Superstore Performance Dataset – Exploratory and Descriptive Analysis

In this notebook, we focus on the **exploratory and descriptive analysis** of the cleaned version of the **Superstore Performance Dataset**, a popular dataset often used for practicing data analysis, visualization, and business intelligence tasks. The dataset contains retail transaction records, customer details, product categories, and regional performance data.

Effective exploratory analysis is crucial for uncovering patterns, trends, and potential issues in the data, which guides further analysis and decision-making. Here, we examine key metrics such as **sales, profit, discount, and quantity**, as well as their distribution across **categories, regions, and time periods**.

We start by importing essential Python libraries for data handling, analysis, and visualization:

- `pandas` for structured data operations.
- `numpy` for numerical operations.
- `os` for interacting with the operating system and directory structures.

```
# Import libraries

import pandas as pd
import numpy as np
import os
```

Define and Create Directory Paths

To ensure reproducibility and organized storage, we programmatically create directories for:

- **raw data**
- **processed data**
- **results**

- **documentation**

These directories will store intermediate and final outputs for reproducibility.

```
# Get working directory
current_dir = os.getcwd()
# Go one directory up to the root directory
project_root_dir = os.path.dirname(current_dir)
# define paths to the data files
data_dir = os.path.join(project_root_dir, 'data')
raw_dir = os.path.join(data_dir, 'raw')
processed_dir = os.path.join(data_dir, 'processed')
# Define paths to the results folder
results_dir  = os.path.join(project_root_dir, 'results')
# Define paths to the docs folder
docs_dir = os.path.join(project_root_dir, 'docs')

# create directories if they do not exist
os.makedirs(raw_dir, exist_ok = True )
os.makedirs(processed_dir, exist_ok = True )
os.makedirs(results_dir, exist_ok = True)
os.makedirs(docs_dir, exist_ok = True)
```

Read in the data

We load the **Superstore Performance Dataset** as an Excel file (`Superstore.xlsx`). The dataset consists of multiple sheets: `Orders`, `Returns`, and `People`. These sheets are read separately and then merged to form a comprehensive dataset for analysis.

Key considerations

- **File structure:** The dataset is stored in a multi-sheet Excel file. It's important to read each sheet carefully and ensure the correct relationships between them.
- **Merging data:**
  - The `Orders` and `Returns` sheets are merged using `Order ID` to identify which orders were returned. We use a **left join** to keep all orders and add return information where available.
  - The `People` sheet is merged on `Region` to associate each order with the responsible regional manager.

- **Data integrity:** The merge operations may introduce missing values (e.g., orders that weren't returned or regions with no associated manager). These should be handled carefully in later analysis.
- **File path handling:** The use of `os.path.join` ensures that file paths are constructed dynamically, improving portability across different systems and directory structures.
- **Previewing the data:** Displaying the first 10 rows (`head(10)`) helps confirm that the data has been loaded and merged as expected before further analysis.

```python
store_data_filename = os.path.join(raw_dir, "Superstore.xlsx")

# Load the sheets
orders_df = pd.read_excel(store_data_filename, sheet_name='Orders')
returns_df = pd.read_excel(store_data_filename, sheet_name='Returns')
people_df = pd.read_excel(store_data_filename, sheet_name='People')

# Merge orders with returns on Order ID
orders_with_returns = pd.merge(orders_df, returns_df, on='Order ID', how='left')

# Merge with people on Region
merged_store = pd.merge(orders_with_returns, people_df, on='Region', how='left')

# Show first 10 rows
merged_store.head(10)
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell
5	6	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman
6	7	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman
7	8	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman
8	9	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman
9	10	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman

```python
merged_store.shape
```

```
(9994, 23)
```

```python
merged_store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9994 non-null   object
 2   Order Date     9994 non-null   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
 21  Returned       800 non-null    object
 22  Person         9994 non-null   object
dtypes: datetime64[ns](2), float64(3), int64(3), object(15)
memory usage: 1.8+ MB
```

Data Cleaning

1. Understanding the Dataset

Before proceeding with data cleaning, it is essential to understand the variables in the **Superstore** dataset. This helps us determine appropriate cleaning and transformation strategies. The table below summarizes the types, descriptions, and typical values or ranges of the variables in our dataset.

Table 1: Summary of variables in the dataset

Variable	Type	Description	Example Values / Range
Customer ID	Categorical	Unique identifier for each customer	CG-12520, TB-20145
Customer Name	Categorical	Full name of the customer	Claire Gute, Sean Miller
Segment	Categorical	Market segment the customer belongs to	Consumer, Corporate, Home Office
Country	Categorical	Country of transaction	United States
City	Categorical	City where the order was placed	Los Angeles, New York
State	Categorical	State where the order was placed	California, Texas
Postal Code	Categorical/Numeric	Postal code of customer location	90036, 10024
Region	Categorical	Regional division of the business	West, East, Central, South
Product ID	Categorical	Unique identifier for product	FUR-BO-10001798
Category	Categorical	Product category	Furniture, Office Supplies, Technology
Sub-Category	Categorical	Product sub-category	Bookcases, Chairs, Phones
Product Name	Categorical	Name of the product	Bush Somerset Collection Bookcase
Sales	Numeric	Dollar amount of sales	2.99 – 22,638.48
Quantity	Numeric	Number of units sold	1 – 14
Discount	Numeric	Discount applied to sale (0 to 1)	0.0 – 0.8
Profit	Numeric	Profit amount in dollars	-6599.98 – 8399.98
Returned	Categorical	Indicates if product was returned	Yes, No, (can be missing)
Person	Categorical	Salesperson or handler	Cassandra Brandow, Anna Andreadi

```python
merged_store.columns
```

```
Index(['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode',
       'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State',
       'Postal Code', 'Region', 'Product ID', 'Category', 'Sub-Category',
       'Product Name', 'Sales', 'Quantity', 'Discount', 'Profit', 'Returned',
       'Person'],
      dtype='object')
```

2. Deal with missing values

To ensure the quality of our dataset, we performed an initial check for missing values in the merged dataset `merged_store`. The following code was used to identify any null values across all columns:

```python
merged_store.isnull().sum()
```

```
Row ID              0
Order ID            0
Order Date          0
Ship Date           0
Ship Mode           0
Customer ID         0
Customer Name       0
Segment             0
Country             0
City                0
State               0
Postal Code         0
Region              0
Product ID          0
Category            0
Sub-Category        0
Product Name        0
Sales               0
Quantity            0
Discount            0
Profit              0
Returned         9194
Person              0
dtype: int64
```

Using the `.isnull().sum()` function, we identified columns with missing values in the **Superstore** dataset. The analysis revealed:

- `Returned` has **9,194 missing values**

To handle this, we applied the following data cleaning approach:

- Imputed missing values in the **`Returned`** column with `"NO"` to indicate that these transactions are considered not returned, in the absence of return info].fillna('NO')

```
merged_store['Returned'] = merged_store['Returned'].fillna('NO')
```

```
merged_store.isnull().sum()
```

```
Row ID           0
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country          0
City             0
State            0
Postal Code      0
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
Quantity         0
Discount         0
Profit           0
Returned         0
Person           0
dtype: int64
```

3. Removing Duplicates

Duplicates can distort statistical summaries and model performance. Using `.duplicated().sum()`, we count duplicate records.

We confirm that we have no duplicates in the dataset.

```
merged_store.duplicated().sum()
```

```
0
```

We also examined the current structure of the dataset and confirmed that it contains **9,994 rows** and **23 columns**, representing a comprehensive collection of retail transaction records.

```
merged_store.shape
```

```
(9994, 23)
```

Finally, we save the clean, processed dataset as a CSV file in our processed directory for future modelling and analysis.

```
merged_store.to_csv('SuperStore-Cleaned.csv', index=False)
```