INTERNSHIP PROJECT REPORT

Title: Data Analysis and Cleaning of SCMS Delivery History Dataset

Declaration by Author

This is to declare that this report has been written by me. No part of the report is plagiarized from other sources. All information included from other sources has been duly acknowledged. I aver that if any part of the report is found to be plagiarized, I shall take full responsibility for it.

Signature:
Name: [Ansari Abdullah Shamim]
Place: [Bhiwandi]
Date: [05-12-2024]

Abstract

The aim of this project was to clean and analyze the SCMS Delivery History Dataset to facilitate effective data-driven decision-making. The dataset, consisting of 10,324 records and 33 columns, contained missing values, duplicate records, and inconsistencies in data types. The project involved the application of Python libraries such as Pandas, Matplotlib, and Seaborn for data cleaning and exploration. The key steps included loading the dataset, identifying missing values, handling inconsistencies, removing duplicates, and enforcing uniform data types. A cleaned version of the dataset was generated as the final deliverable. The insights obtained through exploratory analysis included patterns in shipment modes, distribution of numerical features, and correlation among key variables.

TABLE OF CONTENTS

Introduction

1.1 Problem Addressed

The SCMS Delivery History Dataset is a comprehensive collection of logistics data used to analyze delivery performance and optimize supply chain processes. The dataset contained several challenges such as missing values, duplicate records, and inconsistencies in data formats, which could hinder accurate decision-making.

1.2 Importance of the Problem

Effective logistics management relies on clean and accurate datasets to optimize processes, reduce delays, and improve cost efficiency. Addressing data quality issues ensures reliable insights for stakeholders.

1.3 Scope of the Project

The project focused on cleaning and preprocessing the dataset to remove errors and inconsistencies. It also explored patterns in shipment modes, vendor performance, and product delivery trends.

Approach Used

2.1 Dataset Exploration

The dataset, consisting of 33 columns, was first loaded into a Pandas DataFrame. Key steps in exploration included:

- Displaying summary statistics of numerical and categorical columns.
- Identifying missing values in Shipment Mode, Dosage, and Line Item Insurance.
- Generating histograms and a correlation heatmap to explore data distributions and relationships.

2.2 Data Cleaning and Preprocessing

Key cleaning steps:

1. **Handling Missing Values:**
    - Columns with more than 50% missing data were dropped.
    - Missing numerical values were filled with the column median, and missing categorical values were filled with the mode.
2. **Removing Duplicates:** Duplicate records were removed, ensuring unique entries for analysis.
3. **Data Type Enforcement:** Ensured consistent data types for numerical and categorical columns.
4. **Index Reset:** Re-indexed the cleaned dataset for better readability and usage.

Results and Discussion

3.1 Insights from Data Exploration

- **Missing Values:**
  - Columns like Shipment Mode had 360 missing entries, which were filled or dropped based on thresholds.
  - Dosage and Line Item Insurance were treated with median and mode imputation.
- **Numerical Data Distributions:**
  - Columns like Unit of Measure, Line Item Value, and Pack Price exhibited right-skewed distributions.
  - Outliers were identified in Pack Price and Unit Price.
- **Correlations:**
  - Strong positive correlation observed between Line Item Value and Line Item Quantity.
- **Categorical Data Analysis:**
  - Project Code revealed 768 entries associated with the most frequent code, while others were sparsely represented.

3.2 Improvements Achieved

- Reduced data inconsistencies by filling missing values.
- Generated a cleaned dataset with improved usability for further analysis.
- Insights into logistics trends and performance metrics were made accessible.

Conclusions and Recommendations

Conclusions

The SCMS Delivery History Dataset was cleaned to eliminate inconsistencies, enabling accurate insights. Key findings included trends in shipment modes and correlations among logistics variables.

Recommendations

Future work could include:

1. Automating the cleaning process for real-time data updates.
2. Enhancing analysis with predictive modeling to optimize delivery schedules.

Appendices

Appendix A: Python Code for Data Exploration

Step 1 :- Import Libraries

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

- **import pandas as pd**: Imports the pandas library, which is used for data manipulation and analysis. It's abbreviated as pd for easier usage.
- **import matplotlib.pyplot as plt**: Imports the matplotlib library for creating static, interactive, and animated visualizations. plt is a common shorthand.
- **import seaborn as sns**: Imports the seaborn library, built on top of matplotlib, which makes it easier to create aesthetically pleasing statistical plots.

Step 2 :- Load the Dataset

```
file_path = "F:\\Downloads\\archive (3)\\SCMS_Delivery_History_Dataset.csv"

try:

    data = pd.read_csv(file_path)

    print("Dataset loaded successfully!")

except Exception as e:

    print(f"Error loading dataset: {e}")
```

```
... Dataset loaded successfully!
```

- **file_path**: Specifies the path to the dataset file stored on your computer.

- **pd.read_csv(file_path)**: Reads the CSV file into a pandas DataFrame called data. This function automatically parses the file's structure into a tabular format.
- **try...except block**: Handles errors during file loading:
  - If the file loads successfully, it prints: *"Dataset loaded successfully!"*
  - If there's an error (e.g., file not found), it prints the error message.

Step 3 :- Dataset Overview

```
# Display the first few rows
data.head()
# Display dataset information
print("\n--- Dataset Information ---")
data.info()
# Display basic statistics
print("\n--- Basic Statistics ---")
data.describe()
data.tail()
data
data.columns
data.dtypes
```

```
...
--- Dataset Information ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10324 entries, 0 to 10323
Data columns (total 33 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          10324 non-null  int64
 1   Project Code                10324 non-null  object
 2   PQ #                        10324 non-null  object
 3   PO / SO #                   10324 non-null  object
 4   ASN/DN #                    10324 non-null  object
 5   Country                     10324 non-null  object
 6   Managed By                  10324 non-null  object
 7   Fulfill Via                 10324 non-null  object
 8   Vendor INCO Term            10324 non-null  object
 9   Shipment Mode               9964 non-null   object
 10  PQ First Sent to Client Date 10324 non-null object
 11  PO Sent to Vendor Date      10324 non-null  object
 12  Scheduled Delivery Date     10324 non-null  object
 13  Delivered to Client Date    10324 non-null  object
 14  Delivery Recorded Date      10324 non-null  object
 15  Product Group               10324 non-null  object
 16  Sub Classification          10324 non-null  object
 17  Vendor                      10324 non-null  object
...
dtypes: float64(4), int64(3), object(26)
memory usage: 2.6+ MB
```

```
--- Basic Statistics ---
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

	ID	Unit of Measure (Per Pack)	Line Item Quantity	Line Item Value	Pack Price	Unit Price	Line Item Insurance (USD)
count	10324.000000	10324.000000	10324.000000	1.032400e+04	10324.000000	10324.000000	10037.000000
mean	51098.968229	77.990895	18332.534870	1.576506e+05	21.910241	0.611701	240.117626
std	31944.332496	76.579764	40035.302961	3.452921e+05	45.609223	3.275808	500.190568
min	1.000000	1.000000	1.000000	0.000000e+00	0.000000	0.000000	0.000000
25%	12795.750000	30.000000	408.000000	4.314593e+03	4.120000	0.080000	6.510000
50%	57540.500000	60.000000	3000.000000	3.047147e+04	9.300000	0.160000	47.040000
75%	83648.250000	90.000000	17039.750000	1.664471e+05	23.592500	0.470000	252.400000
max	86823.000000	1000.000000	619999.000000	5.951990e+06	1345.640000	238.650000	7708.440000

	ID	Project Code	PQ #	PO / SO #	ASN/DN #	Country	Managed By	Fulfill Via	Vendor INCO Term	Shipment Mode	...	Unit of Measure (Per Pack)	Line Item Quantity	Line Item Value	Pack Price	Unit Price	Manufacturing Site	First Line Designation	Weight (Kilograms)	Frei Cost (U
10319	86818	103-ZW-T30	FPQ-15197	SO-50020	DN-4307	Zimbabwe	PMO - US	From RDC	N/A - From RDC	Truck	...	60	166571	599655.60	3.60	0.06	Mylan, H-12 & H-13, India	No	See DN-4307 (ID#:83920)	See I 4 (ID#:83!
10320	86819	104-CI-T30	FPQ-15259	SO-50102	DN-4313	Côte d'Ivoire	PMO - US	From RDC	N/A - From RDC	Truck	...	60	21072	137389.44	6.52	0.11	Hetero Unit III Hyderabad IN	No	See DN-4313 (ID#:83921)	See I 4 (ID#:83!
10321	86821	110-ZM-T30	FPQ-14784	SO-49600	DN-4316	Zambia	PMO - US	From RDC	N/A - From RDC	Truck	...	30	514526	5140114.74	9.99	0.33	Cipla Ltd A-42 MIDC Mahar. IN	No	Weight Captured Separately	Fre Include Commo (
10322	86822	200-ZW-T30	FPQ-16523	SO-51680	DN-4334	Zimbabwe	PMO - US	From RDC	N/A - From RDC	Truck	...	60	17465	113871.80	6.52	0.11	Mylan (formerly Matrix) Nashik	Yes	1392	Fre Include Commo (
10323	86823	103-ZW-T30	FPQ-15197	SO-50022	DN-4336	Zimbabwe	PMO - US	From RDC	N/A - From RDC	Truck	...	60	36639	72911.61	1.99	0.03	Cipla, Goa, India	No	Weight Captured Separately	Fre Include Commo (

5 rows × 33 columns

	ID	Project Code	PQ #	PO / SO #	ASN/DN #	Country	Managed By	Fulfill Via	Vendor INCO Term	Shipment Mode	...	Unit of Measure (Per Pack)	Line Item Quantity	Line Item Value	Pack Price	Unit Price	Manufacturing Site	First Line Designation	Weight (Kilograms)	Freight Cost (USD)	Line Item Insurance (USD)
0	1	100-CI-T01	Pre-PQ Process	SCMS-4	ASN-8	Côte d'Ivoire	PMO - US	Direct Drop	EXW	Air	...	30	19	551.00	29.00	0.97	Ranbaxy Fine Chemicals LTD	Yes	13	780.34	NaN
1	3	108-VN-T01	Pre-PQ Process	SCMS-13	ASN-85	Vietnam	PMO - US	Direct Drop	EXW	Air	...	240	1000	6200.00	6.20	0.03	Aurobindo Unit III, India	Yes	358	4521.5	NaN
2	4	100-CI-T01	Pre-PQ Process	SCMS-20	ASN-14	Côte d'Ivoire	PMO - US	Direct Drop	FCA	Air	...	100	500	40000.00	80.00	0.80	ABBVIE GmbH & Co.KG Wiesbaden	Yes	171	1653.78	NaN
3	15	108-VN-T01	Pre-PQ Process	SCMS-78	ASN-50	Vietnam	PMO - US	Direct Drop	EXW	Air	...	60	31920	127360.80	3.99	0.07	Ranbaxy, Paonta Shahib, India	Yes	1855	16007.06	NaN
4	16	108-VN-T01	Pre-PQ Process	SCMS-81	ASN-55	Vietnam	PMO - US	Direct Drop	EXW	Air	...	60	38000	121600.00	3.20	0.05	Aurobindo Unit III, India	Yes	7590	45450.08	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10319	86818	103-ZW-T30	FPQ-15197	SO-50020	DN-4307	Zimbabwe	PMO - US	From RDC	N/A - From RDC	Truck	...	60	166571	599655.60	3.60	0.06	Mylan, H-12 & H-13, India	No	See DN-4307 (ID#:83920)	See DN-4307 (ID#:83920)	705.79
10320	86819	104-CI-T30	FPQ-15259	SO-50102	DN-4313	Côte d'Ivoire	PMO - US	From RDC	N/A - From RDC	Truck	...	60	21072	137389.44	6.52	0.11	Hetero Unit III Hyderabad IN	No	See DN-4313 (ID#:83921)	See DN-4313 (ID#:83921)	161.71
10321	86821	110-ZM-T30	FPQ-14784	SO-49600	DN-4316	Zambia	PMO - US	From RDC	N/A - From RDC	Truck	...	30	514526	5140114.74	9.99	0.33	Cipla Ltd A-42 MIDC Mahar. IN	No	Weight Captured Separately	Freight Included in Commodity Cost	5284.04
10322	86822	200-ZW-T30	FPQ-16523	SO-51680	DN-4334	Zimbabwe	PMO - US	From RDC	N/A - From RDC	Truck	...	60	17465	113871.80	6.52	0.11	Mylan (formerly Matrix) Nashik	Yes	1392	Freight Included in Commodity Cost	134.03
10323	86823	103-ZW-T30	FPQ-15197	SO-50022	DN-4336	Zimbabwe	PMO - US	From RDC	N/A - From RDC	Truck	...	60	36639	72911.61	1.99	0.03	Cipla, Goa, India	No	Weight Captured Separately	Freight Included in Commodity Cost	85.82

10324 rows × 33 columns

```
Index(['ID', 'Project Code', 'PQ #', 'PO / SO #', 'ASN/DN #', 'Country',
       'Managed By', 'Fulfill Via', 'Vendor INCO Term', 'Shipment Mode',
       'PQ First Sent to Client Date', 'PO Sent to Vendor Date',
       'Scheduled Delivery Date', 'Delivered to Client Date',
       'Delivery Recorded Date', 'Product Group', 'Sub Classification',
       'Vendor', 'Item Description', 'Molecule/Test Type', 'Brand', 'Dosage',
       'Dosage Form', 'Unit of Measure (Per Pack)', 'Line Item Quantity',
       'Line Item Value', 'Pack Price', 'Unit Price', 'Manufacturing Site',
       'First Line Designation', 'Weight (Kilograms)', 'Freight Cost (USD)',
       'Line Item Insurance (USD)'],
      dtype='object')
```

```
... ID                              int64
    Project Code                   object
    PQ #                           object
    PO / SO #                      object
    ASN/DN #                       object
    Country                        object
    Managed By                     object
    Fulfill Via                    object
    Vendor INCO Term               object
    Shipment Mode                  object
    PQ First Sent to Client Date   object
    PO Sent to Vendor Date         object
    Scheduled Delivery Date        object
    Delivered to Client Date       object
    Delivery Recorded Date         object
    Product Group                  object
    Sub Classification             object
    Vendor                         object
    Item Description               object
    Molecule/Test Type             object
    Brand                          object
    Dosage                         object
    Dosage Form                    object
    Unit of Measure (Per Pack)      int64
    Line Item Quantity              int64
    ...
    First Line Designation         object
    Weight (Kilograms)             object
    Freight Cost (USD)             object
    Line Item Insurance (USD)     float64
    dtype: object
    Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

data.head()

- Purpose: Displays the first 5 rows of the dataset.
- Use: Helps in getting a quick look at the data structure, column names, and initial values.

data.info()

- Purpose: Provides a summary of the dataset, including:
  - Number of columns and rows.
  - Column names.
  - Non-null counts (useful for identifying missing values).
  - Data types (e.g., int64, float64, object).
- Use: Helps assess the dataset structure and detect potential data issues like missing values or incorrect types.

data.describe()

- Purpose: Displays basic statistics for numerical columns:
  - Count, mean, standard deviation, min, max, and quartiles (25%, 50%, 75%).
- Use: Useful for understanding the data's range and distribution.

data.tail()

- Purpose: Displays the last 5 rows of the dataset.

- Use: Similar to head(), but shows the end of the dataset, which might contain edge cases or incomplete records.

data

- Purpose: Displays the entire dataset as a table.
- Use: Useful for small datasets to visually inspect all the data, but avoid using it for large datasets (as it can slow down performance).

data.columns

- Purpose: Lists all the column names in the dataset.
- Use: Quickly see the names of the columns, especially when there are too many to view in head() or info().

data.dtypes

- Purpose: Displays the data type of each column (e.g., int64, float64, object).
- Use: Helps verify that each column's data type aligns with its intended purpose (e.g., numeric for calculations, strings for text).

Step 4 :- Check for Missing Values

```
missing_values = data.isnull().sum()
# Display columns with missing values
print("\n--- Missing Values ---")
missing_values[missing_values > 0]
```

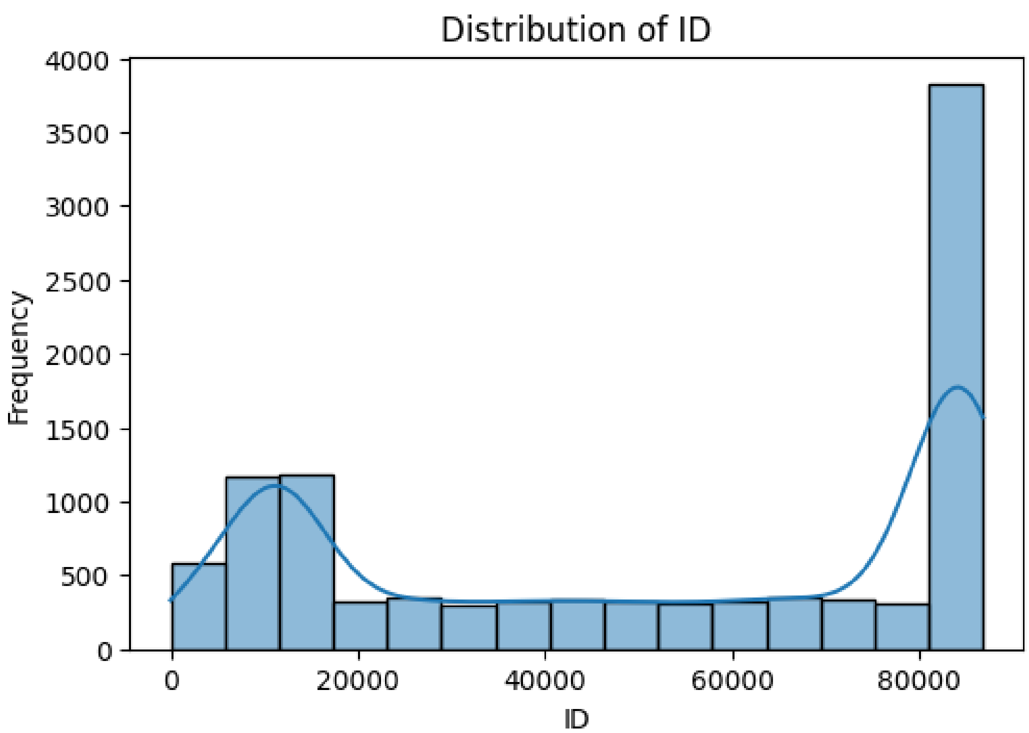
```
...
    --- Missing Values ---

... Shipment Mode                  360
    Dosage                        1736
    Line Item Insurance (USD)      287
    dtype: int64
```

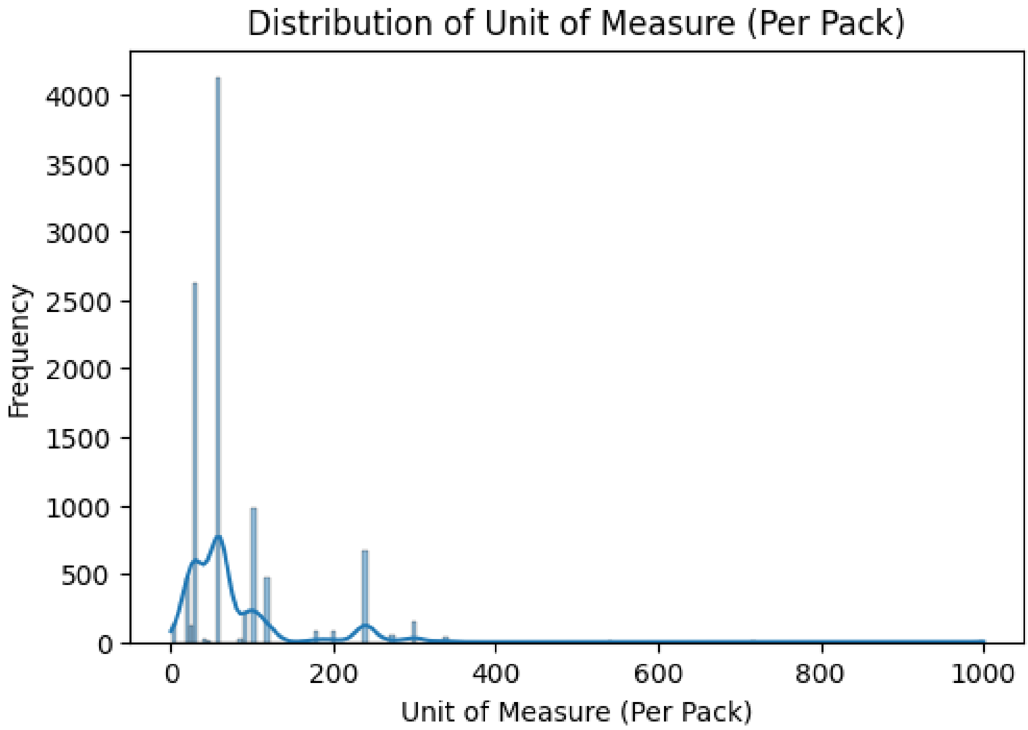
- **data.isnull().sum()**: Calculates the total number of missing (null) values in each column.

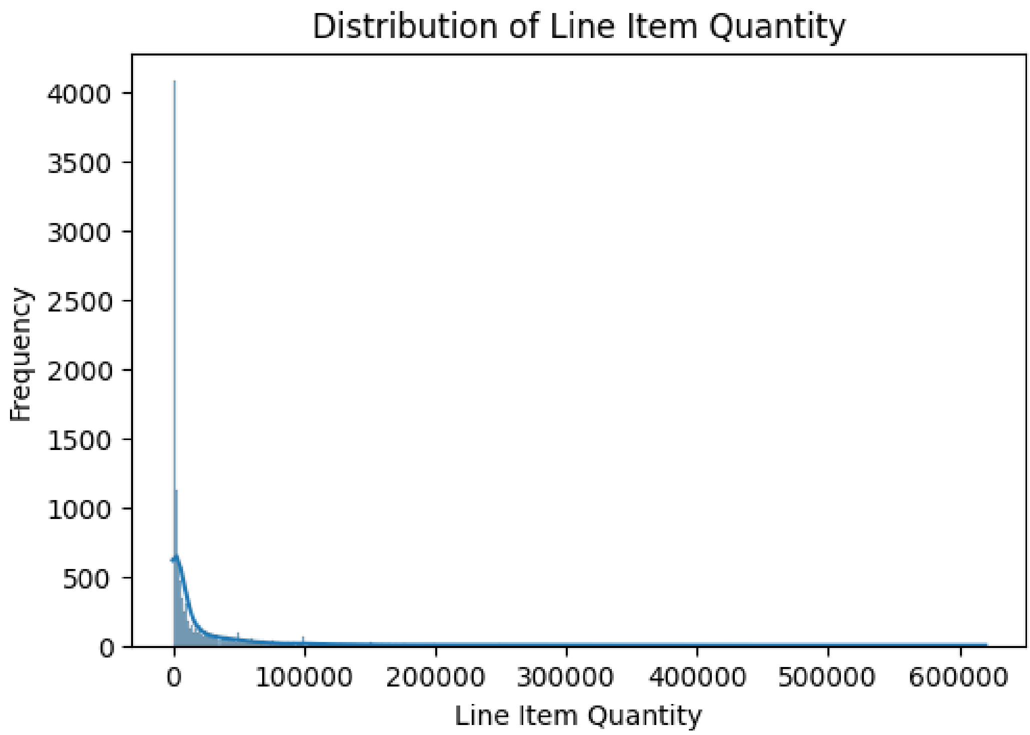
- **Purpose**: Identify which columns have missing values and the extent of the missing data.
- **missing_values[missing_values > 0]**: Filters the output to display only the columns with missing values (ignoring columns with no missing values).
- **Use**: Helps prioritize missing value handling for data cleaning.

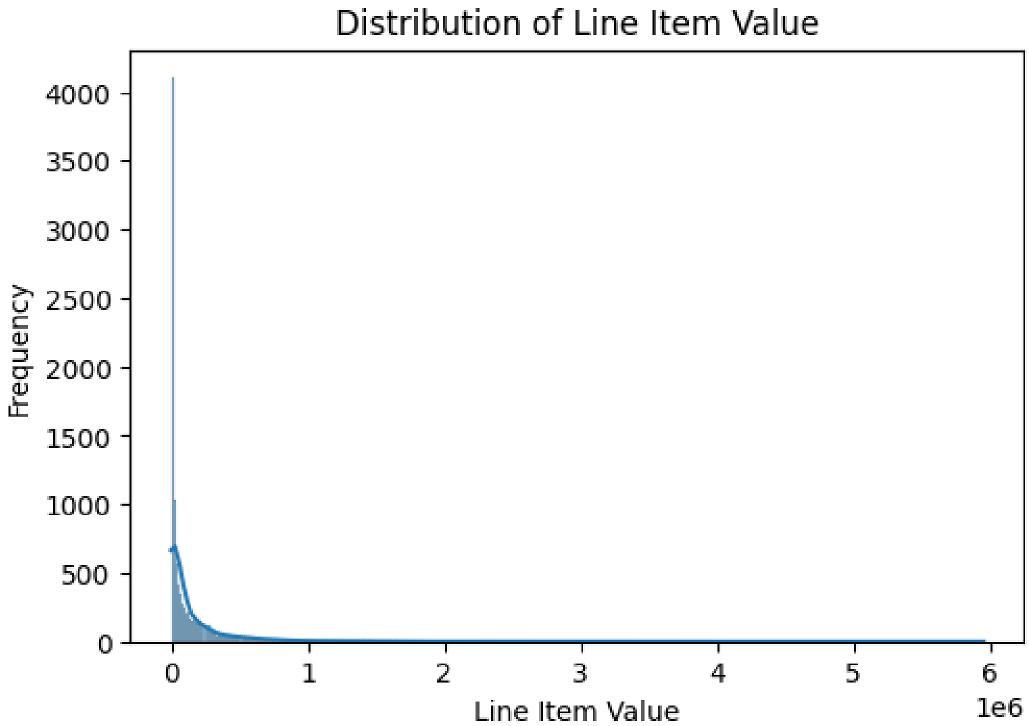
Step 5 :- Plotting Numerical Columns

```
num_cols = data.select_dtypes(include=['float64', 'int64']).columns
for col in num_cols:
    plt.figure(figsize=(6, 4))
    sns.histplot(data[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

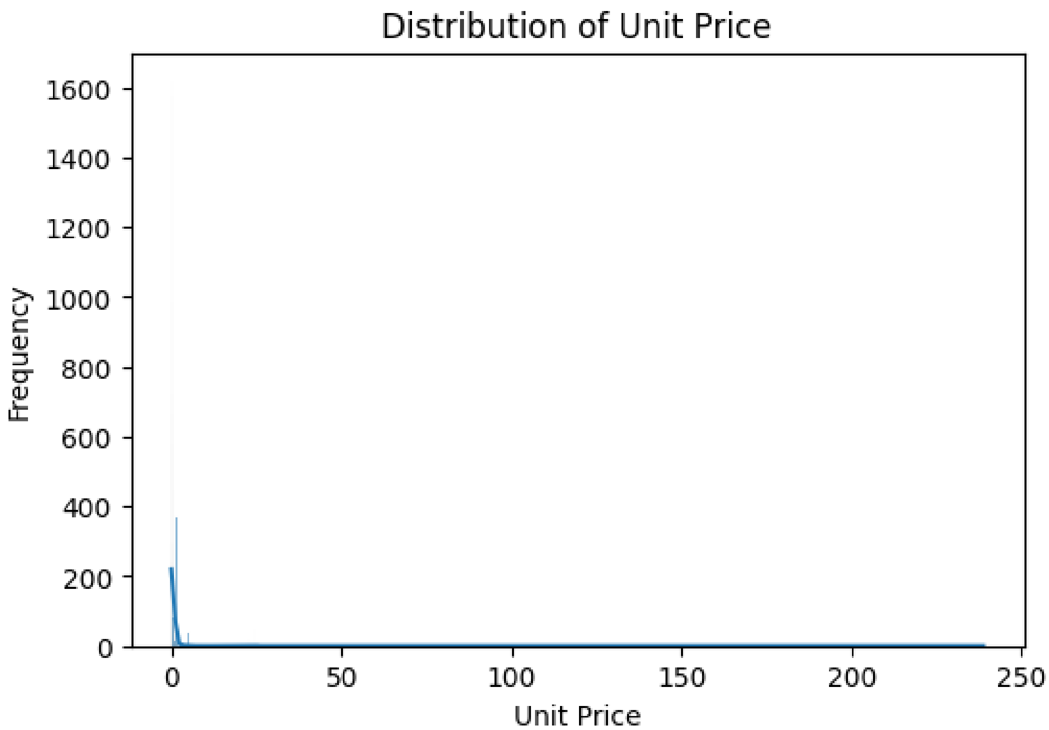



Distribution of Unit of Measure (Per Pack)
Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
0
200
400
600
800
1000
Unit of Measure (Per Pack)

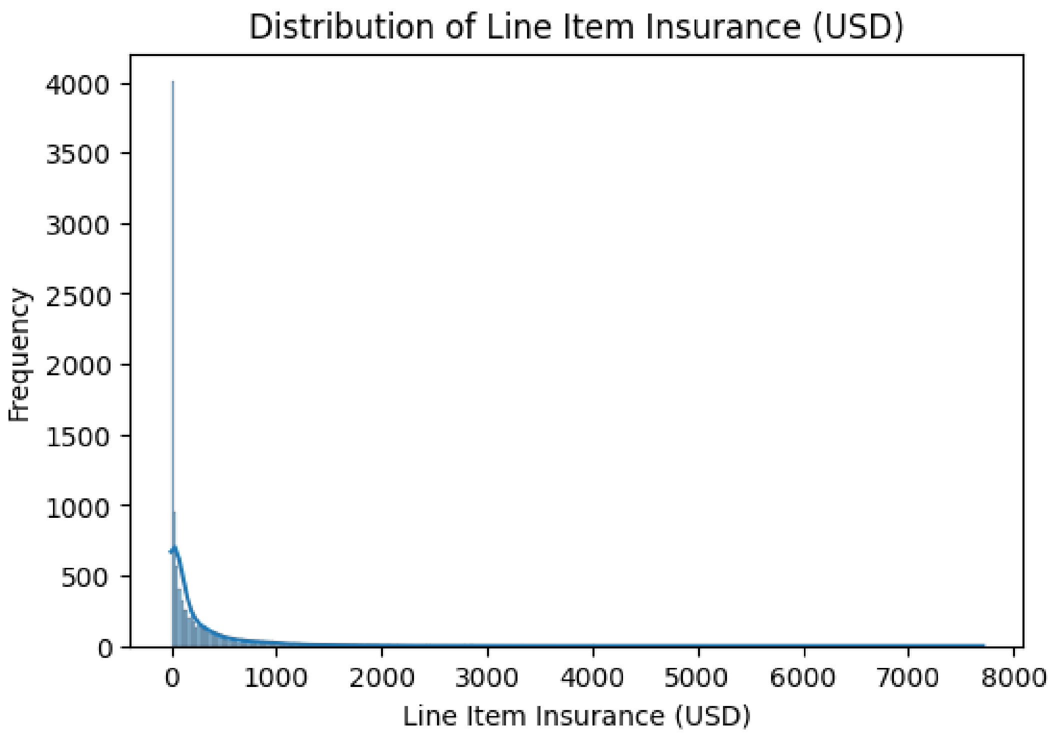

Distribution of Line Item Quantity
Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
0
100000
200000
300000
400000
500000
600000
Line Item Quantity



Distribution of Line Item Value
Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
6
1e6
Line Item Value



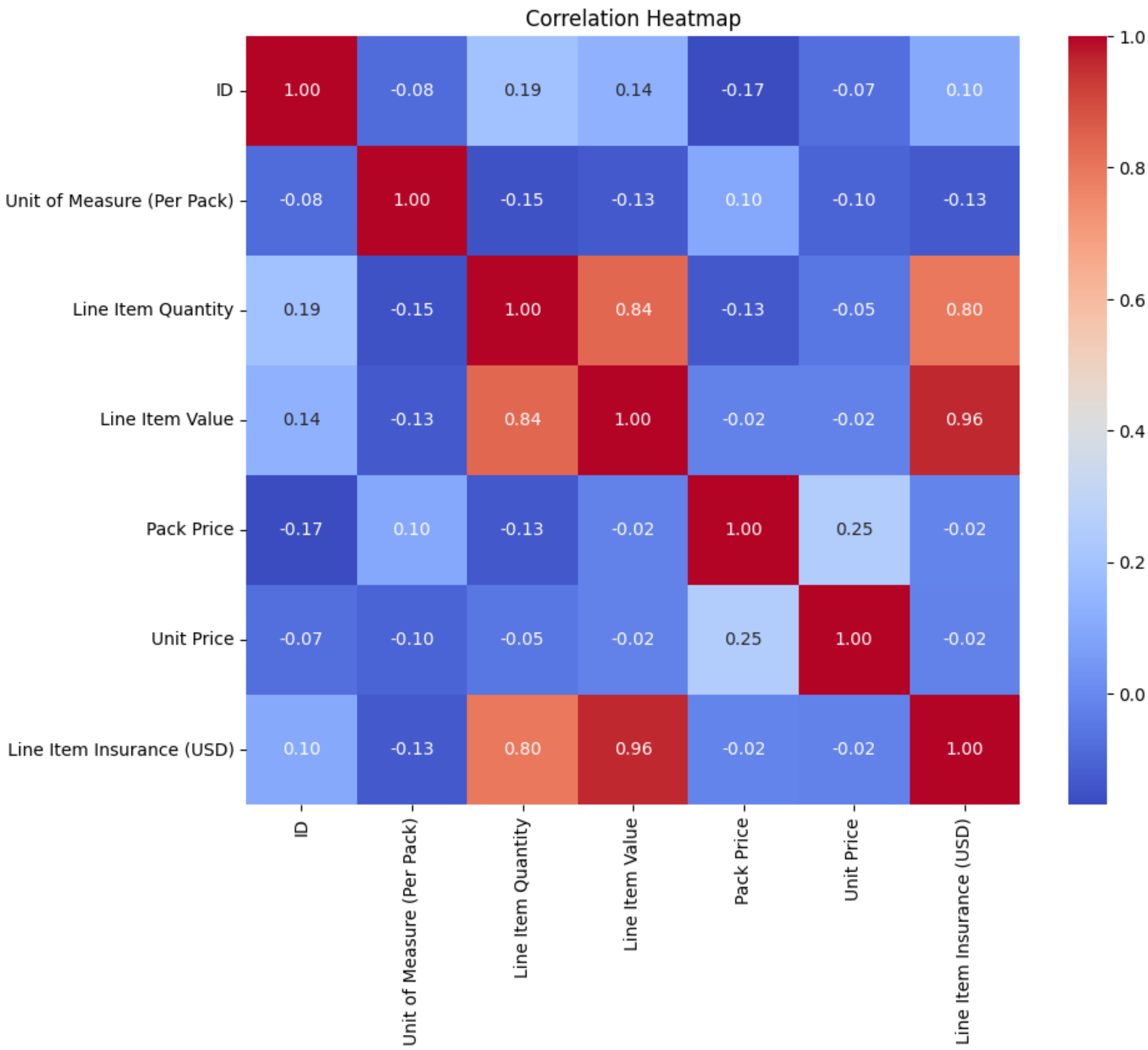
Distribution of Unit Price
Frequency
1600
1400
1200
1000
800
600
400
200
0
0
50
100
150
200
250
Unit Price



- **data.select_dtypes(include=['float64', 'int64'])**: Selects only numerical columns (floats and integers) for analysis.
- **Loop through num_cols**: Iterates over all numerical columns to create individual plots.
- **sns.histplot()**: Creates a histogram with a kernel density estimate (KDE) to show the data distribution.
- **Purpose**: Understand the distribution and spread of each numerical column (e.g., uniform, normal, skewed).

Step 6 :- Heatmap for Correlation

```
numeric_data = data.select_dtypes(include=['float64', 'int64'])
if not numeric_data.empty:
    plt.figure(figsize=(10, 8))
    sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm', fmt=".2f")
    plt.title("Correlation Heatmap")
    plt.show()
else:
    print("No numeric columns available for correlation heatmap.")
```



- **numeric_data.corr()**: Calculates pairwise correlation coefficients between numerical columns.
- **sns.heatmap()**: Creates a heatmap to visualize these correlations.
    - **annot=True**: Displays the correlation values on the heatmap.
    - **cmap='coolwarm'**: Defines the color scheme for the heatmap.
- **Purpose**: Identify relationships between numerical features (e.g., positive or negative correlations) for insights or feature selection.

Step 7 :- Analyze Categorical Columns

```
cat_cols = data.select_dtypes(include='object').columns
for col in cat_cols:
    print(f"\n--- {col} ---")
    print(data[col].value_counts())
```

```
...
--- Project Code ---
Project Code
116-ZA-T30    768
104-CI-T30    729
151-NG-T30    628
114-UG-T30    596
108-VN-T30    522
             ...
100-SN-T01      1
201-UG-T30      1
100-GN-T30      1
A02-SN-T50      1
104-SZ-T30      1
Name: count, Length: 142, dtype: int64

--- PQ # ---
PQ #
Pre-PQ Process    2681
FPQ-14942          205
FPQ-12522          154
FPQ-13973          110
FPQ-4537            98
                  ...
FPQ-12933            1
...
7616.19                             1
12793.7                             1
See DN-4282 (ID#:83919)             1
Name: count, Length: 6733, dtype: int64
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- **data.select_dtypes(include='object'):** Selects only categorical columns for analysis.
- **data[col].value_counts()**: Counts the occurrences of each unique value in the column.
- **Purpose:** Understand the distribution of categories (e.g., frequency of different cities or product types).

Appendix B : Python Code for Data Cleaning

Step 1 :- Handling Missing Values

```
def handle_missing_values(data):
    print("\n--- Handling Missing Values ---")
    missing_values = data.isnull().sum()
    print("Columns with missing values:")
    print(missing_values[missing_values > 0])
    # Drop columns with excessive missing values
    threshold = 0.5 * len(data)
    data = data.dropna(thresh=threshold, axis=1)
    print("\nColumns with excessive missing values dropped.")
    # Fill missing numerical columns with the median
    num_cols = data.select_dtypes(include=['float64', 'int64']).columns
    for col in num_cols:
        if data[col].isnull().sum() > 0:
            data[col] = data[col].fillna(data[col].median())
            print(f"Filled missing values in {col} with median.")
 # Fill missing categorical columns with the mode
    cat_cols = data.select_dtypes(include=['object']).columns
    for col in cat_cols:
        if data[col].isnull().sum() > 0:
            data[col] = data[col].fillna(data[col].mode()[0])
            print(f"Filled missing values in {col} with mode.")
    return data
data = handle_missing_values(data)
```

```
...
--- Handling Missing Values ---
Columns with missing values:
Series([], dtype: int64)

Columns with excessive missing values dropped.
```

- **Purpose**: Handles missing values systematically by:
    - Dropping columns with excessive missing values.
    - Imputing missing numerical values with the median.
    - Imputing missing categorical values with the mode.

Step 2 :- Removing Duplicates

```
def remove_duplicates(data):
    print("\n--- Removing Duplicates ---")
    before = len(data)
    data = data.drop_duplicates()
    after = len(data)
    print(f"Removed {before - after} duplicate rows.")
    return data
data = remove_duplicates(data)
```

```
...
    --- Removing Duplicates ---
    Removed 0 duplicate rows.
```

- **data.drop_duplicates()**: Removes duplicate rows from the dataset.
- **Purpose**: Ensures data integrity and avoids redundancy.

Step 3:- Resetting Index

```
def enforce_data_types(data):
    import pandas as pd
    print("\n--- Ensuring Consistent Data Types ---")
    for col in data.columns:
        if data[col].dtype == 'object':
            try:
                # Explicitly handle conversion
```

```
            data[col] = pd.to_numeric(data[col], errors='coerce')  # Convert to numeric where possible
            print(f"Converted column '{col}' to numeric.")
        except Exception as e:
            print(f"Could not convert column '{col}': {e}")
    return data
def reset_index(data):
    print("\n--- Resetting Index ---")
    data = data.reset_index(drop=True)
    return data
data = reset_index(data)
```

```
...
--- Resetting Index ---
```

- **data.reset_index(drop=True)**: Resets the index of the DataFrame after cleaning operations (e.g., dropping rows).
- **Purpose**: Maintains a clean and sequential index.

Step 4 :- Save Cleaned Data

```
cleaned_file_path = "F:\\AI\\content\\SCMS_Cleaned_Dataset.csv"
data.to_csv(cleaned_file_path, index=False)
print(f"\nCleaned data saved to '{cleaned_file_path}'.")
```

```
...
Cleaned data saved to 'F:\AI\content\SCMS_Cleaned_Dataset.csv'.
```

- **data.to_csv()**: Saves the cleaned dataset to a CSV file.
- **Purpose**: Stores the cleaned data for future use.

Step 5 :- Display the first few rows

```
data.head()
```

	ID	Project Code	PQ #	PO / SO #	ASN/DN #	Country	Managed By	Fulfill Via	Vendor INCO Term	Shipment Mode	...	Unit of Measure (Per Pack)	Line Item Quantity	Line Item Value	Pack Price	Unit Price	Manufacturing Site	First Line Designation	Weight (Kilograms)	Freight Cost (USD)	Line It Insura (U:
0	1	100-CI-T01	Pre-PQ Process	SCMS-4	ASN-8	Côte d'Ivoire	PMO - US	Direct Drop	EXW	Air	...	30	19	551.0	29.00	0.97	Ranbaxy Fine Chemicals LTD	Yes	13	780.34	47
1	3	108-VN-T01	Pre-PQ Process	SCMS-13	ASN-85	Vietnam	PMO - US	Direct Drop	EXW	Air	...	240	1000	6200.0	6.20	0.03	Aurobindo Unit III, India	Yes	358	4521.5	47
2	4	100-CI-T01	Pre-PQ Process	SCMS-20	ASN-14	Côte d'Ivoire	PMO - US	Direct Drop	FCA	Air	...	100	500	40000.0	80.00	0.80	ABBVIE GmbH & Co.KG Wiesbaden	Yes	171	1653.78	47
3	15	108-VN-T01	Pre-PQ Process	SCMS-78	ASN-50	Vietnam	PMO - US	Direct Drop	EXW	Air	...	60	31920	127360.8	3.99	0.07	Ranbaxy, Paonta Shahib, India	Yes	1855	16007.06	47
4	16	108-VN-T01	Pre-PQ Process	SCMS-81	ASN-55	Vietnam	PMO - US	Direct Drop	EXW	Air	...	60	38000	121600.0	3.20	0.05	Aurobindo Unit III, India	Yes	7590	45450.08	47

5 rows × 33 columns

- Displays the first 5 rows of the cleaned dataset to verify the results after cleaning.

References

1. McKinney, W. *Python for Data Analysis*. O'Reilly Media, 2017.
2. Seaborn Documentation. Retrieved from https://seaborn.pydata.org/.
3. Pandas Documentation. Retrieved from https://pandas.pydata.org/.