# Medical inventory optimization

## Exploratory Data Analysis (python) by Vaka Prasanna

### Displaying the data

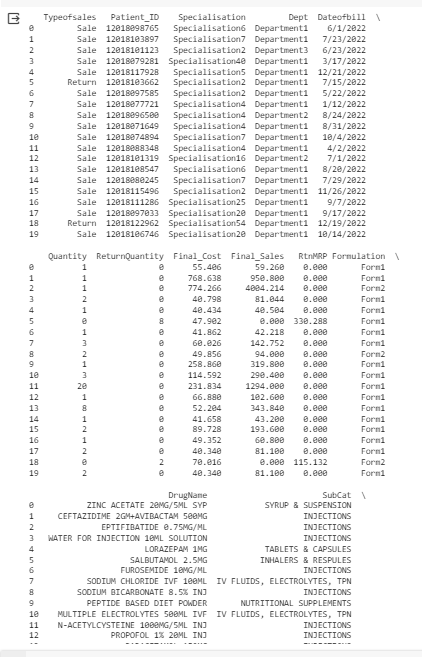
import pandas as pd

# Assuming your CSV file is named 'data.csv' and in the same directory data = pd.read\_csv('MIODataset.csv')

first=data.head(20)

# Print the first few rows of the data print(first)

### Output:



**Calculating the first moment (measures of central tendency such as mean, median, mode) for the dataset.**

import pandas as pd import statistics

# Sample data (replace with your actual data) data =pd.read\_csv('MIODataset.csv')

# Create pandas DataFrame df = pd.DataFrame(data)

# Specify the column names column\_list = ['Quantity',

'ReturnQuantity','Final\_Cost','Final\_Sales','RtnMRP']

# Calculate mean, median, mode for each column for col in column\_list:

# Check if data type is numeric for using statistics methods if pd.api.types.is\_numeric\_dtype(df[col]):

mean\_value = statistics.mean(df[col]) median\_value = statistics.median(df[col]) mode\_value = statistics.mode(df[col])

else:

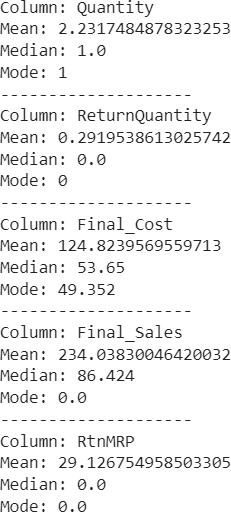
# Use mode function from pandas for categorical data

mode\_value = df[col].mode()[0] # Assuming there's only one mode # You can explore other options for handling categorical data as

needed

mean\_value = None median\_value = None

print("Column:", col) print("Mean:", mean\_value) print("Median:", median\_value) print("Mode:", mode\_value) print("-"\*20)



### Calculating the second moment (measures of dispersion such as variance, standard deviation, range) for the dataset.

import pandas as pd

# Sample data (replace with your actual data) data = pd.read\_csv('MIODataset.csv')

# Create pandas DataFrame df = pd.DataFrame(data)

# Specify the column names column\_list = ['Quantity',

'ReturnQuantity','Final\_Cost','Final\_Sales','RtnMRP']

# Calculate variance, standard deviation, range for each column for col in column\_list:

# Check if data type is numeric for calculations if pd.api.types.is\_numeric\_dtype(df[col]):

variance = df[col].var() std\_dev = df[col].std()

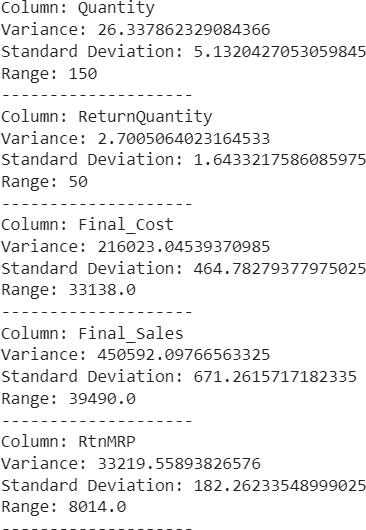
data\_range = df[col].max() - df[col].min() else:

variance = None std\_dev = None data\_range = None

print("Column:", col) print("Variance:", variance) print("Standard Deviation:", std\_dev) print("Range:", data\_range)

print("-"\*20)

**output :**



### 1. Calculating the third moment (skewness) for the dataset. Skewness:

import pandas as pd

# Sample data (replace with your actual data) data =pd.read\_csv('MIODataset.csv')

# Create pandas DataFrame df = pd.DataFrame(data)

# Specify the column names column\_list = ['Quantity',

'ReturnQuantity','Final\_Cost','Final\_Sales','RtnMRP']

# Calculate skewness for each column for col in column\_list:

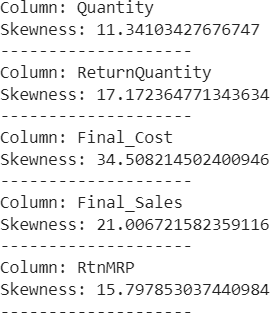
# Check if data type is numeric for skewness calculation if pd.api.types.is\_numeric\_dtype(df[col]):

skewness = df[col].skew() else:

skewness = None # Skewness not applicable for categorical data print("Column:", col)

print("Skewness:", skewness) print("-"\*20)

**Output:**



### Calculating the fourth moment (kurtosis) for the dataset. Kurtosis:

import pandas as pd

# Sample data (replace with your actual data) data =pd.read\_csv('MIODataset.csv')

# Create pandas DataFrame df = pd.DataFrame(data)

# Specify the column names column\_list = ['Quantity',

'ReturnQuantity','Final\_Cost','Final\_Sales','RtnMRP']

# Calculate kurtosis for each column for col in column\_list:

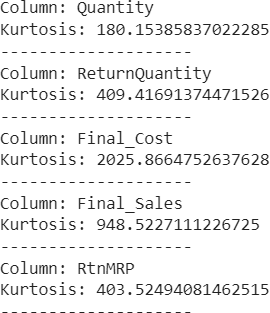
# Check if data type is numeric for kurtosis calculation if pd.api.types.is\_numeric\_dtype(df[col]):

kurtosis = df[col].kurt() else:

kurtosis = None # Kurtosis not applicable for categorical data print("Column:", col)

print("Kurtosis:", kurtosis) print("-"\*20)

**Output:**



# Medical inventory optimization

## Pre-processing Code (Python) by Vaka Prasanna

### Displaying the data.

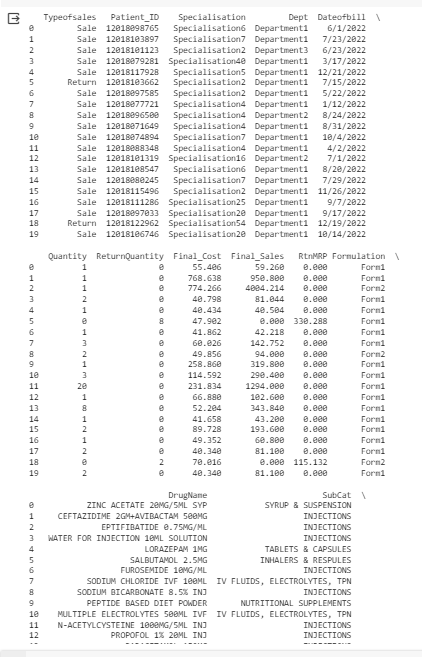
import pandas as pd

# Assuming your CSV file is named 'data.csv' and in the same directory data = pd.read\_csv('MIODataset.csv')

first=data.head(20)

# Print the first few rows of the data print(first)

### Output:

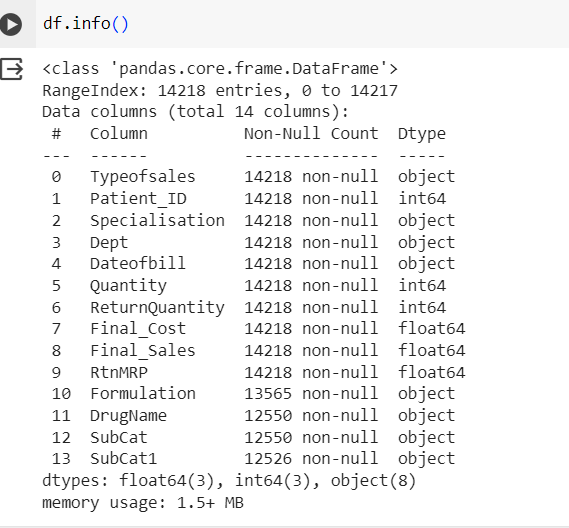


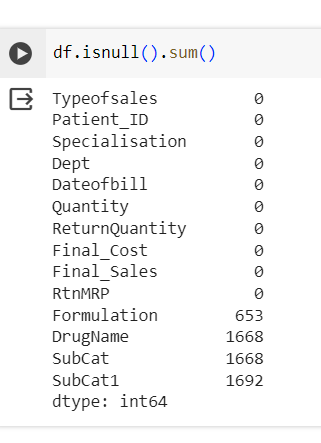
**Observation:** The 'Dateofbill' column in the dataset exhibits inconsistent date formats, with a combination of forward slashes ("/") and dashes ("-"). To ensure uniformity, we can store dates in a consistent format by replacing the forward slashes with dashes.

### Checking the dataset to ensure that all columns have the correct format.

df.info()

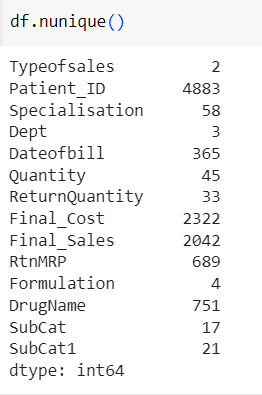
### Output:





df.isnull().sum()

**output:**

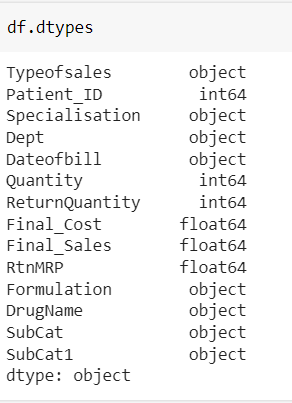


df.nunique()

**output:**

df.dtypes

## output:



### Fixing inconsistent date formats in the Dateofbill column

import pandas as pd import scipy import numpy as np

from sklearn.preprocessing import MinMaxScaler import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Assuming you have your data loaded into a pandas DataFrame named 'data'

# Define a function to handle date formatting (modify format string if needed)

def format\_date(date\_str):

return pd.to\_datetime(date\_str.str.replace('/', '-'), format='%m-%d-

%Y')

# Create a new DataFrame with selected columns and formatted date clean\_data = pd.DataFrame({

'Typeofsales': data['Typeofsales'], 'Patient\_ID': data['Patient\_ID'], 'Specialisation': data['Specialisation'], 'Dept': data['Dept'],

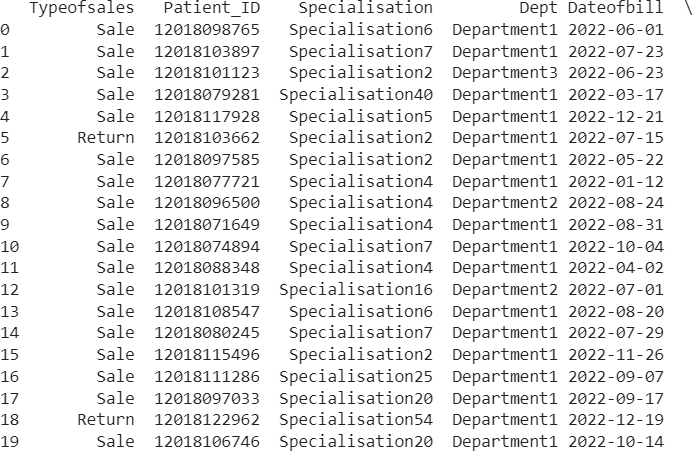
'Dateofbill': format\_date(data['Dateofbill']), 'Quantity': data['Quantity'], 'ReturnQuantity': data['ReturnQuantity'], 'Final\_Cost': data['Final\_Cost'], 'Final\_Sales': data['Final\_Sales'],

'RtnMRP': data['RtnMRP'], 'Formulation': data['Formulation'], 'DrugName': data['DrugName'], 'SubCat': data['SubCat'], 'SubCat1': data['SubCat1']

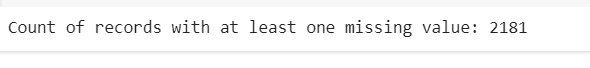
})

# You can now work with the 'clean\_data' DataFrame print(clean\_data.head(20)) # Display the first few rows

### Datatype and format of 'Dateofbill' column after Formatting: Output:



**Counting the missing and non-missing values for each column and the total number of rows.**



import pandas as pd

# Sample data (replace with your actual data)

# Create pandas DataFrame df = pd.DataFrame(data)

# Count rows with at least one missing value missing\_counts = df.isnull().sum(axis=1)

count\_with\_missing = missing\_counts[missing\_counts > 0].count() # Count entries with non-zero missing values

# Print the count

print(f"Count of records with at least one missing value: count\_with\_missing}"

**output:**

### Output:Replacing the missing values with ‘unknown’ in the columns Formulation, DrugName, SubCat and SubCat1.

import pandas as pd

# Assuming your data is loaded into a pandas DataFrame named 'data' # Specify the columns to handle missing values

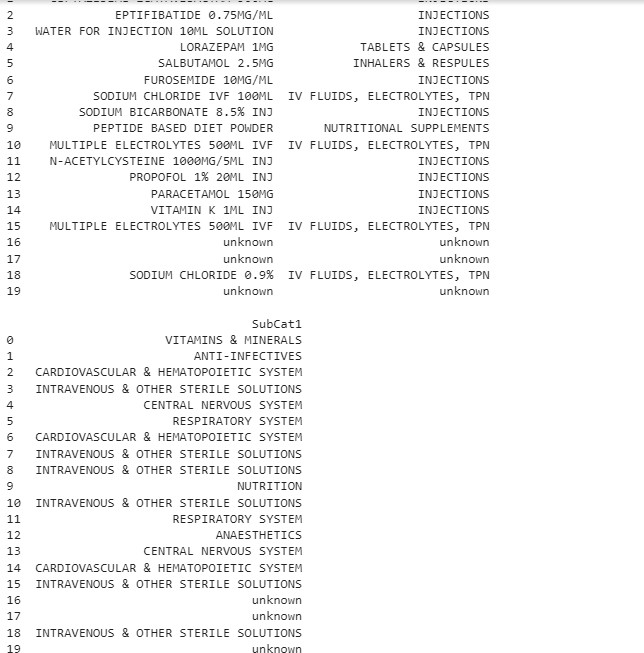
columns\_with\_missing = ['Formulation', 'DrugName', 'SubCat', 'SubCat1']

# Replace missing values with 'unknown' using a dictionary comprehension data\_updated = data.copy() # Create a copy to avoid modifying the original data

data\_updated[columns\_with\_missing] = data\_updated[columns\_with\_missing].fillna('unknown')

# You can now work with the 'data\_updated' DataFrame print(data\_updated.head(20)) # Display the first few rows

**Showing the columns after replacing the missing values with ‘unknown’:**



import pandas as pd

# Create pandas DataFrame df = pd.DataFrame(data)

# Convert Dateofbill to datetime format (assuming it's a string) df['Dateofbill'] = pd.to\_datetime(df['Dateofbill'])

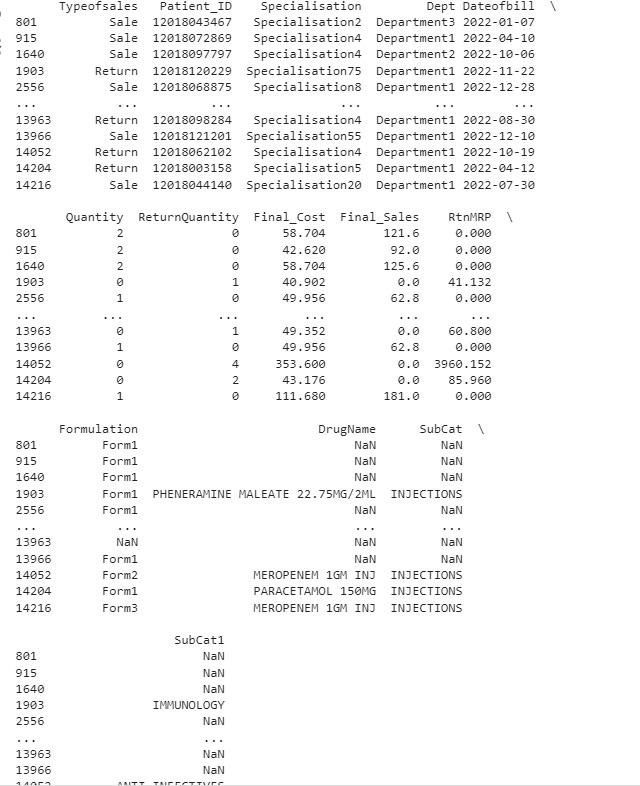
# Filter to exclude rows with DrugName 'unknown' filtered\_df = df[df['DrugName'] != 'unknown']

# Identify duplicates based on Patient\_ID, Dateofbill, and DrugName duplicates = filtered\_df.loc[filtered\_df.duplicated(subset=['Patient\_ID', 'Dateofbill', 'DrugName'], keep='first')]

# Print the duplicates print(duplicates)

### Showing missing\_values table and count of records with at least one or more missing values:

**Output:**



### Identifying duplicate rows based on Patient\_ID, Dateofbill, and DrugName excluding rows where the DrugName is 'unknown'.

import pandas as pd

import numpy as np # for standard deviation calculations

# Assuming you have your data loaded into a pandas DataFrame named 'data'

# Define columns for filtering (modify as needed) columns\_to\_filter = ['Quantity', 'ReturnQuantity', 'Final\_Cost', 'Final\_Sales', 'RtnMRP']

def filter\_by\_outliers(data, column):

"""Filters data based on three standard deviations from the mean""" mean\_value = data[column].mean()

std\_dev = data[column].std() lower\_bound = mean\_value - 3 \* std\_dev upper\_bound = mean\_value + 3 \* std\_dev

return data[(data[column] >= lower\_bound) & (data[column] <= upper\_bound)]

# Filter data for each column within the defined threshold filtered\_data = data.copy() # Create a copy to avoid modifying the original data

for col in columns\_to\_filter:

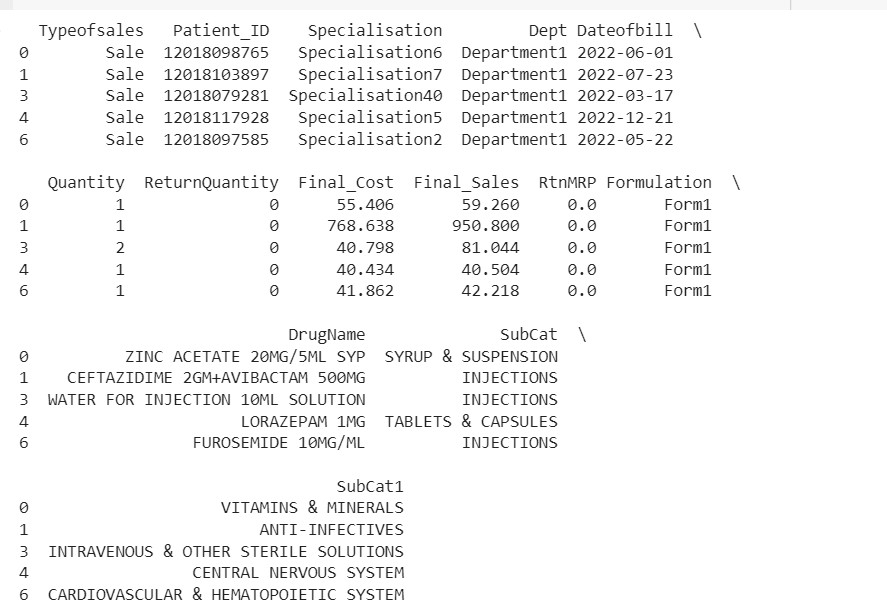
filtered\_data = filter\_by\_outliers(filtered\_data, col)

# The 'filtered\_data' DataFrame now contains rows where specified columns fall within +- 3 standard deviations of the mean

# You can now work with the 'filtered\_data' DataFrame for further analysis

print(filtered\_data.head()) # Display the first few rows

### Output:



**Removing the duplicate rows from data and counting the remainingrows.**

import pandas as pd

# Create pandas DataFrame df = pd.DataFrame(data)

# Convert Dateofbill to datetime format (assuming it's a string) df['Dateofbill'] = pd.to\_datetime(df['Dateofbill'])

# Filter to exclude rows with DrugName 'unknown' (optional, adjust as needed)

# filtered\_df = df[df['DrugName'] != 'unknown'] # Uncomment if needed

# Drop duplicates based on Patient\_ID, Dateofbill, and DrugName (keep='first' keeps the first occurrence)

df\_without\_duplicates = df.drop\_duplicates(subset=['Patient\_ID', 'Dateofbill', 'DrugName'], keep='first')

# Count the remaining rows

number\_of\_rows = df\_without\_duplicates.shape[0]

# Print the count

print(f"Number of rows after removing duplicates: {number\_of\_rows}")

### Output:



**1. In a normal distribution, approximately 68%, 95%, and 99.7% of the data falls within one, two, and three standard deviations of the mean respectively. By using three standard deviations as the threshold for removing outliers, we are effectively removing data points that are more than three standard deviations away from the mean.**

import pandas as pd

import numpy as np # for standard deviation calculations

# Assuming you have your data loaded into a pandas DataFrame named 'data'

# Define columns for filtering (modify as needed) columns\_to\_filter = ['Quantity', 'ReturnQuantity', 'Final\_Cost', 'Final\_Sales', 'RtnMRP']

def filter\_by\_outliers(data, column):

"""Filters data based on three standard deviations from the mean""" mean\_value = data[column].mean()

std\_dev = data[column].std() lower\_bound = mean\_value - 3 \* std\_dev upper\_bound = mean\_value + 3 \* std\_dev

return data[(data[column] >= lower\_bound) & (data[column] <= upper\_bound)]

# Filter data for each column within the defined threshold filtered\_data = data.copy() # Create a copy to avoid modifying the original data

for col in columns\_to\_filter:

filtered\_data = filtered\_data[filtered\_data[col] >= (mean\_value - 3 \* std\_dev)]

filtered\_data = filtered\_data[filtered\_data[col] <= (mean\_value + 3 \* std\_dev)]

# Count the number of rows in the filtered data number\_of\_rows = filtered\_data.shape[0]

# Print the count

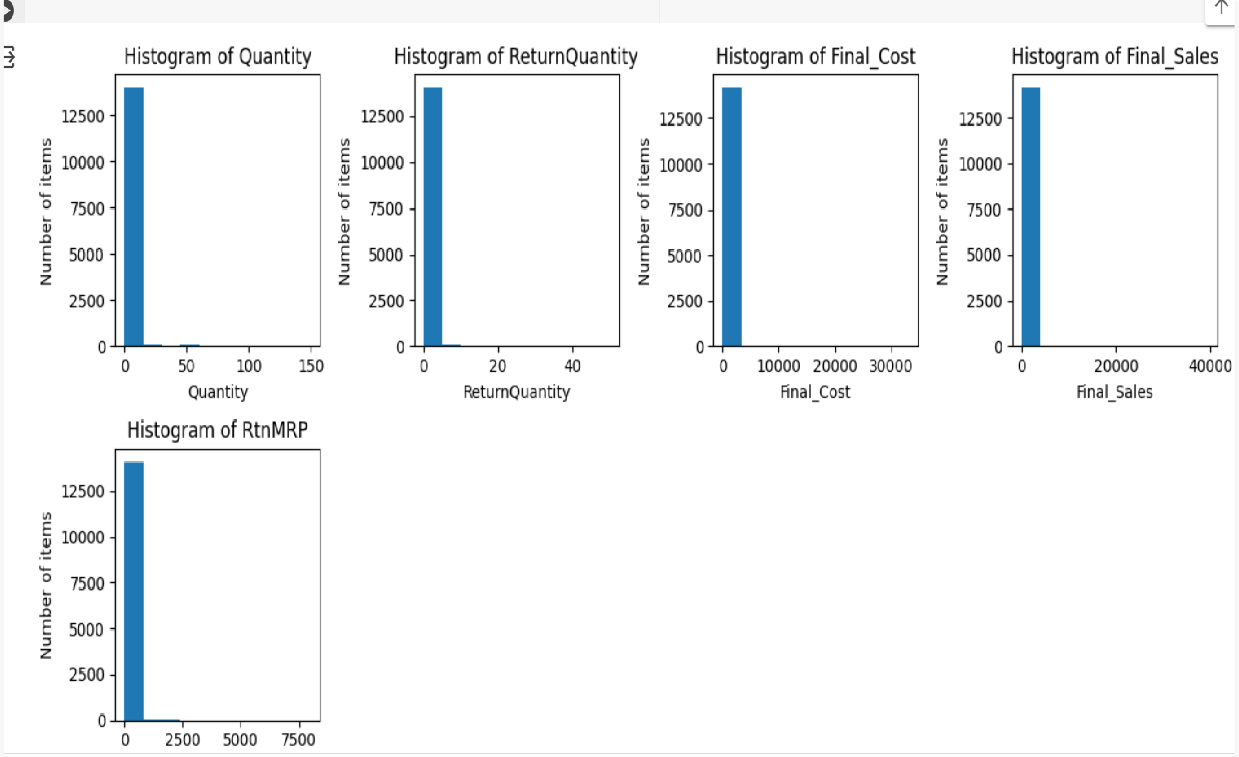
print(f"Number of rows after removing outliers: {number\_of\_rows}")

### Showing the count of rows after removing the outliers:

**Output:**



Histogram:



import matplotlib.pyplot as plt import pandas as pd

# Assuming your data is loaded into a pandas DataFrame named 'data'

# Specify the column names for the histograms (modify as needed) columns\_for\_histograms = ['Quantity', 'ReturnQuantity', 'Final\_Cost', 'Final\_Sales', 'RtnMRP']

# Create a figure for plotting multiple histograms plt.figure(figsize=(12, 6)) # Adjust figure size as desired

# Loop through each column and create a subplot with histogram for i, col in enumerate(columns\_for\_histograms):

# Create subplot at position i+1 (1-based indexing)

plt.subplot(2, 4, i+1) # Adjust grid layout (2 rows, 2 columns) as needed

plt.hist(data[col]) plt.xlabel(col) plt.ylabel('Number of items')

plt.title(f'Histogram of {col}') # Include column name in title

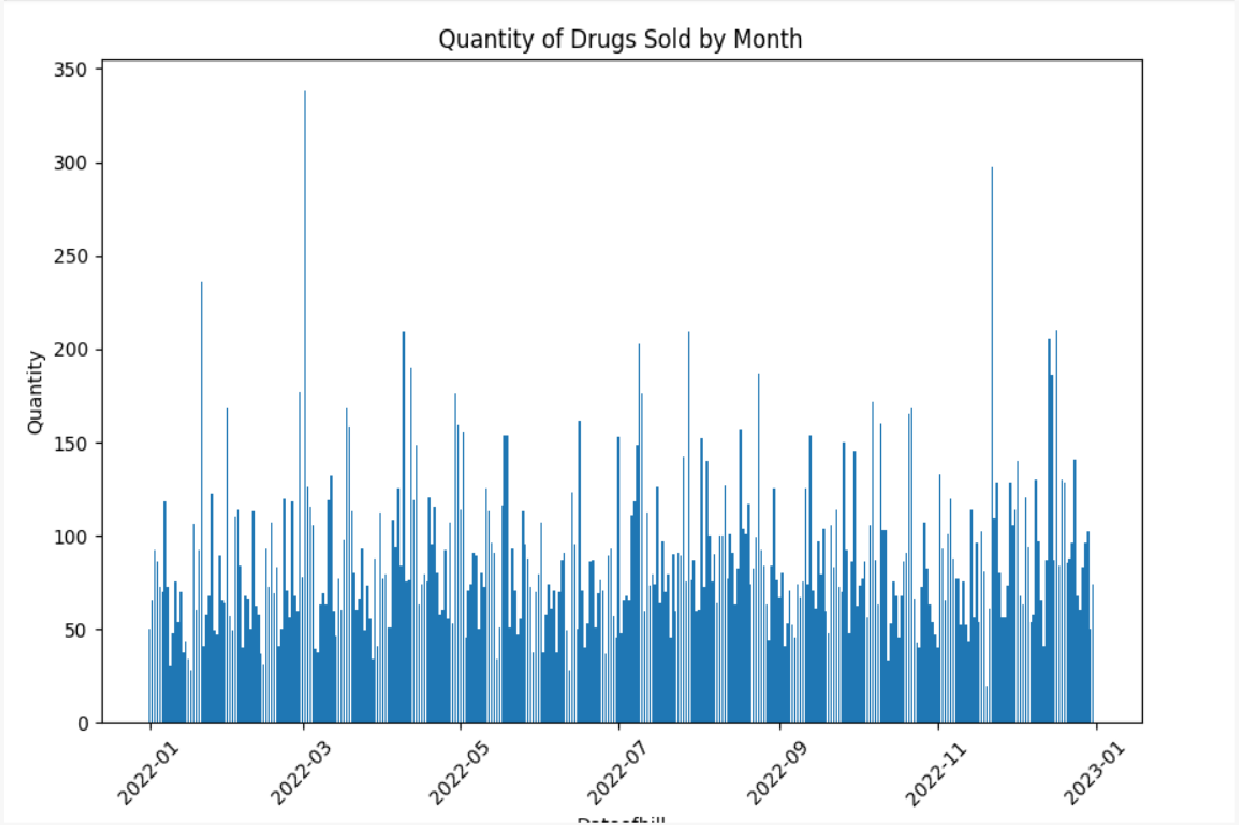
# Adjust layout (optional)

plt.tight\_layout() # Adjust spacing between subplots

# Display the plot plt.show()

#### output:

Bar plot:



import pandas as pd

import matplotlib.pyplot as plt

# Assuming your data is loaded into a pandas DataFrame named 'data'

# Specify columns containing 'Month' and 'Quantity' (modify as needed) month\_column = 'Dateofbill'

quantity\_column = 'Quantity'

# Group data by month and sum the quantity

data\_by\_month = data.groupby(month\_column)[quantity\_column].sum()

# Create a bar plot

plt.figure(figsize=(10, 6)) # Adjust figure size as desired plt.bar(data\_by\_month.index, data\_by\_month.values) plt.xlabel(month\_column)

plt.ylabel(quantity\_column) plt.title('Quantity of Drugs Sold by Month')

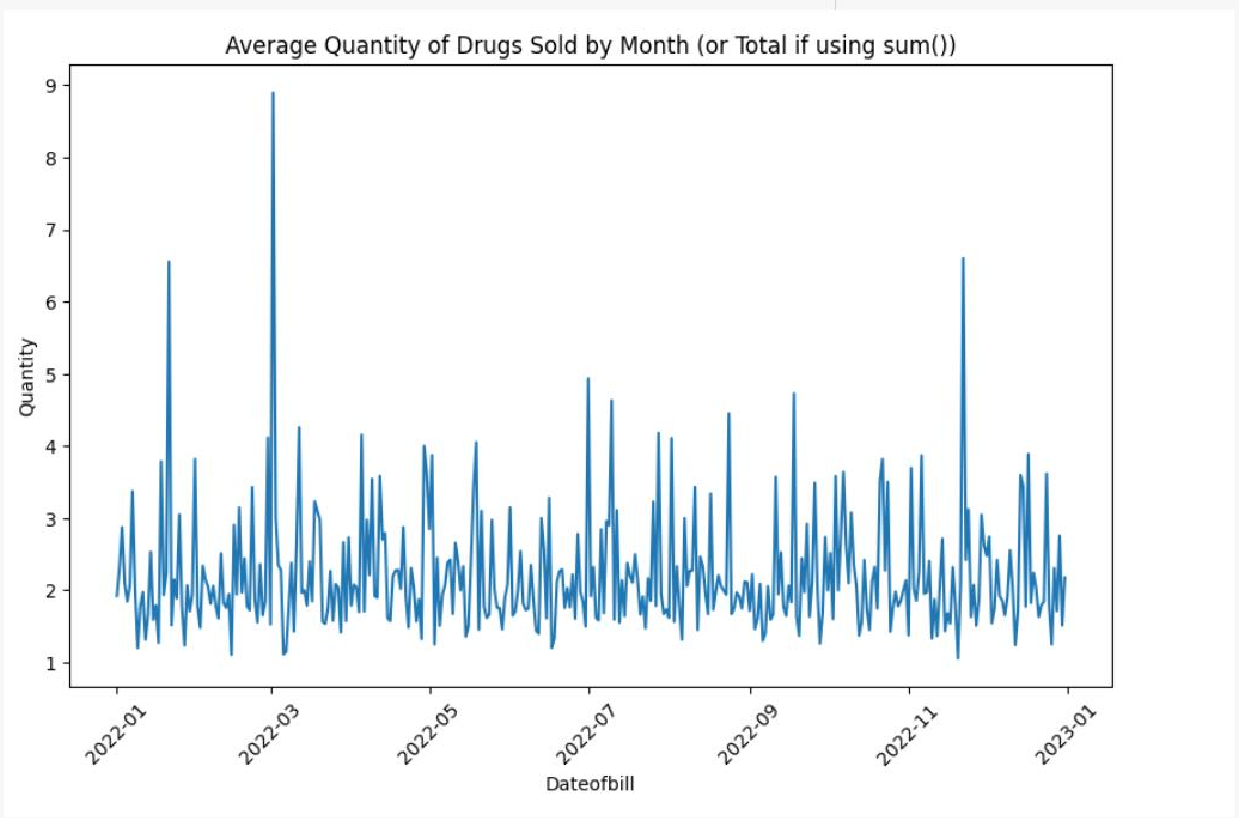
# Rotate x-axis labels for readability if there are many months if len(data\_by\_month.index) > 10:

plt.xticks(rotation=45)

# Display the plot plt.show()

#### output:

Line plot:



import pandas as pd

import matplotlib.pyplot as plt

# Assuming your data is loaded into a pandas DataFrame named 'data'

# Specify columns containing 'Month' and 'Quantity' (modify as needed) month\_column = 'Dateofbill'

quantity\_column = 'Quantity'

# Group data by month and calculate average quantity (or modify for sum) data\_by\_month = data.groupby(month\_column)[quantity\_column].mean() # Use mean() for average, sum() for total

# Create a line chart

plt.figure(figsize=(10, 6)) # Adjust figure size as desired plt.plot(data\_by\_month.index, data\_by\_month.values) plt.xlabel(month\_column)

plt.ylabel(quantity\_column)

plt.title('Average Quantity of Drugs Sold by Month (or Total if using sum())')

# Rotate x-axis labels for readability if there are many months if len(data\_by\_month.index) > 10:

plt.xticks(rotation=45)

# Display the plot plt.show()

#### output:

Box plot:

import seaborn as sns import pandas as pd

import matplotlib.pyplot as plt

# Sample data (replace with your actual data)

# Create a DataFrame

df = pd.DataFrame(data)

# Create a box plot of all columns sns.boxplot(

data=df,

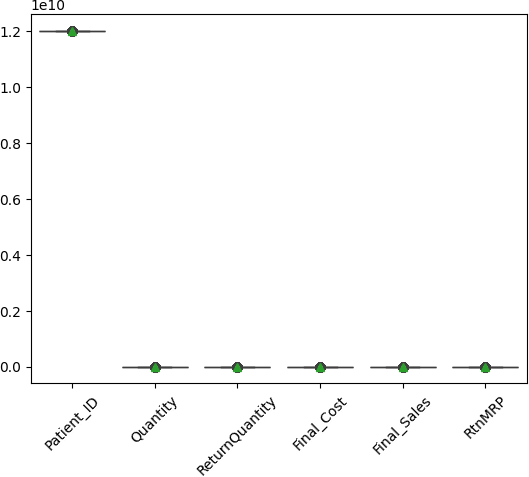
orient="v", # Vertical orientation showmeans=True # Show mean with boxplot

)

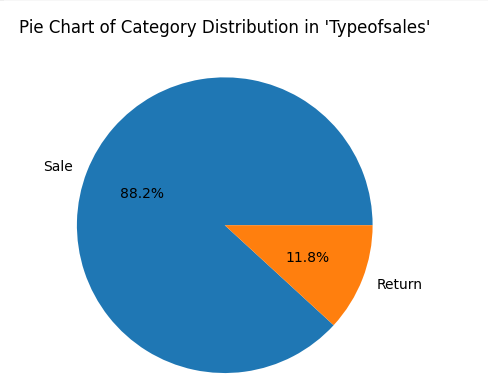
# Rotate x labels for readability if there are many categories plt.xticks(rotation=45)

plt.show()

**output:**



Pie chart :



import pandas as pd

import matplotlib.pyplot as plt

# Assuming your data is loaded into a pandas DataFrame named 'data'

# Get all categorical columns (assuming you want pie charts only for categorical data)

categorical\_columns = [col for col in data.columns if data[col].dtype == 'object']

# Iterate over each categorical column for col in categorical\_columns:

# Get the number of items in each category n\_item\_per\_category =

data[col].value\_counts().sort\_values(ascending=False)

# Get pie chart slice labels

pie\_chart\_slice\_labels = n\_item\_per\_category.index.to\_numpy()

# Get pie chart slice values

pie\_chart\_slice\_values = n\_item\_per\_category.to\_numpy()

# Create a pie chart for the current column plt.figure() # Create a new figure for each pie chart

plt.pie(pie\_chart\_slice\_values, labels=pie\_chart\_slice\_labels, autopct="%1.1f%%")

plt.title(f"Pie Chart of Category Distribution in '{col}'") plt.show()

#### output:

