Superstore Dataset - Data Cleaning and Preprocessing

In this notebook, we focus on data preparation, cleaning, and preprocessing for the Superstore dataset, a comprehensive dataset often used for sales analysis, customer segmentation, and profit prediction tasks based on various order, product, and customer attributes.

Good data preprocessing is crucial for reliable and interpretable results in business intelligence and analytics workflows. Here, I address common data issues such as missing values, duplicates, and inconsistent categorical labels, while creating derived features to improve downstream analysis.

I start by importing essential Python libraries for data handling and manipulation.

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

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

Define and Create Directory Paths

To ensure reproducibility andorganized storage, we programmatically create directories for:

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

These directories will store intermediate and final outputs for reproducibility.

Define and Create Paths

```
# Get current working directory
current_dir = os.getcwd()

# Go one directory up (assuming script is inside a subfolder like 'notebooks')
project_root_dir = os.path.dirname(current_dir)

# Define key folder paths
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')
results_dir = os.path.join(project_root_dir, 'results')
docs_dir = os.path.join(project_root_dir, 'docs')

# Create directories if they don't 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)
```

Load Datasets

Three key datasets—'Orders', 'Returns', and 'People'—are loaded from the Superstore.xlsx Excel file into separate pandas DataFrames.

```
# Define the full path to your Excel file
excel_file_path = os.path.join(raw_dir, "Superstore.xlsx")

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


print("\nOrders DataFrame Head:")
print(orders_df.head())


print("\nReturns DataFrame Head:")
```

```python
print(returns_df.head())


print("\nPeople DataFrame Head:")
print(people_df.head())
```

```
Orders DataFrame Head:
   Row ID        Order ID  Order Date   Ship Date       Ship Mode Customer ID  \
0       1  CA-2016-152156  2016-11-08  2016-11-11    Second Class    CG-12520
1       2  CA-2016-152156  2016-11-08  2016-11-11    Second Class    CG-12520
2       3  CA-2016-138688  2016-06-12  2016-06-16    Second Class    DV-13045
3       4  US-2015-108966  2015-10-11  2015-10-18  Standard Class    SO-20335
4       5  US-2015-108966  2015-10-11  2015-10-18  Standard Class    SO-20335

     Customer Name    Segment        Country             City  ...  \
0      Claire Gute   Consumer  United States        Henderson  ...
1      Claire Gute   Consumer  United States        Henderson  ...
2  Darrin Van Huff  Corporate  United States      Los Angeles  ...
3   Sean O'Donnell   Consumer  United States  Fort Lauderdale  ...
4   Sean O'Donnell   Consumer  United States  Fort Lauderdale  ...

  Postal Code  Region       Product ID         Category Sub-Category  \
0       42420   South  FUR-BO-10001798        Furniture    Bookcases
1       42420   South  FUR-CH-10000454        Furniture       Chairs
2       90036    West  OFF-LA-10000240  Office Supplies       Labels
3       33311   South  FUR-TA-10000577        Furniture       Tables
4       33311   South  OFF-ST-10000760  Office Supplies      Storage

                                        Product Name     Sales  Quantity  \
0                  Bush Somerset Collection Bookcase  261.9600         2
1  Hon Deluxe Fabric Upholstered Stacking Chairs,...  731.9400         3
2  Self-Adhesive Address Labels for Typewriters b...   14.6200         2
3      Bretford CR4500 Series Slim Rectangular Table  957.5775         5
4                     Eldon Fold 'N Roll Cart System   22.3680         2

   Discount    Profit
0      0.00   41.9136
1      0.00  219.5820
2      0.00    6.8714
3      0.45 -383.0310
4      0.20    2.5164
```

```
[5 rows x 21 columns]

Returns DataFrame Head:
  Returned        Order ID
0      Yes  CA-2017-153822
1      Yes  CA-2017-129707
2      Yes  CA-2014-152345
3      Yes  CA-2015-156440
4      Yes  US-2017-155999

People DataFrame Head:
              Person   Region
0      Anna Andreadi     West
1        Chuck Magee     East
2     Kelly Williams  Central
3  Cassandra Brandow    South
```

Data Cleaning

1. Data Merging

This section focuses on integrating the loaded datasets to create a unified DataFrame.

Merge Orders and Returns

The 'Orders' DataFrame is merged with the 'Returns' DataFrame using a left join on 'Order ID'. This ensures that all order records are retained, and return information is added where available.

```
# Merge returns into orders (left join to keep all orders)
merged_df = pd.merge(orders_df, returns_df, on='Order ID', how='left')
merged_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
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 Hu
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	2014-01-21	2014-01-23	Second Class	TB-21400	Tom Boeckenh
9990	9991	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9991	9992	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9992	9993	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9993	9994	CA-2017-119914	2017-05-04	2017-05-09	Second Class	CC-12220	Chris Cortes

Merge with People Data:

The resulting merged DataFrame is then further merged with the 'People' DataFrame. This merge is performed using a left join on the 'Region' column, associating sales representatives with their respective regions.

```
# Merge the result with people data (left join to preserve all order records)
final_merged_df = pd.merge(merged_df, people_df, on='Region', how='left')
final_merged_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
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 Hu
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	2014-01-21	2014-01-23	Second Class	TB-21400	Tom Boeckenh
9990	9991	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9991	9992	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9992	9993	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9993	9994	CA-2017-119914	2017-05-04	2017-05-09	Second Class	CC-12220	Chris Cortes

```
final_merged_df.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
```

```python
final_merged_df.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
final_merged_df.shape
```

```
(9994, 23)
```

```
final_merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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
```

2. Understanding the dataset

Before proceeding with the cleaning, we would like to understand the variables deeply. This would help guide the cleaning process. The subsequent tables detail the types, meaning, and values or ranges of the variables in the Superstore dataset.

Table 1: Summary table of the variables in the dataset

Variable	Type	Description	Values / Range (excluding NaN)
Order ID	Categorical	Unique identifier for each order	Unique alphanumeric codes
Order Date	Date	Date when the order was placed	Dates ranging from 2014 to 2017
Ship Date	Date	Date when the order was shipped	Dates ranging from 2014 to 2017
Ship Mode	Categorical	Shipping method used	'Second Class', 'Standard Class', 'First Class', 'Same Day'
Customer ID	Categorical	Unique identifier for each customer	Unique alphanumeric codes
Customer Name	Categorical	Name of the customer	Text names
Segment	Categorical	Customer segment	'Consumer', 'Corporate', 'Home Office'
Country	Categorical	Country where the order was placed	'United States'
City	Categorical	City where the order was placed	Various city names (e.g., 'New York City', 'Los Angeles')
State	Categorical	State where the order was placed	All 50 U.S. states and D.C.
Postal Code	Numeric	Postal code of the delivery address	10001 – 99301
Region	Categorical	Geographic region	'East', 'Central', 'South', 'West'
Product ID	Categorical	Unique identifier for each product	Unique alphanumeric codes
Category	Categorical	Main product category	'Furniture', 'Office Supplies', 'Technology'
Sub-Category	Categorical	Sub-category of the product	'Bookcases', 'Chairs', 'Phones', 'Storage'
Product Name	Categorical	Name of the product	Various product descriptions
Sales	Numeric	Sales amount for the product	0.444 – 22,638.48
Quantity	Numeric	Quantity of the product ordered	1 – 14
Discount	Numeric	Discount applied to the product	0.0 – 0.8
Profit	Numeric	Profit generated from the product	-6,599.978 – 8,399.976
Returned	Categorical	Indicates if the order was returned	'Yes', 'No'

Person	Categorical	Sales manager responsible for region	'Anna Andrus', 'Chuck Magee', 'Kelly Williams', 'Cassandra Brandow'

Table 2: Categorical Variables Table

Variable	Unique Value	Description
Ship Mode	Second Class	Standard shipping, typically slower than First Class
	Standard Class	Most common and often slowest shipping option
	First Class	Faster shipping option, quicker than Second Class
	Same Day	Fastest shipping option, delivery on the same day
Segment	Consumer	Individual customers purchasing for personal use
	Corporate	Business customers, typically mid-sized companies
	Home Office	Small business or work-from-home customers
Category	Furniture	Products related to furniture
	Office Supplies	Products for office use
	Technology	Electronic devices and related accessories
Returned	Yes	The order was returned
	No	The order was not returned
Person	Anna Andrus	Sales manager for a West region
	Chuck Magee	Sales manager for a East region
	Kelly Williams	Sales manager for a Central region
	Cassandra Brandow	Sales manager for a South region
State	(Various US States)	State where the order was placed (e.g., California, New York)
Region	East	Orders from the Eastern United States
	Central	Orders from the Central United States
	South	Orders from the Southern United States
	West	Orders from the Western United States
Sub-Category	(Various Sub-Categories)	Detailed product classifications (e.g., 'Phones', 'Binders')

```python
print("\nUnique Ship Modes:")
print(np.unique(final_merged_df['Ship Mode'].dropna().to_list()))
```

```
Unique Ship Modes:
['First Class' 'Same Day' 'Second Class' 'Standard Class']
```

```python
# Unique Segments
print("\nUnique Segments:")
print(np.unique(final_merged_df['Segment'].dropna().to_list()))
```

```
Unique Segments:
['Consumer' 'Corporate' 'Home Office']
```

```python
# Unique Categories
print("\nUnique Categories:")
print(np.unique(final_merged_df['Category'].dropna().to_list()))
```

```
Unique Categories:
['Furniture' 'Office Supplies' 'Technology']
```

```python
# Unique Regions
print("\nUnique Regions:")
print(np.unique(final_merged_df['Region'].dropna().to_list()))
```

```
Unique Regions:
['Central' 'East' 'South' 'West']
```

```python
print("\nUnique Sub-Categories:")
print(np.unique(final_merged_df['Sub-Category'].dropna().to_list()))
```

```
Unique Sub-Categories:
['Accessories' 'Appliances' 'Art' 'Binders' 'Bookcases' 'Chairs' 'Copiers'
 'Envelopes' 'Fasteners' 'Furnishings' 'Labels' 'Machines' 'Paper'
 'Phones' 'Storage' 'Supplies' 'Tables']
```

```python
print("\nUnique States:")
print(np.unique(final_merged_df['State'].dropna().to_list()))
```

```
Unique States:
['Alabama' 'Arizona' 'Arkansas' 'California' 'Colorado' 'Connecticut'
 'Delaware' 'District of Columbia' 'Florida' 'Georgia' 'Idaho' 'Illinois'
 'Indiana' 'Iowa' 'Kansas' 'Kentucky' 'Louisiana' 'Maine' 'Maryland'
 'Massachusetts' 'Michigan' 'Minnesota' 'Mississippi' 'Missouri' 'Montana'
 'Nebraska' 'Nevada' 'New Hampshire' 'New Jersey' 'New Mexico' 'New York'
 'North Carolina' 'North Dakota' 'Ohio' 'Oklahoma' 'Oregon' 'Pennsylvania'
 'Rhode Island' 'South Carolina' 'South Dakota' 'Tennessee' 'Texas' 'Utah'
 'Vermont' 'Virginia' 'Washington' 'West Virginia' 'Wisconsin' 'Wyoming']
```

```
print("\nUnique Returned Statuses:")
print(np.unique(final_merged_df['Returned'].dropna().to_list()))
```

```
Unique Returned Statuses:
['Yes']
```

3. Deal with missing values

Handle Missing Values

The 'Returned' column, which contained a significant number of missing values (NaN), is imputed by filling these entries with the string 'No'. This indicates that orders without a return record are considered not returned.

Replace NaN in 'Returned' column with 'No'

```
final_merged_df['Returned'] = final_merged_df['Returned'].fillna('No')
final_merged_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
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 Hu
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	2014-01-21	2014-01-23	Second Class	TB-21400	Tom Boeckenh
9990	9991	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
9991	9992	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9992	9993	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9993	9994	CA-2017-119914	2017-05-04	2017-05-09	Second Class	CC-12220	Chris Cortes

3. Convert Data Types

Create new feature: Shipping Duration in days

The 'Order Date' and 'Ship Date' columns are converted to datetime objects, enabling proper chronological analysis and operations.

```
final_merged_df['Shipping Duration'] = (
    final_merged_df['Ship Date'] - final_merged_df ['Order Date']
).dt.days
final_merged_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
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 Hu
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	2014-01-21	2014-01-23	Second Class	TB-21400	Tom Boeckenh
9990	9991	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9991	9992	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9992	9993	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9993	9994	CA-2017-119914	2017-05-04	2017-05-09	Second Class	CC-12220	Chris Cortes

4. Extract order year and month for trend analysis

This step creates two new columns—Order Year and Order Month by extracting the year and month from the Order Date column using pandas' .dt accessor. These variables are useful for performing time-based trend analysis, such as identifying seasonal patterns or yearly growth in sales.

```
final_merged_df['Order Year'] = final_merged_df['Order Date'].dt.year
final_merged_df['Order Month'] = final_merged_df['Order Date'].dt.month
final_merged_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
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 Hu
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	2014-01-21	2014-01-23	Second Class	TB-21400	Tom Boeckenh
9990	9991	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9991	9992	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9992	9993	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9993	9994	CA-2017-119914	2017-05-04	2017-05-09	Second Class	CC-12220	Chris Cortes

5. Trim text columns of leading/trailing whitespace

This step ensures the consistency and cleanliness of textual data by removing any unnecessary leading or trailing whitespace from string-type columns. This is a crucial universal cleanup practice that prevents issues during data analysis, filtering, or merging operations caused by subtle differences in string values due to whitespace.

The process involves:

- Identifying Text Columns: All columns with an 'object' data type (typically representing strings) are selected from the final_merged_df.
- Applying Whitespace Trim: For each identified text column, the .str.strip() method is applied to every string entry. This method efficiently removes any spaces, tabs, or newlines from the beginning and end of the text, standardizing the data.

```
text_cols = final_merged_df.select_dtypes(include='object').columns
final_merged_df[text_cols] = final_merged_df[text_cols].apply(lambda x: x.str.strip())
final_merged_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Nan
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 Hu
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donne
...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	2014-01-21	2014-01-23	Second Class	TB-21400	Tom Boeckenh
9990	9991	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9991	9992	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9992	9993	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	Dave Brooks
9993	9994	CA-2017-119914	2017-05-04	2017-05-09	Second Class	CC-12220	Chris Cortes

```python
final_merged_df.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
Shipping Duration    0
Order Year           0
Order Month          0
```

```
dtype: int64
```

6. Deal with Duplicates

Duplicate rows across the entire DataFrame are identified and removed to ensure data uniqueness and integrity. The process confirms that no duplicate entries remain after this operation, resulting in a cleaned DataFrame of (9994, 26) dimensions.

```
final_merged_df.duplicated().sum()
```

```
0
```

```
final_merged_df.shape
```

```
(9994, 26)
```

Save the Cleaned DataFrame to a CSV file

The final step involves persisting the cleaned and merged dataset for future use.

Export to CSV: The final_merged_df, now cleaned and preprocessed, is saved as 'final_superstore_cleaned.csv' within the designated 'processed data' directory. The 'index=False' argument ensures that the DataFrame index is not written to the CSV file.

```
# Define the full path for the output CSV
output_file = os.path.join(processed_dir, 'final_superstore_cleanedd.csv')

# Save the cleaned DataFrame
final_merged_df.to_csv(output_file, index=False)
```