01-data_exploration

February 10, 2025

1 Superstore Sales Data Cleaning

This repository contains a project for cleaning and transforming a messy **Superstore Sales Data** dataset. The dataset includes sales records from a retail business and may contain issues such as missing values, duplicates, and inconsistent formatting, which were addressed to prepare the data for analysis and visualization.

1.1 Package Importing

To begin, we import the necessary Python packages required for data exploration and cleaning.

[1]:
```
# In notebook (01-data_exploration.ipynb)
import pandas as pd
import sys
import os
import numpy as np
```

We also configure the environment to ensure the project directory is accessible.

[2]:
```
# Add the parent directory to sys.path
sys.path.append(os.path.abspath(os.path.join(os.getcwd(), '..')))

from superstore_sales.config import RAW_DATA_FILE
import pandas as pd

df_raw = pd.read_csv(RAW_DATA_FILE, encoding='ISO-8859-1')
df_clean = df_raw.copy()
```

1.2 Initial Data Exploration

To understand the dataset, we first examine its structure and content.

1.2.1 Checking Data Structure

[3]:
```
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 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   object
 3   Ship Date      9994 non-null   object
 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
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
```

1.2.2 Initial Observations

- There are no null values in the dataset.
- Some columns require a data type change:
  - Numerical values stored as `object` should be converted to `int` or `float`.
  - Date columns should be converted to `datetime`.
  - Categorical variables can be optimised using the `category` type.

[4]:
```
display(df_clean.iloc[:,0:9].sample(5))
display(df_clean.iloc[:,10:].sample(5))
```

```
      Row ID        Order ID Order Date  Ship Date       Ship Mode  \
7237    7238  CA-2016-164924  7/10/2016  7/10/2016        Same Day
8118    8119  US-2017-106551  7/22/2017  7/27/2017  Standard Class
2385    2386  US-2017-117534  3/25/2017  3/26/2017     First Class
4421    4422  CA-2014-117016   3/4/2014   3/9/2014  Standard Class
6105    6106  CA-2015-147011  6/18/2015  6/22/2015  Standard Class

     Customer ID           Customer Name    Segment        Country
7237    EA-14035            Erin Ashbrook  Corporate  United States
8118    EB-13930             Eric Barreto   Consumer  United States
2385    CV-12295  Christina VanderZanden   Consumer  United States
4421    SC-20095             Sanjit Chand   Consumer  United States
```

```
6105    HR-14830           Harold Ryan  Corporate  United States

         State  Postal Code   Region       Product ID         Category  \
8893     Texas        77590  Central  FUR-TA-10001307        Furniture
2656      Utah        84604     West  OFF-ST-10001272  Office Supplies
996   Kentucky        42420    South  OFF-EN-10003862  Office Supplies
5678  Michigan        49423  Central  OFF-FA-10000735  Office Supplies
3051     Idaho        83642     West  OFF-AP-10004336  Office Supplies

     Sub-Category                                       Product Name   Sales  \
8893       Tables  SAFCO PlanMaster Heigh-Adjustable Drafting Tab…  489.23
2656      Storage       Mini 13-1/2 Capacity Data Binder Rack, Pearl  261.74
996     Envelopes                 Laser & Ink Jet Business Envelopes   10.67
5678    Fasteners                                            Staples   20.44
3051   Appliances  Conquest 14 Commercial Heavy-Duty Upright Vacu…  227.84

      Quantity  Discount   Profit
8893         2       0.3  41.9340
2656         2       0.0  65.4350
996          1       0.0   4.9082
5678         7       0.0   9.1980
3051         4       0.0  66.0736
```

[5]:
```python
df_clean.describe()
```

[5]:
```
            Row ID   Postal Code         Sales     Quantity     Discount  \
count  9994.000000   9994.000000   9994.000000  9994.000000  9994.000000
mean   4997.500000  55190.379428    229.858001     3.789574     0.156203
std    2885.163629  32063.693350    623.245101     2.225110     0.206452
min       1.000000   1040.000000      0.444000     1.000000     0.000000
25%    2499.250000  23223.000000     17.280000     2.000000     0.000000
50%    4997.500000  56430.500000     54.490000     3.000000     0.200000
75%    7495.750000  90008.000000    209.940000     5.000000     0.200000
max    9994.000000  99301.000000  22638.480000    14.000000     0.800000

            Profit
count  9994.000000
mean     28.656896
std     234.260108
min   -6599.978000
25%       1.728750
50%       8.666500
75%      29.364000
max    8399.976000
```

[6]:
```python
df_clean.describe(include='object')
```

[6]:
```
                 Order ID Order Date   Ship Date       Ship Mode Customer ID  \
count                9994       9994        9994            9994        9994
unique               5009       1237        1334               4         793
top        CA-2017-100111   9/5/2016  12/16/2015  Standard Class    WB-21850
freq                   14         38          35            5968          37

        Customer Name   Segment        Country           City       State  \
count            9994      9994           9994           9994        9994
unique            793         3              1            531          49
top     William Brown  Consumer  United States  New York City  California
freq               37      5191           9994            915        2001

       Region        Product ID         Category Sub-Category     Product Name
count    9994              9994             9994         9994             9994
unique      4              1862                3           17             1850
top      West   OFF-PA-10001970  Office Supplies      Binders  Staple envelope
freq     3203                19             6026         1523               48
```

1.2.3 Observations from Data Exploration

1. **Data Sampling:**
   - Two separate random samples were displayed: one from columns 0 to 9 and another from column 10 onwards.
   - This method allows for a quick overview of different sections of the dataset.
2. **Summary Statistics (`df_clean.describe()`):**
   - The dataset contains 9,994 entries.
   - **Sales, Profit, and Discount:**
     - Sales have a wide range, from a minimum of 0.44 to a maximum of 22,638.48.
     - Profit values vary significantly, from -6,599.98 to 8,399.98, indicating potential losses and gains.
     - Discounts range from 0 to 0.8, showing varying discount strategies.
   - **Quantity Distribution:**
     - The quantity per transaction varies from 1 to 14, with a median of 3.
   - **Postal Code Analysis:**
     - The mean postal code is around 55,190, with significant variation (std = 32,063), indicating geographic diversity in the data.
3. **Categorical Data Summary:**
   - The dataset includes categorical fields such as `Order ID`, `Customer ID`, `Product ID`, `Region`, `State`, `City`, `Category`, `Sub-Category`, and `Ship Mode`.
   - Unique counts reveal that there are 5,009 distinct orders, suggesting repeat customers or multi-product orders.
   - The presence of unique customer IDs implies customer-level tracking.
4. **Potential Areas for Further Investigation:**
   - The large standard deviation in profit suggests significant variability in product performance.
   - The presence of negative profits needs further exploration—certain products or regions may be underperforming.
   - Sales and discount correlation analysis could provide insights into pricing strategies.

[7]:
```python
df_clean.columns
```

[7]:
```
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'],
      dtype='object')
```

1.2.4 Data Validation Check

The values for the following transformations were verified to ensure correctness and consistency:

- **Row ID and Postal Code**: Converted to string format and confirmed to have no incorrect or missing values. Postal codes were checked to ensure they follow a uniform 5-digit format.

- **Order Date and Ship Date**: Successfully converted to datetime format, with no invalid or misformatted entries.

- **Categorical Columns**: Verified that 'Ship Mode', 'Segment', 'Country', 'Region', 'Category', and 'Sub-Category' contain only valid and expected values. No unexpected categories or misclassified data were found.

All transformations were validated, and the data is clean and ready for further analysis.

[8]:
```python
# Change numbers to strings
df_clean[['Row ID', 'Postal Code']] = df_clean[['Row ID', 'Postal Code']].
 ↪astype('str')

# Fill the postal codes with leading zeros to ensure a uniform 5-digit format
df_clean['Postal Code'] = df_clean['Postal Code'].str.zfill(5)

# Convert date columns to datetime format
df_clean[['Order Date', 'Ship Date']] = df_clean[['Order Date', 'Ship Date']].
 ↪apply(pd.to_datetime)

# Convert selected columns to categorical data types
df_clean['Ship Mode'] = pd.Categorical(df_clean['Ship Mode'],␣
 ↪categories=df_clean['Ship Mode'].unique(), ordered=False)
df_clean['Segment'] = pd.Categorical(df_clean['Segment'],␣
 ↪categories=df_clean['Segment'].unique(), ordered=False)
df_clean['Country'] = pd.Categorical(df_clean['Country'], categories=['United␣
 ↪States', 'International'], ordered=False)
df_clean['State'] = pd.Categorical(df_clean['State'],␣
 ↪categories=df_clean['State'].unique(), ordered=False)
df_clean['Order ID'] = pd.Categorical(df_clean['Order ID'])
df_clean['Customer ID'] = pd.Categorical(df_clean['Customer ID'])
df_clean['Postal Code'] = pd.Categorical(df_clean['Postal Code'])
```

```python
# Ensure 'Region' is stored as a categorical variable
df_clean['Region'] = pd.Categorical(df_clean['Region'],␣
 ↪categories=df_clean['Region'].unique(), ordered=False)

# Convert 'Category' and 'Sub-Category' to categorical types
df_clean['Category'] = pd.Categorical(df_clean['Category'],␣
 ↪categories=df_clean['Category'].unique(), ordered=False)
df_clean['Sub-Category'] = pd.Categorical(df_clean['Sub-Category'],␣
 ↪categories=df_clean['Sub-Category'].unique(), ordered=False)

# Check for one-to-one relationships between IDs or other key columns

# Verify if each 'Sub-Category' belongs to only one 'Category'
subcat_cat_count = df_clean.groupby('Sub-Category', observed=True)['Category'].
 ↪nunique()  # Count unique categories for each sub-category
display(subcat_cat_count)  # If any count is greater than 1, the sub-category␣
 ↪appears in multiple categories
```

```
Sub-Category
Bookcases      1
Chairs         1
Labels         1
Tables         1
Storage        1
Furnishings    1
Art            1
Phones         1
Binders        1
Appliances     1
Paper          1
Accessories    1
Envelopes      1
Fasteners      1
Supplies       1
Machines       1
Copiers        1
Name: Category, dtype: int64
```

1.2.5 Verification of 'Sub-Category' Consistency

A check was performed to ensure that each **'Sub-Category'** belongs to only one **'Category'**.
The analysis confirmed that no 'Sub-Category' appears in more than one 'Category'.
This validation ensures the data maintains a strict hierarchical relationship between categories and sub-categories.

1.3 Functions for Identifying and Updating Duplicate IDs

1.3.1 `dup_flag(df, id_col, value_col)`

This function identifies whether an ID appears with multiple unique values in the specified column. It returns: - A boolean Series indicating which rows have duplicate IDs. - A DataFrame containing unique combinations of `id_col` and `value_col`.

This is useful for detecting inconsistencies in datasets where each ID should ideally map to a single value.

1.3.2 `update_id(df, id_col, value_col)`

This function appends a numerical suffix to duplicate IDs, ensuring uniqueness while maintaining traceability. It: - Calls `dup_flag()` to identify duplicates. - Assigns a numerical suffix to duplicate occurrences. - Updates the ID column by appending the suffix only for duplicates. - Merges the updated IDs back into the original DataFrame.

This function helps standardize datasets by ensuring IDs remain unique while preserving their original structure.

[9]:
```
def dup_flag(df, id_col, value_col):
    """
    Identifies duplicate IDs based on their associated values.

    Args:
        df (pd.DataFrame): The input DataFrame.
        id_col (str): Column containing IDs.
        value_col (str): Column to check for uniqueness within each ID.

    Returns:
        dup_flags (pd.Series): Boolean Series indicating which rows have␣
↪duplicate IDs.
        unique_combinations (pd.DataFrame): DataFrame with unique (id_col,␣
↪value_col) combinations.
    """
    unique_combinations = df[[id_col, value_col]].drop_duplicates().copy()
    dup_prod = df.groupby(id_col)[value_col].nunique()
    dup_ids = dup_prod[dup_prod > 1].index.to_list()
    dup_flags = df[id_col].isin(dup_ids)
    return dup_flags, unique_combinations

def update_id(df, id_col, value_col):
    """
    Updates duplicate IDs by appending a numerical suffix.

    Args:
        df (pd.DataFrame): The input DataFrame.
        id_col (str): Column containing IDs.
        value_col (str): Column to check for uniqueness within each ID.
```

```
    Returns:
        pd.DataFrame: DataFrame with updated IDs and duplication flags.
    """
    flags = dup_flag(df, id_col, value_col)
    df[id_col + ' dup'] = flags[0]  # Boolean flag for duplicate IDs
    suffixes = flags[1]  # DataFrame with unique ID-value combinations
    suffixes[id_col + ' suffix'] = suffixes.groupby(id_col).cumcount() + 1  #␣
↪Assign incremental suffix
    new_col = id_col + ' updated'
    suffixes[new_col] = suffixes[id_col].astype(str) + "_" + suffixes[id_col +␣
↪' suffix'].astype(str).str.zfill(2)

    # Drop redundant columns if they exist
    df = df.drop(columns=[col for col in [id_col + ' suffix', new_col] if col␣
↪in df.columns], axis=1)

    # Merge updated suffixes with original DataFrame
    df_new = df.merge(suffixes, on=[id_col, value_col], how='left')

    # Keep original ID where no duplicates exist
    df_new[new_col] = np.where(df_new[id_col + " dup"], df_new[new_col],␣
↪df_new[id_col])

    return df_new
```

1.3.3 Handling Duplicate IDs

The focus was placed on the **Product ID**, where duplicate IDs were detected across different products. An updated ID was generated only for those products with duplicates, ensuring that necessary updates were applied efficiently.

As for the **Customer ID**, after running `update_id(df_clean, 'Customer ID', 'Customer Name')`, the results showed that the **Customer ID** requires no modification.

[10]:
```
df_clean=update_id(df_clean,'Product ID', 'Product Name')
df_clean['Product ID updated'] = pd.Categorical(df_clean['Product ID updated'])
df_clean.drop(columns=['Product ID suffix'], inplace=True)
```

[11]:
```
df_clean.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   object
 1   Order ID            9994 non-null   category
```

```
 2   Order Date          9994 non-null   datetime64[ns]
 3   Ship Date           9994 non-null   datetime64[ns]
 4   Ship Mode           9994 non-null   category
 5   Customer ID         9994 non-null   category
 6   Customer Name       9994 non-null   object
 7   Segment             9994 non-null   category
 8   Country             9994 non-null   category
 9   City                9994 non-null   object
 10  State               9994 non-null   category
 11  Postal Code         9994 non-null   category
 12  Region              9994 non-null   category
 13  Product ID          9994 non-null   object
 14  Category            9994 non-null   category
 15  Sub-Category        9994 non-null   category
 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  Product ID dup      9994 non-null   bool
 22  Product ID updated  9994 non-null   category
dtypes: bool(1), category(11), datetime64[ns](2), float64(3), int64(1),
object(5)
memory usage: 1.3+ MB
```

[12]:

```
# Check the Date Range
min_order, max_order = df_clean['Order Date'].min(), df_clean['Order Date'].
 ↪max()
min_ship, max_ship = df_clean['Ship Date'].min(), df_clean['Ship Date'].max()

print(f"Order Date Range:   {min_order.strftime('%Y-%m-%d')}  to  {max_order.
 ↪strftime('%Y-%m-%d')}")
print(f" Ship Date Range:   { min_ship.strftime('%Y-%m-%d')}  to  {max_ship.
 ↪strftime('%Y-%m-%d')}")
```

```
Order Date Range:   2014-01-03  to  2017-12-30
 Ship Date Range:   2014-01-07  to  2018-01-05
```

[13]:

```
# Check for Orders Shipped Before They Were Ordered
df_invalid_dates = df_clean[df_clean['Ship Date'] < df_clean['Order Date']]

if df_invalid_dates.empty:
    print("All shipping dates are valid. No orders were shipped before the␣
 ↪order date.")
else:
    print("Orders with invalid shipping dates found:")
    print(df_invalid_dates)
```

All shipping dates are valid. No orders were shipped before the order date.

[14]:
```
#Check for Outliers (Extremely Long Shipping Times)
df_clean['Shipping Duration'] = df_clean['Ship Date'] - df_clean['Order Date']
print("\n  Shipping Duration Stats (in days):")
print(df_clean['Shipping Duration'].dt.days.describe())
```

```
  Shipping Duration Stats (in days):
count    9994.000000
mean        3.958175
std         1.747567
min         0.000000
25%         3.000000
50%         4.000000
75%         5.000000
max         7.000000
Name: Shipping Duration, dtype: float64
```

[15]:
```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 24 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Row ID             9994 non-null   object
 1   Order ID           9994 non-null   category
 2   Order Date         9994 non-null   datetime64[ns]
 3   Ship Date          9994 non-null   datetime64[ns]
 4   Ship Mode          9994 non-null   category
 5   Customer ID        9994 non-null   category
 6   Customer Name      9994 non-null   object
 7   Segment            9994 non-null   category
 8   Country            9994 non-null   category
 9   City               9994 non-null   object
 10  State              9994 non-null   category
 11  Postal Code        9994 non-null   category
 12  Region             9994 non-null   category
 13  Product ID         9994 non-null   object
 14  Category           9994 non-null   category
 15  Sub-Category       9994 non-null   category
 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  Product ID dup     9994 non-null   bool
 22  Product ID updated 9994 non-null   category
```

```
 23  Shipping Duration   9994 non-null   timedelta64[ns]
dtypes: bool(1), category(11), datetime64[ns](2), float64(3), int64(1),
object(5), timedelta64[ns](1)
memory usage: 1.4+ MB
```

[16]:
```
df_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 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   object
 3   Ship Date      9994 non-null   object
 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
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
```

[ ]:
```

```