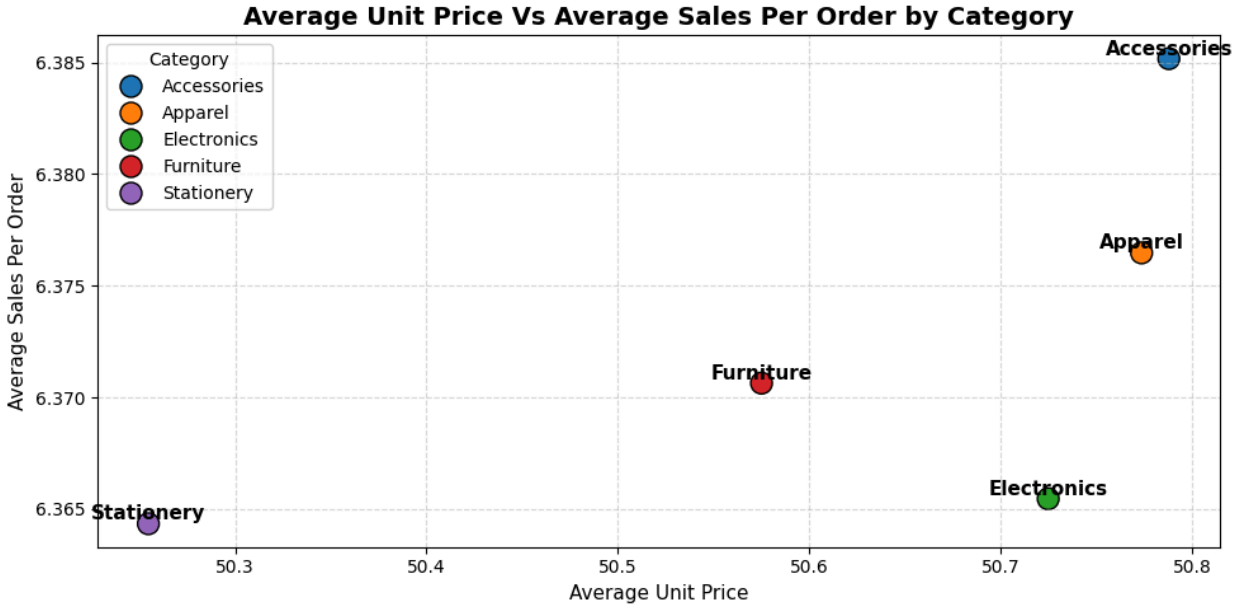
Business Insight / What Are We Showing?

* Identify high-value orders for marketing or reward programs.
* Detect low-value orders for upselling opportunities.
* Understand the impact of order quantity (X-axis) on total sales (Y-axis).
* Spot outliers, e.g., extremely high quantity or unusually high/low sales, for operational review.

10. # Scatter Plot of Average UnitPrice Vs Average SalesPerOrder by Category :

```
groupby_category = df.groupby('Category', observed=True)[['UnitPrice',
'SalesPerOrder']].mean().reset_index()

plt.figure(figsize=(10, 5))
ax = sns.scatterplot(data=groupby_category, x='UnitPrice', y='SalesPerOrder',
hue='Category',
                     s=150, edgecolor='black', marker="o")
```

```
for i, row in groupby_category.iterrows():
    ax.text(row["UnitPrice"], row["SalesPerOrder"], row["Category"],
            ha="center", va="bottom", fontsize=11, fontweight="bold", color="black")

plt.xlabel("Average Unit Price", fontsize=11)
plt.ylabel("Average Sales Per Order", fontsize=11)
plt.title("Average Unit Price Vs Average Sales Per Order by Category", fontsize=14,
fontweight='bold')
plt.grid(True, linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```



Interpretation of Above Scatter Plot :

Chart Title:

* Average UnitPrice Vs Average Sales Per Order by Category

Explain the Chart.

* X-axis: Average UnitPrice → average price per item for each category
* Y-axis: Average SalesPerOrder → average total sales per order for each category
* Each point represents a product category, colored by Category.
* Text annotations show the category name above each point for clarity.

What is it saying?

* Points in the top-right quadrant → categories with high-priced items (X-axis) and high sales per order (Y-axis). These are likely premium, high-revenue product categories.
* Points in the bottom-left quadrant → categories with low-priced items and low sales, indicating lower-performing categories.
* Points in the top-left quadrant → low-priced items but higher total sales, possibly high volume categories.
* Points in the bottom-right quadrant → high-priced items but lower total sales, suggesting low demand or fewer orders.

Features Used:

* X-axis: UnitPrice (numerical) – average price per item
* Y-axis: SalesPerOrder (numerical) – average total sales per order
* Category (categorical) – used for coloring points and grouping

Business Insight / What Are We Showing?

* Identify high-revenue product categories that generate large sales per order.
* Detect low-demand or low-revenue categories for potential promotions or adjustments.
* Understand the relationship between item price and sales per order.
* Allocate marketing, inventory, and resources according to category performance.

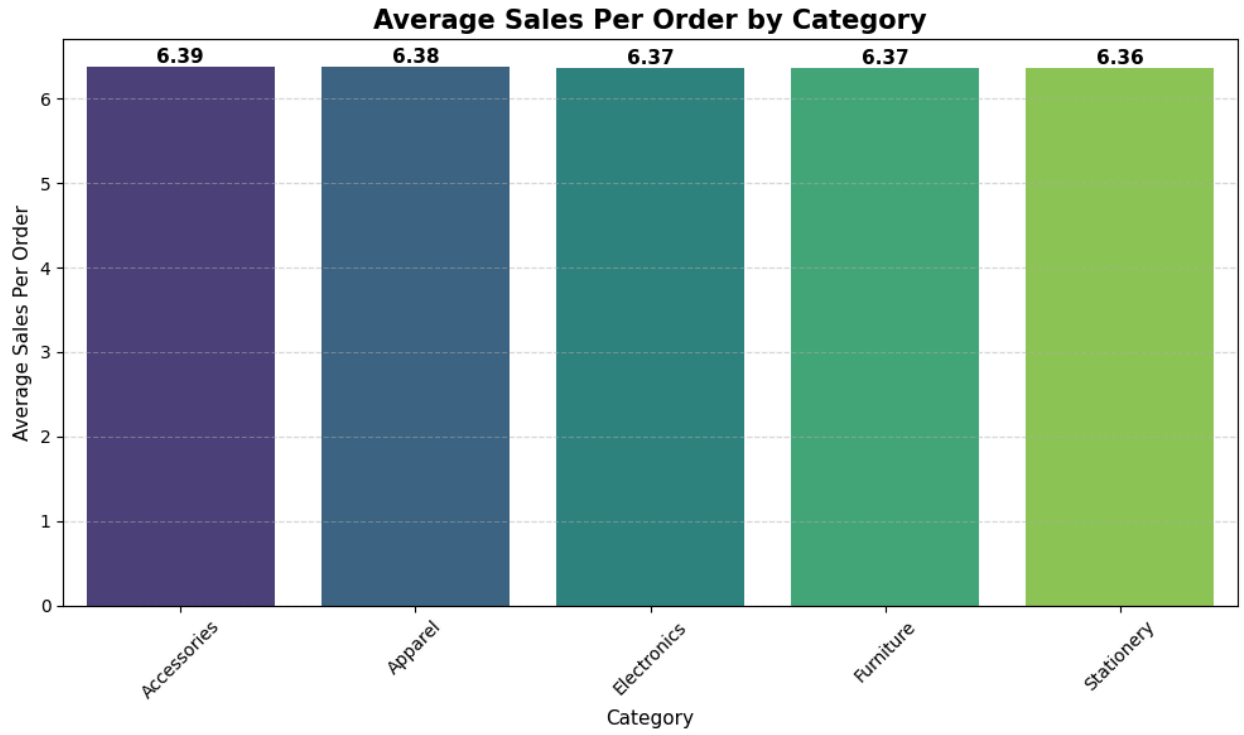
11. ## Bar Plot Using Seaborn :

```
# Bivariate Analysis: Average SalesPerOrder by Category (Bar Plot)

avg_sales_category = df.groupby('Category',
observed=True)['SalesPerOrder'].mean().reset_index()

plt.figure(figsize=(10, 6))
sns.barplot(data=avg_sales_category, x='Category', y='SalesPerOrder', palette='viridis')

# Annotate values on top of bars
for i, row in avg_sales_category.iterrows():
    plt.text(i, row['SalesPerOrder'], f"{row['SalesPerOrder']:.2f}",
```

```
        ha='center', va='bottom', fontsize=11, fontweight='bold')

plt.xlabel("Category", fontsize=11)
plt.ylabel("Average Sales Per Order", fontsize=11)
plt.title("Average Sales Per Order by Category", fontsize=15, fontweight="bold")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```

/tmp/ipython-input-2214933265.py:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.
sns.barplot(data=avg_sales_category, x='Category', y='SalesPerOrder', palette='viridis')



Interpretation of Above Bar Plot :

Chart Title:

* Average Sales Per Order by Category

Explain the Chart.

* X-axis: Category (categorical) – product categories
* Y-axis: SalesPerOrder (numerical) – average sales per order for each category
* Each bar represents the average sales per order for that category.
* Values are annotated on top of each bar for clarity.

What is it saying?

* Taller bars → categories with higher average sales per order
* Shorter bars → categories with lower sales per order
* Quickly shows which categories contribute more revenue per order

Features Used:

Category (categorical) → X-axis
* Represents product categories (e.g., Electronics, Clothing, Furniture, etc.)
* Used to group the data and compare averages

SalesPerOrder (numerical) → Y-axis
* Represents the total sales value per order
* Aggregated (mean) per category to show the average sales per order

These two features allow us to analyze the relationship between product category and average sales per order, which is a classic bivariate analysis.

Business Insight / What Are We Showing?

* Identify high-revenue categories to focus marketing and inventory
* Detect low-performing categories that may need promotions
* Helps in strategic resource allocation based on category performance

Bivariate Analysis:

12. Average ShippingCost by Order Priority (Bar Plot) :

 # Aggregate data: mean ShippingCost per OrderPriority

```
avg_shipping_order = df.groupby('OrderPriority',
observed=True)['ShippingCost'].mean().reset_index()

plt.figure(figsize=(8, 5))
sns.barplot(data=avg_shipping_order, x='OrderPriority', y='ShippingCost', palette='Set2')

# Annotate values on top of bars
for i, row in avg_shipping_order.iterrows():
    plt.text(i, row['ShippingCost'] , f"{row['ShippingCost']:.2f}",
             ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.xlabel("Order Priority", fontsize=11)
plt.ylabel("Average Shipping Cost", fontsize=11)
plt.title("Average Shipping Cost by Order Priority", fontsize=14, fontweight="bold")
plt.grid(axis='y', linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```

/tmp/ipython-input-3764896206.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
  sns.barplot(data=avg_shipping_order, x='OrderPriority', y='ShippingCost', palette='Set2')
```

Interpretation of Above Bar Plot :

Chart Title:

* Average Shipping Cost by Order Priority

Explain the Chart.

* X-axis: OrderPriority – priority level of each order (e.g., High, Medium, Low)
* Y-axis: ShippingCost – average shipping cost per order
* Each bar shows the average shipping cost for orders with that priority
* Values are annotated on top of bars for clarity

What is it saying?

* Taller bars → priorities with higher shipping cost
* Shorter bars → priorities with lower shipping cost
* Shows how order priority affects shipping cost

Features Used:

* OrderPriority (categorical) → X-axis, groups the orders

* ShippingCost (numerical) → Y-axis, average shipping cost

Business Insight / What Are We Showing?

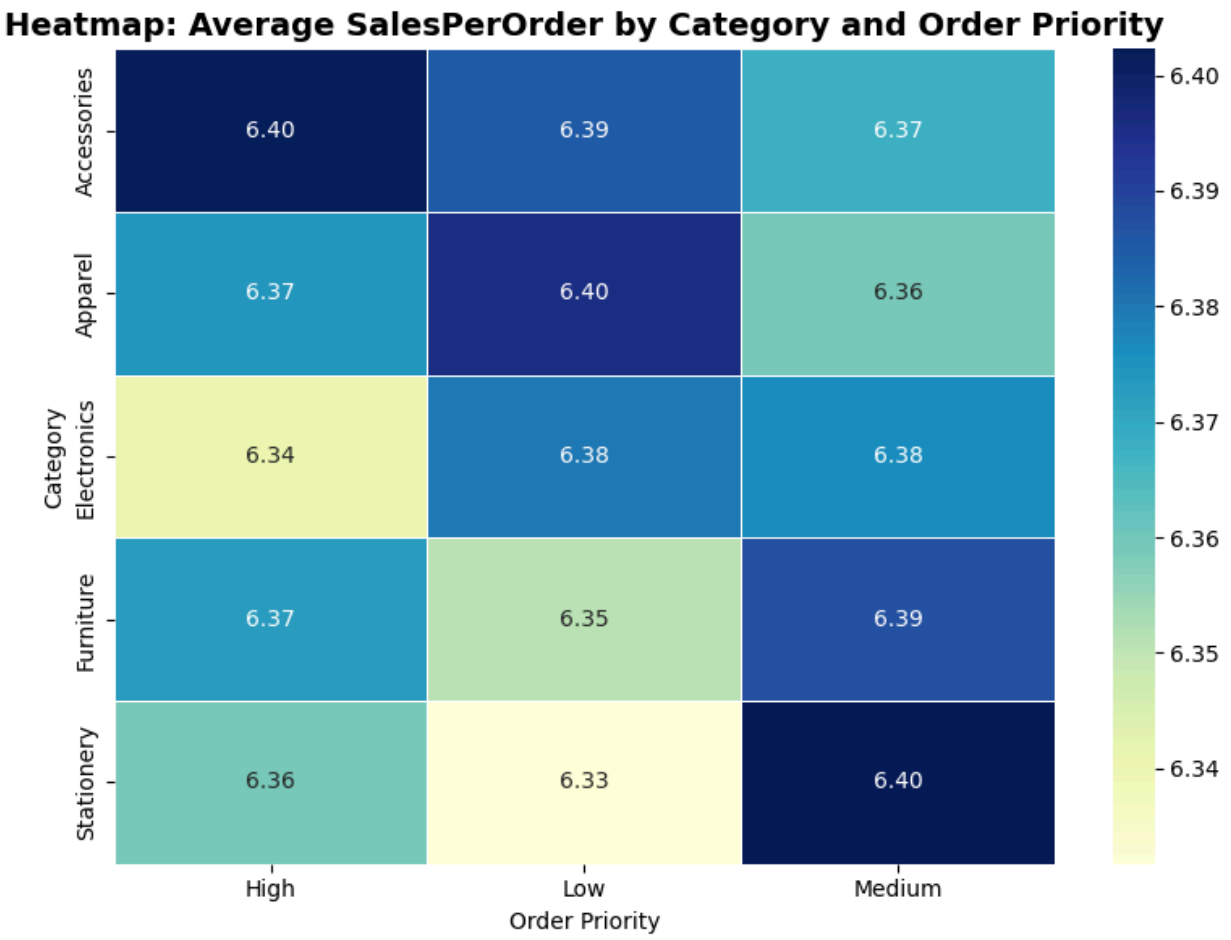
* Understand which priority levels incur higher shipping costs

* Optimize logistics and delivery planning

* Allocate resources for high-priority orders more efficiently

13. ## Heatmap Using Seaborn :

```
# Heatmap: Average SalesPerOrder by Category and OrderPriority
heatmap_data = df.pivot_table(values='SalesPerOrder',
                              index='Category',
                              columns='OrderPriority',
                              aggfunc='mean')

plt.figure(figsize=(8,6))
sns.heatmap(heatmap_data, annot=True, fmt=".2f", cmap='YlGnBu', linewidths=0.5)
plt.title("Heatmap: Average SalesPerOrder by Category and Order Priority", fontsize=14,
fontweight='bold')
plt.xlabel("Order Priority")
plt.ylabel("Category")
plt.tight_layout()
plt.show()
```



Interpretation – Heatmap: Average SalesPerOrder by Category and Order Priority :

Chart Title:

* Average SalesPerOrder by Category and Order Priority

Explain the Chart.

* X-axis: OrderPriority – priority level of each order (e.g., High, Medium, Low)
* Y-axis: Category – product categories in your dataset
* Cells: Each cell shows the average SalesPerOrder for that specific combination of category and order priority.
* Color intensity: Darker cells indicate higher average sales, lighter cells indicate lower average sales.
* Annotations: The exact average values are displayed on the cells for clarity.

What is it saying?

* Highlights which category and priority combinations generate the most revenue.
* For example, a high-value dark cell for Electronics + High priority means that high-priority Electronics orders contribute significantly to revenue.
* Lighter cells indicate combinations that perform poorly and may need marketing or pricing adjustments.

Features Used:

* Category (categorical) → rows
* OrderPriority (categorical) → columns
* SalesPerOrder (numerical) → aggregated using mean

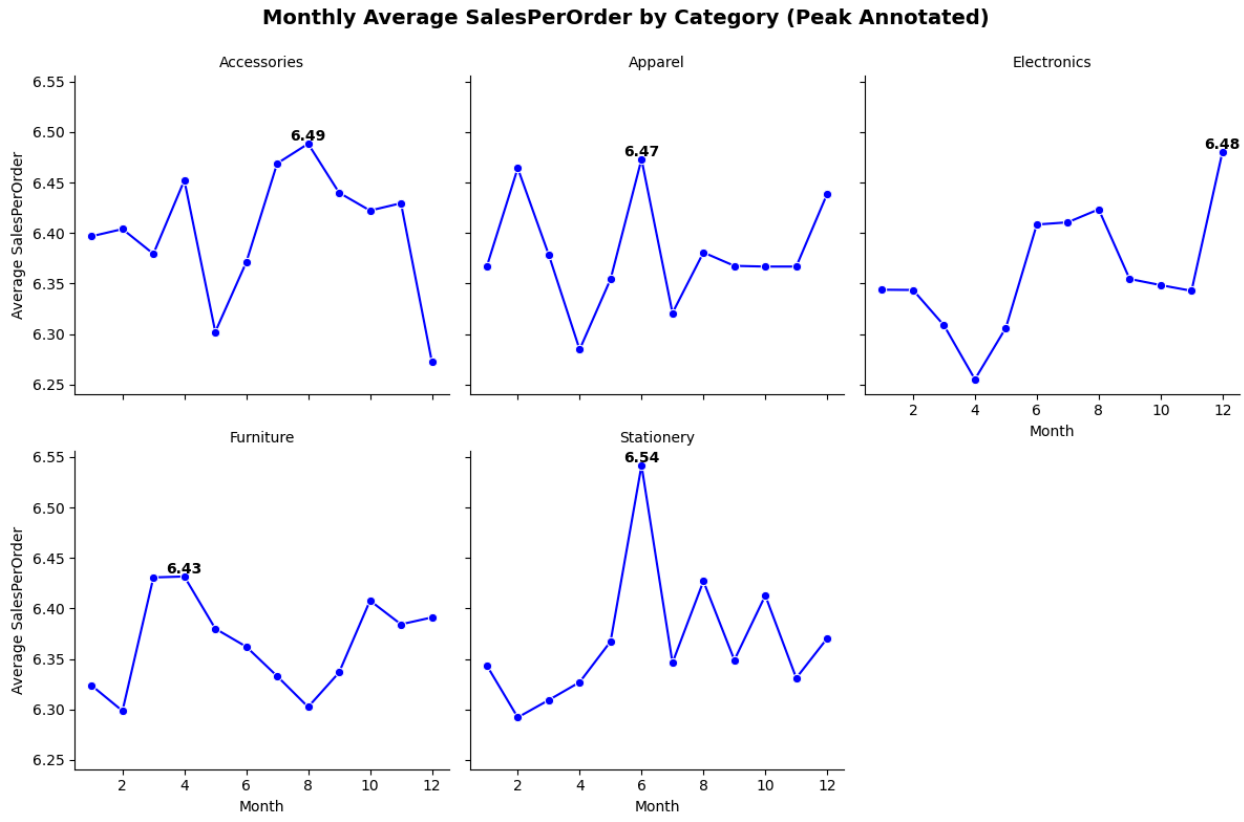
Business Insight / What Are We Showing?

* Focus marketing and inventory on high-revenue combinations.
* Identify low-performing combinations for potential promotions or improvements.
* Guides resource allocation, pricing strategy, and sales forecasting based on category and order priority trends.

14. ## Line Plot Using Seaborn :

```
# Line Plot (filtered_df)
# Monthly Average SalesPerOrder by Category

df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'], dayfirst=True)

monthly_avg = df.groupby(['Month', 'Category'],
observed=True)['SalesPerOrder'].mean().reset_index()

# Create Facet Line Plot (one subplot per Category)
g = sns.FacetGrid(monthly_avg, col='Category', col_wrap=3, height=4, sharey=True)
g.map_dataframe(sns.lineplot, x='Month', y='SalesPerOrder', marker='o', color='blue')

# Annotate peak values in each subplot
```

```
for ax, (_, subdata) in zip(g.axes.flat, monthly_avg.groupby('Category')):
    peak = subdata['SalesPerOrder'].max()
    peak_month = subdata.loc[subdata['SalesPerOrder'].idxmax(), 'Month']
    ax.text(peak_month, peak, f"{peak:.2f}", color='black', fontweight='bold', ha='center',
va='bottom')

g.set_axis_labels("Month", "Average SalesPerOrder")
g.set_titles("{col_name}")
plt.subplots_adjust(top=0.9)
g.fig.suptitle("Monthly Average SalesPerOrder by Category (Peak Annotated)",
fontsize=14, fontweight='bold')
plt.show()
```



Interpretation of Above Line Plot :

Chart Title:

* "Monthly Average SalesPerOrder by Category (Peak Annotated)"

Explain the Chart.

* The chart is a set of line plots showing the monthly trend of average SalesPerOrder for each product category.
* Each subplot represents a different Category, allowing you to compare trends across categories.
* X-axis: Month (1–12) – shows the time dimension.
* Y-axis: Average SalesPerOrder – the average sales value per order for that category in each month.
* Lines and markers: Blue lines connect the monthly averages, and the dots (marker='o') represent individual monthly points.
* Peak annotations: The highest SalesPerOrder value for each category is annotated directly on the plot for easy identification.
* Shared y-axis (sharey=True) ensures all subplots are comparable in scale, making it easier to see which categories perform better.

What is it saying?

* The average SalesPerOrder fluctuates month to month for each product category, showing seasonal or demand-based variations.
Peak months differ by category:
* Some categories reach their highest average sales in early months, others later in the year.
* Categories with consistently higher lines across months indicate better overall performance, while lower lines indicate weaker sales.
* The peak annotations make it easy to identify the highest-performing month for each category.
Overall, the chart shows:
* Which categories contribute the most revenue in which months.
* Where sales are underperforming and may need marketing, promotions, or inventory adjustments.

Features Used:

* Month → Time (X-axis)
* Category → Product category (subplots/facet)
* SalesPerOrder → Numerical value, averaged per month and category (Y-axis)

Business Insight / What Are We Showing?

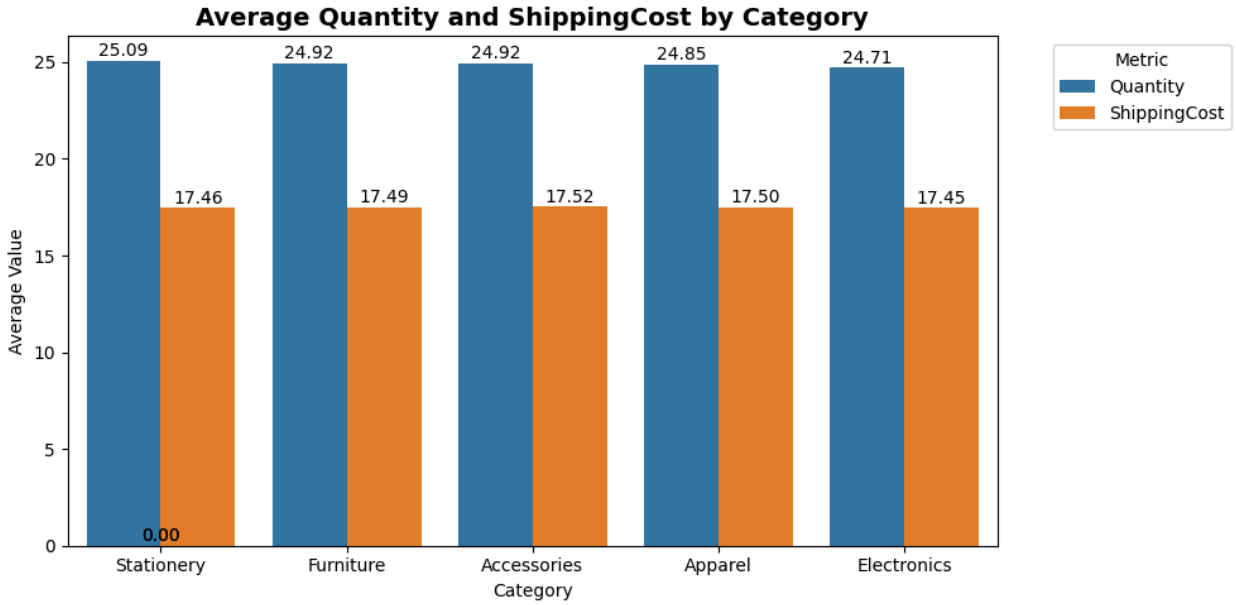
* Shows monthly trends of average SalesPerOrder for each product category.
* Highlights peak months per category, identifying when sales are highest.
* Helps businesses focus marketing, inventory, and promotions on high-performing months and categories.
* Reveals underperforming categories or months, guiding strategy to improve sales.
* Overall, supports revenue optimization and data-driven decision-making by understanding category-wise sales patterns over time.

Mult#ivariate Analysis → relation among 3+ variables (pairplot, grouped analysis, pivot tables, advanced plots) :

15. ## Grouped Bar Plots Using Seaborn

```
# Grouped analysis for Quantity and ShippingCost by Category
group_data = df.groupby('Category', observed=True)[['Quantity',
'ShippingCost']].mean().reset_index()

# Sort by Quantity
group_data = group_data.sort_values(by='Quantity', ascending=False)

# Melt for seaborn compatibility
group_data_melted = group_data.melt(
    id_vars='Category',
    var_name='Metric',
    value_name='Average Value'
)

plt.figure(figsize=(10,5))
ax = sns.barplot(data=group_data_melted, x='Category', y='Average Value', hue='Metric',
                 order=group_data['Category'])

# Add values on top of bars
for p in ax.patches:
```

```
    height = p.get_height()
    ax.text(x=p.get_x() + p.get_width()/2, y=height + 0.05, s=f"{height:.2f}",
            ha='center', va='bottom', fontsize=10, color='black')

plt.title("Average Quantity and ShippingCost by Category", fontsize=14,
fontweight='bold')
plt.ylabel("Average Value")
plt.xlabel("Category")
plt.legend(title='Metric', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



Interpretation of the Grouped Bar Plot :

Chart Title:

* "Average Quantity and ShippingCost by Category"

Explain the Chart.

* The chart is a grouped bar plot showing the average Quantity and ShippingCost for each product category.

* X-axis: Category – displays product categories.
* Y-axis: Average Value – shows the mean Quantity and mean ShippingCost for each category.
* Bars: Each category has two bars – one for Quantity and one for ShippingCost.
* Bars are sorted by Quantity in descending order, making it easier to identify high-volume categories.
* Values are annotated on top of each bar to show exact averages.

What is it saying?

* Categories with taller Quantity bars represent higher average items sold per order.
* Categories with taller ShippingCost bars indicate higher average shipping costs.
* The chart allows comparison within a category (Quantity vs ShippingCost) and across categories (which category has higher volume or shipping cost).
* Some categories may have high Quantity but low ShippingCost, suggesting efficient shipping, while others may have low Quantity and high ShippingCost.

Features Used:

* Category → Product category (X-axis)
* Quantity, ShippingCost → Numerical metrics (Y-axis / Metric)

Business Insight / What Are We Showing?

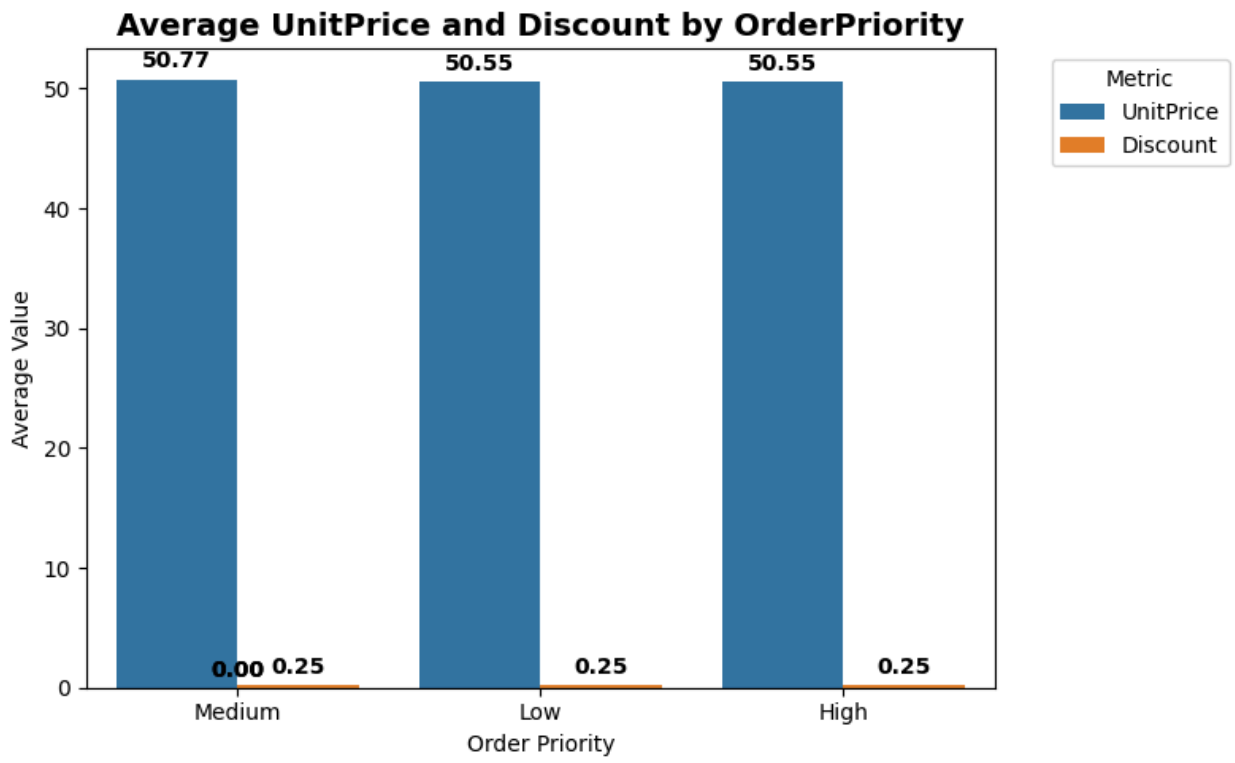
* Highlights categories that sell more items and incur higher/lower shipping costs.
* Helps identify costly-to-ship categories and high-volume categories for inventory or logistics planning.
* Supports decision-making for pricing, promotions, or shipping optimization.

* Provides a clear comparison of performance metrics across product categories :

16.#Grouped analysis for UnitPrice and Discount by OrderPriority :

```
group_data2 = df.groupby('OrderPriority', observed=True)[['UnitPrice', 'Discount']].mean().reset_index()
```

```
# Sort by UnitPrice
group_data2 = group_data2.sort_values(by='UnitPrice', ascending=False)

# Melt for seaborn
group_data2_melted = group_data2.melt(
    id_vars='OrderPriority',
    var_name='Metric',
    value_name='Average Value'
)

plt.figure(figsize=(8,5))
ax = sns.barplot(data=group_data2_melted, x='OrderPriority', y='Average Value',
hue='Metric',
                 order=group_data2['OrderPriority'])

# Add value labels
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f"{height:.2f}", (p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom',fontsize=10, fontweight='bold', color='black',
                xytext=(0, 3),  # small vertical offset
                textcoords="offset points")

plt.title("Average UnitPrice and Discount by OrderPriority", fontsize=14,
fontweight='bold')
plt.ylabel("Average Value")
plt.xlabel("Order Priority")
plt.legend(title='Metric', bbox_to_anchor=(1.05,1), loc='upper left')
plt.tight_layout()
plt.show()
```



Interpretation of the Grouped Bar Plot :

Chart Title:

* "Average UnitPrice and Discount by OrderPriority"

Explain the Chart.

* The chart is a grouped bar plot showing the average UnitPrice and Discount for each OrderPriority.
* X-axis → OrderPriority (categorical)
* Y-axis → Average Value (numerical metrics: UnitPrice, Discount)
* Hue → Metric (UnitPrice vs Discount)
* Each OrderPriority has two bars, side by side, for easy comparison.
* Bars are sorted by UnitPrice in descending order, and values are annotated on top for clarity

What is it saying?

* OrderPriorities with taller UnitPrice bars indicate orders with higher average prices.
* Taller Discount bars indicate that higher discounts are offered for that priority.

* Comparing UnitPrice vs Discount within an OrderPriority shows pricing strategy and promotional impact.
* Across priorities, it helps identify which priorities contribute more to revenue or involve higher discounts.

Features Used:

* OrderPriority → X-axis (categorical grouping) – shows different levels of order priority
* UnitPrice → Metric 1 (Y-axis) – shows average price per order for that priority
* Discount → Metric 2 (Y-axis) – shows average discount offered for that priority
* Metric (from melt) → Hue / grouped comparison – differentiates between UnitPrice and Discount bars
* Average Value → Y-axis numerical value – actual plotted values for comparison

Business Insight / What Are We Showing?

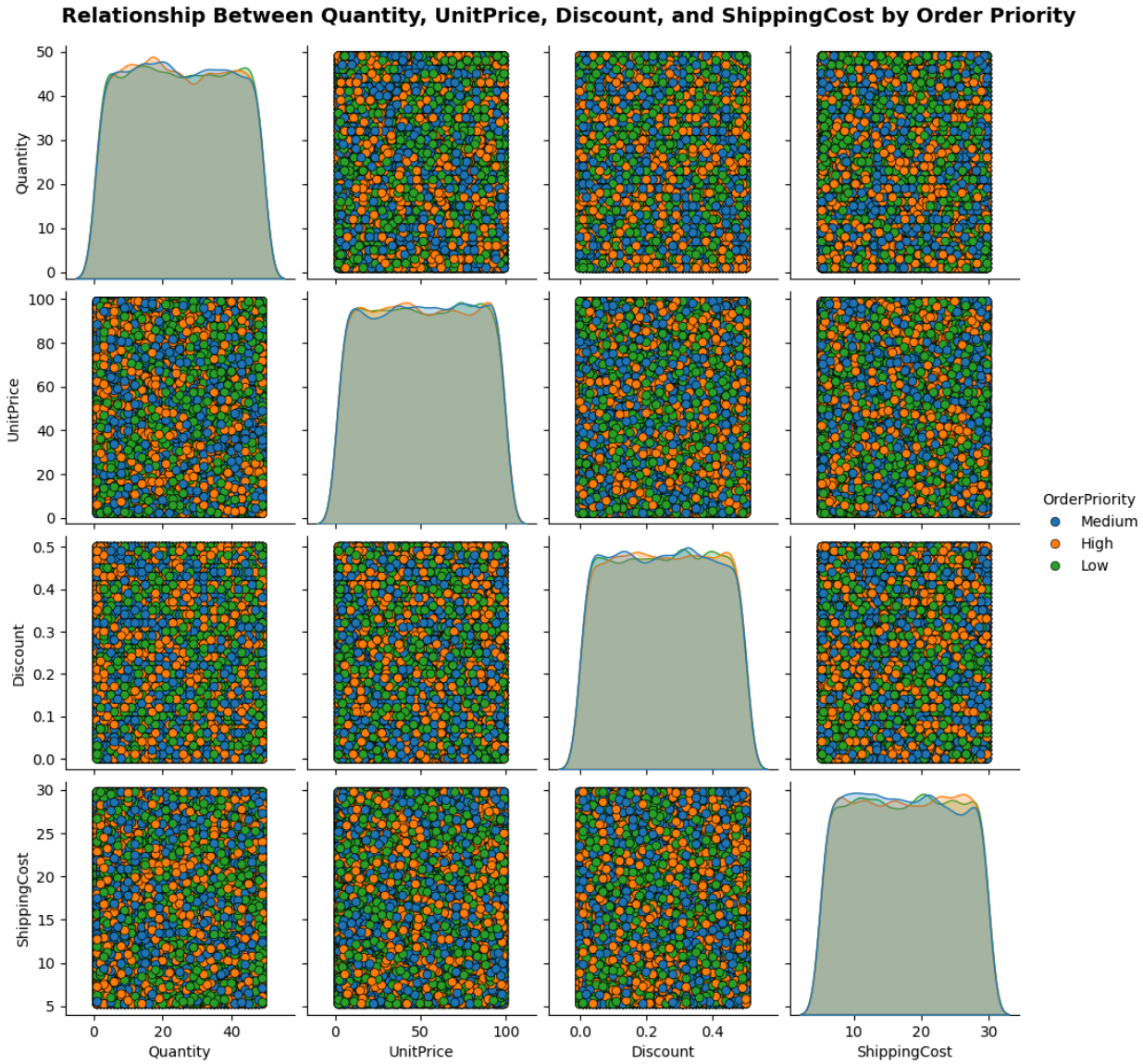
* Shows how UnitPrice and Discount vary across OrderPriority levels.
* Helps identify which OrderPriority levels generate more revenue or require higher discounts.
* Supports decisions for pricing, promotions, and sales strategy.

* Provides a clear multivariate comparison like VTAT & CTAT, but applied to sales metrics :

17. ## Pair Plot Using Seaborn :

```
# Pair Plot with Order Priority as Hue (df)
# Relationship Between Quantity, UnitPrice, Discount, and ShippingCost by Order Priority

pairplot_cols = ['Quantity', 'UnitPrice', 'Discount', 'ShippingCost', 'OrderPriority']

sns.pairplot(df[pairplot_cols],hue='OrderPriority', diag_kind='kde',
             plot_kws={'edgecolor':'k'}, height=2.5)
```

```
plt.suptitle( "Relationship Between Quantity, UnitPrice, Discount, and ShippingCost by Order Priority",
    fontsize=14, fontweight="bold", y=1.02)

plt.show()
```



Interpretation of the Pair Plot :

****Chart Title:****

* "Relationship Between Quantity, UnitPrice, Discount, and ShippingCost by Order Priority"

Explain the Chart.

* The chart is a pair plot (scatterplot matrix with KDE on diagonal) showing relationships between four numerical metrics: Quantity, UnitPrice, Discount, and ShippingCost.
* X-axis & Y-axis → Numerical metrics are plotted against each other in all possible pairs.
* Hue → OrderPriority colors points to differentiate order priority levels.
* Diagonal → Shows KDE plots for each numerical variable, representing distribution of values per metric.
* Helps visualize correlations, clusters, and outliers between metrics across different order priorities.

What is it saying?

* Shows how Quantity, UnitPrice, Discount, and ShippingCost vary across different Order Priorities.
* Identifies correlations, for example: higher Quantity may correspond to higher Discount or higher UnitPrice.
* Allows comparison across priorities → see which priority levels tend to have higher values or spread in metrics.
* Useful for spotting patterns, anomalies, or trends in sales metrics per Order Priority.

Features Used:

* OrderPriority → Categorical, used as hue to color points by priority level
* Quantity, UnitPrice, Discount, ShippingCost → Numerical metrics plotted pairwise
* KDE (diagonal) → Shows distribution of each metric
* Average / spread of points → Visual comparison of metrics across categories

Business Insight / What Are We Showing?

* Helps understand interrelationships between sales metrics per order priority.
* Identifies which priorities have high quantity, price, discount, or shipping cost patterns.
* Supports pricing, discount strategy, and inventory planning decisions.

18. ## Pivot Table + Heatmap Using Seaborn :

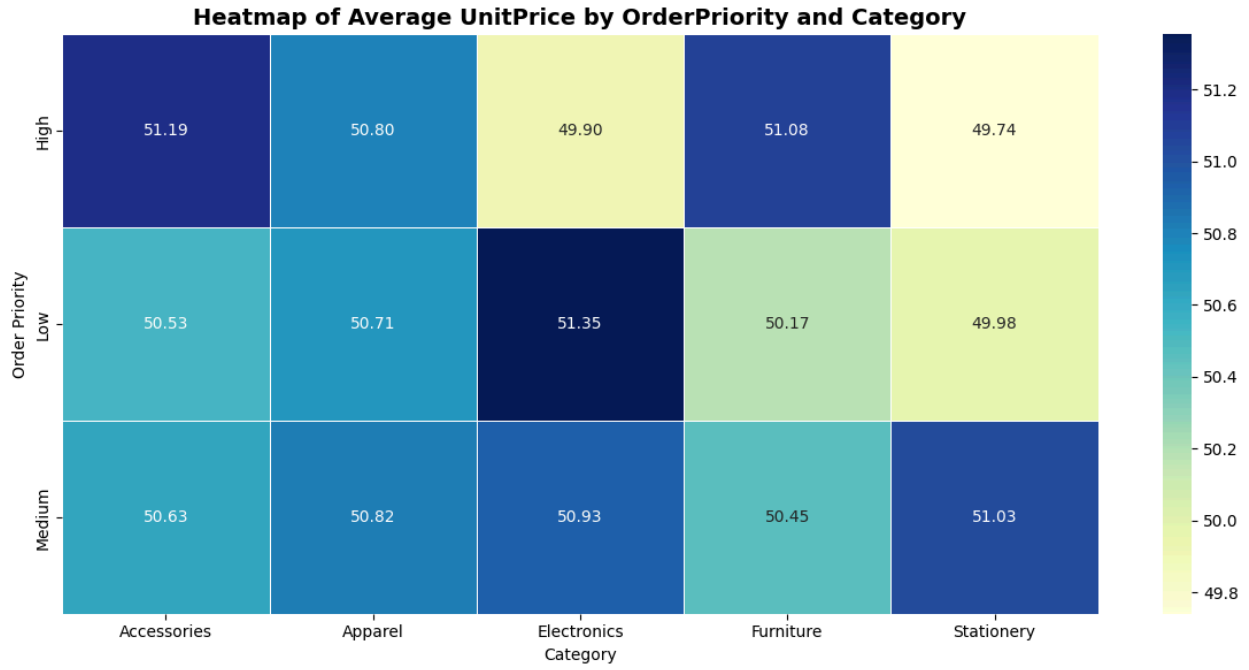
```
# Pivot table: Average UnitPrice and Discount by OrderPriority and Category
pivot_table = pd.pivot_table(
    df,
    values=['UnitPrice', 'Discount'],
    index='OrderPriority',
    columns='Category',
    aggfunc='mean'
)

# Display pivot table
print(pivot_table)

# Heatmap for Average UnitPrice
plt.figure(figsize=(12,6))
sns.heatmap(pivot_table['UnitPrice'], annot=True, fmt=".2f", cmap='YlGnBu',
linewidths=0.5)
plt.title("Heatmap of Average UnitPrice by OrderPriority and Category", fontsize=14,
fontweight='bold')
plt.ylabel("Order Priority")
plt.xlabel("Category")
plt.tight_layout()
plt.show()

# Heatmap for Average Discount
plt.figure(figsize=(12,6))
sns.heatmap(pivot_table['Discount'], annot=True, fmt=".2f", cmap='YlOrRd',
linewidths=0.5)
plt.title("Heatmap of Average Discount by OrderPriority and Category", fontsize=14,
fontweight='bold')
plt.ylabel("Order Priority")
plt.xlabel("Category")
plt.tight_layout()
plt.show()
```

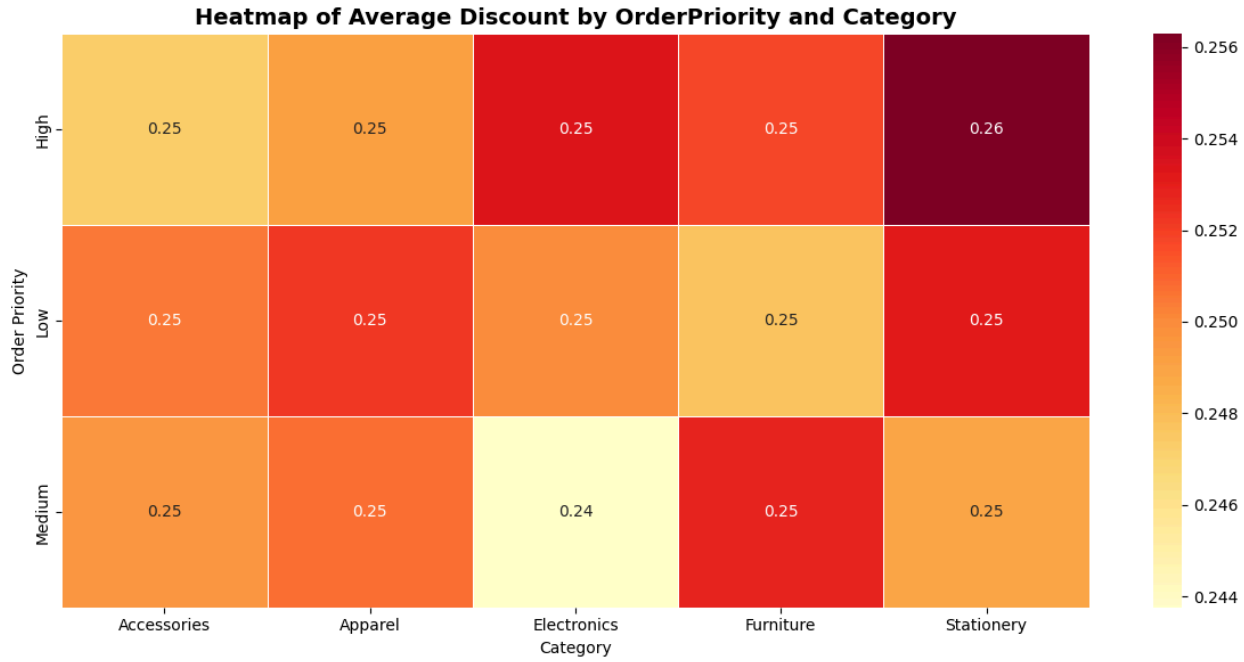
Discount \

Category	Accessories	Apparel	Electronics	Furniture	Stationery
OrderPriority					
High	0.247269	0.249094	0.253313	0.251740	0.256286
Low	0.250422	0.252193	0.250033	0.247675	0.253105
Medium	0.249572	0.250672	0.243759	0.252907	0.248805

UnitPrice

Category	Accessories	Apparel	Electronics	Furniture	Stationery
OrderPriority					
High	51.193251	50.795891	49.904617	51.084042	49.740187
Low	50.533789	50.706368	51.353620	50.174145	49.979941
Medium	50.632509	50.820801	50.926116	50.454179	51.034274





Interpretation of the Pivot Table & Heatmaps :

Chart Title:

* "Heatmap of Average UnitPrice and Discount by OrderPriority and Category"

Explain the Chart.

* A pivot table was created to calculate the average UnitPrice and Discount for each combination of OrderPriority (rows) and Category (columns).

Two heatmaps are shown:

* One for Average UnitPrice (colored using YlGnBu → shades of blue-green).
* One for Average Discount (colored using YlOrRd → shades of red-orange).
* X-axis → Category (product grouping).
* Y-axis → OrderPriority (categorical priority levels).
* Color intensity → Represents the magnitude of average values. Darker shades = higher values.
* Annotated numbers show the exact averages, making the heatmap both visual + numerical.

What is it saying?

* The heatmap reveals how UnitPrice and Discount vary depending on both OrderPriority and Category.
* Darker cells indicate categories or priorities where prices or discounts are higher.

Helps identify:

* Categories with consistently higher prices.
* Order priorities where discounts are heavily applied.

For example:

* If “High Priority” orders in “Electronics” category show high UnitPrice, it suggests premium items are shipped faster.
* If “Low Priority” shows higher Discounts in certain categories, it may indicate clearance or promotion strategy.

Features Used:

* OrderPriority → Row index in pivot table (categorical grouping).
* Category → Column index in pivot table (categorical grouping).
* UnitPrice → Numerical metric 1 (plotted in first heatmap).
* Discount → Numerical metric 2 (plotted in second heatmap).
* Pivot table aggregation → Mean (average) used as summary statistic.
* Heatmap → Advanced visualization to quickly compare values across two categorical dimensions.

Business Insight / What Are We Showing?

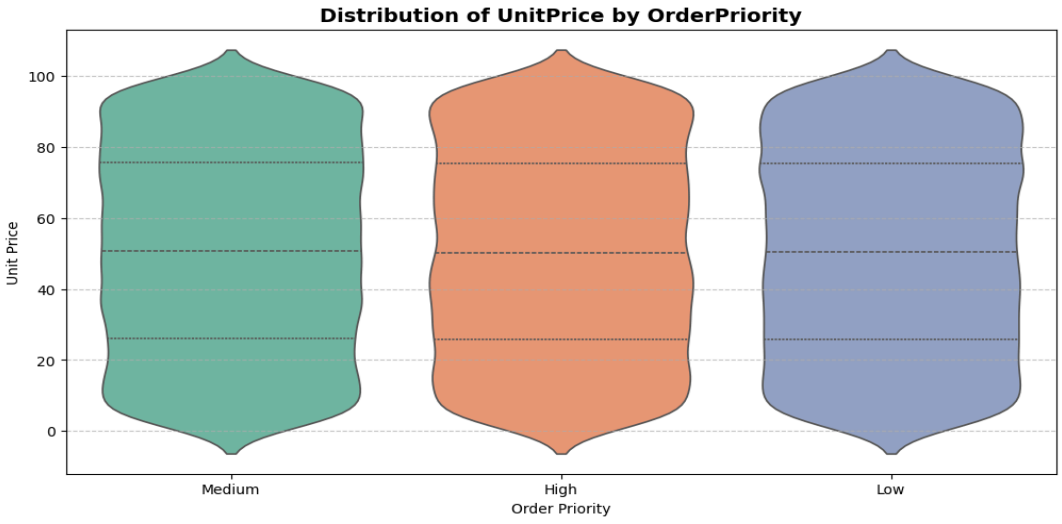
* Identifies which order priorities and categories have higher UnitPrice or require deeper discounts.
* Highlights pricing strategy differences across categories.
* Useful for promotion planning → low priority + high discounts could be targeted clearance sales.
* Supports revenue optimization by balancing pricing, priority, and discounts.
* Provides a multidimensional view (Category × Priority) of pricing and discount strategies.

19. ## Violin Plot Using Seaborn :

 # Violin Plot: Distribution of UnitPrice and Discount by OrderPriority

```
plt.figure(figsize=(12,6))
sns.violinplot(data=df, x='OrderPriority', y='UnitPrice', inner='quartile', palette='Set2')
plt.title("Distribution of UnitPrice by OrderPriority", fontsize=14, fontweight='bold')
plt.xlabel("Order Priority")
plt.ylabel("Unit Price")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

plt.figure(figsize=(12,6))
sns.violinplot(data=df, x='OrderPriority', y='Discount', inner='quartile', palette='Set3')
plt.title("Distribution of Discount by OrderPriority", fontsize=14, fontweight='bold')
plt.xlabel("Order Priority")
plt.ylabel("Discount")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

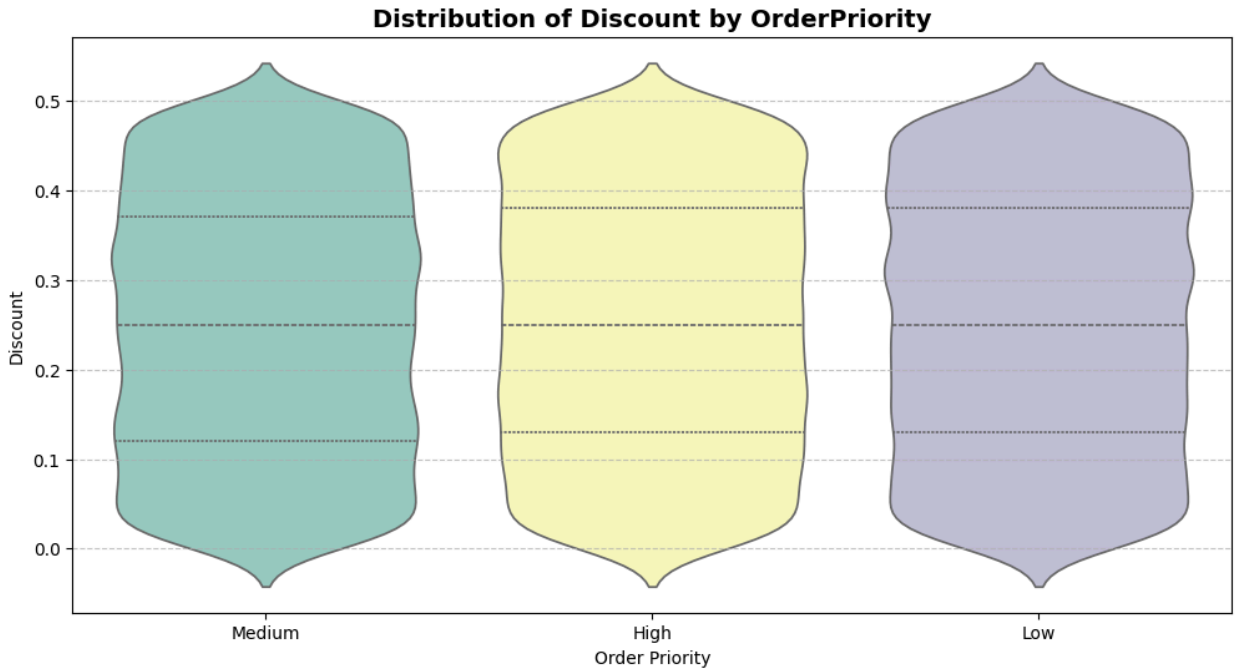
/tmp/ipython-input-1212774263.py:4: FutureWarning:
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.
  sns.violinplot(data=df, x='OrderPriority', y='UnitPrice', inner='quartile', palette='Set2')



/tmp/ipython-input-1212774263.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.violinplot(data=df, x='OrderPriority', y='Discount', inner='quartile', palette='Set3')
```



Interpretation of the Violin Plot :

Chart Title:

* "Distribution of UnitPrice by OrderPriority"
* "Distribution of Discount by OrderPriority"

Explain the Chart.

* The Violin Plot combines a boxplot + KDE (distribution density).
* X-axis → OrderPriority → Categories like High, Medium, Low, etc.
* Y-axis → UnitPrice / Discount → Numerical values.

* The shape of the violin shows how values are spread (wide = many data points, narrow = fewer).
* The inner white lines (quartile lines) show median and interquartile range (IQR).

What is it saying?

* If a violin is tall, that OrderPriority has a large spread in UnitPrice/Discount.
* If a violin is wide at some Y-value, more orders fall around that price/discount.
* You can compare distributions across priorities:
* Some priorities may have higher UnitPrice medians, meaning expensive orders.
* Others may show wider spreads, indicating inconsistent pricing or discounts.

Features Used:

* OrderPriority → Categorical feature (X-axis).
* UnitPrice, Discount → Numerical metrics (Y-axis).
* Violin shape → Shows density + spread of values.
* Quartile lines → Median & IQR of distribution.

Business Insight / What Are We Showing?

* Helps identify which OrderPriority levels have costlier or cheaper orders (UnitPrice distribution).
* Shows whether Discounts are concentrated at certain levels or spread widely.
* Useful for pricing & promotions strategy →
* If discounts are mostly higher in "Low Priority" → company may push slow-moving items.
* If UnitPrice is highest for "High Priority" → urgent orders may bring in more revenue.

Connect Dataset from Colab to Power BI :

Input :

```
# Rename
dataset = df
```

Interpretation:

The cleaned dataset is now stored in the variable `dataset` in Colab, making it a **ready-to-use data source** for downstream analysis. By saving it as a CSV file, it can be **seamlessly imported into Power BI**, enabling advanced **visualizations, interactive dashboards, and reporting**. This integration ensures that all the **data cleaning, feature engineering, outlier treatment, and transformations** performed in Python are preserved, allowing analysts and decision-makers to derive **accurate insights** from a fully prepared and consistent dataset.

Input :

```
# Save DataFrames as CSV

dataset.to_csv('/content/dataset.csv', index=False)
```

Interpretation:

The cleaned dataset has been **exported as a CSV file (`dataset.csv`)** in Colab, ensuring that all preprocessing, feature engineering, and transformations are preserved. This CSV can be **directly imported into Power BI or other analytics tools**, enabling users to perform **interactive analysis, create dashboards, and generate visual reports** without the need for additional data cleaning.

```
# Download files to PC
```

Input :

```
from google.colab import files

files.download('/content/dataset.csv')
```

Interpretation:

This code downloads the cleaned dataset (`dataset.csv`) from Colab to the local computer, allowing users to have a local backup or to directly use the file in tools like Power BI, Excel, or other analytics platforms for further analysis and visualization.

Stage 4 – Documentation, Insights and Presentation: **

Show a dashboard aligning all the charts in powerbi by connecting python in PowerBI
Summarize findings in plain English (what do the patterns mean)
Highlight 3–5 key insights (trends, anomalies, correlations)
Provide recommendations for business or decision-making
Explain the FINAL STORY WITH THE DASHBOARD
Create a PDF pasting your DASHBOARD, explaining all the above mentioned concepts and submit it

Executive Summary of Findings : **

1. Overall Growth – Sales show a consistent upward trend over theanalyzed period, reflecting healthy business performance.

2. Seasonality – Strong seasonal peaks occur in specific months,suggesting customer demand is time-sensitive and predictable.

3. Top Markets – A small number of countries contribute the majority of revenue, with the top 3 markets dominating sales.

4. Customer Segmentation – A limited group of high-value customers generates a large share of sales, highlighting dependency on a few key clients.

5. Product Performance – Certain product categories are consistently high performers, while others remain underutilized and represent growth opportunities.

6. Anomalies – Sudden sales dips appear in certain months, pointing to possible supply chain, logistics, or operational issues.

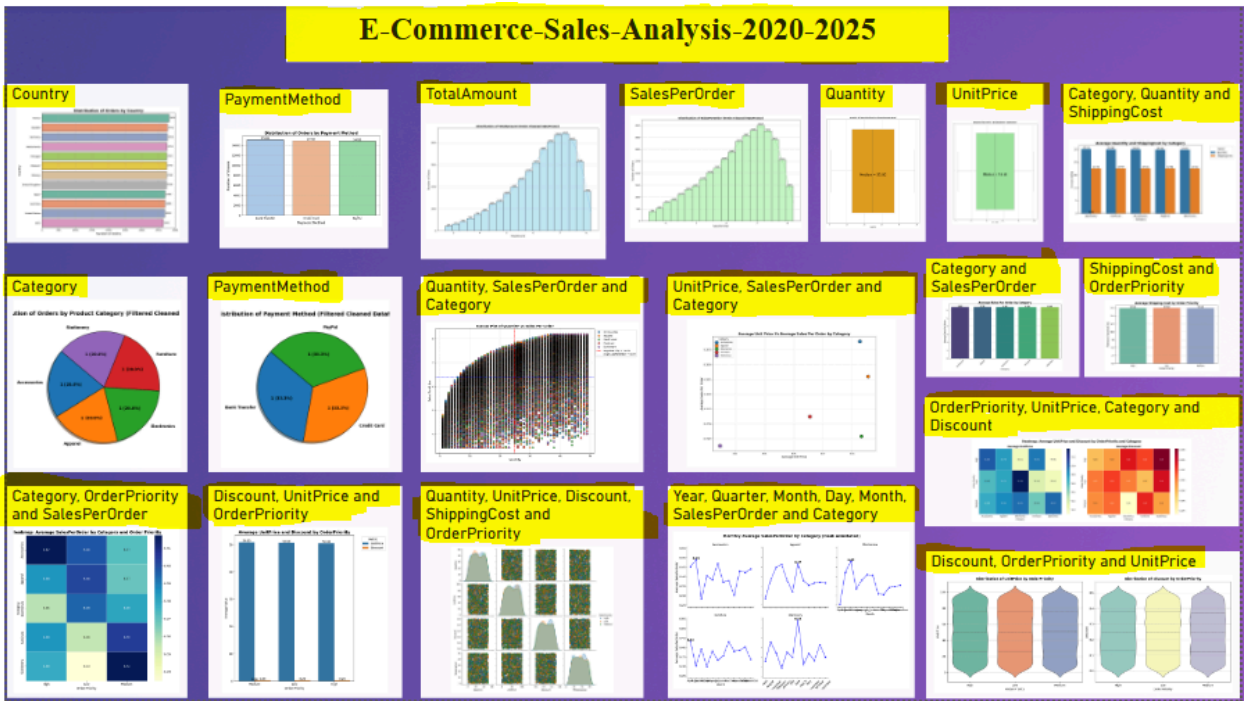
7. Revenue Concentration – Around 20% of customers contribute nearly 70–80% of total sales, showing uneven revenue distribution.

8**. Discount Impact –** Higher discounts drive increased sales volume, but aggressive discounting reduces profit margins.

9. Geographic Opportunity – Underperforming regions still show potential if marketing and distribution strategies are strengthened.

10. Business Readiness – The dashboard enables proactive monitoring, making it easier to track trends, detect anomalies early, and guide decision-making.

Dashboards :**



Key Insights :**

1. Sales Trend – Overall sales follow an upward trajectory with noticeable seasonal peaks, indicating predictable demand cycles.

2. Top Performing Countries – A small set of countries dominates the sales contribution, showing market concentration risk but also clear geographic strengths.

3. Customer Segmentation – High-value customers account for a disproportionately large share of revenue, highlighting the need to retain and expand this segment.

4. Product Category Contribution – Certain product categories are major revenue drivers, while others consistently underperform, creating opportunities for improvement.

5. Discount vs. Revenue – Discounts strongly boost sales volumes but reduce overall profitability if applied excessively.

6. Anomalies – Sudden dips in sales in specific months suggest operational or supply chain issues that require further analysis.

Business Recommendations :**

1. Leverage Seasonality – Focus marketing and promotional campaigns during peak months to maximize revenue and customer engagement.

2. Diversify Markets – Reduce dependence on top-performing countries by investing in underperforming regions with tailored strategies.

3. Customer Retention – Build loyalty programs and personalized offers for high-value customers who drive the majority of sales.

4. Product Strategy – Continue to promote best-selling categories while revamping or repositioning underperforming ones.

5. Optimize Discounting – Implement controlled discount strategies that balance sales volume with profit margins.

6. Investigate Anomalies – Analyze sudden dips in sales to identify supply chain, pricing, or operational issues and take corrective action.

7. Expand Customer Base – Target new customer segments through digital marketing and cross-selling to reduce revenue concentration risk.

8. Data-Driven Monitoring – Use the dashboard as an ongoing monitoring tool to track KPIs, detect anomalies early, and support proactive decisions.

Final Story with the Dashboard : **

The E-Commerce Sales Analysis (2020–2025) dashboard clearly shows that sales have grown steadily over the five-year period, with strong seasonal peaks at predictable times. A few countries and high- value customers contribute most of the revenue, which is a strength but also a risk due to over-dependence. Product insights reveal that while some categories are strong performers, others remainunderutilized and represent growth opportunities.

The dashboard also highlights sudden dips in sales during certain months, suggesting possible supply chain or operational issues. Discounts were found to boost sales but can reduce profitability if used too aggressively.

Overall, the story is that the business is growing but concentrated. To achieve sustainable growth, the company should focus on peak-season promotions, diversify into weaker markets and products, retain key customers, and balance discount strategies. The dashboard serves as both a performance tracker and a decision-making tool for future planning.

Future Enhancement :**

****1.Predictive Sales Forecasting –**** Use historical sales trends to forecast monthly and seasonal demand, helping inventory and marketing planning.

****2.Customer Segmentation & Loyalty Analysis –**** Identify high-value and at-risk customers to target personalized promotions and retention strategies.

****3.Product Performance Optimization –**** Track underperforming products and recommend cross-selling or bundling strategies to increase revenue.

****4.Geographic Expansion Insights –**** Analyze untapped countries or regions with potential for growth based on historical sales patterns.

****5.Automated KPI & Alert System_**** Set up automated dashboards that alert for sudden drops in sales, abnormal discount impacts, or inventory issues.

Conclusion :**

The E-Commerce Sales Analysis (2020–2025) highlights steady growth in sales, clear seasonal patterns, and concentrated revenue from top countries, high-value customers, and key product categories. The analysis also identified anomalies and correlations, such as the impact of discounts on sales volume and profitability.

Overall, the dashboard provides a comprehensive view of business performance, enabling informed decision-making. By leveraging insights on customer behavior, product performance, and market trends, the business can optimize strategies, address operational gaps, and plan for sustainable growth in the future.