

# GLOBAL SUPERSTORE DATASET SALES ANALYSIS

By Student ID: 202410472

## 1. Task 1

### Import required Python Libraries

```
In [ ]: # pip install packages
import sys
!{sys.executable} -m pip install word2number
!{sys.executable} -m pip install geopandas
!{sys.executable} -m pip install adjustText

# import libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
import socket
import requests
import geopandas as gpd

from matplotlib import ticker as mtick
from tabulate import tabulate
from scipy.stats import chi2_contingency
from word2number import w2n
from adjustText import adjust_text

# Letting pandas to show max columns
pd.set_option('display.max_columns', None)

# Set the style for seaborn plots
sns.set_theme(style="whitegrid")
```

Defaulting to user installation because normal site-packages is not writeable  
Requirement already satisfied: word2number in c:\users\nares\appdata\roaming\python\python311\site-packages (1.1)  
Defaulting to user installation because normal site-packages is not writeable  
Requirement already satisfied: geopandas in c:\users\nares\appdata\roaming\python\python311\site-packages (0.14.3)  
Requirement already satisfied: fiona>=1.8.21 in c:\users\nares\appdata\roaming\python\python311\site-packages (from geopandas) (1.9.6)  
Requirement already satisfied: packaging in c:\programdata\anaconda3\lib\site-packages (from geopandas) (23.1)  
Requirement already satisfied: pandas>=1.4.0 in c:\programdata\anaconda3\lib\site-packages (from geopandas) (2.0.3)  
Requirement already satisfied: pyproj>=3.3.0 in c:\users\nares\appdata\roaming\python\python311\site-packages (from geopandas) (3.6.1)  
Requirement already satisfied: shapely>=1.8.0 in c:\users\nares\appdata\roaming\python\python311\site-packages (from geopandas) (2.0.4)  
Requirement already satisfied: attrs>=19.2.0 in c:\programdata\anaconda3\lib\site-packages (from fiona>=1.8.21->geopandas) (22.1.0)  
Requirement already satisfied: certifi in c:\programdata\anaconda3\lib\site-packages (from fiona>=1.8.21->geopandas) (2024.2.2)  
Requirement already satisfied: click~=8.0 in c:\programdata\anaconda3\lib\site-packages (from fiona>=1.8.21->geopandas) (8.0.4)  
Requirement already satisfied: click-plugins>=1.0 in c:\users\nares\appdata\roaming\python\python311\site-packages (from fiona>=1.8.21->geopandas) (1.1.1)  
Requirement already satisfied: cligj>=0.5 in c:\users\nares\appdata\roaming\python\python311\site-packages (from fiona>=1.8.21->geopandas) (0.7.2)  
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from fiona>=1.8.21->geopandas) (1.16.0)  
Requirement already satisfied: python-dateutil>=2.8.2 in c:\programdata\anaconda3\lib\site-packages (from pandas>=1.4.0->geopandas) (2.8.2)  
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas>=1.4.0->geopandas) (2023.3.post1)  
Requirement already satisfied: tzdata>=2022.1 in c:\programdata\anaconda3\lib\site-packages (from pandas>=1.4.0->geopandas) (2023.3)  
Requirement already satisfied: numpy>=1.21.0 in c:\programdata\anaconda3\lib\site-packages (from pandas>=1.4.0->geopandas) (1.24.3)  
Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from click~=8.0->fiona>=1.8.21->geopandas) (0.4.6)  
Defaulting to user installation because normal site-packages is not writeable  
Requirement already satisfied: adjustText in c:\users\nares\appdata\roaming\python\python311\site-packages (1.1.1)  
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from adjustText) (1.24.3)  
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from adjustText) (3.7.2)  
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from adjustText) (1.11.1)  
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (1.0.5)  
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (0.11.0)  
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (4.25.0)  
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (1.4.4)  
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (23.1)  
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (10.0.1)  
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (3.0.9)  
Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\anaconda3\lib\site-packages (from matplotlib->adjustText) (2.8.2)

```
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib->adjustText) (1.16.0)
```

## Load first 10 records

```
In [ ]: # Reading CSV file and assigning into a dataframe ss_data
global_super_store_df_org = pd.read_csv('sample-superstore 2023 T3.csv')

# Copy the dataframe before processing
global_super_store_df = global_super_store_df_org.copy()

# Get the first 1000 records only for the EDA
global_super_store_df = global_super_store_df.head(n=1000)

# Set the head to 10 to retrieve the first 10 records
first_10_rows = global_super_store_df.head(n=10)
print(tabulate(first_10_rows, headers='keys', tablefmt='pretty', stralign ='right'),
```

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	
7773	CA-2016-108196	25/11/2016	12/02/2016	Standard Class	CS-12505
Cindy Stewart	Consumer	United States	Lancaster	Ohio	
43130	Est	TEC-MA-1000418	Technology	Machines	
Cubify CubeX 3D Printer Double Head Print	599.978	4499.985	5	0.7	-6
684	US-2017-168116	11/04/2017	11/04/2017	Same Day	GT-14635
Grant Thornton	Corporate	United States	Burlington	North Carolina	
"27217"	South	TEC-MA-10004125	Technology	Machines	
Cubify CubeX 3D Printer Triple Head Print	39.9904	7999.98	4	0.5	-38
9775	CA-2014-169019	26/07/2014	30/07/2014	Standard Class	LF-17185
Luke Foster	Consumer	United States	San Antonio	Texas	
78207	Central	OFF-BI-10004995	Office Supplies	Binders	
GBC DocuBind P400 Electric Binding System	01.8928	2177.584	8	0.8	-37
3012	CA-2017-134845	17/04/2017	24/04/2017	Standard Class	SR-20425
Sharelle Roach	Home Office	United States	Louisville	Colorado	
80027	West	TEC-MA-10000822	Technology	Machines	
Lexmark MX611dhe Monochrome Laser Printer	3399.98	2549.985	5	0.7	-
4992	US-2017-122714	12/07/2017	13/12/2017	Standard Class	HG-14965
Henry Goldwyn	Corporate	United States	Chicago	Illinois	
60653	Central	OFF-BI-10001120	Office Supplies	Binders	
Ibico EPK-21 Electric Binding System	45	1889.99	5	0.8	-2929.48
3152	CA-2015-147830	15/12/2015	18/12/2015	First Class	NF-18385
Natalie Fritzler	Consumer	United States	Newark	Ohio	
43055	East	TEC-MA-10000418	Technology	Machines	
Cubify CubeX 3D Printer Double Head Print	9.9912"	1799.994	Two	0.7	"-263
5311	CA-2017-131254	19/11/2017	21/11/2017	First Class	NC-18415
Nathan Cano	Consumer	United States	Houston	Texas	
77095	Central	OFF-BI-10003527	Office Supplies	Binders	Fellowes PB50
0 Electric Punch Plastic Comb Binding Machine with Manual Bind	6	1525.188			
9640	CA-2015-116638	28/01/2015	nan	Second Class	JH-15985
Joseph Holt	Consumer	United States	Concord	North Carolina	
28027	South	FUR-TA-10000198	Frnture	Tables	
Chromcraft Bull-Nose Wood Oval Conference Tables & Bases	0.4	4297.644	Thirteen		
1200	CA-2016-130946	04/08/2016	04/12/2016	Standard Class	ZC-21910
Zuschuss Carroll	Consumer	United States	Houston	Texas	
77041	Central	OFF-BI-10004995	Office Supplies	Binders	
GBC DocuBind P400 Electric Binding System	50.9464	1088.792	4	0.8	-18
2698	CA-2014-145317	18/03/2014	23/03/2014	Standard Class	SM-20320
Sean Miller	Home Office	nan	Jacksonville	Florida	
32216	Southh	TEC-MA-10002412	Technology	Machines	
Cisco TelePresence System EX90 Videoconferencing Unit	0.5	22638.48	6		
-1811.0784					

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+
```

## Understanding of the dataset

### Information of descriptive statistics

Using info() and describe() function to get the descriptive statistics

```
In [ ]: # Get the metadata information about the dataset
# Grouping data types by category
numerical_columns = global_super_store_df.select_dtypes(include=['int64', 'float64'])
categorical_columns = global_super_store_df.select_dtypes(include=['object']).columns
datetime_columns = global_super_store_df.select_dtypes(include=['datetime64']).columns

# Define a table width and print the header row with a dotted line
table_width = 250
print("-" * table_width)

# Print each row of the table with content and a dash line
content_list = [f"We are working with a {global_super_store_df.shape} sized dataset",
                f"Numerical columns: {''.join(numerical_columns)}",
                f"Categorical columns: {''.join(categorical_columns)}",
                f"Date Time columns: {''.join(datetime_columns)}"]

for content_row in content_list:
    print(" | {:<246} |".format(content_row))
    print("-" * table_width)

# Print the table
print(tabulate(global_super_store_df.describe(), headers='keys', tablefmt='pretty',
```

```
-----  
| We are working with a (1000, 21) sized dataset.  
|  
-----  
|  
-----  
| Numerical columns: Row ID, Sales, Discount  
|  
-----  
|  
-----  
| Categorical columns: Order ID, Order Date, Ship Date, Ship Mode, Customer ID, Cu  
stomer Name, Segment, Country, City, State, Postal Code, Region, Product ID, Categ  
ory, Sub-Category, Product Name, Quantity, Profit  
|  
-----  
|  
-----  
| Date Time columns:  
|  
-----  
|  
-----  
+-----+-----+-----+  
|       Row ID |           Sales |        Discount |  
+-----+-----+-----+  
|      1000.0 |         999.0 |        999.0 |  
| 5033.065 | 415.95479939939935 | 0.4533433433433433 |  
| 2955.719828481633 | 940.7397657620787 | 0.22030138136310903 |  
|      4.0 |         8.652 |        0.15 |  
| 2435.0 |        73.8495 |        0.2 |  
| 5014.0 |        218.352 |        0.4 |  
| 7727.75 |        475.779 |        0.7 |  
| 9963.0 |       22638.48 |        0.8 |  
+-----+-----+-----+
```

```
In [ ]: # Print metatype information about the dataset  
print(global_super_store_df.info())
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Row ID             1000 non-null   int64  
 1   Order ID           999 non-null    object  
 2   Order Date          998 non-null    object  
 3   Ship Date           997 non-null    object  
 4   Ship Mode            997 non-null    object  
 5   Customer ID         1000 non-null    object  
 6   Customer Name        997 non-null    object  
 7   Segment              997 non-null    object  
 8   Country              996 non-null    object  
 9   City                 999 non-null    object  
 10  State                998 non-null    object  
 11  Postal Code          998 non-null    object  
 12  Region               999 non-null    object  
 13  Product ID           998 non-null    object  
 14  Category              999 non-null    object  
 15  Sub-Category          996 non-null    object  
 16  Product Name          997 non-null    object  
 17  Sales                 999 non-null    float64 
 18  Quantity              995 non-null    object  
 19  Discount              999 non-null    float64 
 20  Profit                 989 non-null    object  
dtypes: float64(2), int64(1), object(18)
memory usage: 164.2+ KB
None

```

## Missing Data Analysis

```

In [ ]: # Count the number of missing values in each column
missing_values_per_column = global_super_store_df.isna().sum()

# Convert the Series to a DataFrame for tabulation
missing_values_df = missing_values_per_column.to_frame().reset_index()
missing_values_df.columns = ['Column', 'Missing Values']

# Print the tabulated missing values per column
print("Number of missing values per column:\n")
print(tabulate(missing_values_df, headers='keys', tablefmt='pretty', stralign ='right'))

# Count the total number of missing values across all columns
total_missing_values = missing_values_per_column.sum()

# Print the total number of missing values
print("\nTotal number of missing values:", total_missing_values)

```

Number of missing values per column:

Column	Missing Values
Row ID	0
Order ID	1
Order Date	2
Ship Date	3
Ship Mode	3
Customer ID	0
Customer Name	3
Segment	3
Country	4
City	1
State	2
Postal Code	2
Region	1
Product ID	2
Category	1
Sub-Category	4
Product Name	3
Sales	1
Quantity	5
Discount	1
Profit	11

Total number of missing values: 53

The primary key of these records are a system-generated, and denoted as column: *RowID*

The datatypes of the dataset are following:

- int64(1)
- float64(2)
- object(18)

A few records of *Quantity* and *Profit* columns has the datatype of object, but it must be float64, thus needs to be cleansed or transformed.

*Ship Date* and *Order Date* columns are represented as strings, those needs to be converted as datetime.

Once cleansed, the descriptive statistics can be applied to the numeral columns, and they are Sales, Quantity, Discount and Profit/Loss.

## Histograms based on categorical features

Function to generate a histogram plot for the given dataset.

```
In [ ]: def plot_histogram(data, xlabel, ylabel, title):
    """
    plot_histogram generates a histogram plot for the given dataset.
    Parameters:
        data: Dataset for the histogram plot (pandas Series or any iterable).
        xlabel: Label for the x-axis.
        ylabel: Label for the y-axis.
        title: Title of the plot.
    """
```

```

Returns:
    None
"""

plt.figure(figsize=(12, 10))

bars = plt.bar(data.index, data)

plt.title(title)
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.xticks(rotation=90)

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, height, ha='center', va='bottom')
plt.show()

```

```

In [ ]: # Histogram for 'Ship Mode'
orders_per_segment = global_super_store_df['Ship Mode'].value_counts(ascending=True)
plot_histogram(orders_per_segment, 'Ship Mode', 'Number of Orders', 'Number of Orders')

# Histogram for 'Segment'
orders_per_segment = global_super_store_df['Segment'].value_counts(ascending=True)
plot_histogram(orders_per_segment, 'Segment', 'Number of Orders', 'Number of Orders')

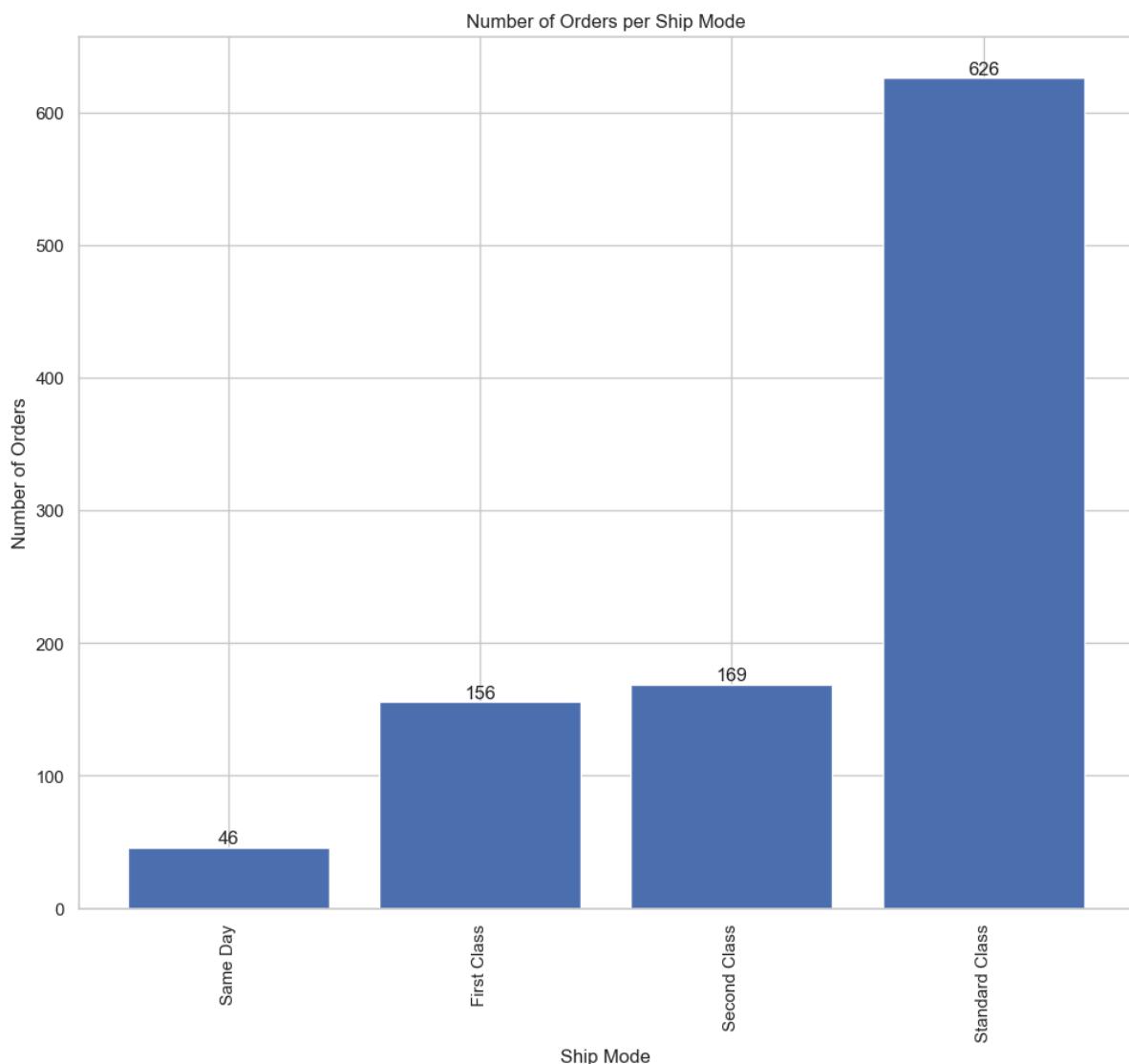
# Histogram for 'Category'
orders_per_category = global_super_store_df['Category'].value_counts(ascending=True)
plot_histogram(orders_per_category, 'Category', 'Number of Orders', 'Number of Orders')

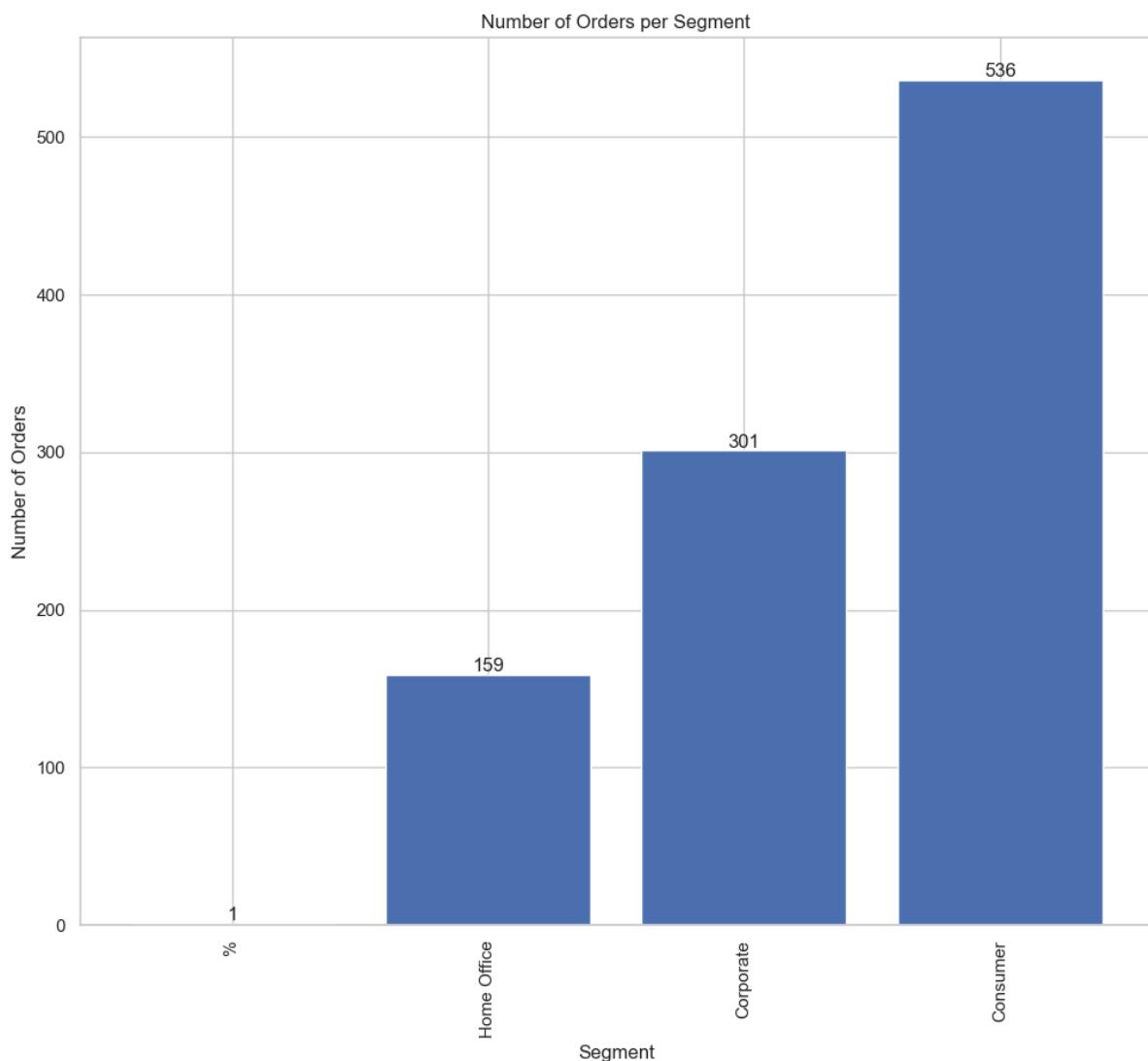
# Histogram for 'Sub-Category'
orders_per_sub_category = global_super_store_df['Sub-Category'].value_counts(ascending=True)
plot_histogram(orders_per_sub_category, 'Sub-Category', 'Number of Orders', 'Number of Orders')

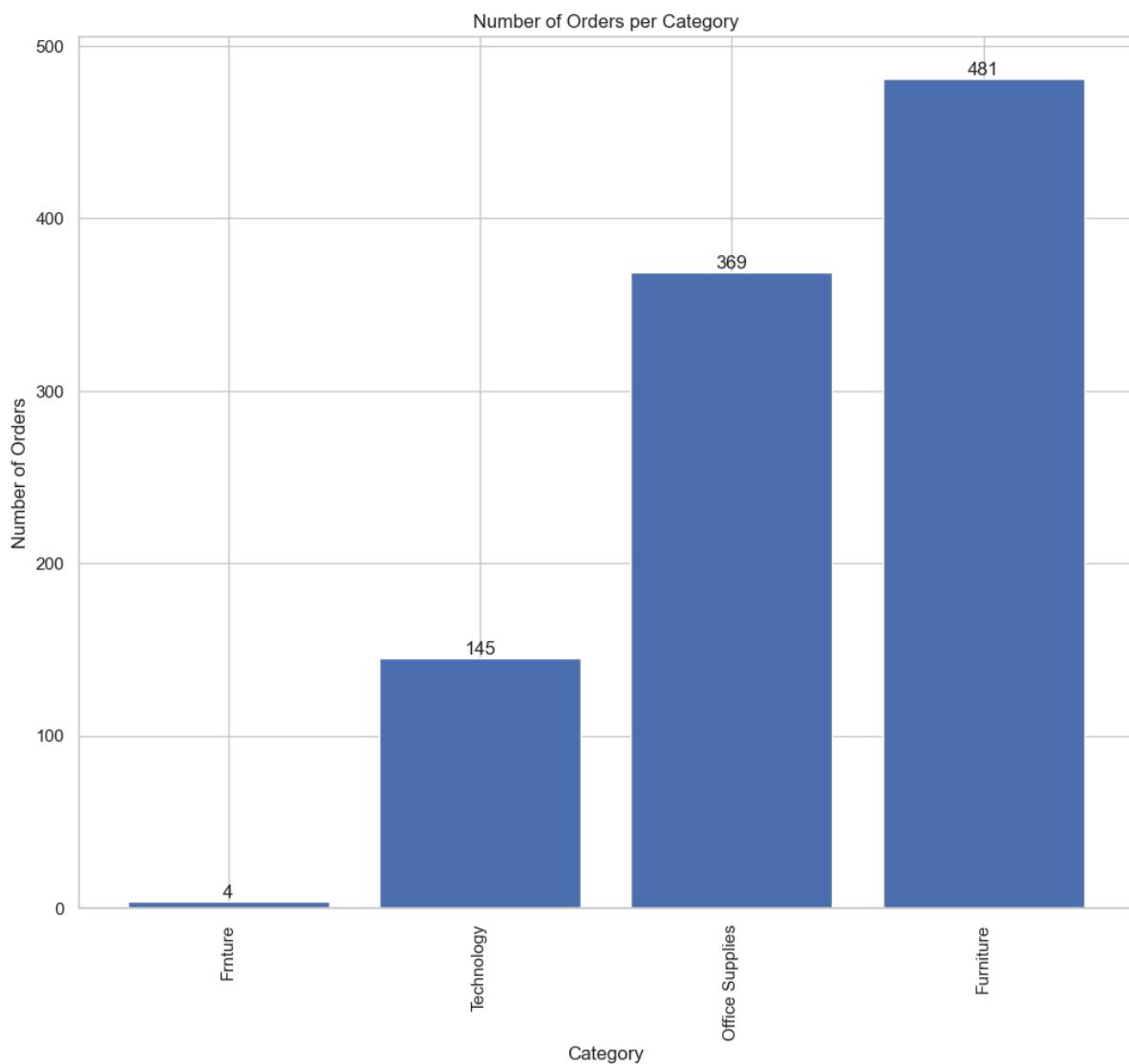
# Histogram for 'Region'
orders_per_sub_category = global_super_store_df['Region'].value_counts(ascending=True)
plot_histogram(orders_per_sub_category, 'Region', 'Number of Orders', 'Number of Orders')

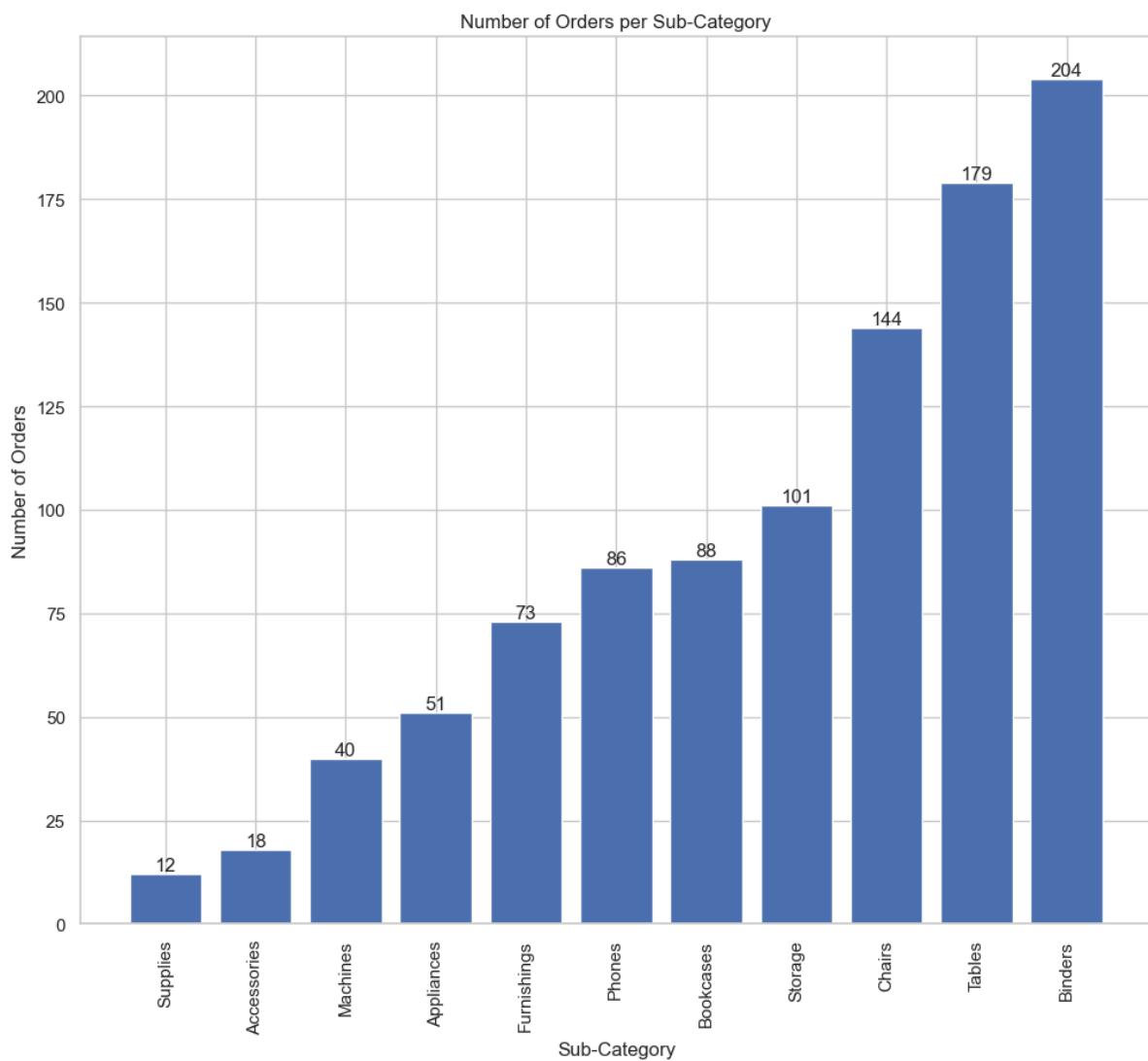
# Histogram for 'State'
orders_per_sub_category = global_super_store_df['State'].value_counts(ascending=True)
plot_histogram(orders_per_sub_category, 'State', 'Number of Orders', 'Number of Orders')

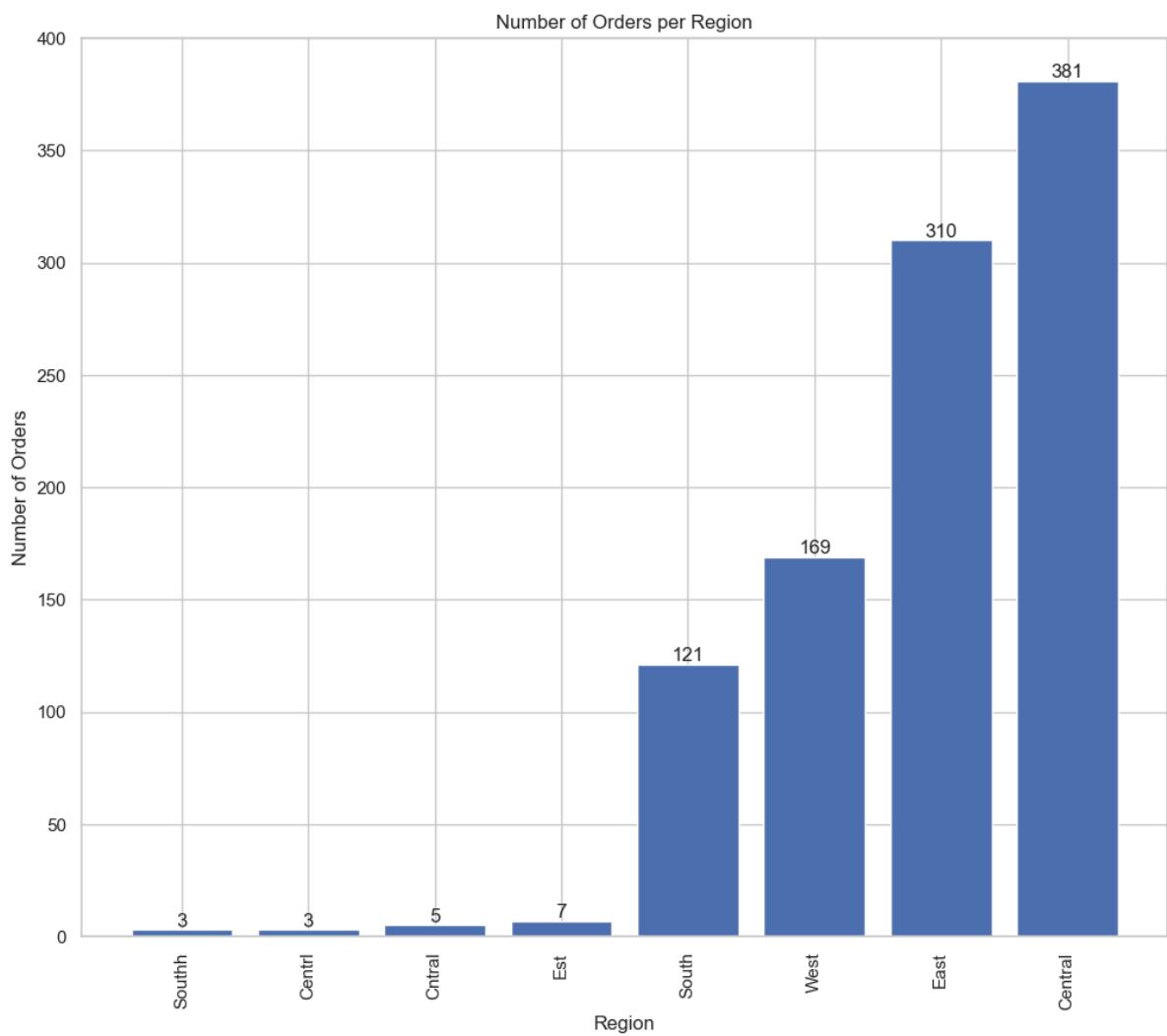
```

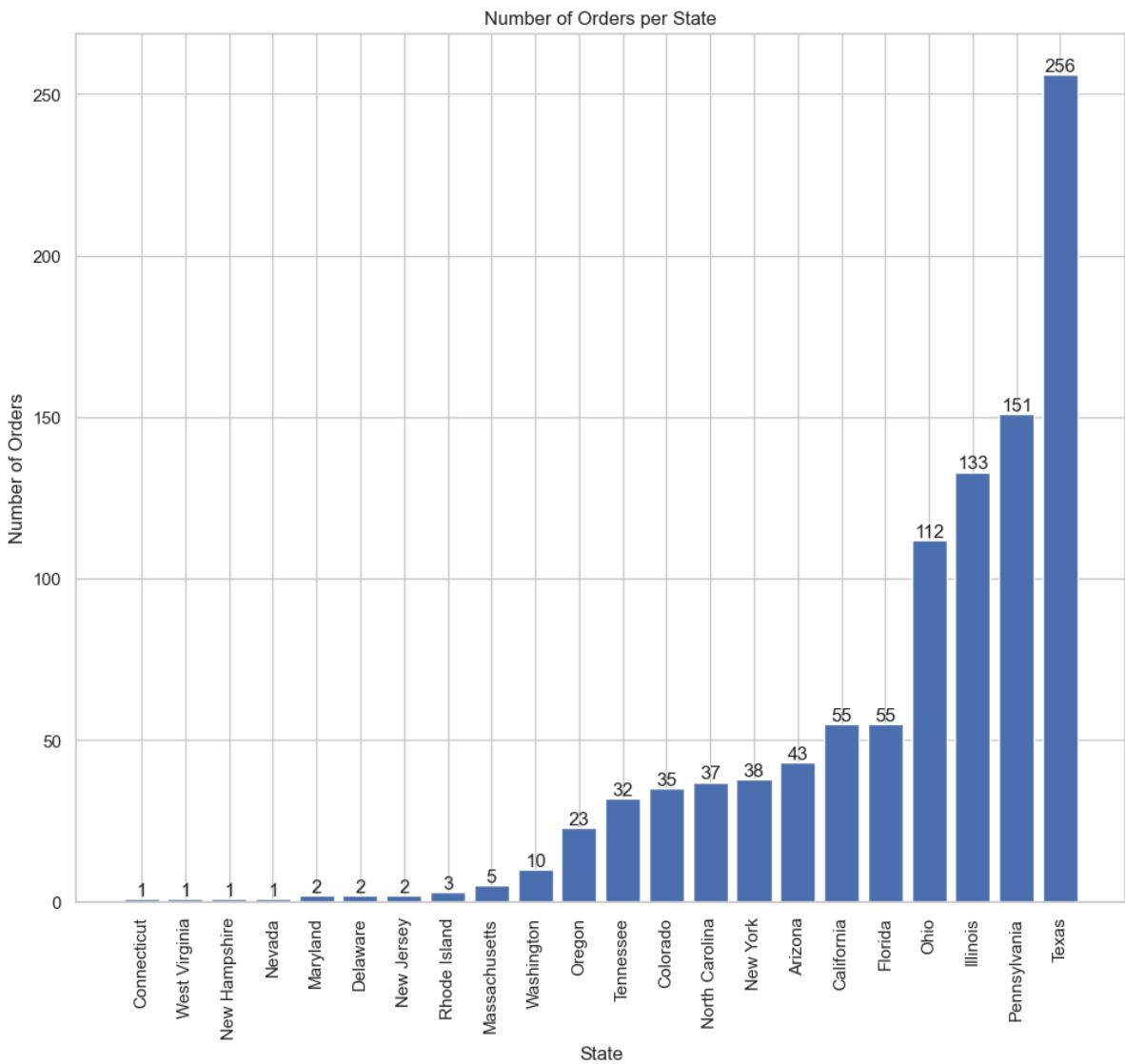












The `plot_histogram()` function can be run after the data cleansing.

## 2. Task 2

### Handling missing values in the dataset

#### Helper functions

Function to convert alphabetic values in a DataFrame column to numbers and update the DataFrame.

```
In [ ]: def convert_alphabetic_to_float(df, numerical_feature):
    """
    Convert alphabetic values in a DataFrame column to numbers and update the DataFrame.

    Parameters:
        df (pandas.DataFrame): The DataFrame containing the column to be processed.
        numerical_feature (str): The name of the column containing the values to be converted.

    Returns:
        None. The function modifies the input DataFrame in place.
    """
    pass
```

```

# Create alphabetic filter using regex
alphabetic_filter = df[numerical_feature].str.contains(r'^[a-zA-Z]+$', na=False)
alphabetic_filtered_df = df.loc[alphabetic_filter]

# Convert alphabetic values to numbers using w2n.word_to_num library
alphabetic_filtered_df.loc[:, numerical_feature] = alphabetic_filtered_df[numerical_feature].apply(w2n.word_to_num)

# Update the original DataFrame with the filtered results
df.update(alphabetic_filtered_df)

# Convert the column to float
df[numerical_feature] = df[numerical_feature].astype(float)

```

Function to check if the machine is connected to internet, so its used in

`get_city_from_postal_code` and `get_state_from_postal_code` functions respectively.

```

In [ ]: def is_internet_connected():
    """
    Check if the machine is connected to the internet by attempting to connect to Google's DNS server.
    The function is used to connect to post code API to retrieve City and State.
    Returns:
        bool: True if the machine is connected to the internet, False otherwise.
    """
    try:
        # Attempt to connect to Google's DNS server
        socket.create_connection(("8.8.8.8", 53), timeout=3)
        return True
    except OSError:
        pass
    return False

```

Function to retrieve the city name from postal code using zippopotam API.

```

In [ ]: def get_city_from_postal_code(postal_code):
    """
    get_city_from_postal_code retrieves the city name from postal code using zippopotam API.
    Parameters:
        postal_code: Postal code for the city.
    Returns:
        str or None: City name corresponding to the given postal code, or None if no city found.
    """
    if not is_internet_connected():
        return None

    if postal_code == '':
        return None

    url = f"http://api.zippopotam.us/us/{postal_code}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        city = data['places'][0]['place name']
        return city
    else:
        return None

```

Function to retrieve the state name from postal code using zippopotam API.

```

In [ ]: def get_state_from_postal_code(postal_code):
    """

```

```

get_state_from_postal_code retrieves the state name from postal code using zipp
Parameters:
    postal_code: Postal code for the state.
Returns:
    str or None: State name corresponding to the given postal code, or None if
    """
    """
if not is_internet_connected():
    return None

if postal_code == '':
    return None

url = f"http://api.zippopotam.us/us/{postal_code}"
response = requests.get(url)
if response.status_code == 200:
    data = response.json()
    state = data['places'][0]['state']
    return state
else:
    return None

```

Function to format a date string to the format 'yyyy-mm-dd'.

```

In [ ]: def format_date_in_yyyyymmdd(date_str):
    """
    Formats a date string to the format 'yyyy-mm-dd'.

    Parameters:
    - date_str (str or pd.NA): A string representing a date.

    Returns:
    - str or pd.NA: The standardized date string in 'yyyy-mm-dd' format, or pd.NA i
    """
    if pd.notna(date_str) and date_str.strip(): # Check if date is not blank or Not
        try:
            # Attempt to parse as yyyy-mm-dd with dayfirst=False
            return pd.to_datetime(date_str, dayfirst=False, errors='coerce').strftime('%Y-%m-%d')
        except ValueError:
            # If parsing fails, attempt to parse as dd-mm-yyyy
            return pd.to_datetime(date_str, format='%d-%m-%Y', errors='coerce').strftime('%Y-%m-%d')
    else:
        return date_str # Return the original blank or NaN value

```

## Dropping unwanted columns

```

In [ ]: # Columns to drop
columns_to_drop = ['Row ID', 'Order ID', 'Customer ID', 'Product ID', 'Product Name',
                   'Sales', 'Quantity', 'Profit']

# Row ID is not needed for the analysis, hence dropping the column
global_super_store_df.drop(columns=columns_to_drop, inplace=True)

```

## Replacing faulty values

```

In [ ]: # Removing "?" from Quantity column
global_super_store_df['Quantity'] = global_super_store_df['Quantity'].str.replace('?', '')

# Removing "''' from profit column
global_super_store_df['Profit'] = global_super_store_df['Profit'].str.replace('''', '')

# Removing "''' from Postal Code column

```

```

global_super_store_df['Postal Code'] = global_super_store_df['Postal Code'].str.replace('99999-0000', '99999-0000')

# Make all records as Country = United States
global_super_store_df['Country'] = 'United States'

# Correcting spelling mistakes on Category column
global_super_store_df['Category'] = global_super_store_df['Category'].replace('Front', 'Front')

# Datafix on Category based on subcategories
# Apply the condition element-wise
condition = (global_super_store_df['Category'] == 'NO_CATEGORY') & \
            (global_super_store_df['Sub-Category'].isin(['Binders', 'Storage']))

# Update 'Category' where the condition is True
global_super_store_df.loc[condition, 'Category'] = 'Office Supplies'

# Update empty and wrong records on Segment Column
global_super_store_df['Segment'] = global_super_store_df['Segment'].replace('%', 'N/A')

# Cleanse the Regions
central_regions_to_replace = ['Centrl', 'Cntral']
east_regions_to_replace = ['Est']
south_regions_to_replace = ['Southh']

global_super_store_df['Region'] = global_super_store_df['Region'].replace(central_regions_to_replace)
global_super_store_df['Region'] = global_super_store_df['Region'].replace(east_regions_to_replace)
global_super_store_df['Region'] = global_super_store_df['Region'].replace(south_regions_to_replace)

# Cleanse the Order Date column
global_super_store_df['Order Date'] = global_super_store_df['Order Date'].str.replace('/', '-')

# Cleanse the Customer Name column
global_super_store_df['Customer Name'] = global_super_store_df['Customer Name'].str.replace(' ', '')

```

## Filling empty values with default values

```

In [ ]: # Filling values on empty Customer Name records
global_super_store_df['Customer Name'] = global_super_store_df['Customer Name'].fillna('Unknown')

# Assuming zero values for NaN on Profit
global_super_store_df['Profit'] = global_super_store_df['Profit'].fillna(0.00)

# Filling values on empty Category and Sub-Category records
global_super_store_df['Category'] = global_super_store_df['Category'].fillna('NO_CATEGORY')
global_super_store_df['Sub-Category'] = global_super_store_df['Sub-Category'].fillna('N/A')

# Filling Quantity as 1 as sold quantity cannot be zero
global_super_store_df['Quantity'] = global_super_store_df['Quantity'].fillna(1.00)

# Filling Discount as 0
global_super_store_df['Discount'] = global_super_store_df['Discount'].fillna(0.00)

# Filling Sales as 0
global_super_store_df['Sales'] = global_super_store_df['Sales'].fillna(0.00)

# Filling values on empty Ship Mode records
global_super_store_df['Ship Mode'] = global_super_store_df['Ship Mode'].fillna('No Mode')

# Filling values on empty Postal Code records
global_super_store_df['Postal Code'] = global_super_store_df['Postal Code'].fillna('99999-0000')

# Filling values on empty Region records
global_super_store_df['Region'] = global_super_store_df['Region'].fillna('NO_REGION')

```

```
# Filling values on empty Segment records
global_super_store_df['Segment'] = global_super_store_df['Segment'].fillna('NO_SEGM
```

## Applying helper functions to fill the data

```
In [ ]: # Filter the empty state rows
state_filtered_na = global_super_store_df.loc[pd.isna(global_super_store_df['State'])]
state_filtered_na

# Apply the function to fill the missing value via API
global_super_store_df.loc[pd.isna(global_super_store_df['State'])], 'State'] = state_fi
```

```
In [ ]: # Filter the empty city rows
city_filtered_na = global_super_store_df.loc[pd.isna(global_super_store_df['City'])]
city_filtered_na

# Apply the function to fill the missing value via API
global_super_store_df.loc[pd.isna(global_super_store_df['City'])], 'City'] = city_fi
```

```
In [ ]: # Call the function to convert the the Quantity and Profit columns which has alphabetic values to float
convert_alphabetic_to_float(global_super_store_df, 'Quantity')
convert_alphabetic_to_float(global_super_store_df, 'Profit')
```

## Order Date / Ship Date fixes

```
In [ ]: # Get the all the records out of that filter to fix
date_fix_condition = (~global_super_store_df['Ship Date'].isna()) & \
                    (~global_super_store_df['Order Date'].isna())

# Standardize Order Date column
global_super_store_df.loc[date_fix_condition, 'Order Date'] = global_super_store_df['O
```

```
# Standardize Ship Date column
global_super_store_df.loc[date_fix_condition, 'Ship Date'] = global_super_store_df['S
```

```
# Swap Order Date and Ship Date if necessary
for index, row in global_super_store_df.iterrows():
    order_date = pd.to_datetime(row['Order Date'], errors='coerce') # Coerce errors
    ship_date = pd.to_datetime(row['Ship Date'], errors='coerce') # Coerce errors

    if not pd.isna(order_date) and not pd.isna(ship_date) and order_date > ship_dat
        global_super_store_df.at[index, 'Order Date'], global_super_store_df.at[inc
```

```
C:\Users\nares\AppData\Local\Temp\ipykernel_21212\991193335.py:14: UserWarning: Pa
rsing dates in %d/%m/%Y format when dayfirst=False (the default) was specified. Pa
ss `dayfirst=True` or specify a format to silence this warning.
    return pd.to_datetime(date_str, dayfirst=False, errors='coerce').strftime('%Y-%m
-%d')
C:\Users\nares\AppData\Local\Temp\ipykernel_21212\991193335.py:14: UserWarning: Pa
rsing dates in %d/%m/%Y format when dayfirst=False (the default) was specified. Pa
ss `dayfirst=True` or specify a format to silence this warning.
    return pd.to_datetime(date_str, dayfirst=False, errors='coerce').strftime('%Y-%m
-%d')
C:\Users\nares\AppData\Local\Temp\ipykernel_21212\171948261.py:13: UserWarning: Pa
rsing dates in %d/%m/%Y format when dayfirst=False (the default) was specified. Pa
ss `dayfirst=True` or specify a format to silence this warning.
    order_date = pd.to_datetime(row['Order Date'], errors='coerce') # Coerce errors
to NaT for comparison
```

```
In [ ]: # Convert columns to datetime64 dtype
global_super_store_df['Order Date'] = pd.to_datetime(global_super_store_df['Order Date'])
global_super_store_df['Ship Date'] = pd.to_datetime(global_super_store_df['Ship Date'])

# Filling 01/01/1970 date as a NO_DATA_DATE date
global_super_store_df['Order Date'] = global_super_store_df['Order Date'].fillna('1970-01-01')
global_super_store_df['Ship Date'] = global_super_store_df['Ship Date'].fillna('1970-01-01')
```

## New columns to Analyze

After cleansing the data, the following columns have been added to perform EDA and visualizations.

Column Name	Data Type	Description
Loss	float64	Negative loss will be marked as loss
Shipment Days	float64	Calculate the days between Order Date and Shipped Date
Order Year	int32	Extracted year from Order Date to plot the sales/loss trends
Ship Year	int32	Extracted year from Ship Date to plot the sales/loss trends

```
In [ ]: # New column Shipment Days
global_super_store_df['Shipment Days'] = (global_super_store_df['Ship Date'] - global_super_store_df['Order Date']).dt.days

# New columns: Order Year and Ship Year. extract year from 'Order Date' and 'Ship Date'
global_super_store_df['Order Year'] = pd.DatetimeIndex(global_super_store_df['Order Date']).year
global_super_store_df['Ship Year'] = pd.DatetimeIndex(global_super_store_df['Ship Date']).year

# Fill NaN values with 0 (assuming missing years should be represented as 0)
global_super_store_df[['Order Year', 'Ship Year']] = global_super_store_df[['Order Year', 'Ship Year']].fillna(0)

# Convert year columns to integer dtype
global_super_store_df[['Order Year', 'Ship Year']] = global_super_store_df[['Order Year', 'Ship Year']].astype(int)

# New column 'Loss' where Loss is negative
global_super_store_df['Loss'] = global_super_store_df['Profit'].apply(lambda x: -x if x < 0 else x)
```

## Removing the filler values

Function to filter the Dataframe to apply the filler values.

```
In [ ]: def filter_out_filler_values_count(df, column, filler_value):
    """
    Filters out rows from the DataFrame where the value in the specified column matches the filler value.

    Parameters:
        df (DataFrame): The pandas DataFrame containing the data.
        column (str): The name of the column to filter on.
        filler_value: The value to match in the specified column for filtering.

    Returns:
        int: The count of filtered values.
    """
    filtered_count = len(df[df[column] == filler_value])
    return filtered_count
```

Function to drop the rows based on a filler value.

```
In [ ]: def remove.filler_values_on_df(df, key_column, key_value):
    """
        Filters out rows from the DataFrame where the value in the key column matches the key_value.

    Parameters:
        df (DataFrame): The pandas DataFrame containing the data.
        key_column (str): The name of the column to filter on.
        key_value: The value to match in the key column for filtering.

    Returns:
        None
    """
    # Filter out rows matching the condition
    df.drop(df[df[key_column] == key_value].index, inplace=True)
```

```
In [ ]: # Define the dictionary of column names and filler values
filter.filler_values_dict = {
    'Category': 'NO_CATEGORY',
    'Sub-Category': 'NO_SUB_CATEGORY',
    'Segment': 'NO_SEGMENT',
    'Ship Mode': 'NO_SHIP_MODE',
    'Customer Name': 'NO_CUSTOMER_NAME',
    'Postal Code': 'NO_POSTAL_CODE',
    'Region': 'NO_REGION',
    'Order Date': '1970/01/01',
    'Ship Date': '1970/01/01'
}
```

```
In [ ]: # Initialize a list to store the filtered count results
filter_count_results = []

# Iterate over the dictionary filter.filler_values_dict items and apply the filtering
for column, filler_value in filter.filler_values_dict.items():
    filtered_count = filter_out.filler_values_count(global_super_store_df, column, filler_value)
    filter_count_results.append([column, filler_value, filtered_count])

# Print the results as a table
print(tabulate(filter_count_results, headers=['Column', 'Filler Value', 'Filtered Count']))
```

Column	Filler Value	Filtered Count
Category	NO_CATEGORY	1
Sub-Category	NO_SUB_CATEGORY	4
Segment	NO_SEGMENT	4
Ship Mode	NO_SHIP_MODE	3
Customer Name	NO_CUSTOMER_NAME	3
Postal Code	NO_POSTAL_CODE	2
Region	NO_REGION	1
Order Date	1970/01/01	3
Ship Date	1970/01/01	3

The filler values are pretty small when compared to the dataset (1000 rows). These records are removed from the `global_super_store_df` dataframe.

```
In [ ]: # Iterate over the dictionary items and apply the filtering
for column, filler_value in filter.filler_values_dict.items():
    remove.filler_values_on_df(global_super_store_df, column, filler_value)
```

## Plotting histograms after cleansing data

```
In [ ]: # Histogram for 'Ship Mode'
orders_per_segment = global_super_store_df['Ship Mode'].value_counts(ascending=True)
plot_histogram(orders_per_segment, 'Ship Mode', 'Number of Orders', 'Number of Orders')

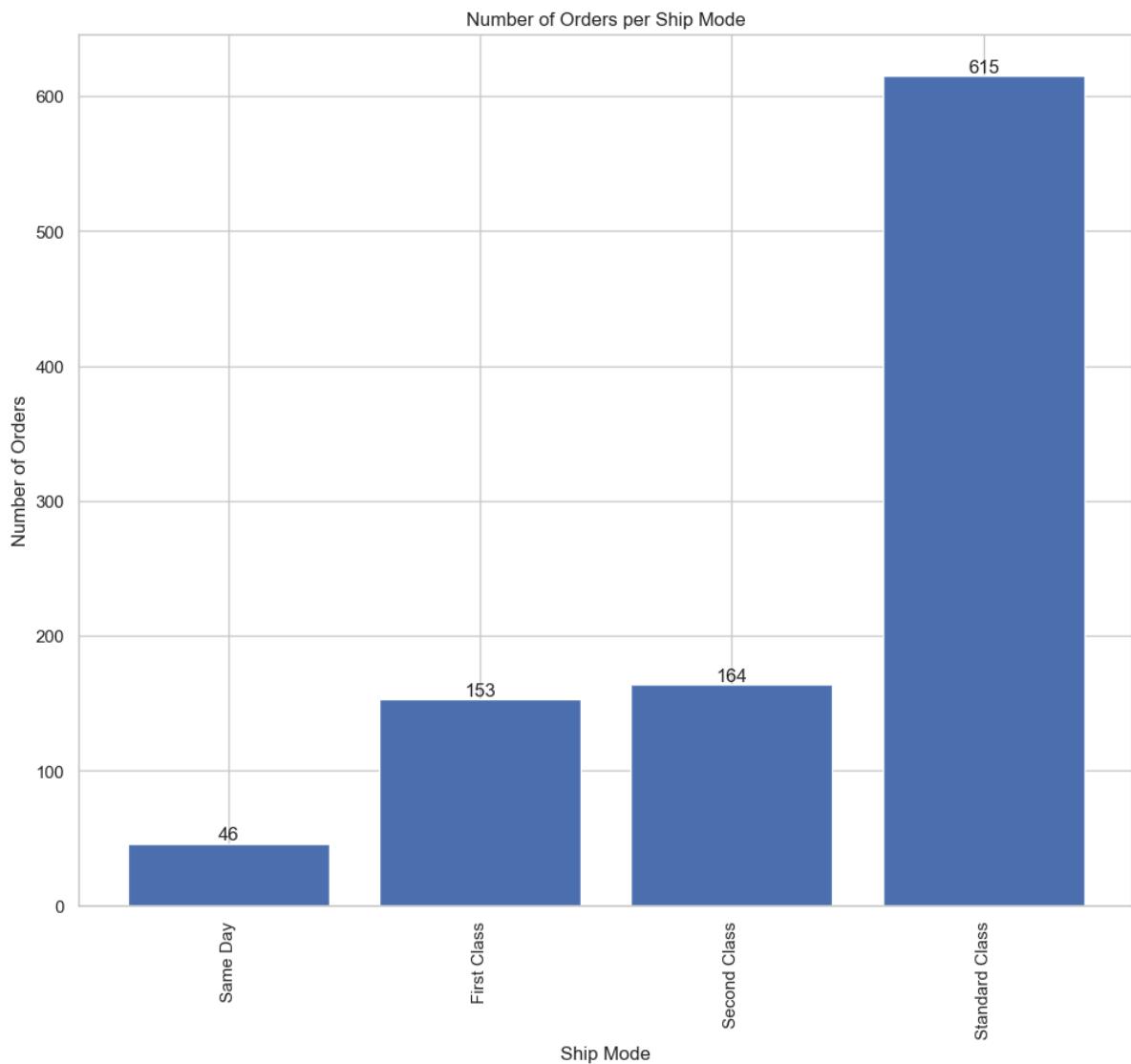
# Histogram for 'Segment'
orders_per_segment = global_super_store_df['Segment'].value_counts(ascending=True)
plot_histogram(orders_per_segment, 'Segment', 'Number of Orders', 'Number of Orders')

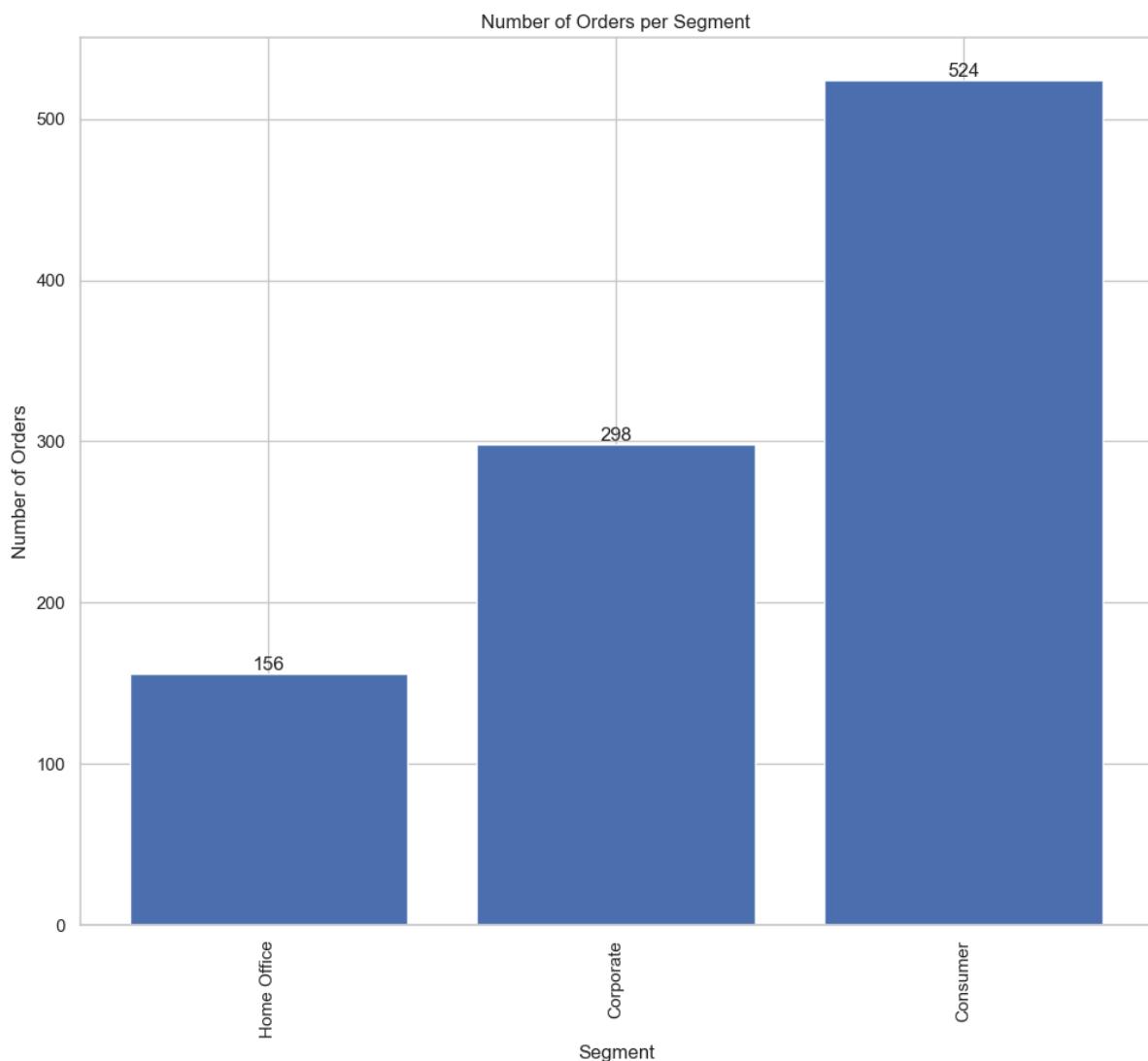
# Histogram for 'Category'
orders_per_category = global_super_store_df['Category'].value_counts(ascending=True)
plot_histogram(orders_per_category, 'Category', 'Number of Orders', 'Number of Orders')

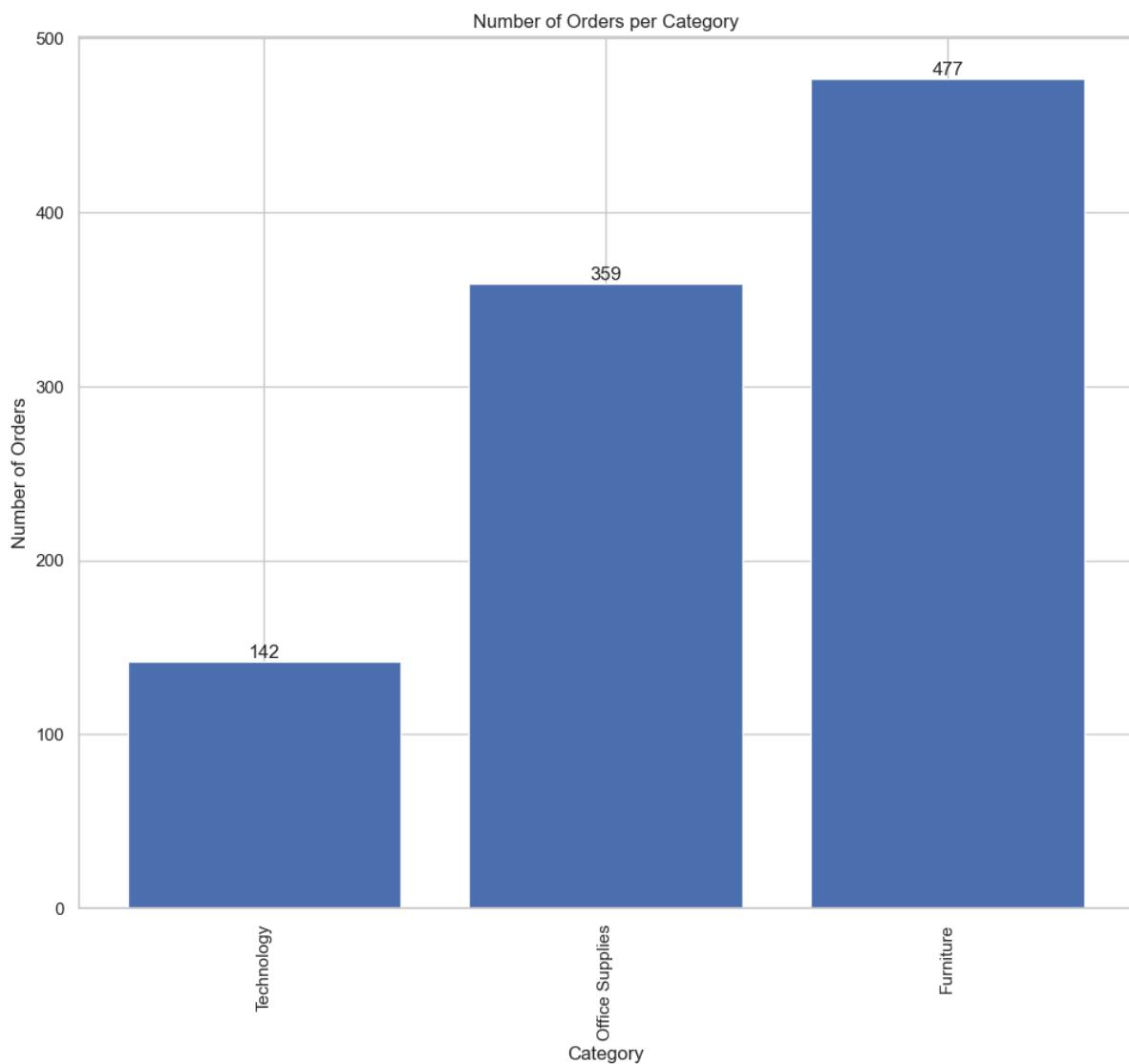
# Histogram for 'Sub-Category'
orders_per_sub_category = global_super_store_df['Sub-Category'].value_counts(ascending=True)
plot_histogram(orders_per_sub_category, 'Sub-Category', 'Number of Orders', 'Number of Orders')

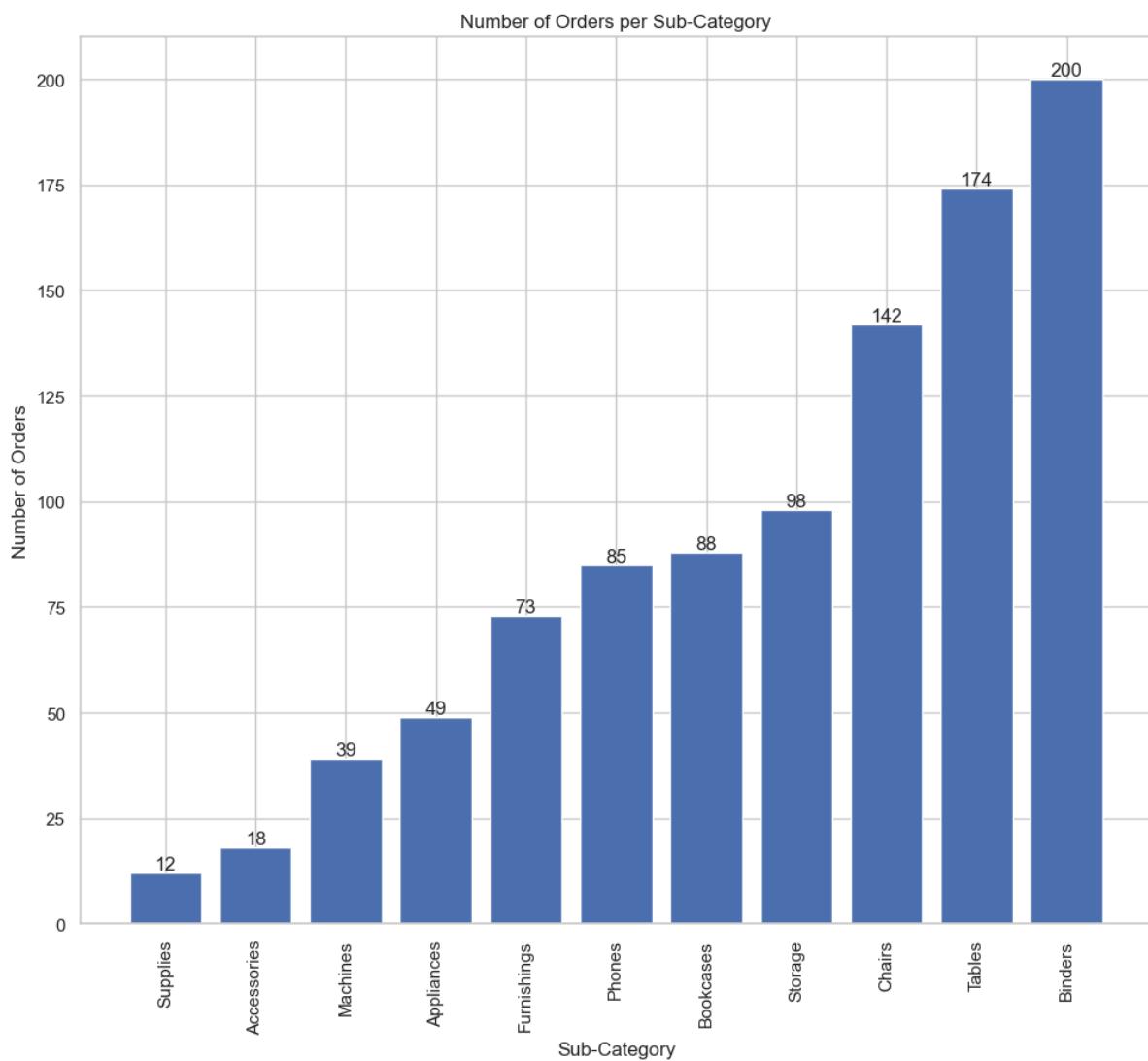
# Histogram for 'Region'
orders_per_sub_category = global_super_store_df['Region'].value_counts(ascending=True)
plot_histogram(orders_per_sub_category, 'Region', 'Number of Orders', 'Number of Orders')

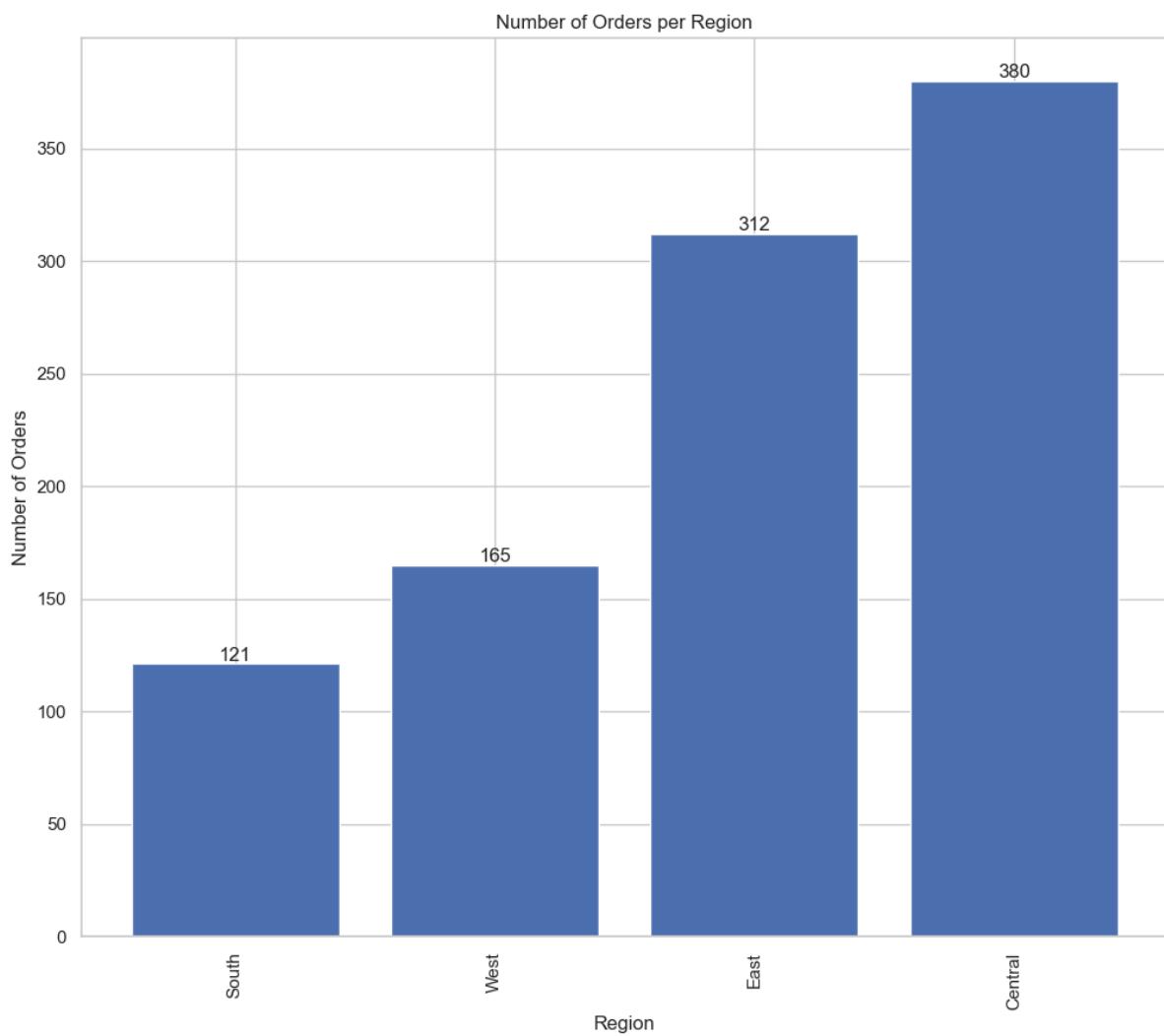
# Histogram for 'State'
orders_per_sub_category = global_super_store_df['State'].value_counts(ascending=True)
plot_histogram(orders_per_sub_category, 'State', 'Number of Orders', 'Number of Orders')
```

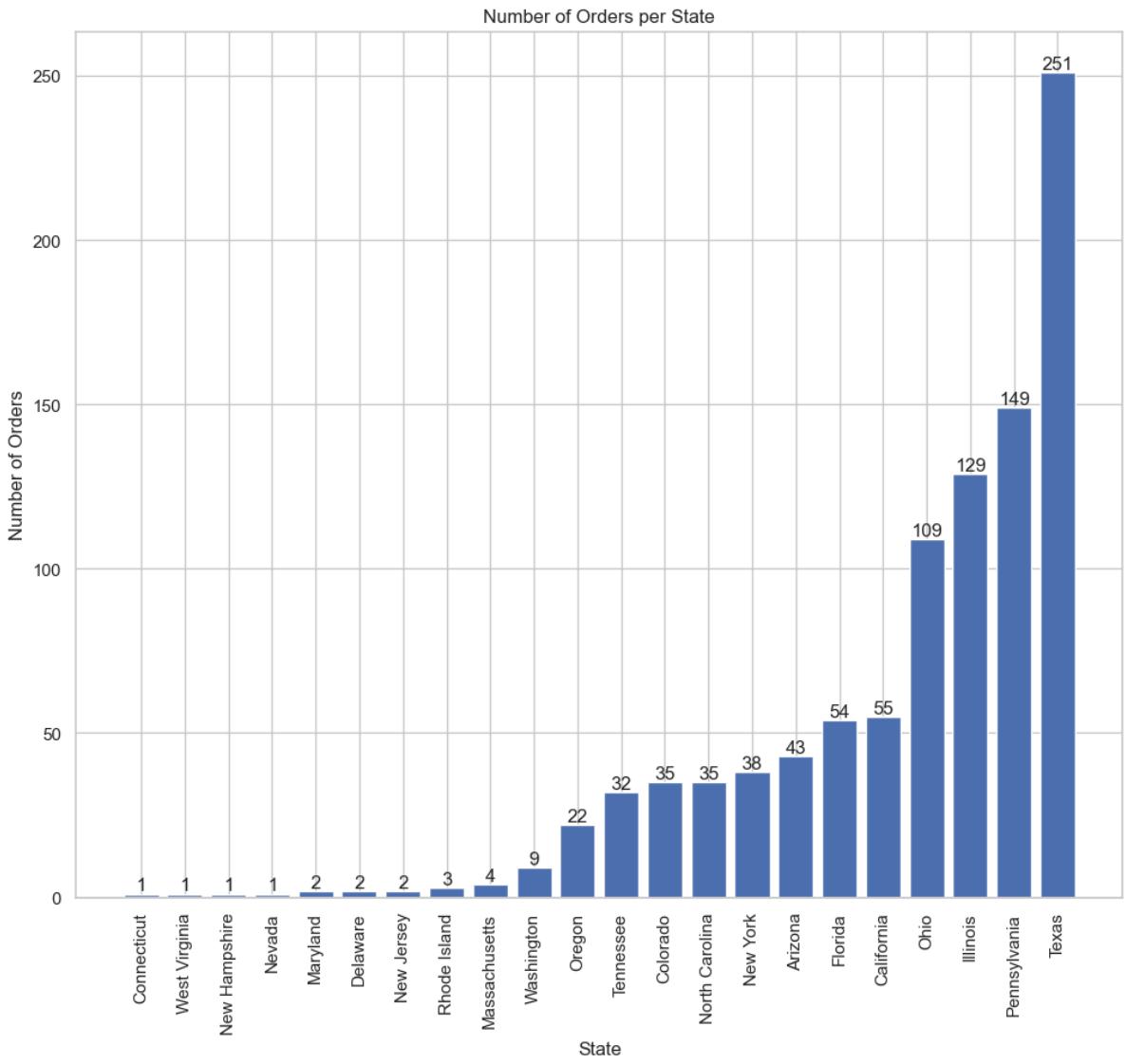












## Descriptive statistics

```
In [ ]: # Descriptive Statistics
descriptive_stats = global_super_store_df.describe().round(2)

# Generate descriptive statistics
print(tabulate(descriptive_stats, headers='keys', tablefmt='pretty', stralign ='right'))
```

			Order Date		Ship Date		Sales
	Quantity	Discount	Profit	Shipment Days	Order Year	Ship Year	Los
	s						s
count				978			978.0
978.0	978.0	978.0	978.0	978.0	978.0	978.0	978.
mean	2016-04-29 21:22:27.239263744	4.24	0.45	-142.19	2016-05-03 21:15:05.521472512	3.99	409.2
10	4.24	0.45	-142.19		2015.73	2015.74	142.1
min	2014-01-07 00:00:00	1.0	0.0	-6599.98	0.0	2014.0	2014.0
0	2014-01-07 00:00:00	0.0	0.0	-6599.98	0.0	2014.0	0.
25%	2015-04-26 00:00:00	3.0	0.2	-120.51	3.0	2015.0	72.88
5	2015-04-26 00:00:00	3.0	0.2	-120.51	3.0	2015.0	26.9
50%	2016-06-14 00:00:00	4.0	0.4	-52.08	4.0	2016.0	215.57
8	2016-06-14 00:00:00	4.0	0.4	-52.08	4.0	2016.0	52.0
75%	2017-05-12 00:00:00	6.0	0.7	-26.95	5.0	2017.0	462.83
11	2017-05-12 00:00:00	6.0	0.7	-26.95	5.0	2017.0	120.5
max	2017-12-29 00:00:00	14.0	0.8	0.0	7.0	2018.0	22638.48
8	2017-12-29 00:00:00	14.0	0.8	0.0	7.0	2018.0	6599.9
std		2.27	0.22	368.8	nan		939.86
8		2.27	0.22	368.8	nan	1.13	1.13
							368.

```
In [ ]: # Calculate the number of missing values for each column
missing_values = global_super_store_df.isnull().sum().to_frame(name='Missing Value')

# Display the missing values using tabulate
print(tabulate(missing_values, headers='keys', tablefmt='pretty', stralign='right',
```

	Missing Value Count
Order Date	0
Ship Date	0
Ship Mode	0
Customer Name	0
Segment	0
Country	0
City	0
State	0
Postal Code	0
Region	0
Category	0
Sub-Category	0
Sales	0
Quantity	0
Discount	0
Profit	0
Shipment Days	0
Order Year	0
Ship Year	0
Loss	0

## Grouping of data

### Helper Functions

Function to print total sales/loss by feature in a tabulate format.

```
In [ ]: def print_sales_loss_by_feature(df, sales_loss_feature, category_column):
    """
    print_sales_loss_by_feature, prints total sales/loss by feature in a tabulate format.

    Parameters:
    - df (DataFrame): The DataFrame containing the sales data.
    - sales_column (str): The name of the column containing sales data.
    - sales_category_column (str): The name of the column to group by.

    Returns:
    - None
    """
    # Group total sales by category
    sales_category = df.groupby(category_column)[sales_loss_feature].sum().round(4)

    # Convert the grouped DataFrame to a list of lists
    sales_category_list = sales_category.values.tolist()

    # Print the table using tabulate
    print(tabulate(sales_category_list, headers=[category_column, f'Total {sales_los

```

### Grouping Sales/Losses based on Categories

```
In [ ]: # group total sales by category from the highest sale.
print_sales_loss_by_feature(global_super_store_df.round(2), 'Sales', 'Category')
```

Category	Total Sales
Furniture	212508.31
Technology	108552.39
Office Supplies	79135.09

```
In [ ]: # group total Loss by category from the highest sale.
print_sales_loss_by_feature(global_super_store_df.round(2), 'Loss', 'Category')
```

Category	Total Loss
Furniture	55390.63
Office Supplies	47105.15
Technology	36561.47

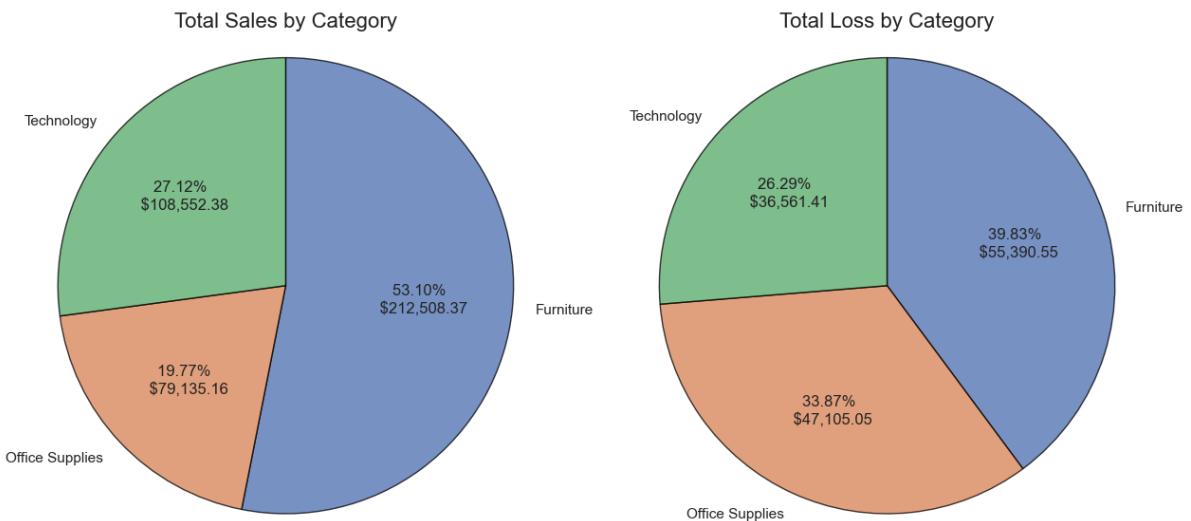
```
In [ ]: # group total sales by category, only considering sales
sales_category = global_super_store_df.groupby('Category')['Sales'].sum()

# group total loss by category, only considering losses
loss_category = global_super_store_df.groupby('Category')['Loss'].sum()

# figure size
plt.figure(figsize=(15,10));

# left total sales pie chart
plt.subplot(1,2,1); # 1 row, 2 columns, the 1st plot.
plt.pie(sales_category.values, labels=sales_category.index, startangle=90, counter-clockwise=True,
        autopct=lambda p:f'{p:.2f}% \n ${p * np.sum(sales_category.values) / 100 :.2f}');
        wedgeprops={'linewidth': 1, 'edgecolor':'black', 'alpha':0.75});
plt.axis('square');
plt.title('Total Sales by Category', fontdict={'fontsize':16});

# right total loss pie chart
plt.subplot(1,2,2); # 1 row, 2 columns, the 2nd plot
plt.pie(loss_category.values, labels=loss_category.index, startangle=90, counter-clockwise=True,
        autopct=lambda p:f'{p:.2f}% \n ${p * np.sum(loss_category.values) / 100 :.2f}');
        wedgeprops={'linewidth': 1, 'edgecolor':'black', 'alpha':0.75});
plt.axis('square');
plt.title('Total Loss by Category', fontdict={'fontsize':16});
```



Total Sales on Categories

1. Furniture at ~53%
2. Technology at ~27%
3. Office Supplies at ~20%

#### Total Losses on Categories

1. Furniture at ~39%
2. Office Supplies at ~34%
3. Technology at ~26%

## Grouping Sales/Losses based on Sub-Categories

```
In [ ]: # group total sales by sub-category from the highest sale.
print_sales_loss_by_feature(global_super_store_df.round(2), 'Sales', 'Sub-Category')
```

Sub-Category	Total Sales
Tables	92530.18
Machines	71632.22
Chairs	67562.95
Bookcases	42565.94
Storage	32034.07
Phones	31868.86
Binders	30556.3
Supplies	13741.19
Furnishings	9849.24
Accessories	5051.31
Appliances	2803.53

```
In [ ]: # group total loss by sub-category from the highest loss.
print_sales_loss_by_feature(global_super_store_df.round(2), 'Loss', 'Sub-Category')
```

Sub-Category	Total Loss
Binders	31432.34
Machines	29296.35
Tables	29176.51
Bookcases	11454.5
Chairs	8989.0
Appliances	7101.94
Phones	6676.66
Furnishings	5770.62
Storage	5659.41
Supplies	2911.46
Accessories	588.46

```
In [ ]: #Grouping the data on category and it's respective sub-categories. Calculating the
sales_per_cat_subcat = global_super_store_df.groupby(['Category', 'Sub-Category']),
sales_per_cat_subcat['Loss %'] = (sales_per_cat_subcat['Loss'] / sales_per_cat_subcat['Sales']) * 100
```

#Sorting the dataframe based on Loss margin

```
sales_per_cat_subcat = sales_per_cat_subcat.sort_values(by=['Loss %'], ascending=False)
print(tabulate(sales_per_cat_subcat.round(2), headers='keys', tablefmt='pretty', str_escapes=True))
```

Category	Sub-Category	Sales	Loss	Loss %
Office Supplies	Appliances	2803.57	7101.9	253.32
Office Supplies	Binders	30556.33	31432.29	102.87
Furniture	Furnishings	9849.27	5770.65	58.59
Technology	Machines	71632.26	29296.32	40.9
Furniture	Tables	92530.2	29176.48	31.53
Furniture	Bookcases	42565.96	11454.45	26.91
Office Supplies	Supplies	13741.18	2911.46	21.19
Technology	Phones	31868.83	6676.65	20.95
Office Supplies	Storage	32034.07	5659.4	17.67
Furniture	Chairs	67562.94	8988.98	13.3
Technology	Accessories	5051.29	588.45	11.65

## Grouping Sales/Losses based on Region

```
In [ ]: # group total sales by regions from the highest sale.
sales_by_segment = global_super_store_df.groupby(['Region'], as_index=False)[['Sales', 'Loss']]
sales_by_segment['Sales %'] = (sales_by_segment['Sales'] / global_super_store_df['Sales'].sum())
sales_by_segment['Loss %'] = (sales_by_segment['Loss'] / sales_by_segment['Sales'])

# Convert numerical values to strings with commas for thousands separators and round to 2 decimal places
for numerical_column in sales_by_segment.select_dtypes(include=['int64', 'float64']):
    sales_by_segment[numerical_column] = sales_by_segment[numerical_column].apply(lambda x: f'{x:,}') if isinstance(x, (int, float)) else x

# List of total sales by states
print(tabulate(sales_by_segment.round(2), headers='keys', tablefmt='pretty', stralign='right'))
```

Region	Sales	Loss	Sales %	Loss %
East	141,134.00	46,631.09	35.27	33.04
Central	118,801.87	49,218.54	29.69	41.43
South	78,709.87	21,888.28	19.67	27.81
West	61,550.17	21,319.12	15.38	34.64

## Grouping Sales/Losses based on States

```
In [ ]: # group total sales by states from the highest sale.
sales_by_states = global_super_store_df.groupby(['State'], as_index=False)[['Sales', 'Loss']]
sales_by_states['Sales %'] = (sales_by_states['Sales'] / global_super_store_df['Sales'].sum())
sales_by_states['Loss %'] = (sales_by_states['Loss'] / sales_by_states['Sales']) * 100

# Convert numerical values to strings with commas for thousands separators and round to 2 decimal places
for numerical_column in sales_by_states.select_dtypes(include=['int64', 'float64']):
    sales_by_states[numerical_column] = sales_by_states[numerical_column].apply(lambda x: f'{x:,}') if isinstance(x, (int, float)) else x

# List of total sales by states
print(tabulate(sales_by_states.round(2), headers='keys', tablefmt='pretty', stralign='right'))
```

State	Sales	Loss	Sales %	Loss %
Texas	89,052.53	31,205.35	22.25	35.04
Pennsylvania	67,815.33	19,998.49	16.95	29.49
Ohio	42,400.36	20,619.95	10.59	48.63
Florida	40,932.20	7,925.21	10.23	19.36
Illinois	29,749.34	18,013.19	7.43	60.55
California	27,976.97	3,326.36	6.99	11.89
New York	23,198.86	4,953.83	5.80	21.35
North Carolina	21,728.61	8,132.08	5.43	37.43
Tennessee	16,049.06	5,830.98	4.01	36.33
Arizona	12,751.76	6,351.07	3.19	49.81
Colorado	11,722.51	8,691.70	2.93	74.15
Oregon	5,521.03	2,514.96	1.38	45.55
Washington	2,903.54	325.45	0.73	11.21
Massachusetts	2,575.26	436.25	0.64	16.94
Rhode Island	1,517.50	216.41	0.38	14.26
New Hampshire	1,053.16	105.32	0.26	10.00
Maryland	789.80	71.12	0.20	9.00
Nevada	674.35	109.58	0.17	16.25
West Virginia	673.34	76.95	0.17	11.43
Delaware	510.28	85.90	0.13	16.83
New Jersey	418.29	51.29	0.10	12.26
Connecticut	181.80	15.58	0.05	8.57

**State made the highest sales %:** Texas \ **State made the lowest sales %:** Connecticut

## Grouping Sales/Losses based on Segment

```
In [ ]: # group total sales by segments from the highest sale.
sales_by_segment = global_super_store_df.groupby(['Segment'], as_index=False)[['Sales', 'Loss']]
sales_by_segment['Sales %'] = (sales_by_segment['Sales'] / global_super_store_df['Sales'].sum())
sales_by_segment['Loss %'] = (sales_by_segment['Loss'] / sales_by_segment['Sales'])

# Convert numerical values to strings with commas for thousands separators and round
for numerical_column in sales_by_segment.select_dtypes(include=['int64', 'float64']):
    sales_by_segment[numerical_column] = sales_by_segment[numerical_column].apply(lambda x: f'{x:,}').round(2)

# List of total sales by states
print(tabulate(sales_by_segment.round(2), headers='keys', tablefmt='pretty', stralign='center'))
```

Segment	Sales	Loss	Sales %	Loss %
Consumer	205,634.12	72,978.65	51.38	35.49
Corporate	118,826.11	42,474.37	29.69	35.74
Home Office	75,735.68	23,604.01	18.92	31.17

**Consumer** Segment has 50% of Sales share, followed by **Corporate** and **Home Office**.

## Grouping Sales/Losses based on Shipping Mode

```
In [ ]: # group total sales by segments from the highest sale.
sales_by_ship_mode = global_super_store_df.groupby(['Ship Mode'], as_index=False)[['Sales', 'Loss']]
sales_by_ship_mode['Sales %'] = (sales_by_ship_mode['Sales'] / global_super_store_df['Sales'].sum())
sales_by_ship_mode['Loss %'] = (sales_by_ship_mode['Loss'] / sales_by_ship_mode['Sales'])

# Convert numerical values to strings with commas for thousands separators and round
for numerical_column in sales_by_segment.select_dtypes(include=['int64', 'float64']):
```

```

sales_by_ship_mode[numerical_column] = sales_by_ship_mode[numerical_column].apply(
    lambda x: x if x != 0 else np.nan)

# List of total sales by states
print(tabulate(sales_by_ship_mode.round(2), headers='keys', tablefmt='pretty', str
+-----+-----+-----+-----+
| Ship Mode | Sales | Loss | Sales % | Loss % |
+-----+-----+-----+-----+
| Standard Class | 257698.95 | 88887.64 | 64.39 | 34.49 |
| Second Class | 67310.74 | 20253.2 | 16.82 | 30.09 |
| First Class | 49908.08 | 21371.32 | 12.47 | 42.82 |
| Same Day | 25278.13 | 8544.86 | 6.32 | 33.8 |
+-----+-----+-----+-----+

```

**Standard** shipping method is preferred as the sales percentage is nearly 64%, followed by **Second Class** and **First Class**. **Same Day** is not an economical option, so only preferred by 6% of the orders.

```

In [ ]: # Filter the filler date 01/01/1970 - the default NO_DATE date
global_super_store_df_date_filtered = global_super_store_df[(global_super_store_df['Order Date'] != 'NO_DATE') & (global_super_store_df['Order Date'] != '1970-01-01')]
grouped_data = global_super_store_df_date_filtered.round(3).reset_index().values.to
print(tabulate(grouped_data, headers=['Ship Mode', 'Average Shipped in Days'], tab
+-----+-----+
| Ship Mode | Average Shipped in Days |
+-----+-----+
| First Class | 2.163 |
| Same Day | 0.022 |
| Second Class | 3.262 |
| Standard Class | 4.943 |
+-----+-----+

```

- **Standard Class** shipping method takes ~5 days to ship from the order date.
- **First Class** and **Second Class** are between 2-3 days.
- **Same Day** is as promised, the orders are shipped with a day.

In conclusion, **Second Class** ship mode is the best value for money, being economical than **First Class** and faster than **Standard Class**.

## Univariate analysis and visualisation

### Histograms

Function to plot the histogram with density plot for a specified column in the DataFrame

```

In [ ]: def plot_histogram_density(df, column, xlim=None):
    """
    Plot histogram with density plot for a specified column in the DataFrame.

    Parameters:
    df (DataFrame): Input DataFrame.
    column (str): Name of the column to plot.
    xlim (tuple): Tuple containing the lower and upper limits of the x-axis.

    Returns:
    None
    """
    # Create subplots for each column

```

```

fig, ax = plt.subplots(figsize=(12, 10))

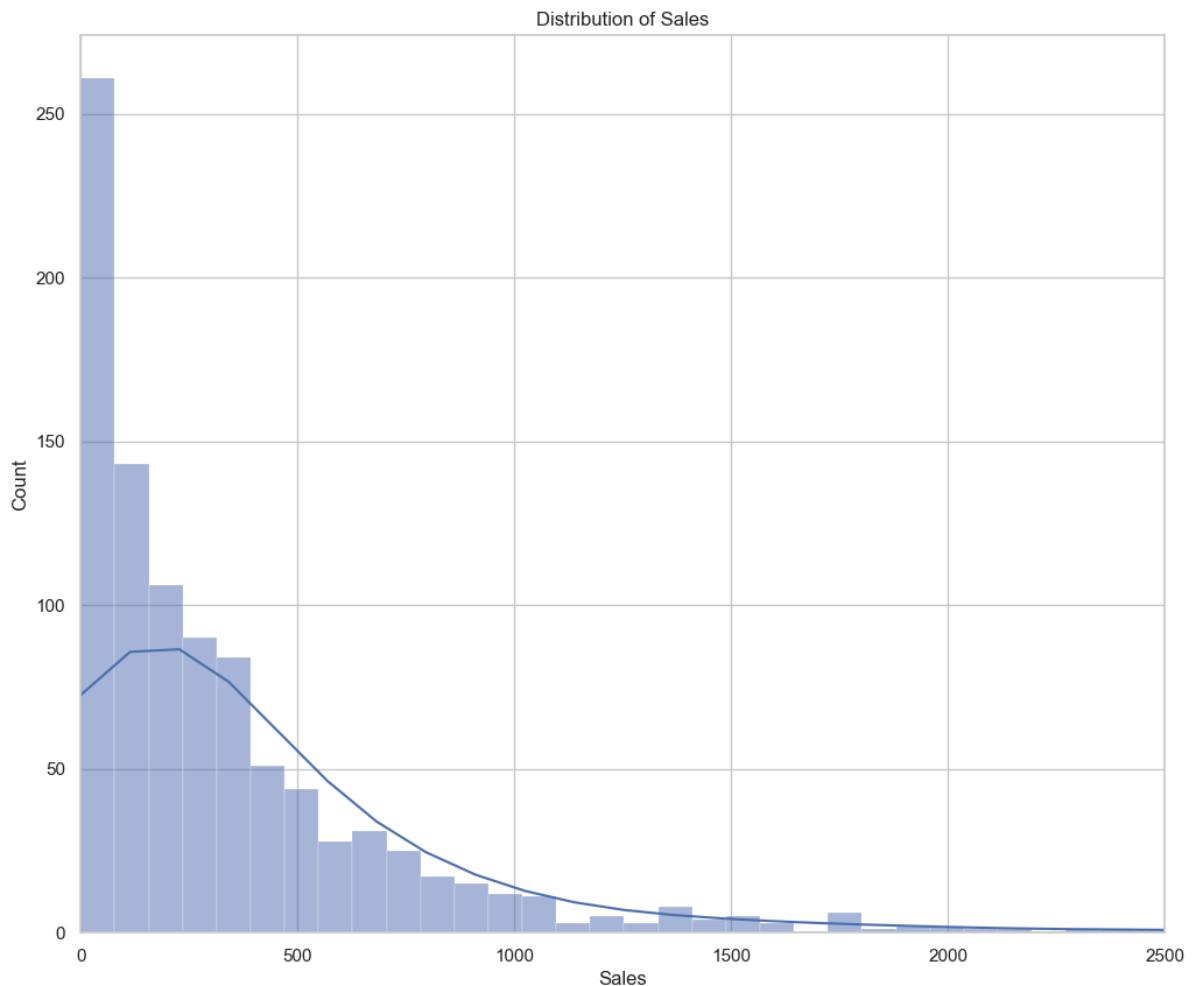
# Plot histogram with density plot
sns.histplot(df[column], kde=True, ax=ax)
ax.set_title(f'Distribution of {column}')

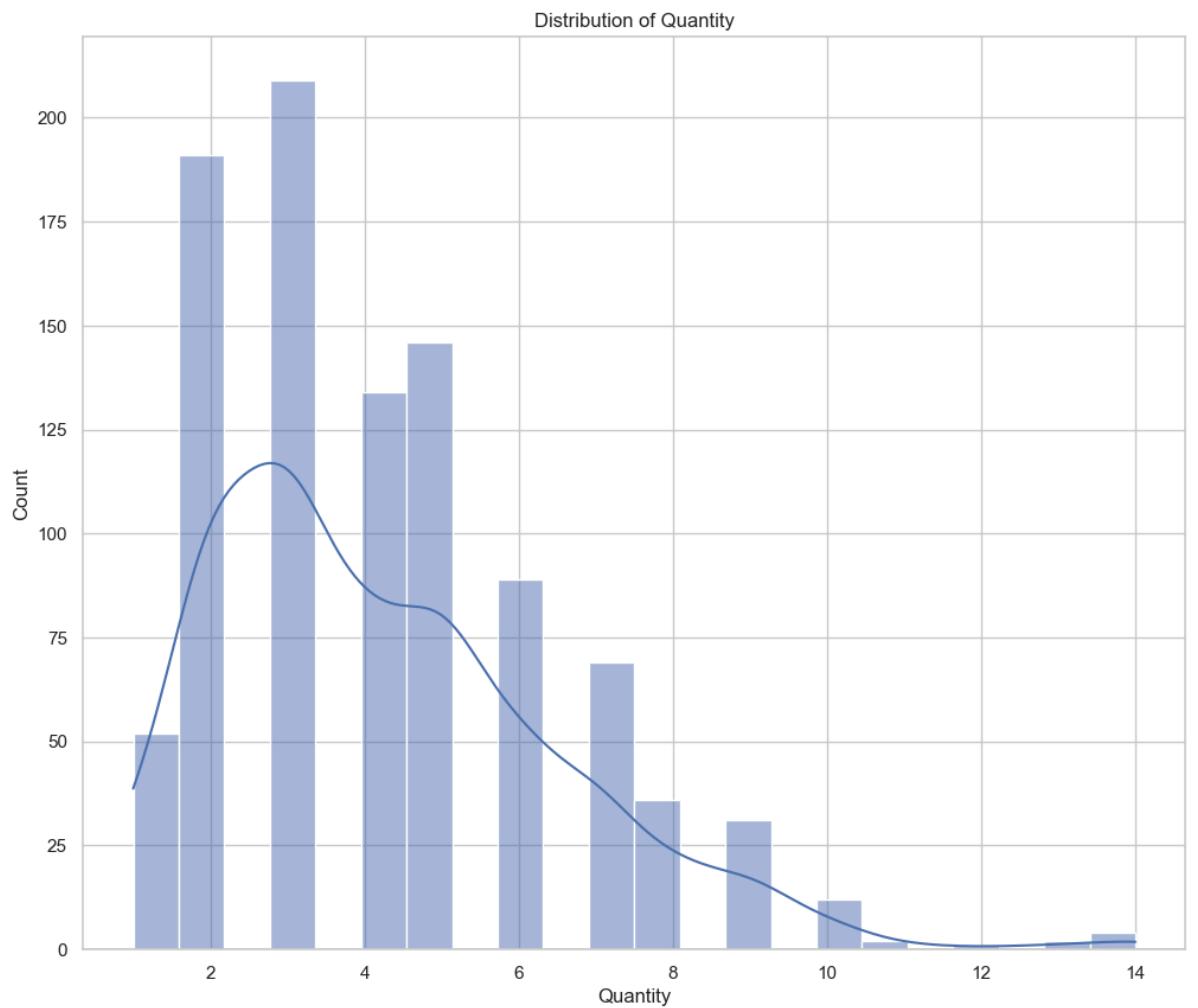
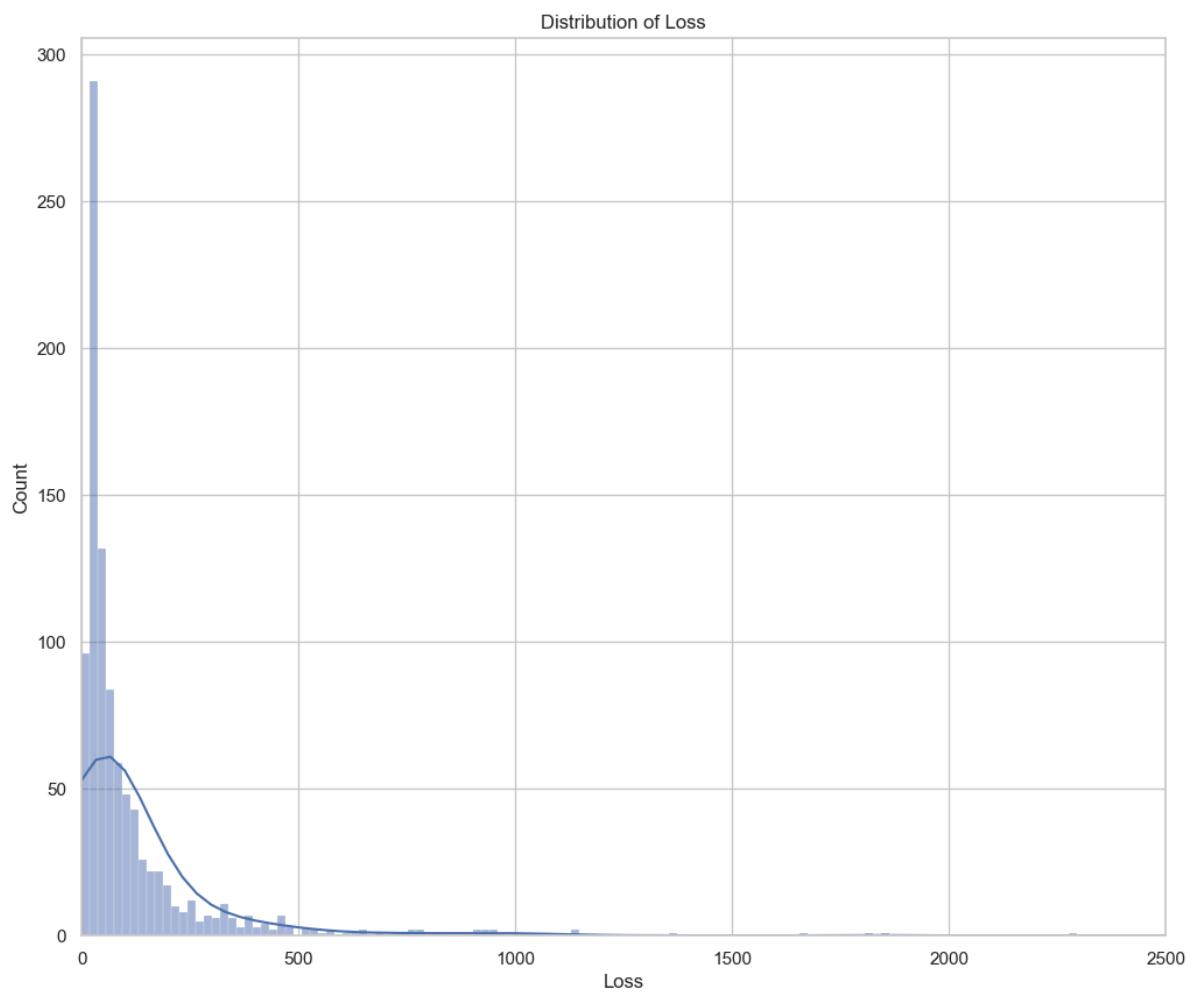
# Set x-axis limits if specified
if xlim:
    plt.xlim(xlim)

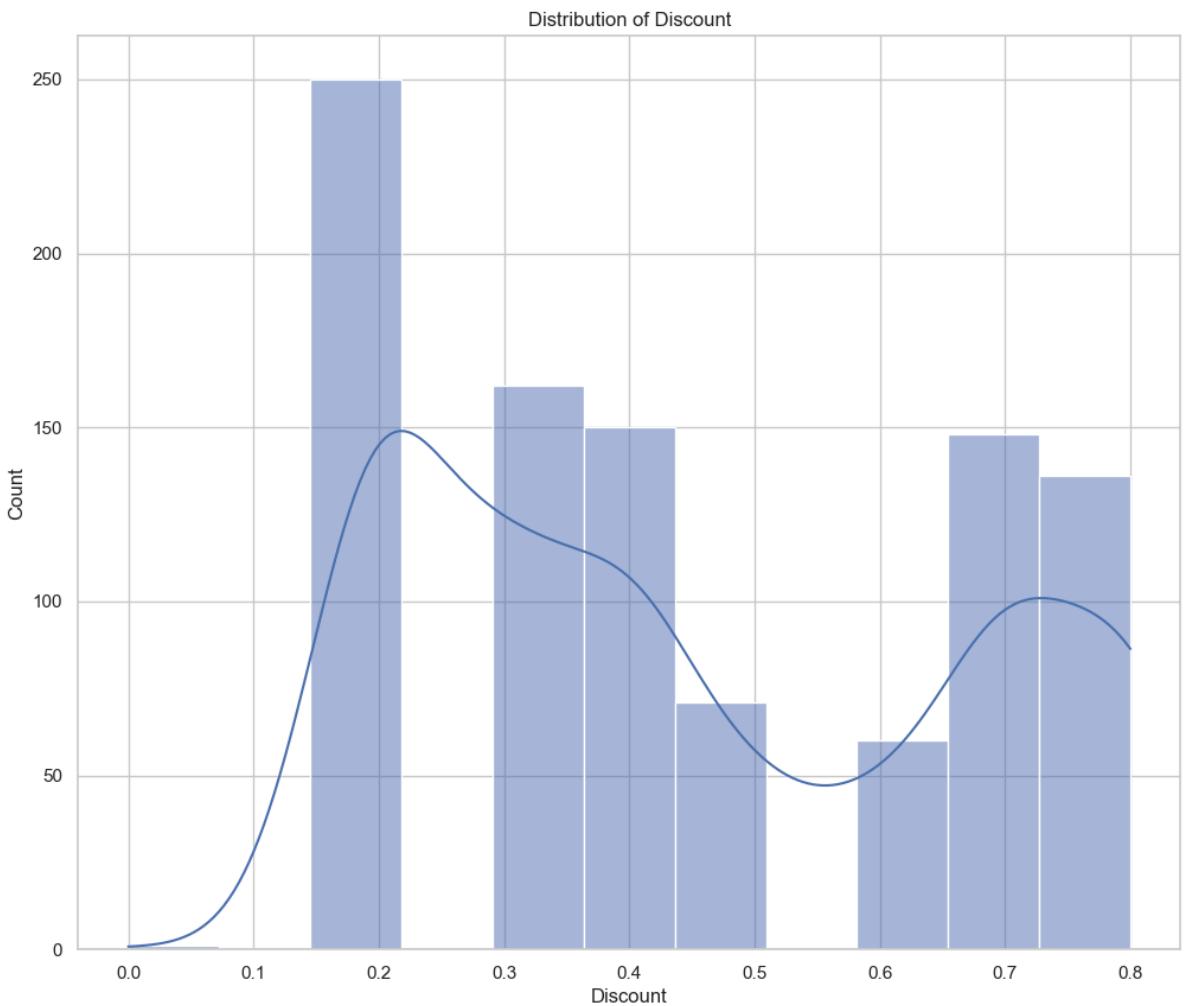
# Show the plot
plt.show()

```

In [ ]: `plot_histogram_density(global_super_store_df, 'Sales', xlim=(0, 2500))  
plot_histogram_density(global_super_store_df, 'Loss', xlim=(0, 2500))  
plot_histogram_density(global_super_store_df, 'Quantity')  
plot_histogram_density(global_super_store_df, 'Discount')`







## Probability Density Function / Cumulative Density Function

```
In [ ]: # Normalize all columns in the DataFrame
numerical_columns = global_super_store_df.select_dtypes(include=['int64', 'float64'])
normalized_df = (global_super_store_df[numerical_columns] - global_super_store_df[numerical_columns].mean()) / global_super_store_df[numerical_columns].std()

# Create subplots for each column
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Plot PDF and CDF for Sales
sns.kdeplot(normalized_df['Sales'], cumulative=False, ax=axes[0, 0], color='blue')
sns.kdeplot(normalized_df['Sales'], cumulative=True, ax=axes[0, 0], color='red', label='CDF')
axes[0, 0].set_title('PDF and CDF for Sales')
axes[0, 0].axvline(x=0, linestyle='--', color='gray') # Add a dotted line at x = 0
axes[0, 0].set_xlim(-3, 3) # Set custom x-axis limits

# Plot PDF and CDF for Quantity
sns.kdeplot(normalized_df['Quantity'], cumulative=False, ax=axes[0, 1], color='blue')
sns.kdeplot(normalized_df['Quantity'], cumulative=True, ax=axes[0, 1], color='red', label='CDF')
axes[0, 1].set_title('PDF and CDF for Quantity')
axes[0, 1].axvline(x=0, linestyle='--', color='gray') # Add a dotted line at x = 0
axes[0, 1].set_xlim(-3, 3) # Set custom x-axis limits

# Plot PDF and CDF for Discount
sns.kdeplot(normalized_df['Discount'], cumulative=False, ax=axes[1, 0], color='blue')
sns.kdeplot(normalized_df['Discount'], cumulative=True, ax=axes[1, 0], color='red', label='CDF')
axes[1, 0].set_title('PDF and CDF for Discount')
axes[1, 0].axvline(x=0, linestyle='--', color='gray') # Add a dotted line at x = 0
axes[1, 0].set_xlim(-3, 3) # Set custom x-axis limits

# Plot PDF and CDF for Loss
```

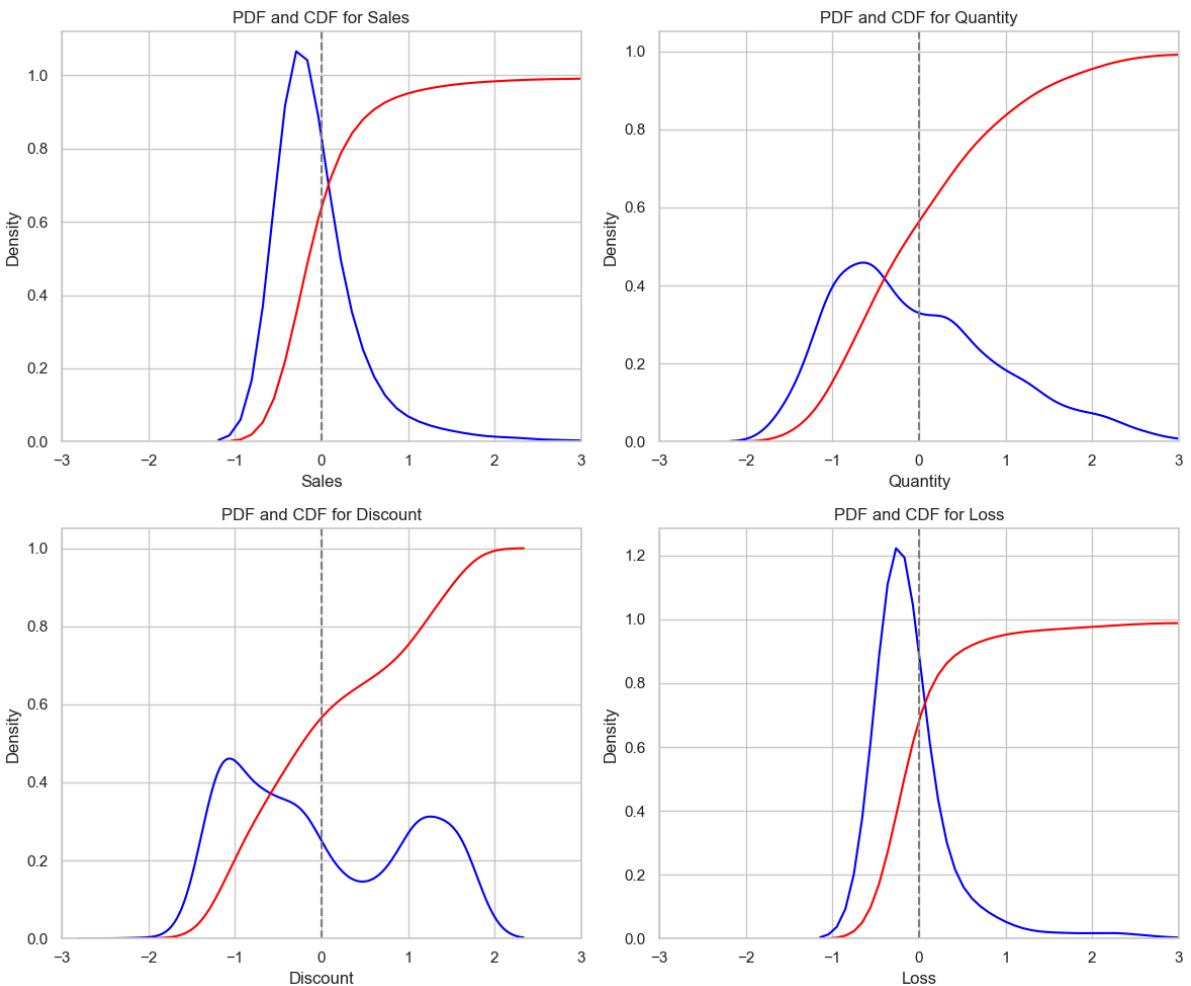
```

sns.kdeplot(normalized_df['Loss'], cumulative=False, ax=axes[1, 1], color='blue', )
sns.kdeplot(normalized_df['Loss'], cumulative=True, ax=axes[1, 1], color='red', label='Loss')
axes[1, 1].set_title('PDF and CDF for Loss')
axes[1, 1].axvline(x=0, linestyle='--', color='gray') # Add a dotted Line at x = 0
axes[1, 1].set_xlim(-3, 3) # Set custom x-axis limits

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()

```



```

In [ ]: # Calculate skewness for each column
skewness = normalized_df[['Sales', 'Loss', 'Quantity', 'Discount']].skew()

print(tabulate(skewness.to_frame().round(4), headers=['Numercial Features', 'Skewness']))

```

Numercial Features	Skewness
Sales	15.2478
Loss	9.6722
Quantity	1.0036
Discount	0.331

## Trends

Function to generate a trend plot for a specified column in a DataFrame, showing the total value of the column over the years.

```
In [ ]: def plot_trend(df, numerical_column, display_dollar_sign=False):
    """
        Generates a trend plot for a specified column in a DataFrame, showing the total
    Parameters:
    - df (DataFrame): The DataFrame containing the data.
    - numerical_column (str): The name of the column for which the trend plot is to
    - display_dollar_sign (bool): Whether to display y-axis values with a dollar si
    Returns:
    None
    """

    # Group by 'Order Year' and calculate the sum of the specified column
    trend_data = df.groupby(['Order Year'])[numerical_column].sum()

    # Create the plot
    fig, ax = plt.subplots(figsize=(15,10))

    # Plot the bar plot
    bars = ax.bar(x=trend_data.index, height=trend_data.values, color='teal')

    # Annotate each bar with its value
    for bar in bars:
        height = bar.get_height()
        if display_dollar_sign:
            ax.annotate('${:.2f}'.format(height),
                        xy=(bar.get_x() + bar.get_width() / 2, height),
                        xytext=(0, 3), # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom')
        else:
            ax.annotate('{:.2f}'.format(height),
                        xy=(bar.get_x() + bar.get_width() / 2, height),
                        xytext=(0, 3), # 3 points vertical offset
                        textcoords="offset points",
                        ha='center', va='bottom')

    # Plot the line plot
    ax.plot(trend_data.index, trend_data.values, color='orange')

    # Turn off scientific notation for y-axis
    ax.yaxis.get_major_formatter().set_scientific(False)

    # Set title and labels
    plt.title(f'{numerical_column} trend over the years', fontsize=18)
    plt.ylabel(f'Total {numerical_column}', fontsize=14)
    plt.xlabel('Order Year', fontsize=14)

    # Convert 'Order Year' column to a list of integers
    years = trend_data.index.tolist()

    # Set x-ticks to only include specific years
    plt.xticks(years, rotation=45)

    # Add gridlines
    plt.grid(axis='x')

    # Show the plot
    plt.show()
```

```
In [ ]: # Call the function for sales
plot_trend(global_super_store_df, 'Sales', display_dollar_sign=True)
```

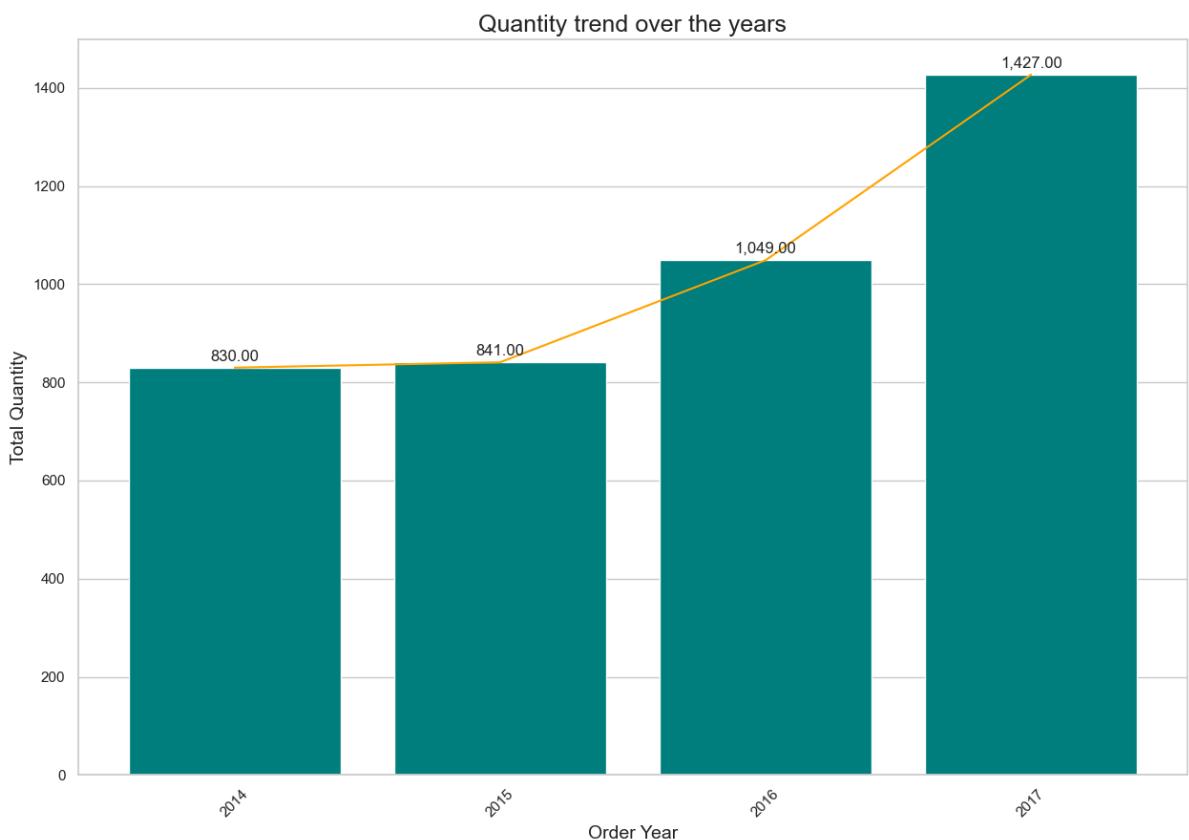
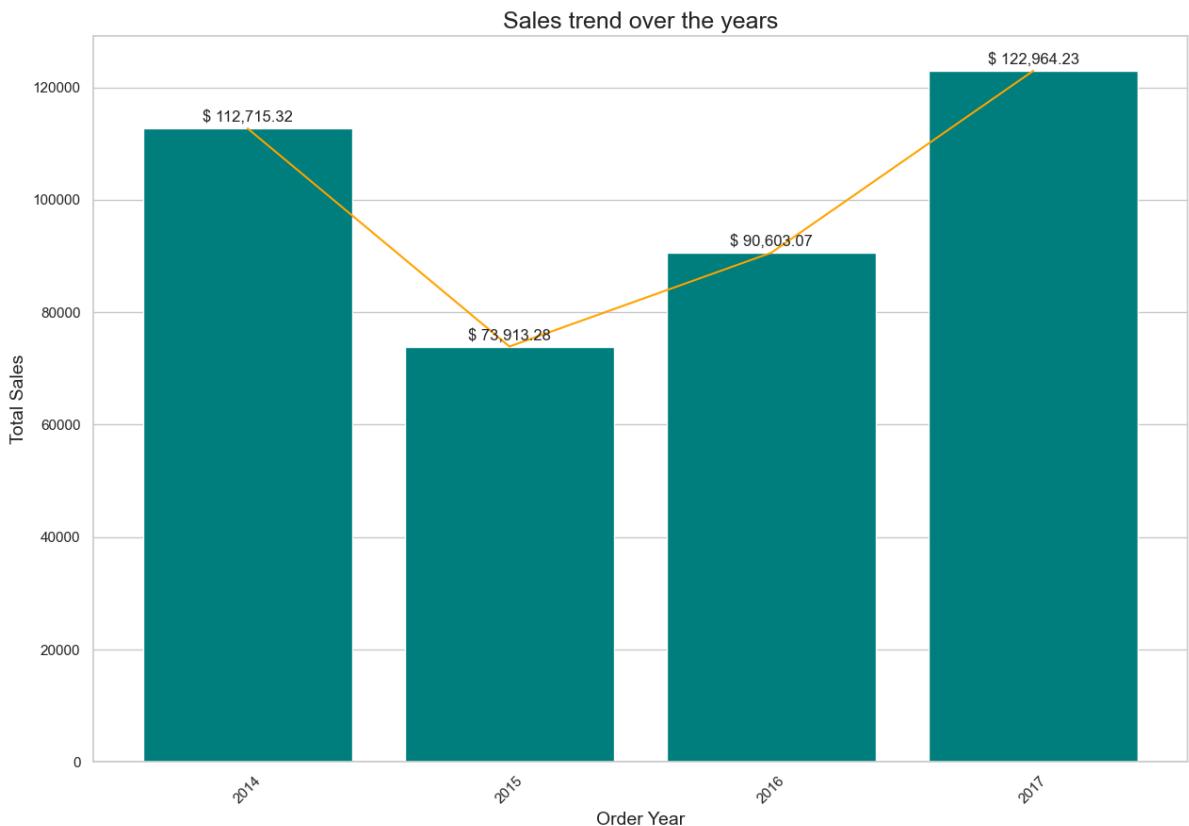
```

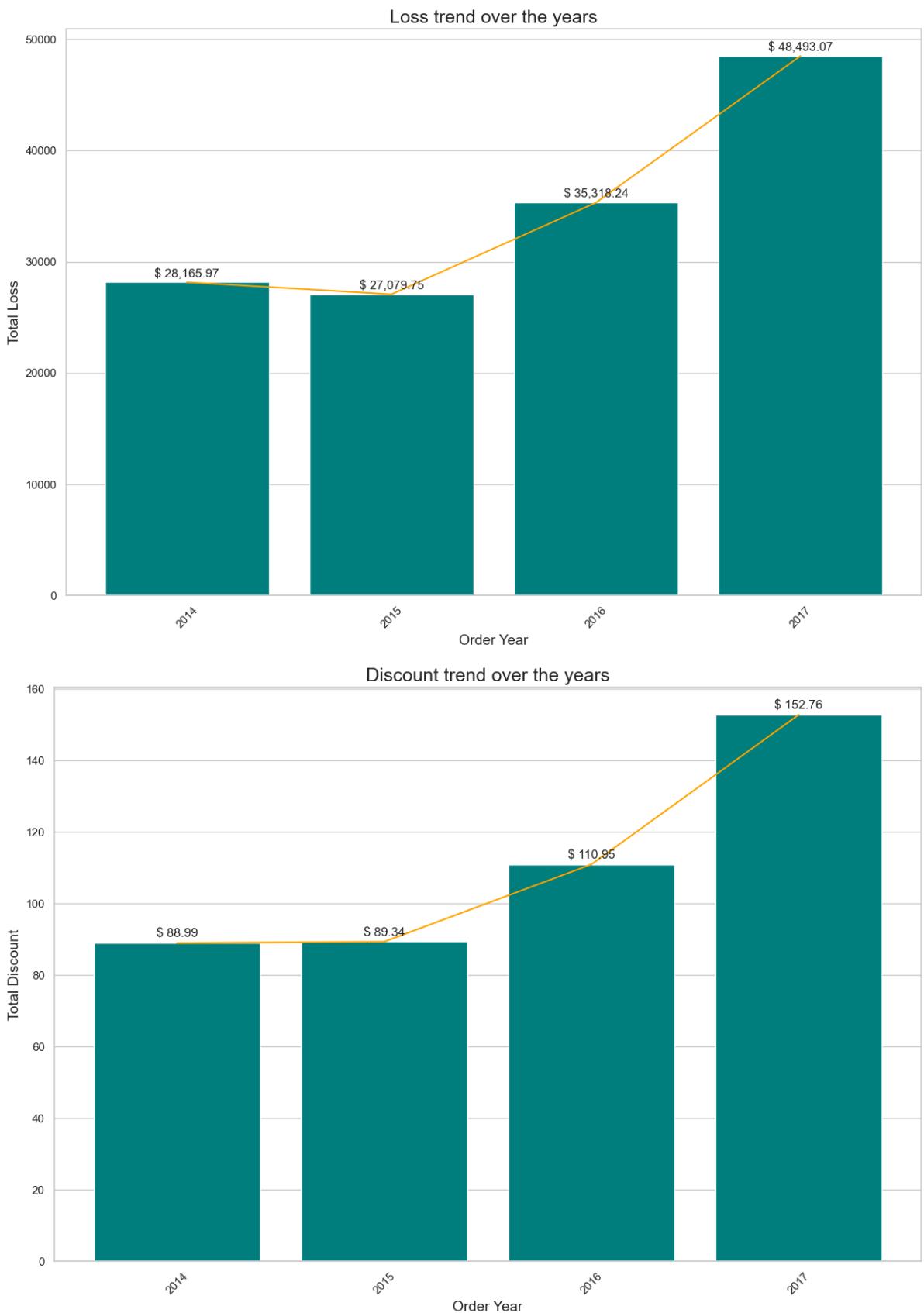
# Call the function for quantities
plot_trend(global_super_store_df, 'Quantity', display_dollar_sign=False)

# Call the function for loss
plot_trend(global_super_store_df, 'Loss', display_dollar_sign=True)

# Call the function for discounts
plot_trend(global_super_store_df, 'Discount', display_dollar_sign=True)

```





## Outlier Treatment

### Interquartile Range (IQR) Method

Function to calculate IQR and plot a boxplot.

In [ ]:

```
def detect_plot_outliers_iqr(df, numerical_feature):
    """
    Plots a boxplot for the specified numerical feature and identifies outliers bas
```

```

Parameters:
    df (DataFrame): The pandas DataFrame containing the data.
    numerical_feature (str): The name of the numerical feature to analyze.

Returns:
    None
"""

q1 = df[numerical_feature].quantile(0.25)
q3 = df[numerical_feature].quantile(0.75)

iqr = q3 - q1
lower_limit = q1 - 1.5 * iqr
upper_limit = q3 + 1.5 * iqr

outliers_df = df[(df[numerical_feature] < lower_limit) | (df[numerical_feature] >
print(f"Outlier numerical feature: {numerical_feature}, Outlier Count: {outlier

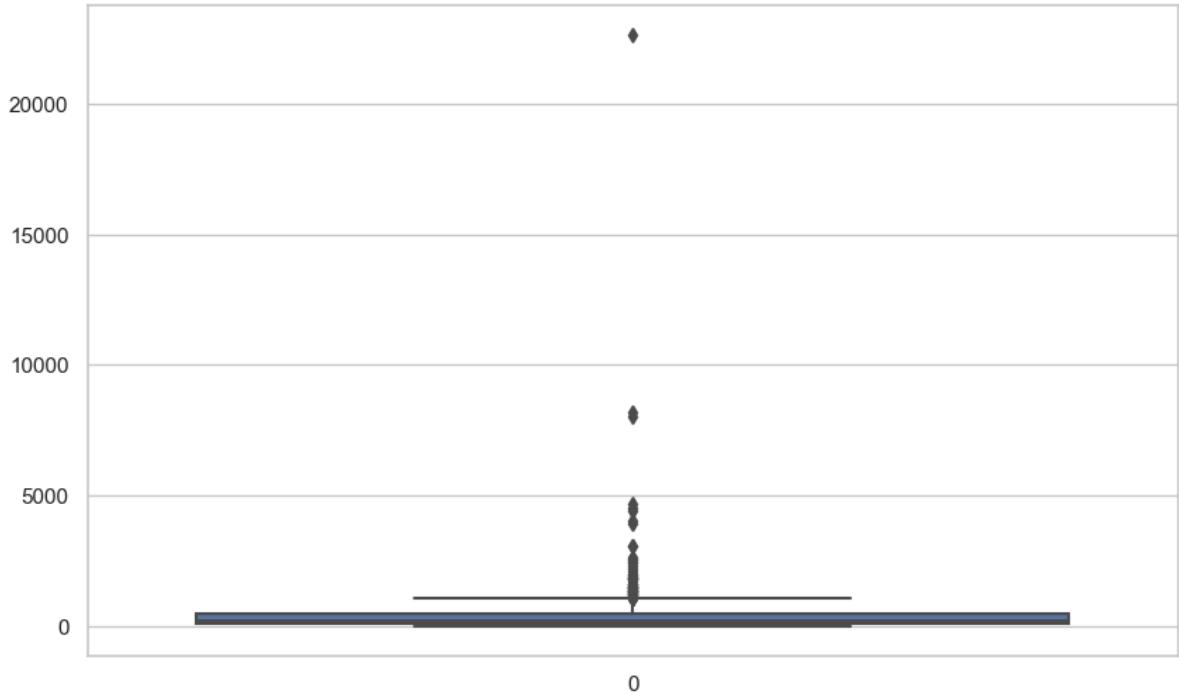
numerical_column = df[numerical_feature]
plt.figure(figsize=(10, 6))
sns.boxplot(data=numerical_column)
plt.title(f'Boxplot of Outliers for {numerical_feature}')
plt.show()

```

```
In [ ]: detect_plot_outliers_iqr(global_super_store_df, 'Sales')
detect_plot_outliers_iqr(global_super_store_df, 'Quantity')
detect_plot_outliers_iqr(global_super_store_df, 'Discount')
detect_plot_outliers_iqr(global_super_store_df, 'Loss')
detect_plot_outliers_iqr(global_super_store_df, 'Shipment Days')
```

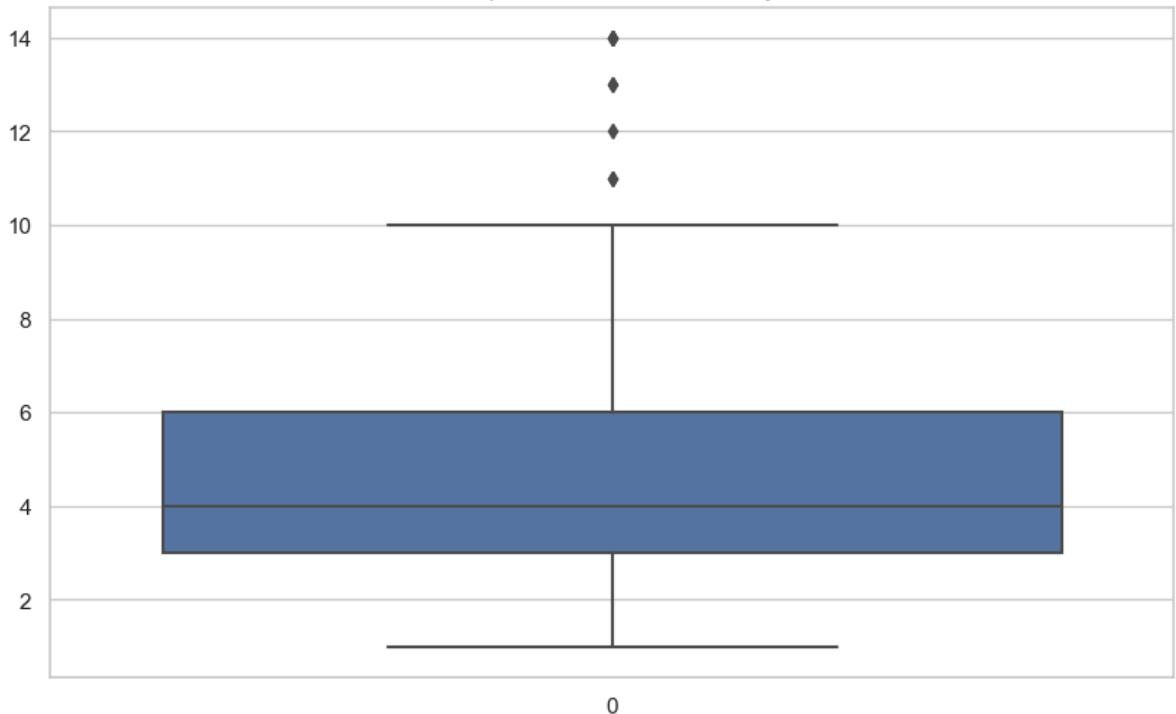
Outlier numerical feature: Sales, Outlier Count: 67

Boxplot of Outliers for Sales



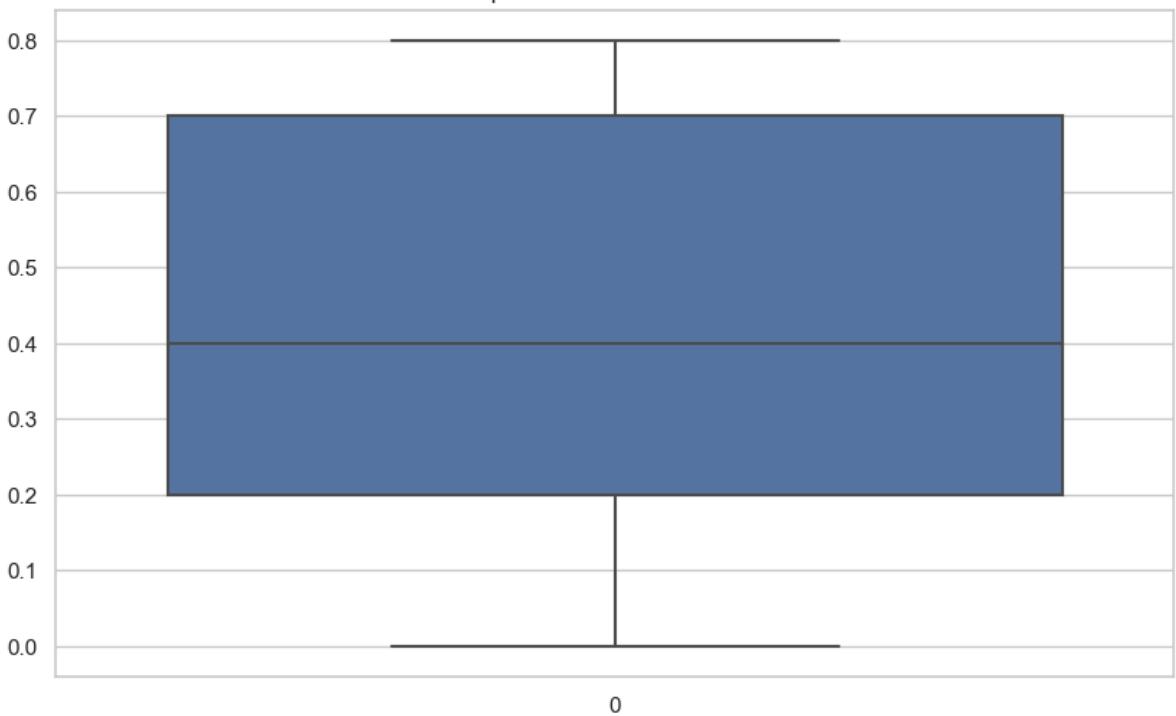
Outlier numerical feature: Quantity, Outlier Count: 9

Boxplot of Outliers for Quantity

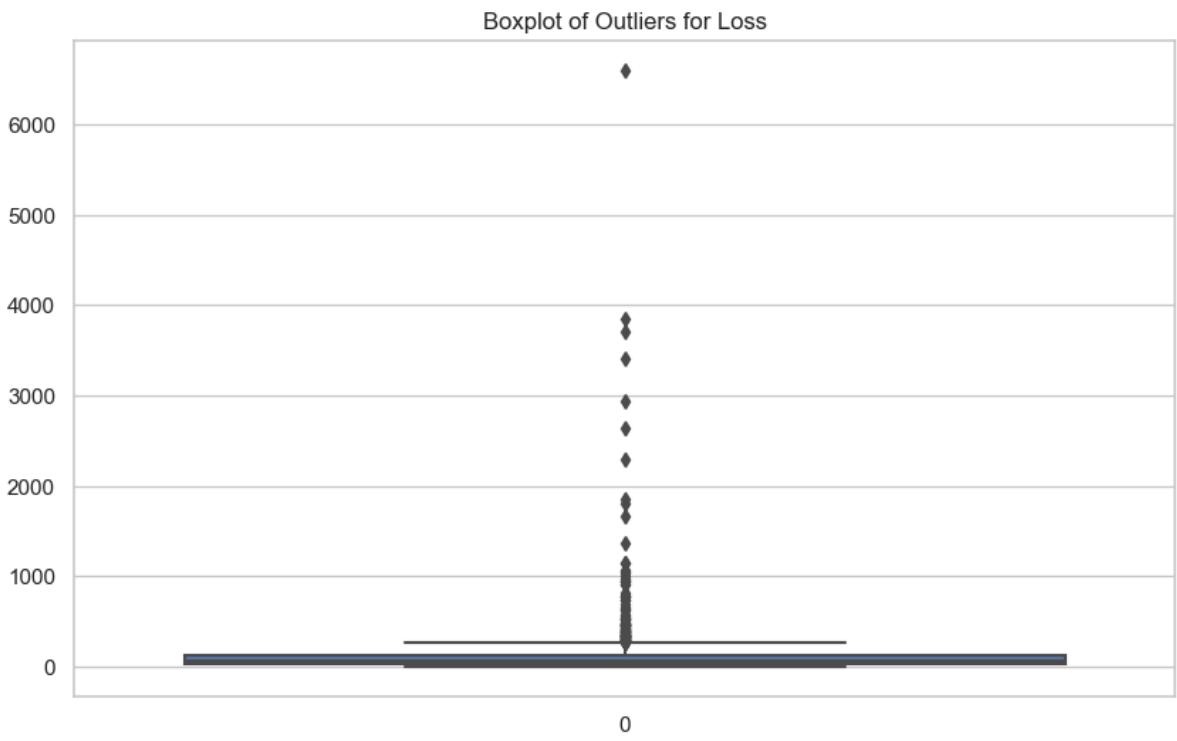


Outlier numerical feature: Discount, Outlier Count: 0

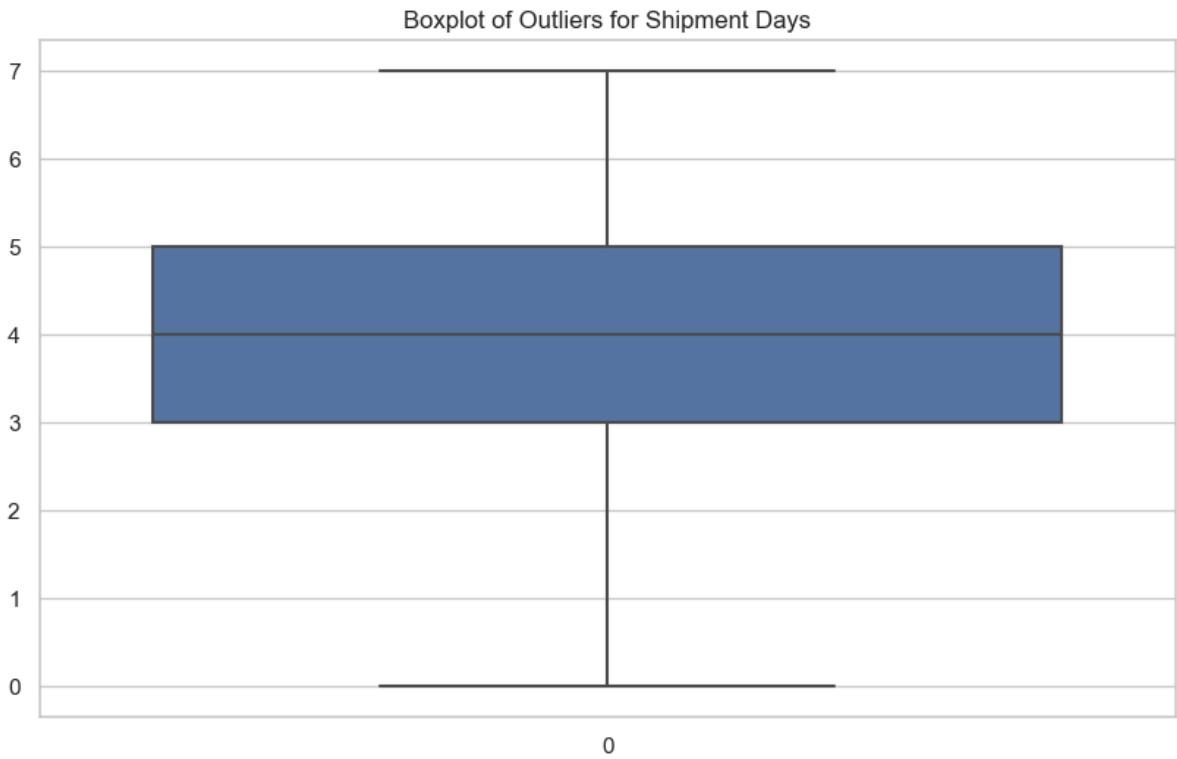
Boxplot of Outliers for Discount



Outlier numerical feature: Loss, Outlier Count: 108



Outlier numerical feature: Shipment Days, Outlier Count: 0



## Z-Score Method

Function to detect the outliers using z-score method.

```
In [ ]: def detect_outliers_zscore(df, numerical_feature, threshold=3):
    """
    Detect outliers in a numerical feature of a DataFrame using the Z-Score method.

    Parameters:
        df (DataFrame): The pandas DataFrame containing the data.
        numerical_feature (str): The name of the numerical feature to analyze.
        threshold (float): The Z-Score threshold for identifying outliers. Default
                           is 3.

    Returns:
        DataFrame: A DataFrame containing the outliers identified by the Z-Score
                   method.
    """
    # Calculate the Z-Score for each row
    df['Z-Score'] = (df[numerical_feature] - df[numerical_feature].mean()) / df[numerical_feature].std()

    # Identify outliers based on the threshold
    outliers = df[df['Z-Score'] > threshold]
    return outliers
```

```

    None
"""
data = df[numerical_feature]
z_scores = ((data - data.mean()) / data.std()).abs()
outliers = z_scores > threshold
outlier_count = outliers.sum()
print(f"Outlier count for {numerical_feature}: {outlier_count}")

```

In [ ]:

```

detect_outliers_zscore(global_super_store_df, 'Sales')
detect_outliers_zscore(global_super_store_df, 'Quantity')
detect_outliers_zscore(global_super_store_df, 'Discount')
detect_outliers_zscore(global_super_store_df, 'Loss')
detect_outliers_zscore(global_super_store_df, 'Shipment Days')

```

```

Outlier count for Sales: 8
Outlier count for Quantity: 7
Outlier count for Discount: 0
Outlier count for Loss: 11
Outlier count for Shipment Days: 0
Outlier count for Loss: 11
Outlier count for Shipment Days: 0

```

### Removing outliers using IQR method

Function to remove the outliers from the original dataframe.

In [ ]:

```

def remove_outliers_iqr(df):
    """
    Removes outliers from all numerical columns in the DataFrame based on the Interquartile Range (IQR) method.

    Parameters:
        df (DataFrame): The pandas DataFrame containing the data.

    Returns:
        DataFrame: The DataFrame with outliers removed.
    """

    # Get numerical columns
    numerical_columns = df.select_dtypes(include=['number']).columns

    # Iterate over numerical columns and remove outliers
    for column in numerical_columns:
        df = remove_outliers_iqr_column(df, column)

    return df

def remove_outliers_iqr_column(df, column):
    """
    Removes outliers from a specific numerical column in the DataFrame based on the IQR method.

    Parameters:
        df (DataFrame): The pandas DataFrame containing the data.
        column (str): The name of the numerical column to remove outliers from.

    Returns:
        DataFrame: The DataFrame with outliers removed.
    """

    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

```

```

# Filter out rows where the column value is outside the bounds
filtered_df = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

return filtered_df

```

In [ ]: `global_super_store_df = remove_outliers_iqr(global_super_store_df)`

## Conclusion

- Handling missing values in dataset
  - The following columns identified as non-meaningful for EDA, hence been dropped from the dataframe:
    - Row ID, Order ID, Customer ID, Product ID, Product Name
  - Replaced the Faulty values with filler values on categorical columns:
    - Ship Mode, Segment, State, Region, Category, Sub-Category
  - Replaced the following numerical columns were type of object, and data cleansed:
    - Quantity, Profit
    - Applied the `convert_alphabetic_to_float()` library to fix Quantity column
  - Cleansed the date using function: `format_date_in_yyyymmdd()`
  - Populated Geo information:
    - Function `get_city_from_postal_code()` is used to populate the city from postcode
    - Function `get_state_from_postal_code()` is used to populate the state from postcode
- Based on the 1000 order sample, the Global Super Store has:
  - An average sale of \$410.
  - Sold ~4 quantities in average.
  - Only made \$369 as profit, rest of them are massive losses.
  - Max order to ship turn around 7 days, average being ~4 days.
- Grouped the dataset by categorical columns against the sales and losses
  - **Furniture** is the best sales category and also made bigger losses.
  - **Tables** is the best sales sub category, and **Binders** made the worst losses.
  - **Texas** has the highest sales and well as highest losses.
  - **Consumer** segment making half of the total sales, but also responsible for one-thirds of the total losses.
  - **Standard** shipping method is preferred as the sales percentage is nearly two-thirds of total shipments.
- Univariate analysis and visualization
  - Based on the PDF/CDF graphs, the skewness of **Sales** and **Loss** columns, it is heavily skewed to the right, with a few very high sales/loss values exerting a disproportionate influence on the distribution.
  - Year on year trends
    - Sales numbers are growing year on year, dipping slightly in 2015.

- Global super store has sold more quantities year on year despite making more losses.

### 3. Task 3

#### Categorical Vs. Categorical

The chi-square test of independence is a statistical test used to determine whether there is a significant association between two categorical variables.

#### Chi-Square Test of Independence

```
In [ ]: def chi_square_test(df, categorical_feature_1, categorical_feature_2, alpha=0.05):
    """
        Perform chi-square test of independence between two categorical variables.

    Parameters:
        df (DataFrame): Input DataFrame containing the data.
        categorical_feature_1 (str): Name of the first categorical variable.
        categorical_feature_2 (str): Name of the second categorical variable.
        alpha (float): Significance level (default is 0.05).
    """
    # Cross-tabulation
    cross_tab = pd.crosstab(df[categorical_feature_1], df[categorical_feature_2])

    # Perform chi-square test
    chi2, p_value, dof, expected = chi2_contingency(cross_tab)

    # Print results
    print(f"Chi-square statistic: {chi2}")
    print(f"P-value: {p_value}")
    print(f"Degrees of freedom: {dof}")

    # Make decision based on p-value
    if p_value <= alpha:
        print(f"Based on the p-value of {p_value}, we reject the null hypothesis.")
        print(f"There is a significant association between {categorical_feature_1}")
    else:
        print(f"Based on the p-value of {p_value}, we fail to reject the null hypothesis.")
        print(f"There is no significant association between {categorical_feature_1}")
```

Performing Chi-Square test of independence.

```
In [ ]: # List of categorical variables
categorical_features = ['Ship Mode', 'Segment', 'Region', 'Category', 'Sub-Category']

# Loop through all combinations of categorical variables to call chi_square_test
for i in range(len(categorical_features)):
    for j in range(i + 1, len(categorical_features)):
        categorical_feature_1 = categorical_features[i]
        categorical_feature_2 = categorical_features[j]
        print(f"Performing chi-square test between {categorical_feature_1} and {categorical_feature_2}")
        chi_square_test(global_super_store_df, categorical_feature_1, categorical_feature_2)
        print("\n")
```

Performing chi-square test between Ship Mode and Segment:  
Chi-square statistic: 5.620633935946101  
P-value: 0.4669980038187912  
Degrees of freedom: 6  
Based on the p-value of 0.4669980038187912, we fail to reject the null hypothesis.  
There is no significant association between Ship Mode and Segment.

Performing chi-square test between Ship Mode and Region:  
Chi-square statistic: 14.294943833673106  
P-value: 0.11221352184272178  
Degrees of freedom: 9  
Based on the p-value of 0.11221352184272178, we fail to reject the null hypothesis.  
There is no significant association between Ship Mode and Region.

Performing chi-square test between Ship Mode and Category:  
Chi-square statistic: 3.1422040665529076  
P-value: 0.7907987938955084  
Degrees of freedom: 6  
Based on the p-value of 0.7907987938955084, we fail to reject the null hypothesis.  
There is no significant association between Ship Mode and Category.

Performing chi-square test between Ship Mode and Sub-Category:  
Chi-square statistic: 35.80469153873701  
P-value: 0.21454116179140625  
Degrees of freedom: 30  
Based on the p-value of 0.21454116179140625, we fail to reject the null hypothesis.  
There is no significant association between Ship Mode and Sub-Category.

Performing chi-square test between Segment and Region:  
Chi-square statistic: 5.372511621886918  
P-value: 0.4969973003222622  
Degrees of freedom: 6  
Based on the p-value of 0.4969973003222622, we fail to reject the null hypothesis.  
There is no significant association between Segment and Region.

Performing chi-square test between Segment and Category:  
Chi-square statistic: 3.330745510287931  
P-value: 0.5040756934811397  
Degrees of freedom: 4  
Based on the p-value of 0.5040756934811397, we fail to reject the null hypothesis.  
There is no significant association between Segment and Category.

Performing chi-square test between Segment and Sub-Category:  
Chi-square statistic: 14.652883814938855  
P-value: 0.7959132032930026  
Degrees of freedom: 20  
Based on the p-value of 0.7959132032930026, we fail to reject the null hypothesis.  
There is no significant association between Segment and Sub-Category.

Performing chi-square test between Region and Category:  
Chi-square statistic: 103.50328539993852  
P-value: 4.6574174516409994e-20  
Degrees of freedom: 6  
Based on the p-value of 4.6574174516409994e-20, we reject the null hypothesis.  
There is a significant association between Region and Category.

```
Performing chi-square test between Region and Sub-Category:  
Chi-square statistic: 259.59669247851286  
P-value: 2.107543089876603e-38  
Degrees of freedom: 30  
Based on the p-value of 2.107543089876603e-38, we reject the null hypothesis.  
There is a significant association between Region and Sub-Category.
```

```
Performing chi-square test between Category and Sub-Category:  
Chi-square statistic: 1541.9999999999998  
P-value: 0.0  
Degrees of freedom: 20  
Based on the p-value of 0.0, we reject the null hypothesis.  
There is a significant association between Category and Sub-Category.
```

According to chi-square test of independence, these categorical features has a significant association:

- *Category and Sub-Category*
- *Region and Category*
- *Region and Sub-Category*

Hence, the bivariate analysis/visualization will be based between these categorical features.

Function: `categorical_analysis()` uses cross-tabulation to plot heatmaps/stacked bar plots.

Function: `bivariate_categorical_sales_figure_analysis()` uses pivot tables to plot heatmaps/stacked bar plots .

## Helper Functions for two categorical variables analysis

Function to plot the cross tabulation between two categorical features.

```
In [ ]: def categorical_analysis(df, categorical_feature_1, categorical_feature_2):  
    """  
        Perform bivariate analysis and visualization between two categorical variables  
  
    Parameters:  
        df (DataFrame): Input DataFrame containing the data.  
        categorical_feature_1 (str): Name of the first categorical variable.  
        categorical_feature_2 (str): Name of the second categorical variable.  
    """  
  
        # Cross-tab  
    cross_tab = pd.crosstab(df[categorical_feature_1], df[categorical_feature_2])  
        # Create subplots  
    fig, axes = plt.subplots(1, 2, figsize=(20, 10))  
  
        # Visualize cross-tabulate using a heatmap  
    sns.heatmap(cross_tab, annot=True, fmt='d', cmap='YlGnBu', ax=axes[0])  
    axes[0].set_title(f'Cross-tabulation between {categorical_feature_1} and {categorical_feature_2}')  
    axes[0].set_xlabel(categorical_feature_2)  
    axes[0].set_ylabel(categorical_feature_1)  
  
        # Stacked bar plot for better visualization
```

```

cross_tab.plot(kind='bar', stacked=True, ax=axes[1])
axes[1].set_title(f'Stacked Bar Plot of {categorical_feature_1} by {categorical_feature_2}')
axes[1].set_xlabel(categorical_feature_1)
axes[1].set_ylabel('Count')
axes[1].set_xticklabels(cross_tab.index, rotation=45)
axes[1].legend(title=categorical_feature_2)

plt.tight_layout()
plt.show()

```

Function to plot the pivot table between two categorical features and a numerical feature.

```

In [ ]: def bivariate_categorical_figure_analysis(df, categorical_feature_1, categorical_feature_2):
    """
    Perform bivariate analysis and visualization between two categorical variables.

    Parameters:
        df (DataFrame): Input DataFrame containing the data.
        categorical_feature_1 (str): Name of the first categorical variable.
        categorical_feature_2 (str): Name of the second categorical variable.
    """

    # Create a pivot table with 'Sales' values
    pivot_table = df.pivot_table(values=numerical_feature, index=categorical_feature_1,
                                 columns=categorical_feature_2, aggfunc='sum')

    # Define color palette
    color_palette = sns.color_palette("YlGnBu")

    # Plotting the pivot table
    plt.figure(figsize=(20, 10))
    sns.heatmap(pivot_table, annot=True, cmap=color_palette, fmt=',.2f', annot_kws={'size': 10})
    plt.title(f'Cross-tabulation of Total {numerical_feature} between {categorical_feature_1} and {categorical_feature_2}')
    plt.xlabel(categorical_feature_2)
    plt.ylabel(categorical_feature_1)
    plt.show()

    # Plotting the pivot table as a stacked bar plot
    ax = pivot_table.plot(kind='bar', stacked=True, figsize=(20, 10), color=color_palette)
    ax.set_title(f'Stacked Bar Plot of Total {numerical_feature} by {categorical_feature_1}')
    ax.set_xlabel(categorical_feature_1)
    ax.set_ylabel(f'Total {numerical_feature}')
    ax.set_xticklabels(pivot_table.index, rotation=45)

    # Format y-axis tick labels as currency
    ax.yaxis.set_major_formatter('${:.2f}'.format)

    ax.legend(title=categorical_feature_2)
    plt.show()

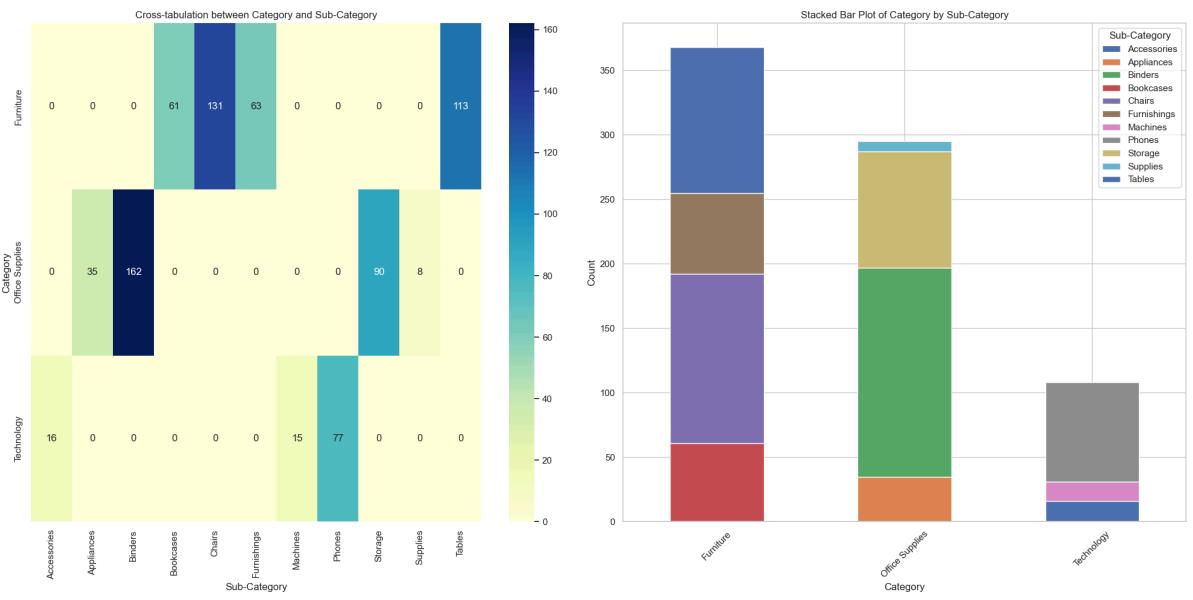
```

## Category Vs. Sub-Category

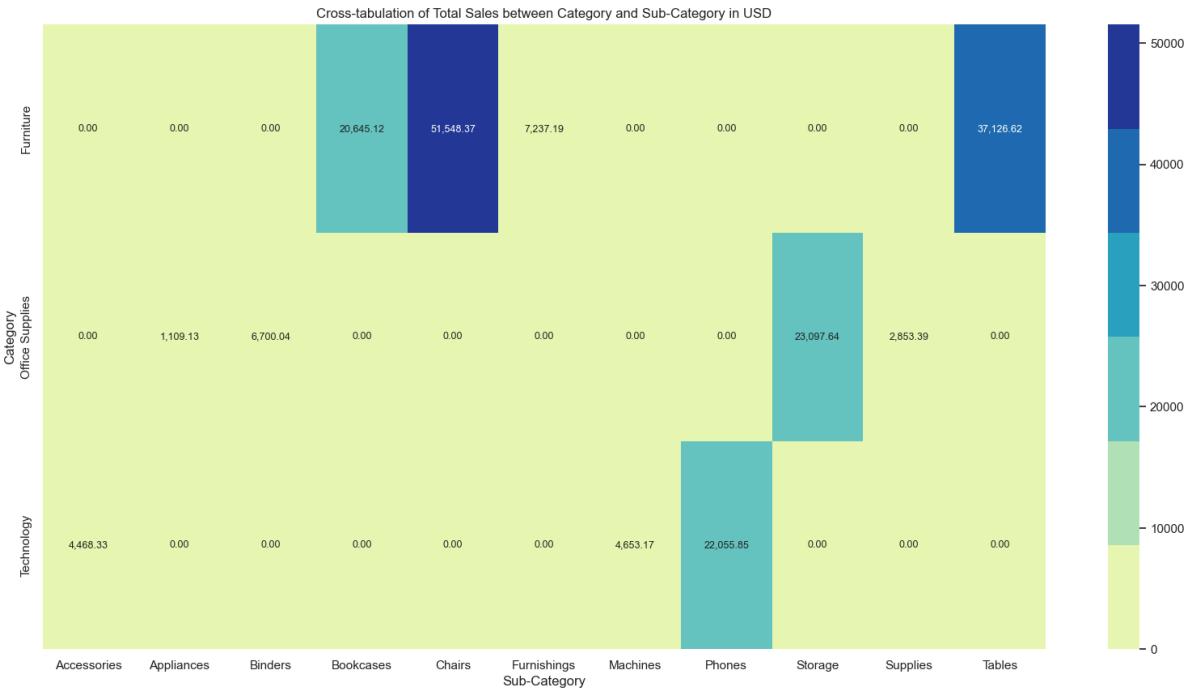
```

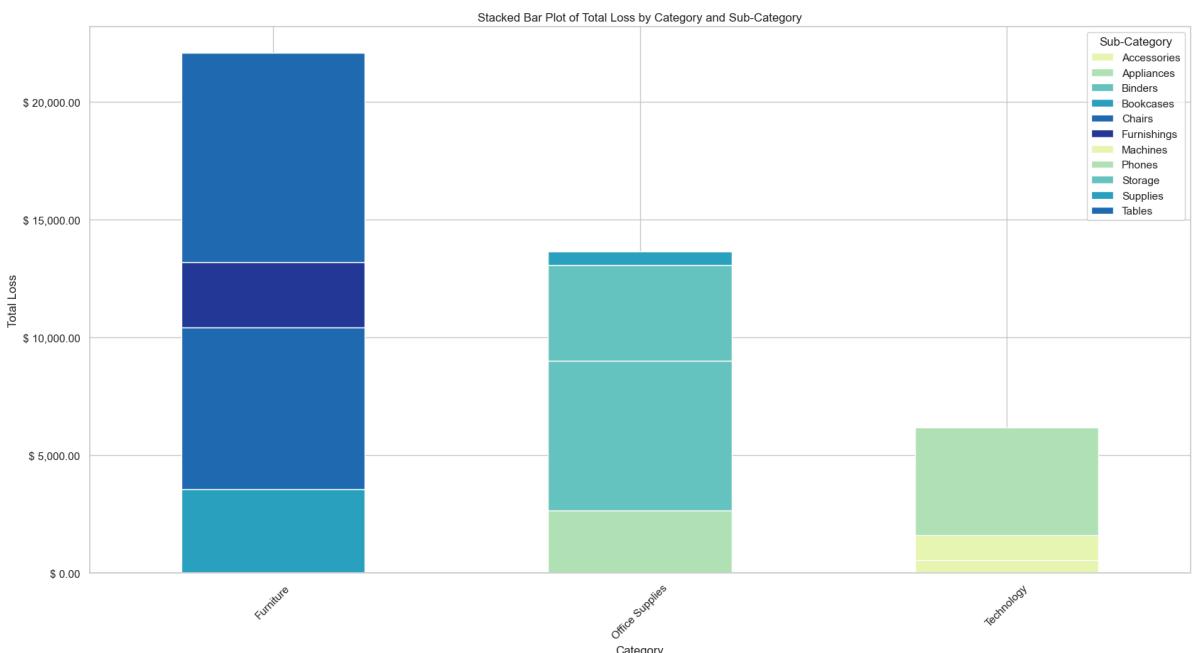
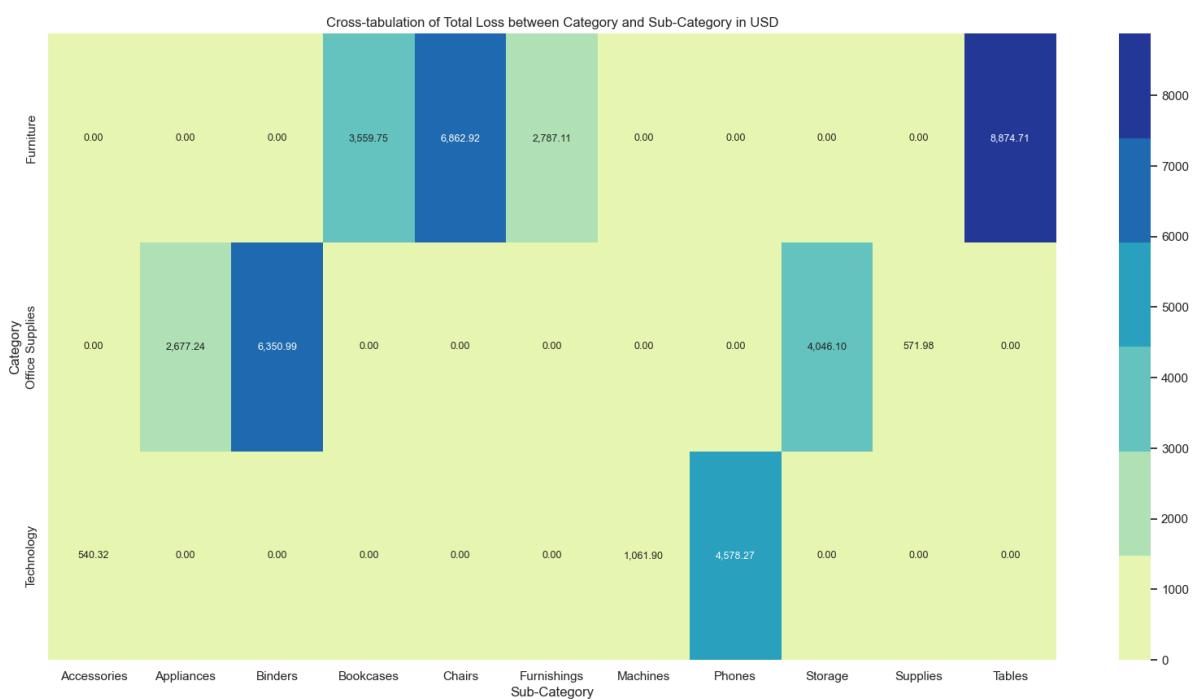
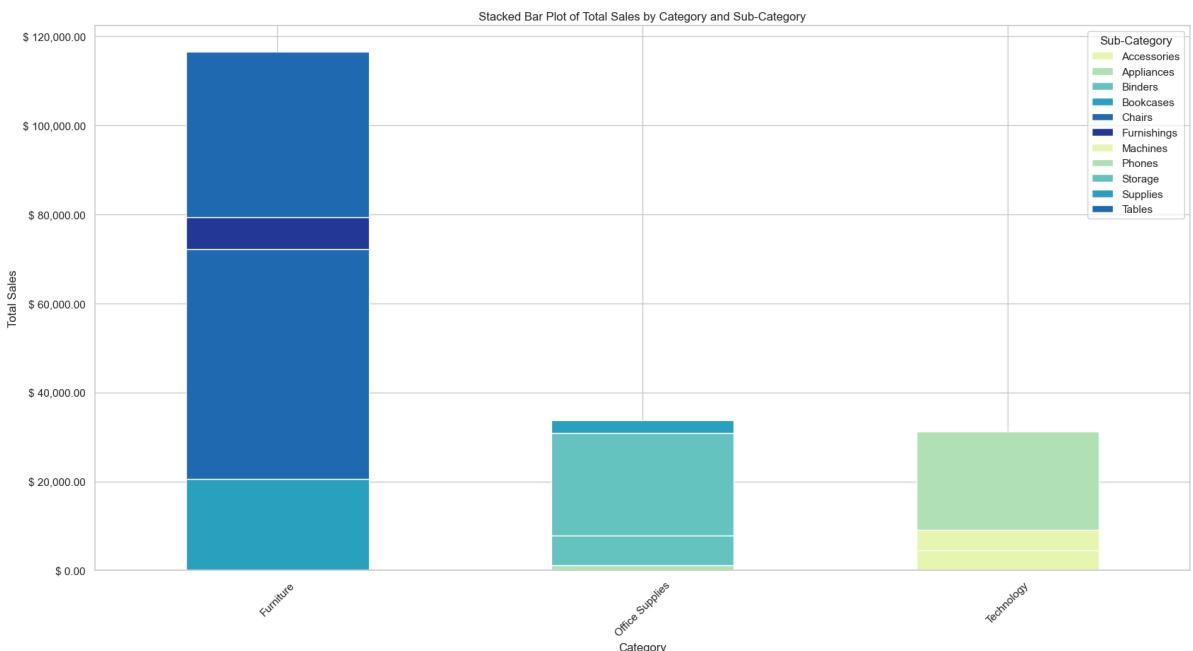
In [ ]: categorical_analysis(global_super_store_df, 'Category', 'Sub-Category')

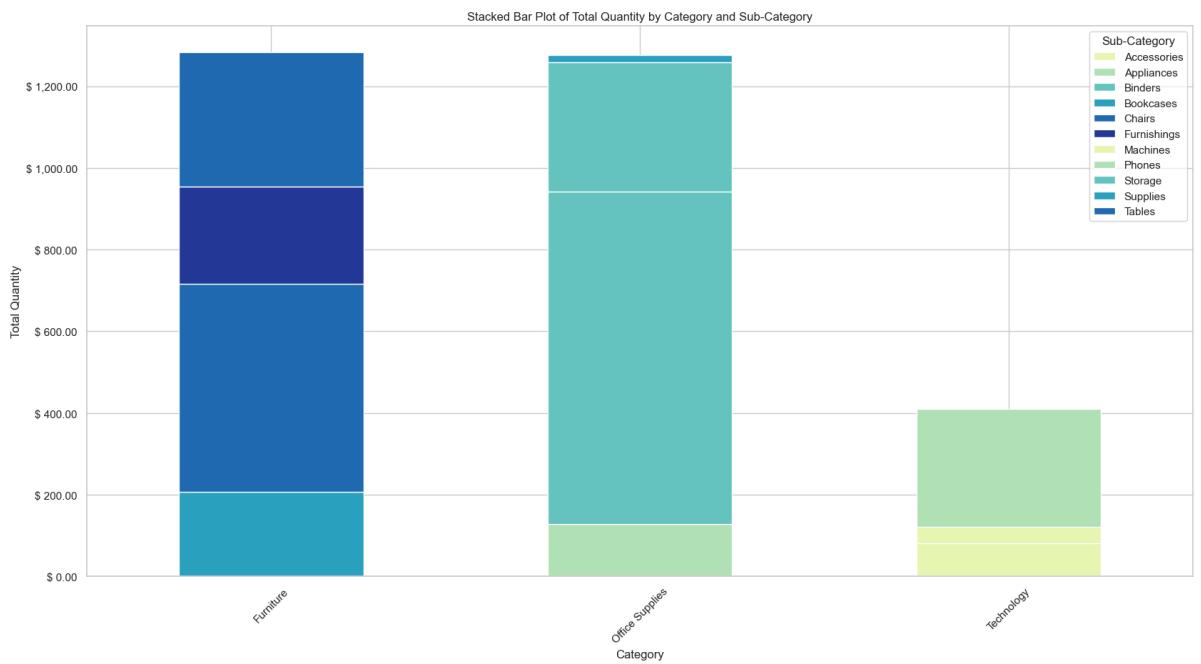
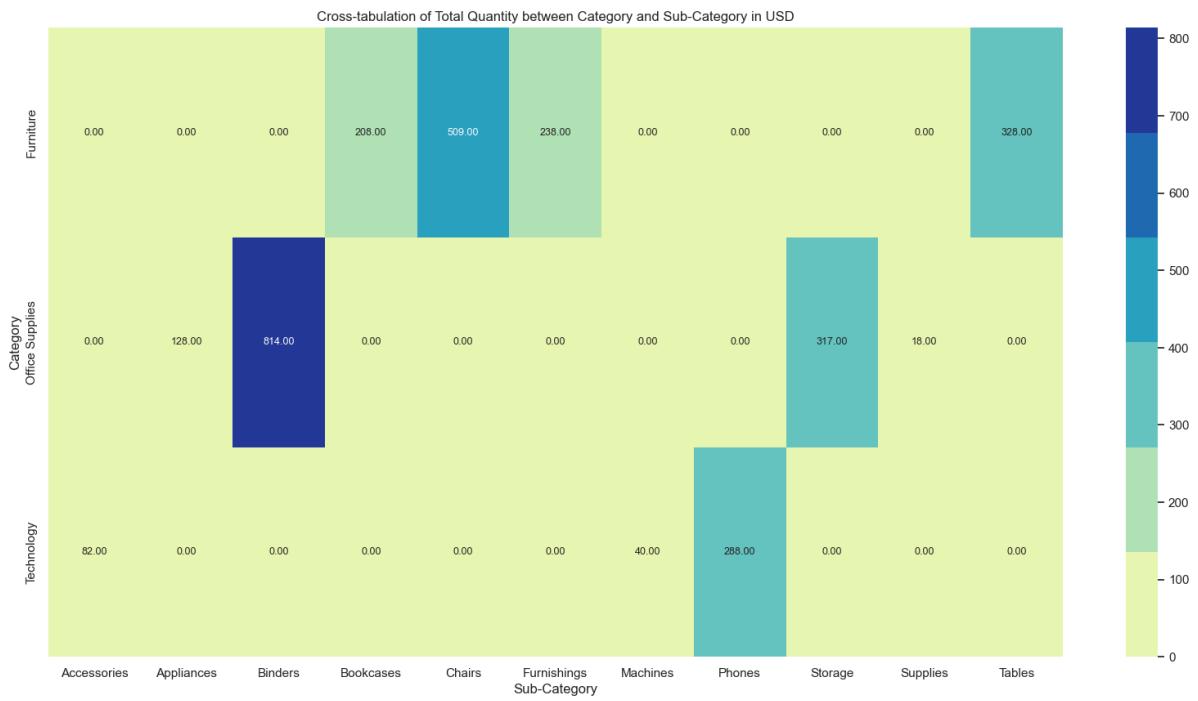
```



```
In [ ]: bivariate_categorical_figure_analysis(global_super_store_df, 'Category', 'Sub-Category')
bivariate_categorical_figure_analysis(global_super_store_df, 'Category', 'Sub-Category')
bivariate_categorical_figure_analysis(global_super_store_df, 'Category', 'Sub-Category')
```

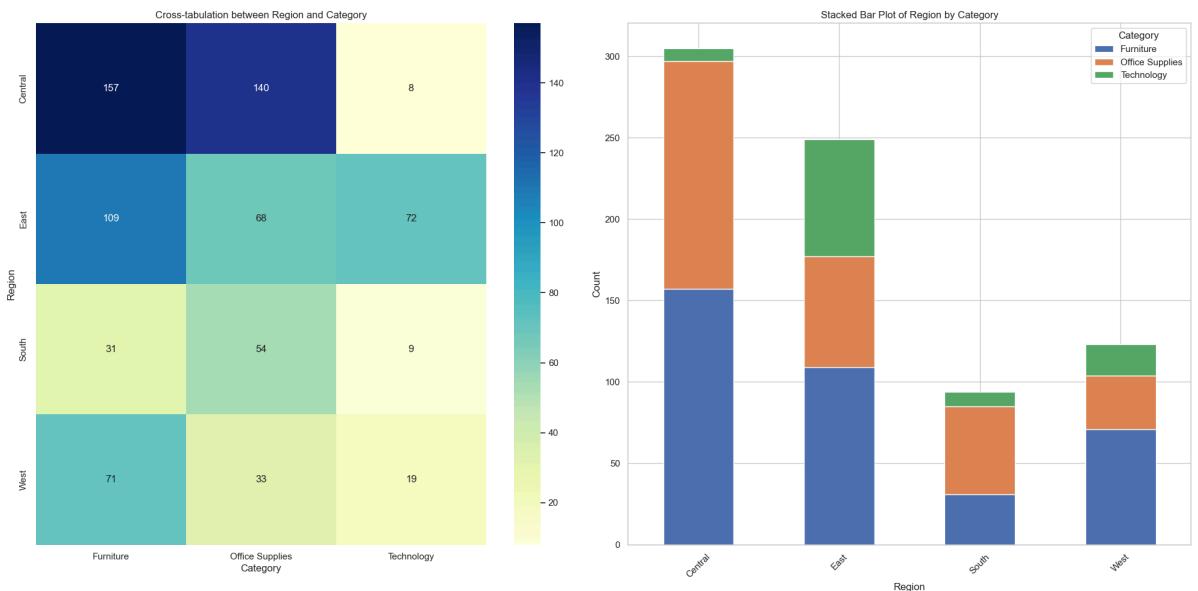




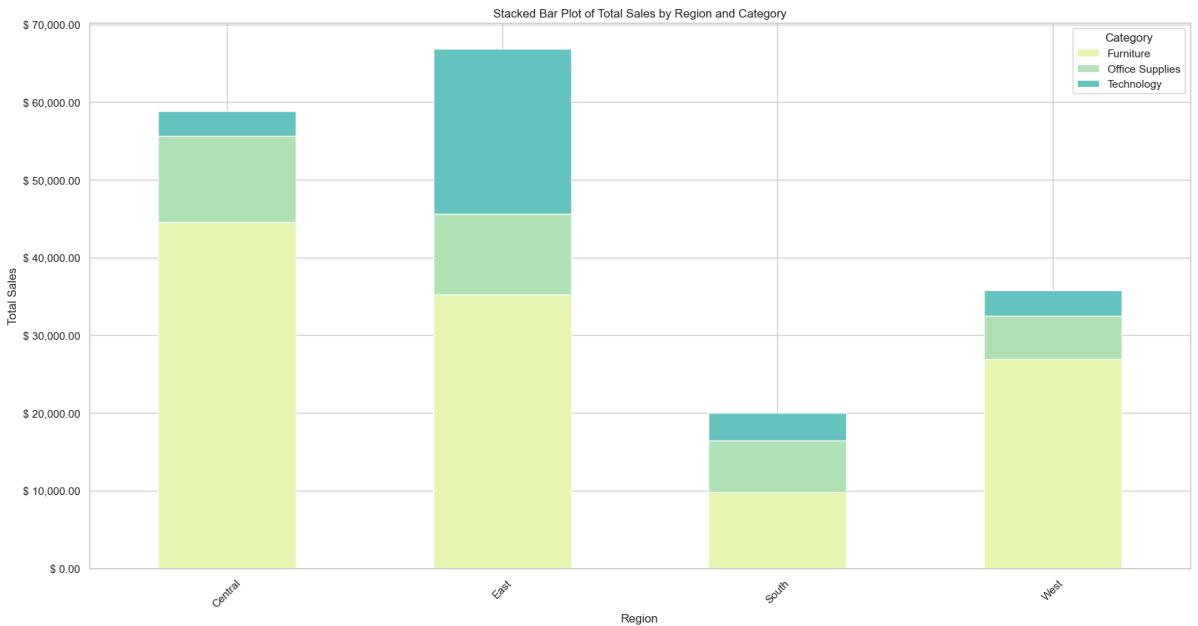
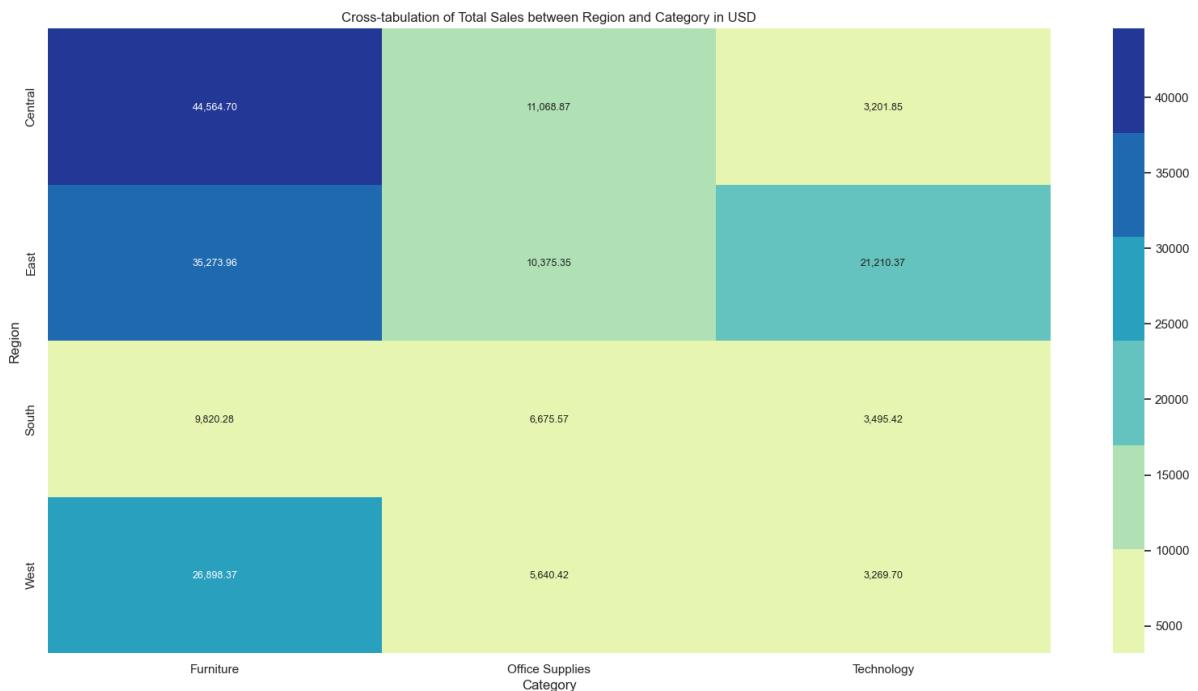


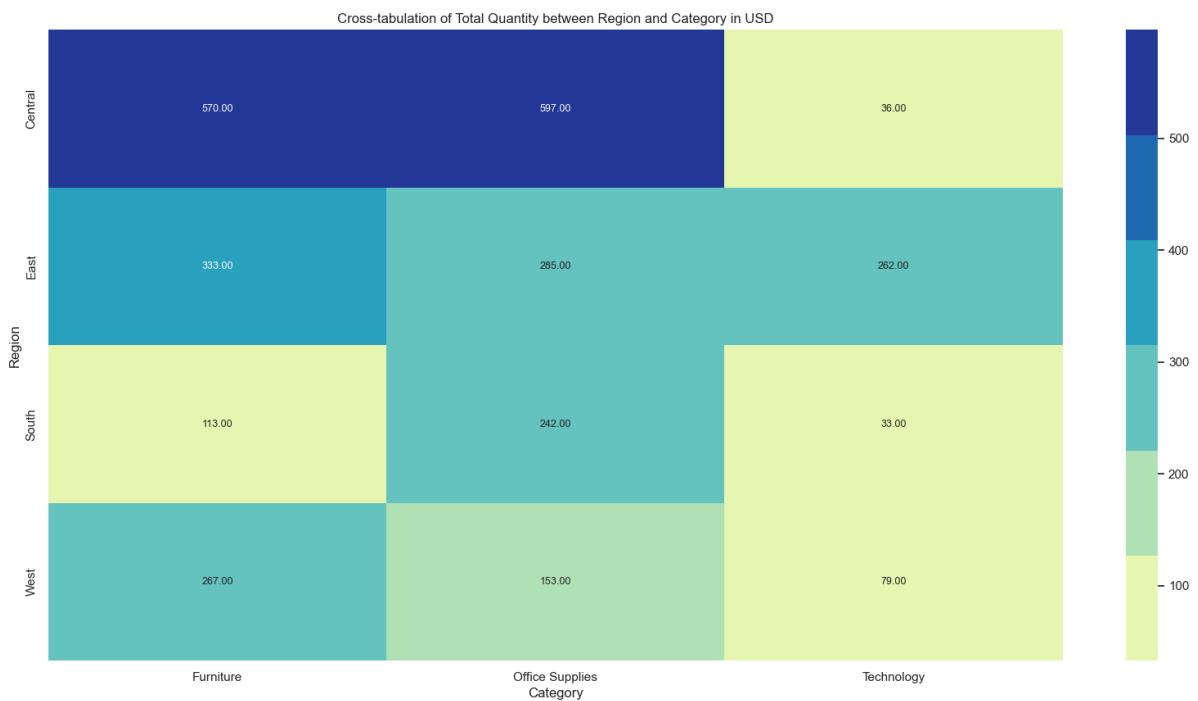
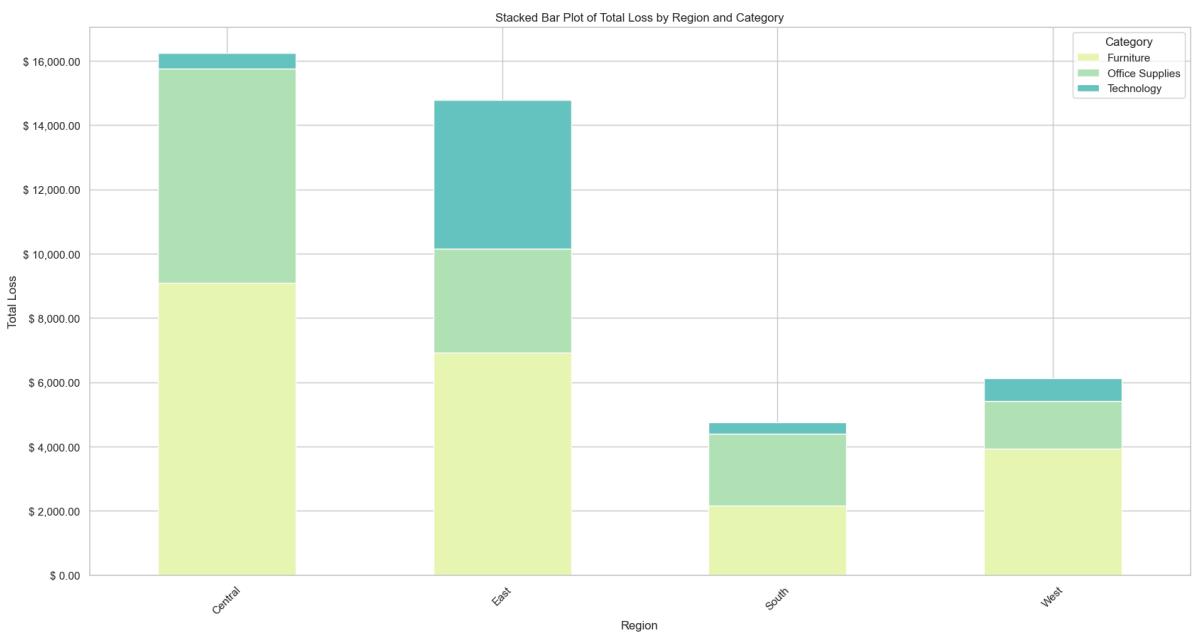
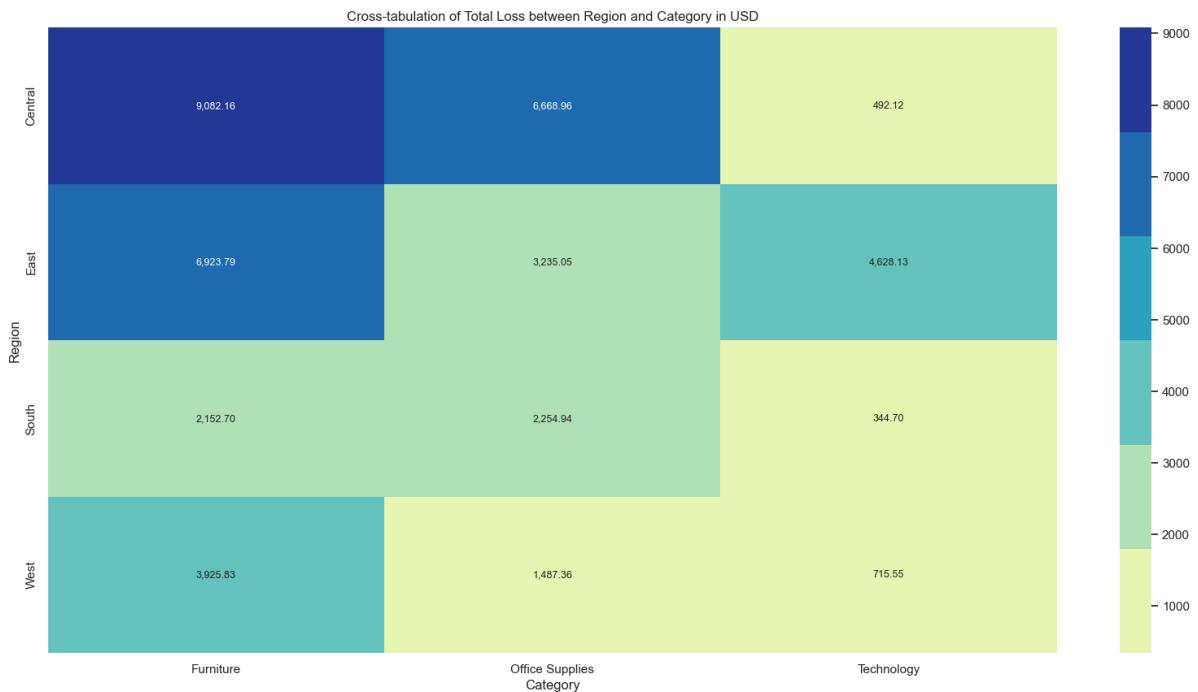
## Region Vs. Category

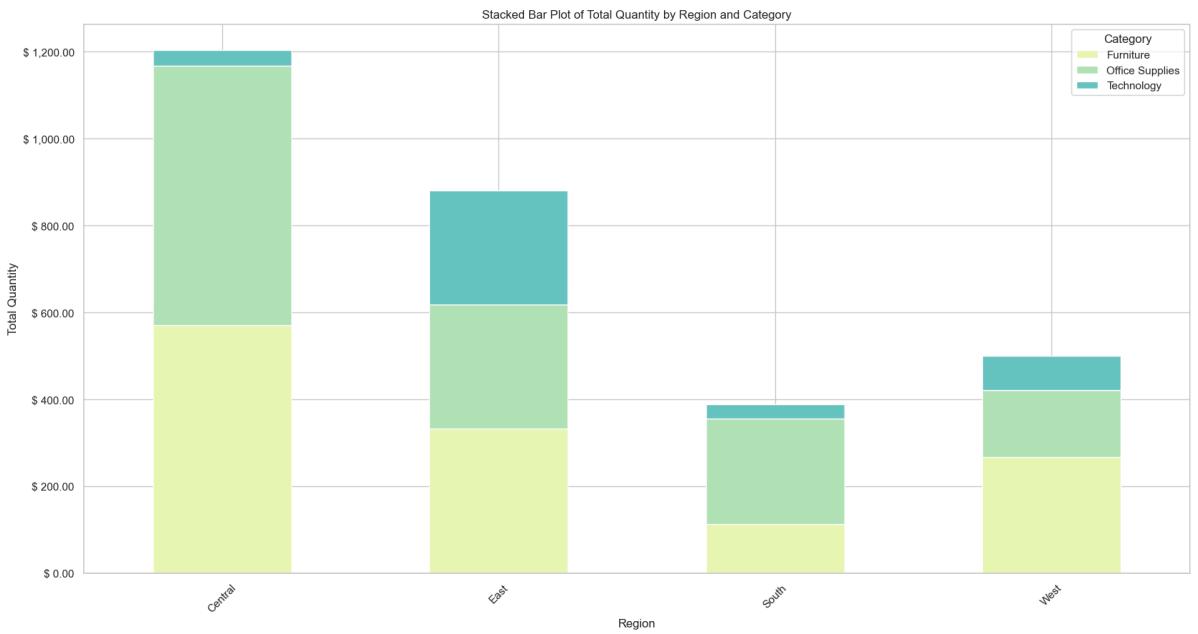
```
In [ ]: categorical_analysis(global_super_store_df, 'Region', 'Category')
```



```
In [ ]: bivariate_categorical_figure_analysis(global_super_store_df, 'Region', 'Category',
bivariate_categorical_figure_analysis(global_super_store_df, 'Region', 'Category',
bivariate_categorical_figure_analysis(global_super_store_df, 'Region', 'Category',
```

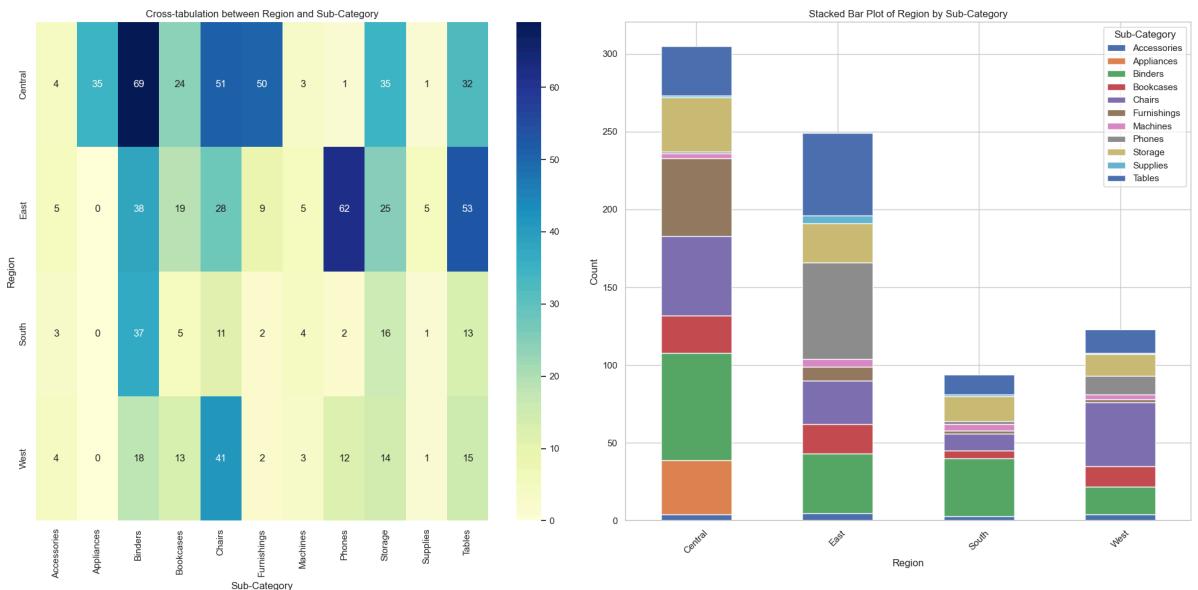




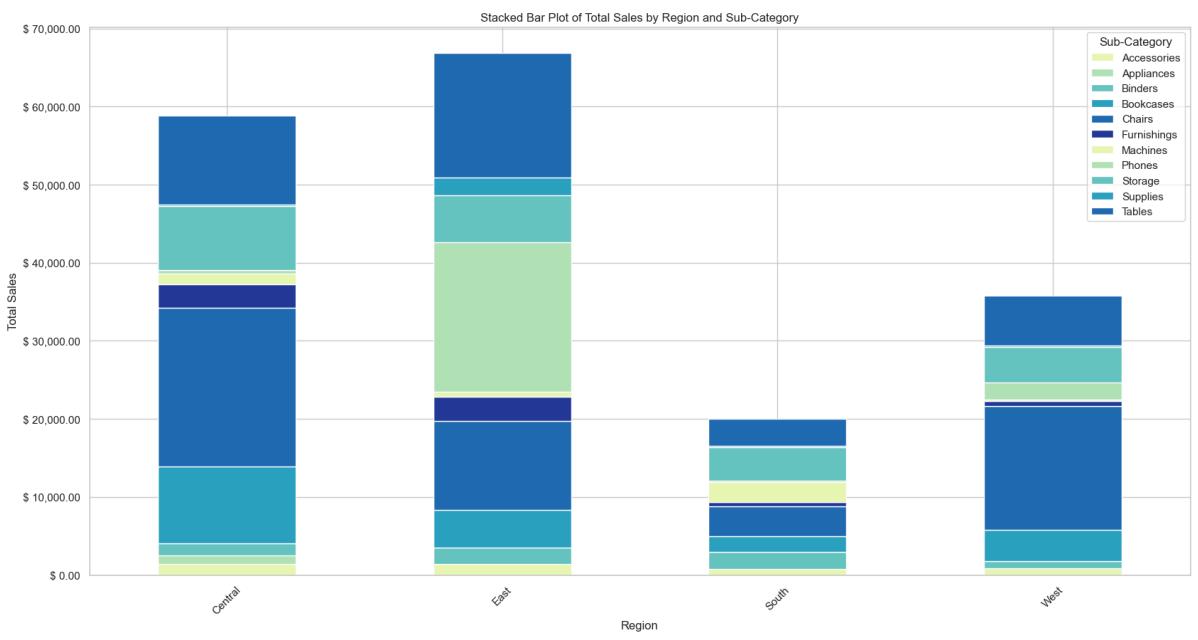


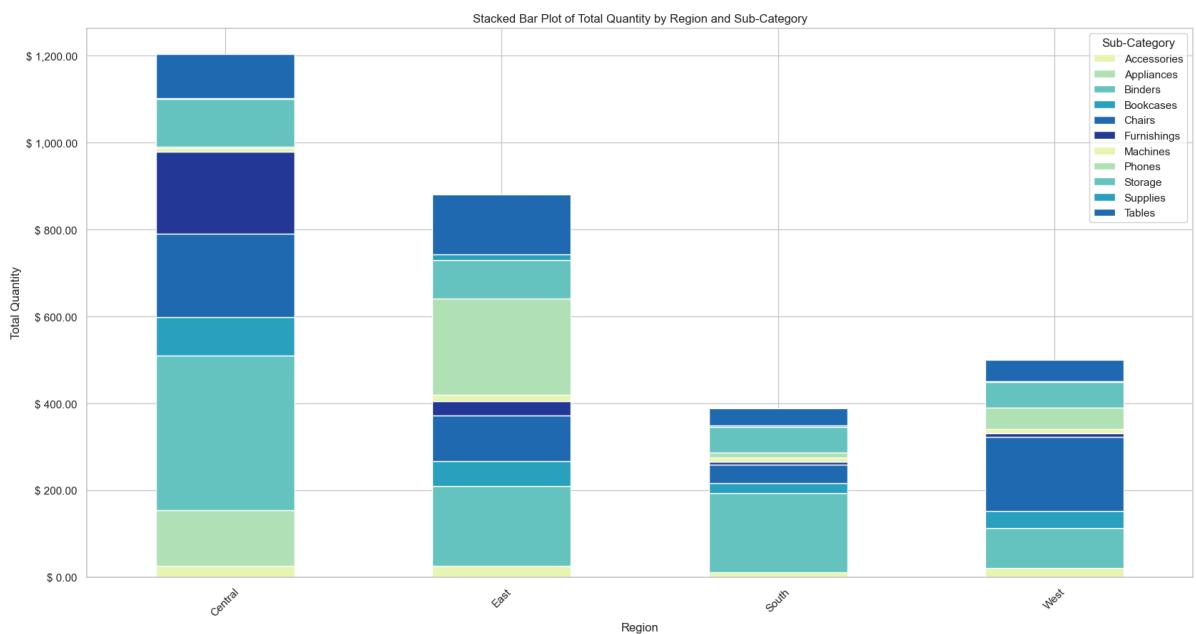
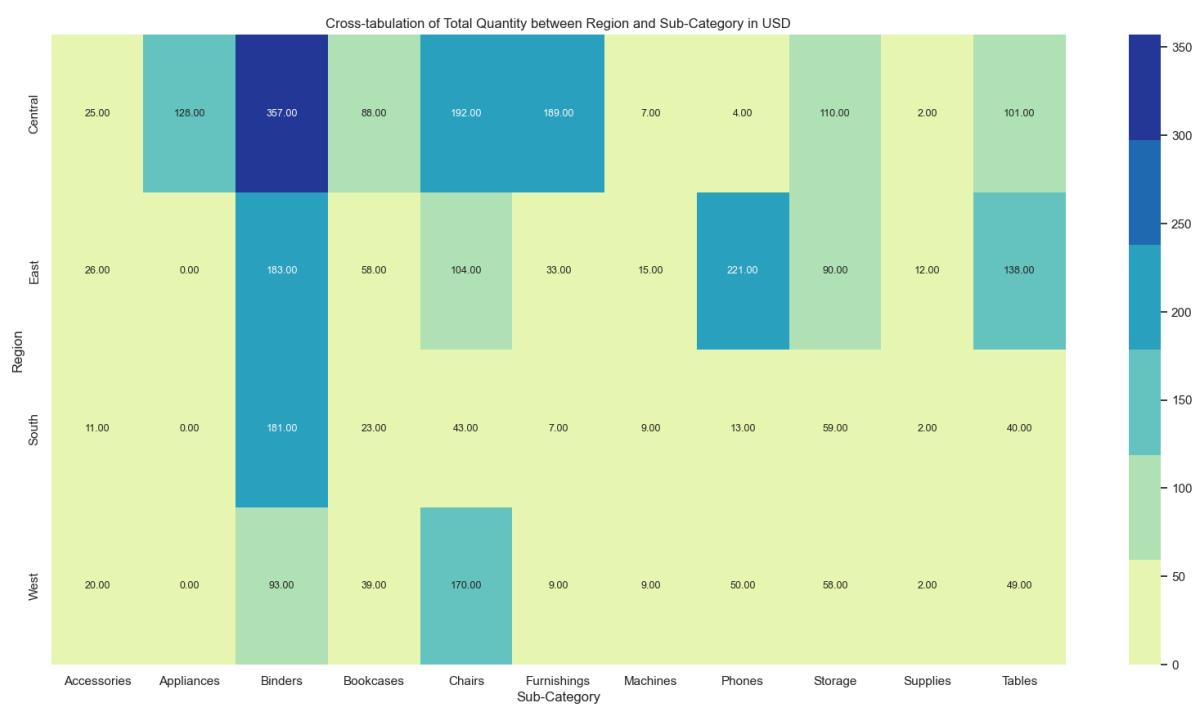
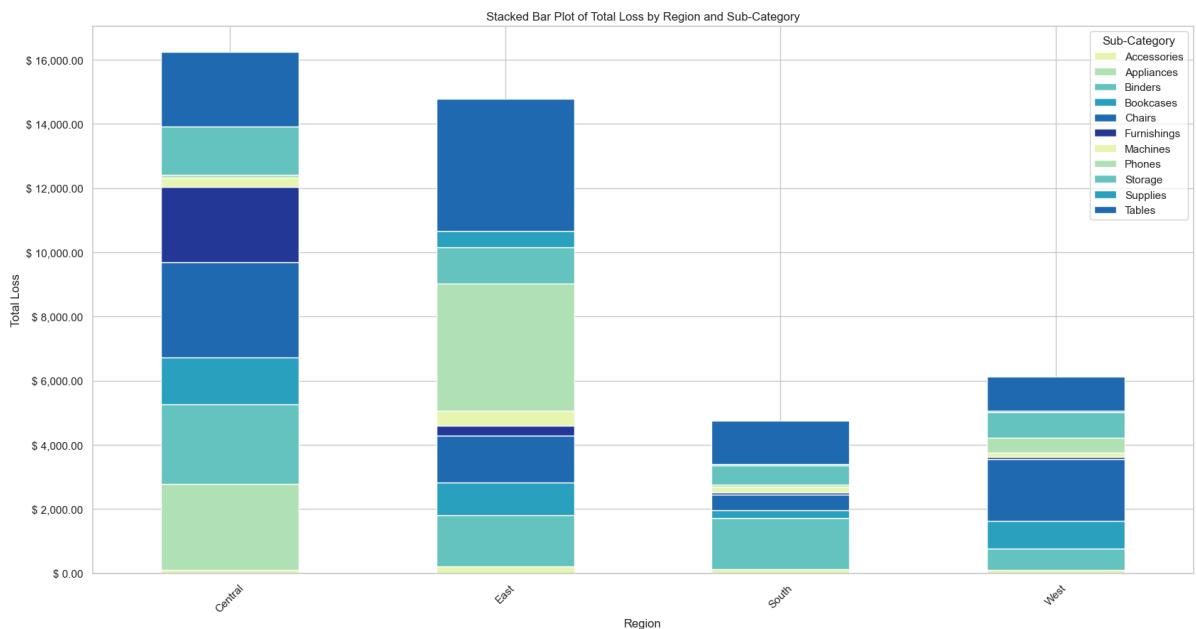
## Region Vs. Sub-Category

```
In [ ]: categorical_analysis(global_super_store_df, 'Region', 'Sub-Category')
```



```
In [ ]: bivariate_categorical_figure_analysis(global_super_store_df, 'Region', 'Sub-Category')
bivariate_categorical_figure_analysis(global_super_store_df, 'Region', 'Sub-Category')
bivariate_categorical_figure_analysis(global_super_store_df, 'Region', 'Sub-Category')
```





## Numerical Vs. Numerical

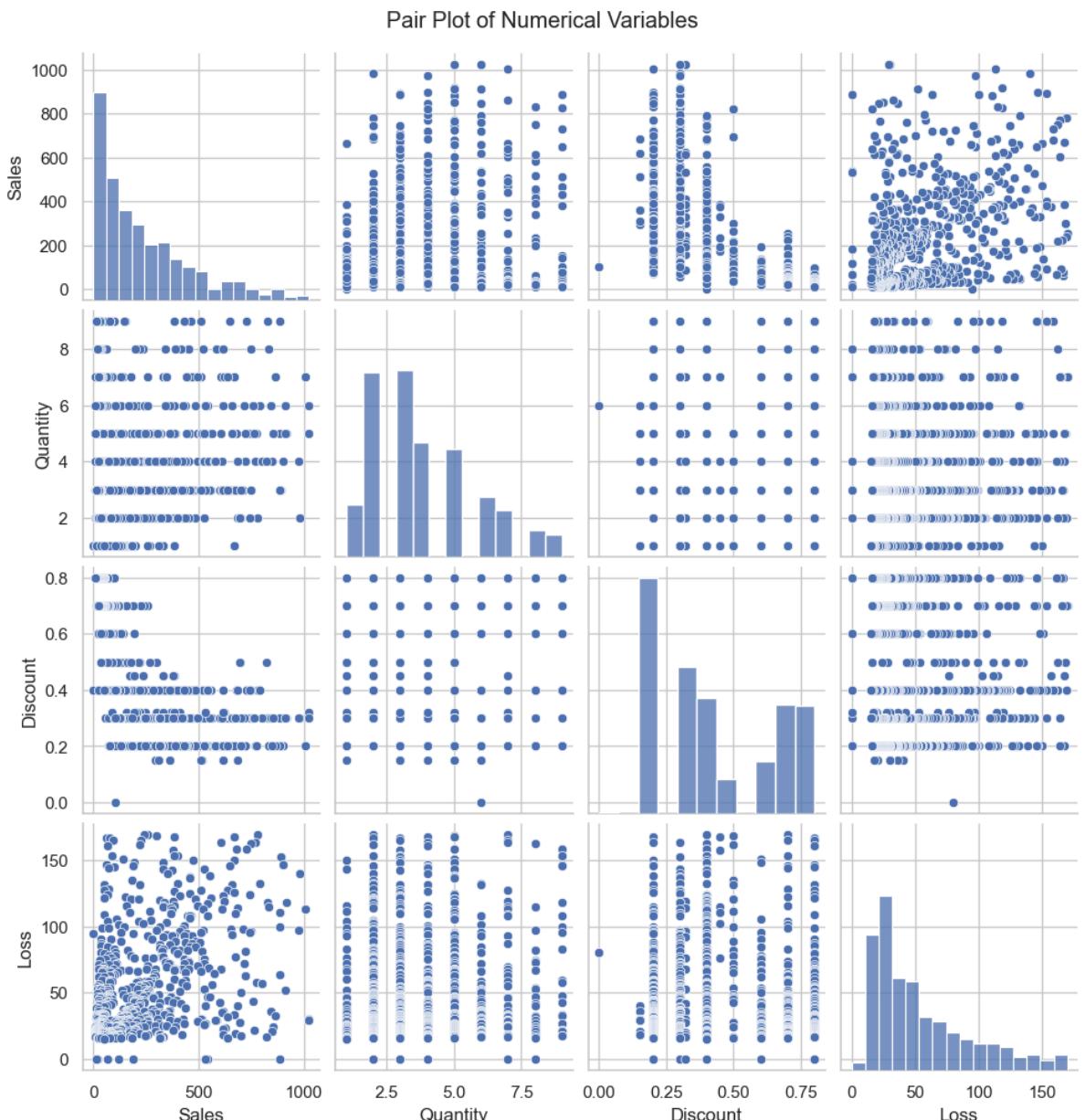
Choosing the numerical features and filter into a new dataframe.

```
In [ ]: #Select Numerical Features on global_super_store_df dataframe  
numerical_features = ['Sales', 'Quantity', 'Discount', 'Loss']  
global_super_store_numerical_data = global_super_store_df[numerical_features]
```

### Pairplots

```
In [ ]: sns.pairplot(global_super_store_numerical_data)  
plt.suptitle('Pair Plot of Numerical Variables', y=1.02)  
plt.show()
```

```
c:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: T  
he figure layout has changed to tight  
    self._figure.tight_layout(*args, **kwargs)
```



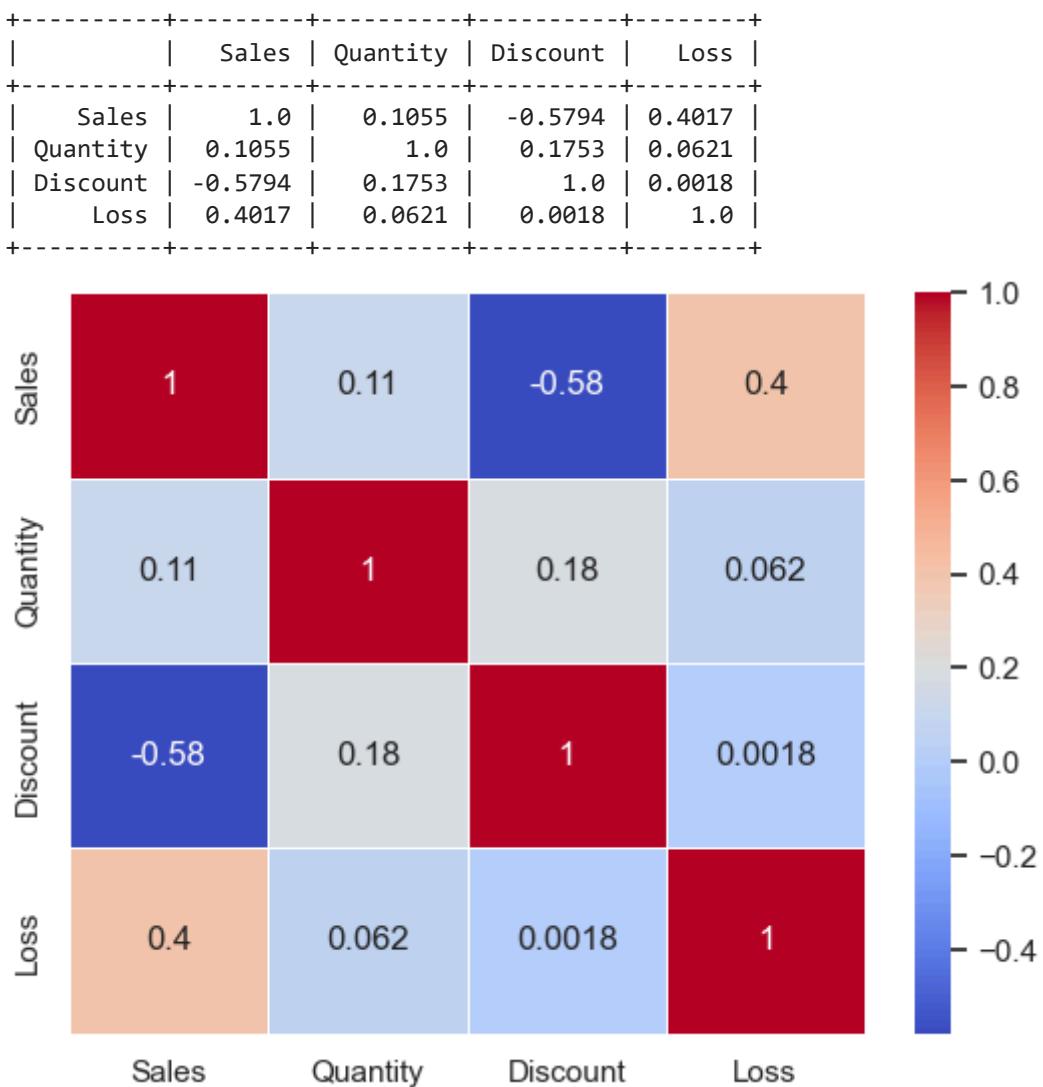
### Correlation Matrix

```
In [ ]: # Create Correlation matrix  
correlation_matrix = global_super_store_numerical_data.corr()
```

```

print(tabulate(correlation_matrix.round(4), headers='keys', tablefmt='pretty', stra
# Plot correlation matrix as heatmap
sns.heatmap(correlation_matrix.round(4), annot=True, cmap='coolwarm', linewidths=0.
plt.show()

```



## Scatter plots

Function to plot scatter and joint plot to perform bivariate analysis between two numerical features.

```

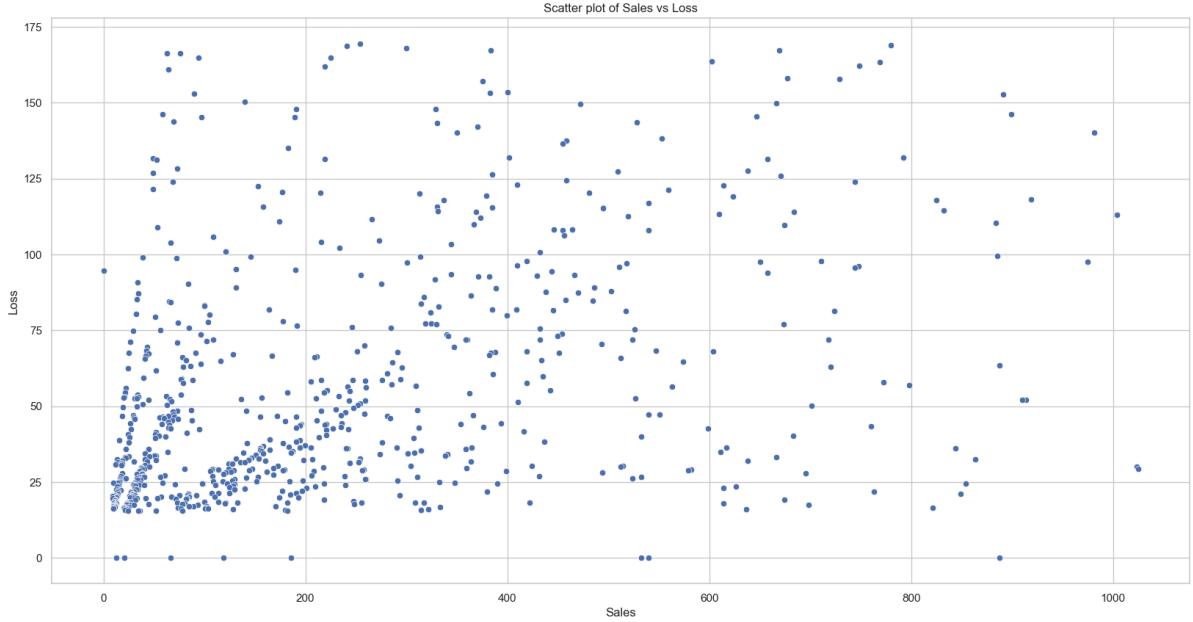
In [ ]: def bivariate_numerical_analysis(df, num_var1, num_var2):
    """
    Perform bivariate analysis and visualization between two numerical variables.

    Parameters:
        df (DataFrame): Input DataFrame containing the data.
        num_var1 (str): Name of the first numerical variable.
        num_var2 (str): Name of the second numerical variable.
    """
    # Scatter plot
    plt.figure(figsize=(20, 10))
    sns.scatterplot(x=num_var1, y=num_var2, data=df)
    plt.title(f'Scatter plot of {num_var1} vs {num_var2}')
    plt.xlabel(num_var1)
    plt.ylabel(num_var2)
    plt.show()

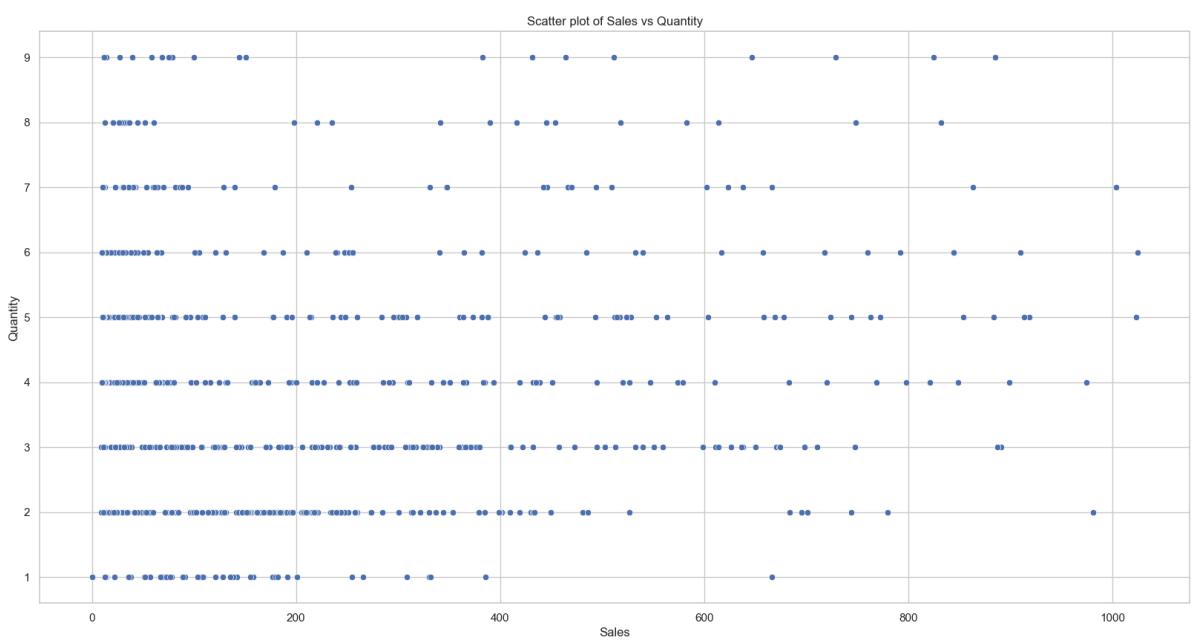
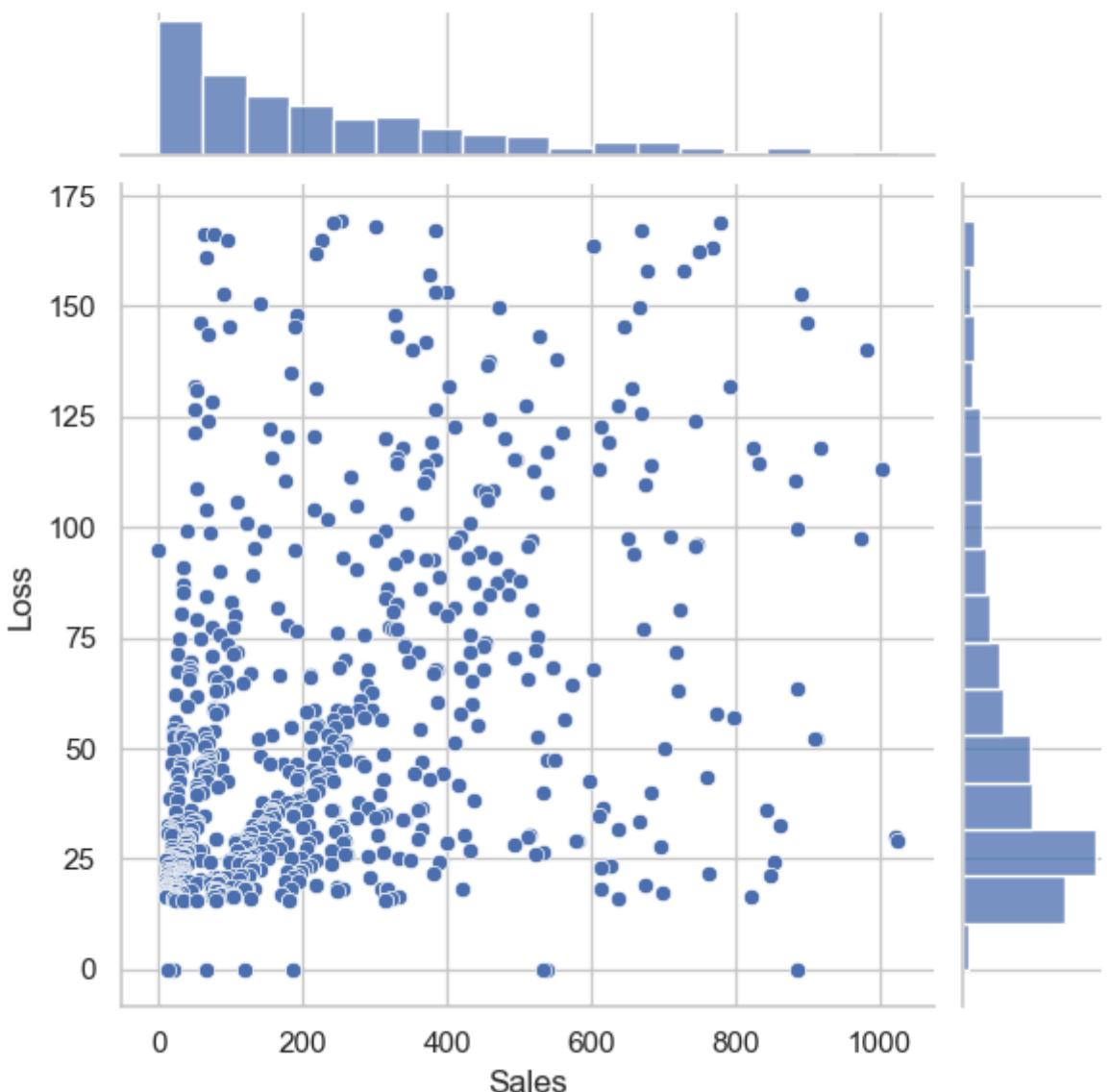
```

```
# Joint Plot
sns.jointplot(x=num_var1, y=num_var2, data=df, kind='scatter')
plt.xlabel(num_var1)
plt.ylabel(num_var2)
plt.suptitle(f'Joint Plot between {num_var1} and {num_var2}', y=1.02)
plt.show()
```

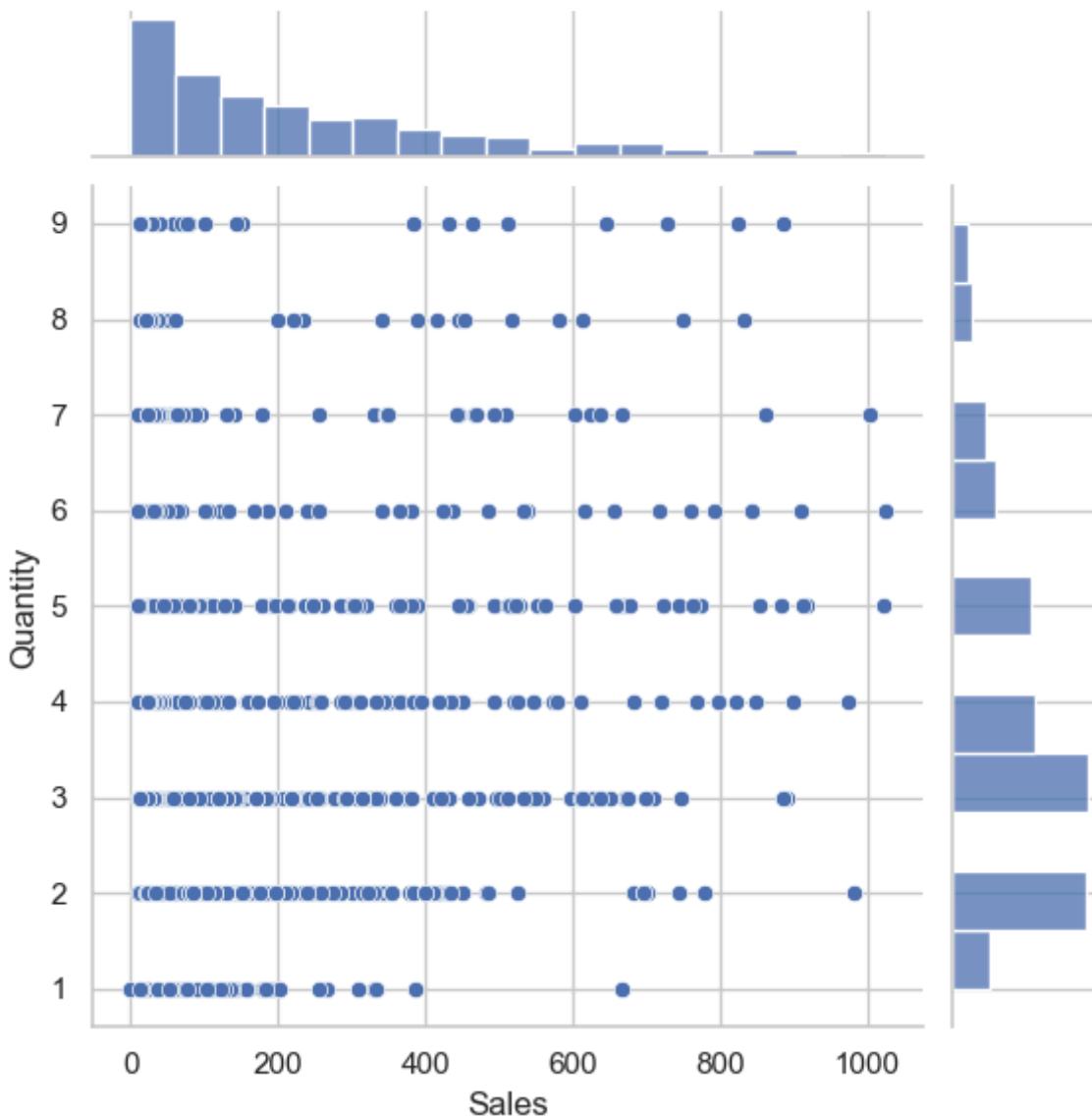
```
In [ ]: bivariate_numerical_analysis(global_super_store_df, 'Sales', 'Loss')
bivariate_numerical_analysis(global_super_store_df, 'Sales', 'Quantity')
```



## Joint Plot between Sales and Loss



## Joint Plot between Sales and Quantity



## Categorical Vs. Numerical

### Helper Functions

The following function `bivariate_analysis_categorical_numerical` will plot the following in a sub-plot between categorical and numerical features:

- Box Plot
- Violin Plot
- Strip Plot
- Bar Plot

```
In [ ]: def bivariate_analysis_categorical_numerical(df, cat_var, num_var):  
    """  
        Perform bivariate analysis and visualization between a categorical variable and  
        Produces Box Plot, Violin Plot, Strip Plot and Bar Plot  
        Parameters:  
        - df (DataFrame): Input DataFrame containing the data.  
        - cat_var (str): Name of the categorical variable.  
        - num_var (str): Name of the numerical variable.  
    """
```

```

fig, axes = plt.subplots(2, 2, figsize=(20, 15))
plt.suptitle(f'{cat_var} Vs. {num_var}', fontsize=16) # Set main title for the plots

# Box plot
sns.boxplot(x=cat_var, y=num_var, data=df, ax=axes[0, 0])
axes[0, 0].set_title(f'Box Plot of {num_var} by {cat_var}')
axes[0, 0].set_xlabel(cat_var)
axes[0, 0].set_ylabel(num_var)

# Violin plot
sns.violinplot(x=cat_var, y=num_var, data=df, ax=axes[0, 1])
axes[0, 1].set_title(f'Violin Plot of {num_var} by {cat_var}')
axes[0, 1].set_xlabel(cat_var)
axes[0, 1].set_ylabel(num_var)

# Strip plot
sns.stripplot(x=cat_var, y=num_var, data=df, ax=axes[1, 0])
axes[1, 0].set_title(f'Strip Plot of {num_var} by {cat_var}')
axes[1, 0].set_xlabel(cat_var)
axes[1, 0].set_ylabel(num_var)

# Bar plot
sns.barplot(x=cat_var, y=num_var, data=df, ax=axes[1, 1])
axes[1, 1].set_title(f'Bar Plot of {num_var} by {cat_var}')
axes[1, 1].set_xlabel(cat_var)
axes[1, 1].set_ylabel(num_var)

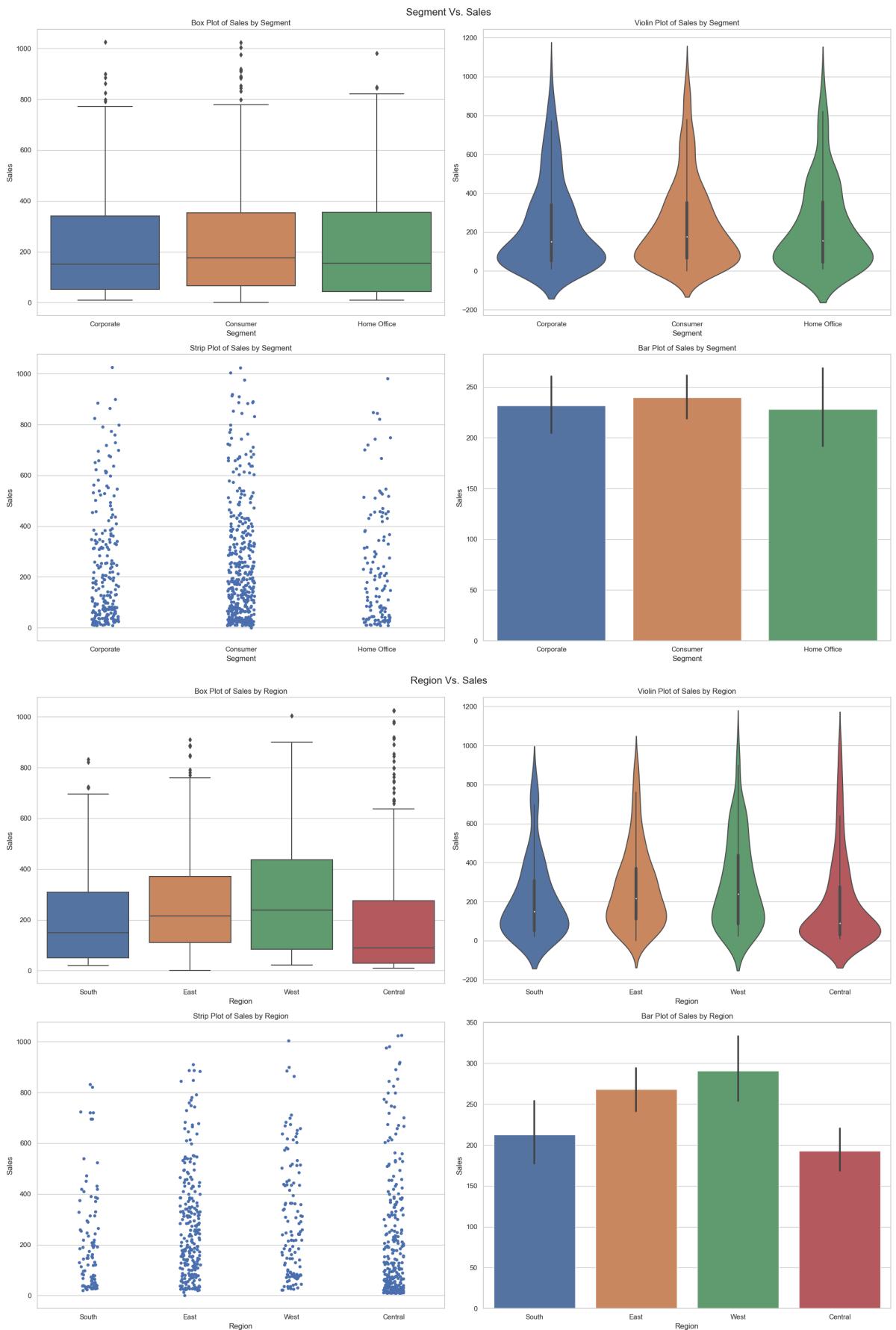
plt.tight_layout()
plt.show()

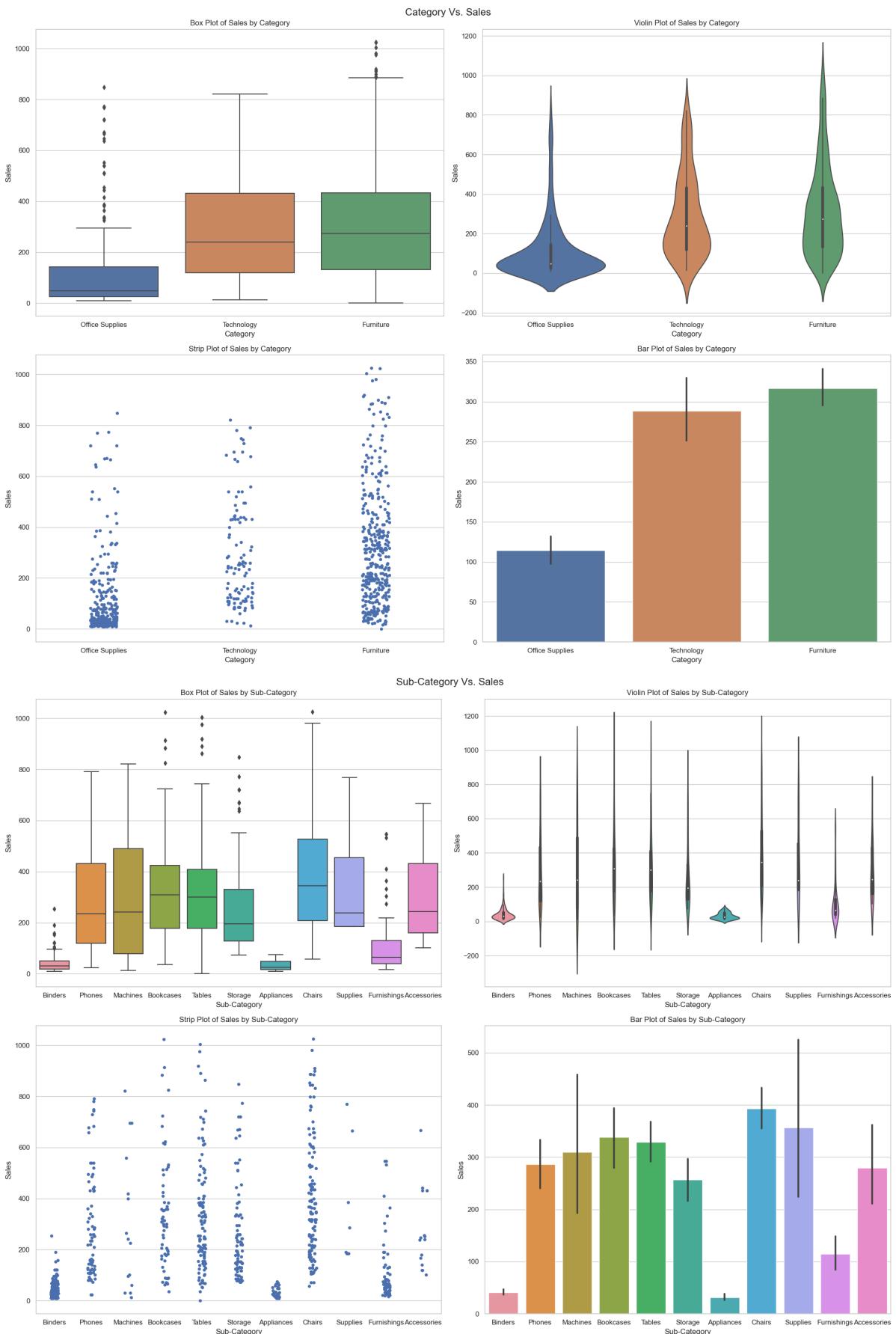
```

Then the function will run for all categories against one numerical feature.

## Categories Vs. Sales

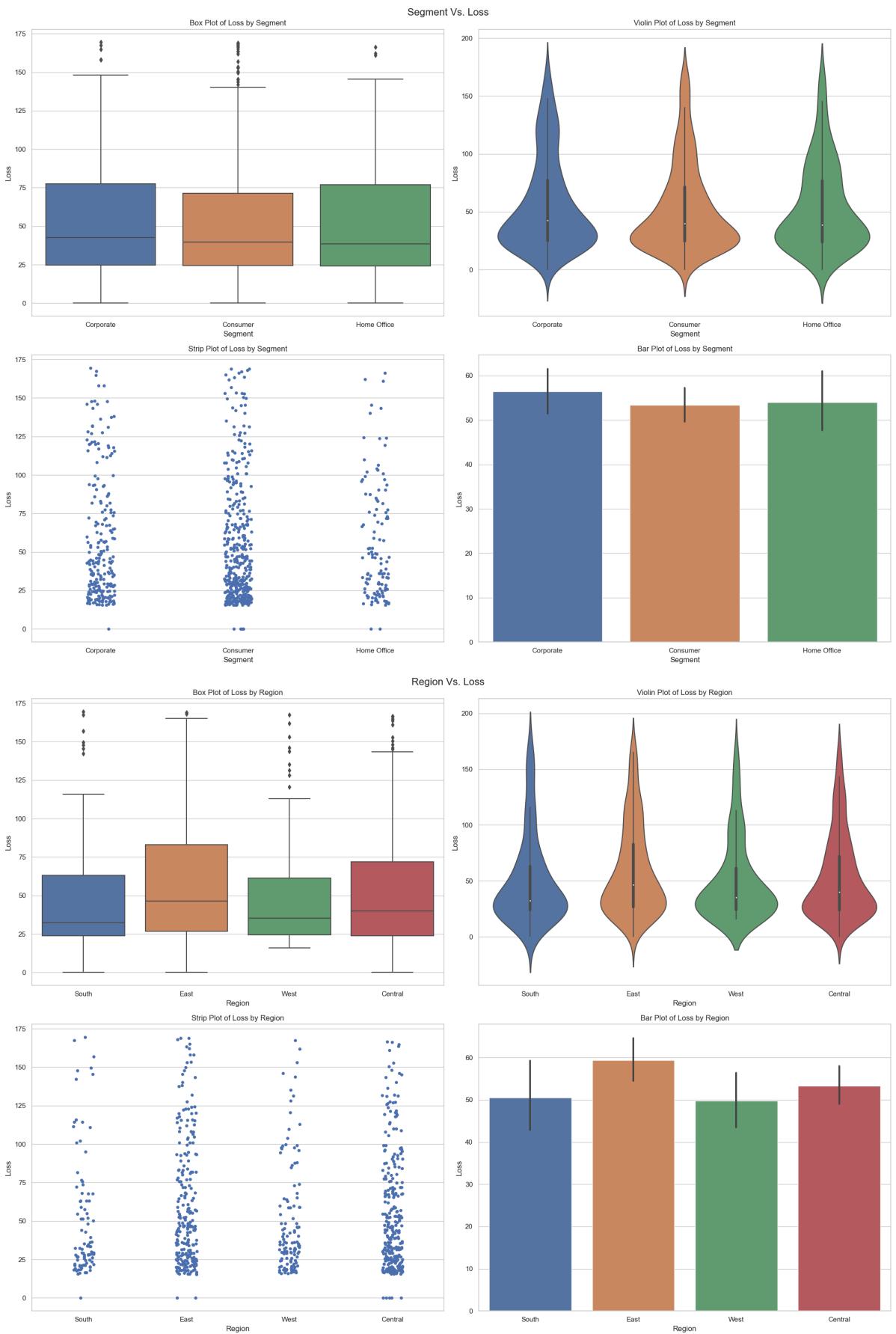
```
In [ ]: bivariate_analysis_categorical_numerical(global_super_store_df, 'Segment', 'Sales')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Region', 'Sales')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Category', 'Sales')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Sub-Category', 'Sales')
```

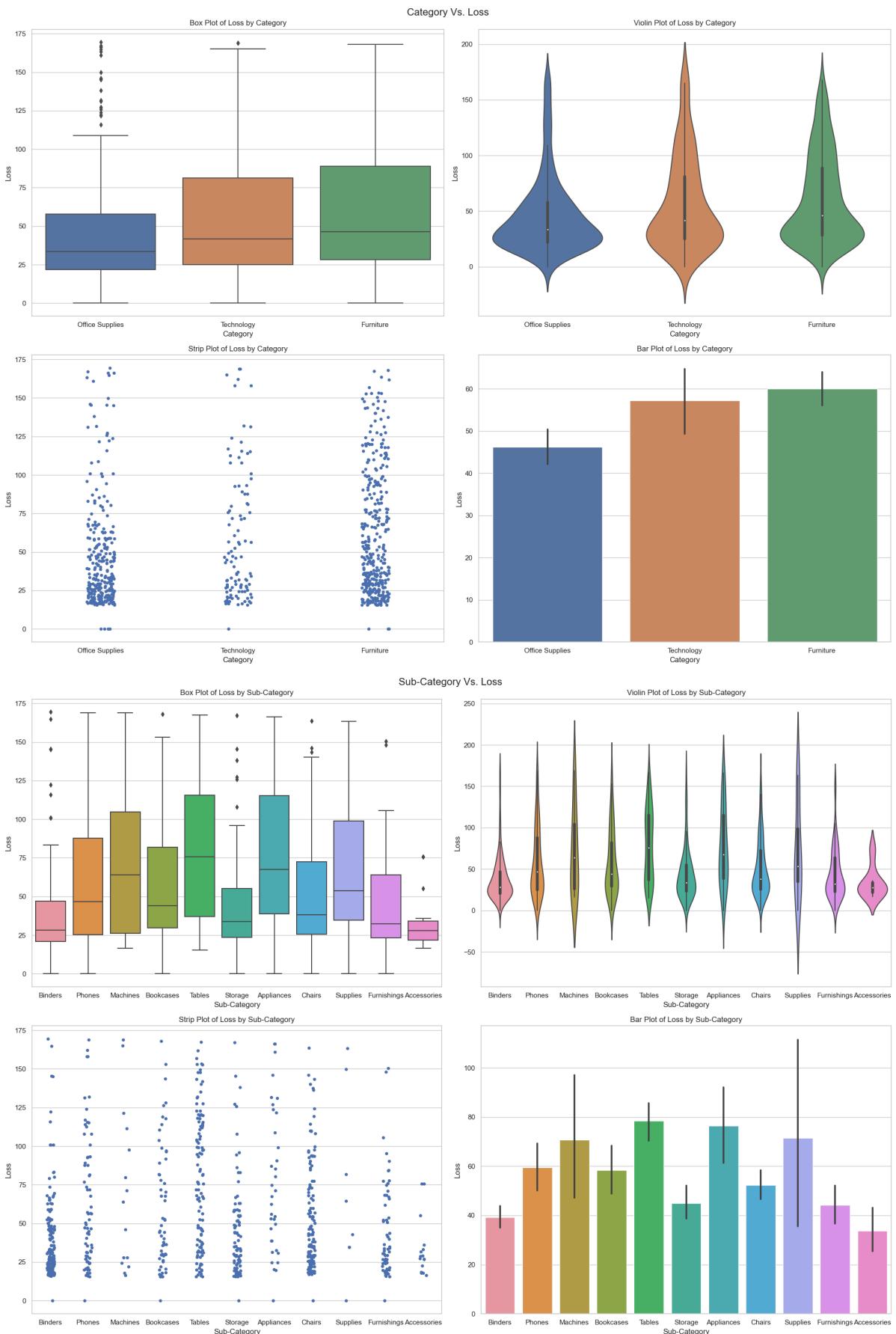




## Categories Vs. Loss

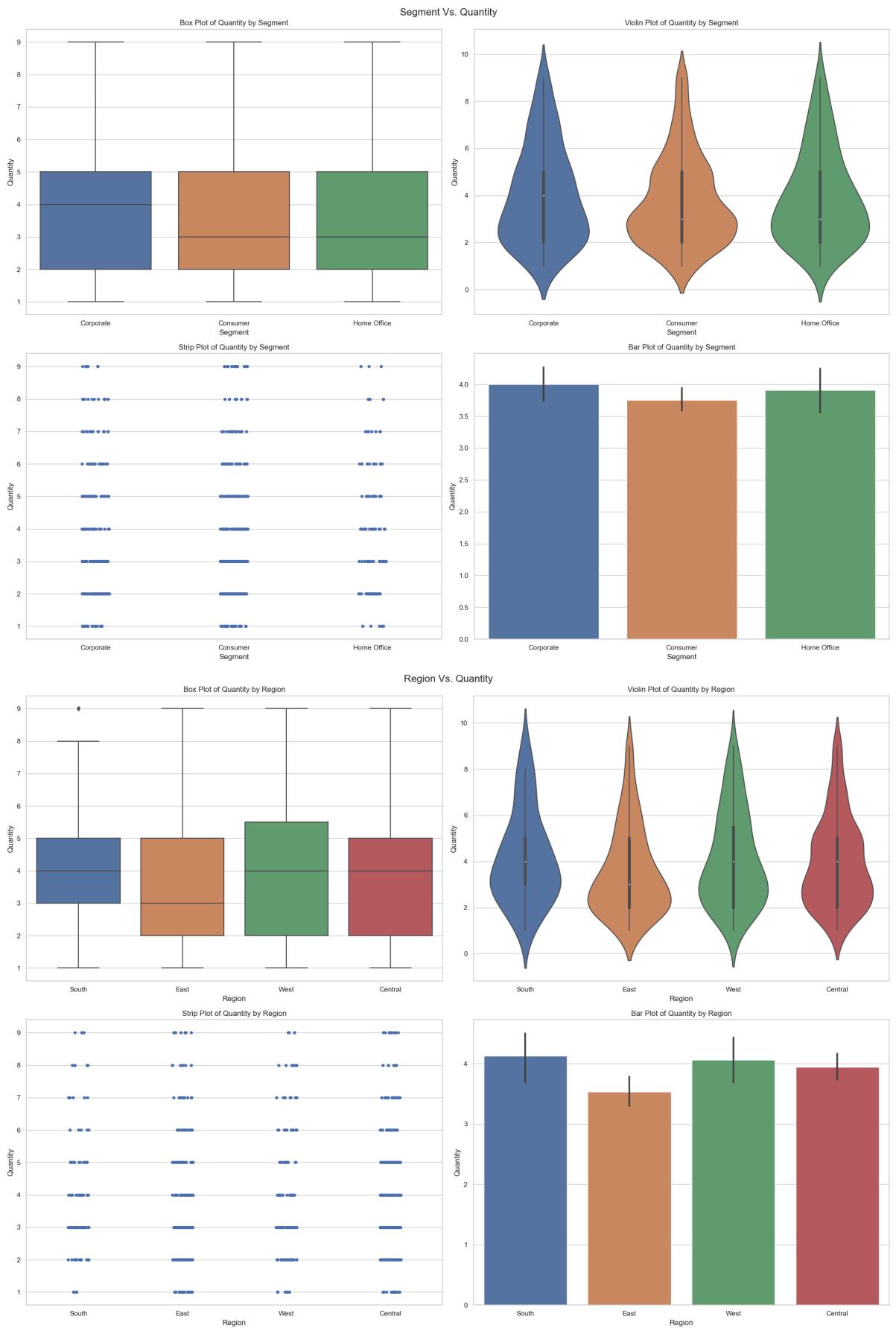
```
In [ ]: bivariate_analysis_categorical_numerical(global_super_store_df, 'Segment', 'Loss')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Region', 'Loss')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Category', 'Loss')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Sub-Category', 'Loss')
```





## Categories Vs. Quantity

```
In [ ]: bivariate_analysis_categorical_numerical(global_super_store_df, 'Segment', 'Quantity')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Region', 'Quantity')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Category', 'Quantity')
bivariate_analysis_categorical_numerical(global_super_store_df, 'Sub-Category', 'Quantity')
```





## Geo Analysis

Function to plot the numerical features on a US map.

```
In [ ]: def geo_analysis_by_state(df, feature_column):
    """
```

```
        Plots the feature percentages on a US map grouped by States
```

```

Args:
- df (DataFrame): DataFrame containing the feature data.
- feature_column (str): Name of the numerical feature column.

Returns:
- None
"""

base_folder_path = os.getcwd()
shp_path = '/data/usa-states-census-2014.shp'

us_states = gpd.read_file(f'{base_folder_path}{shp_path}')
us_states = us_states.to_crs("EPSG:3395")

# Group feature data based on States
feature_per_states = df.groupby(['State'], as_index=False)[[feature_column]].sum()

# Create a new column to see the Feature %
feature_percent_column_name = feature_column + ' %'

# Calculate the sum of the feature_column
total_feature_sum = df[feature_column].sum()

# Check if the total sum is zero
if total_feature_sum == 0:
    # If the total sum is zero, set all percentages to zero or handle the scenario
    feature_per_states[feature_percent_column_name] = 0
else:
    # Calculate percentages only if the total sum is not zero
    feature_per_states[feature_percent_column_name] = (feature_per_states[feature_column] / total_feature_sum) * 100

# Merge sales data with the US map based on state codes or names
merged_data = us_states.merge(feature_per_states, how='left', left_on='NAME', right_index=True)

# Filter out duplicate entries
merged_data = merged_data.drop_duplicates(subset=['NAME'])

# Fill NaN values with 0 for the percentage column
merged_data[feature_percent_column_name].fillna(0, inplace=True)

# Plot the map
fig, ax = plt.subplots(1, 1, figsize=(30, 10))
us_states.plot(ax=ax, color='lightgrey', edgecolor='black')
merged_data.plot(column=feature_percent_column_name, cmap='Spectral_r', linewidth=2)

# Annotate state names and sales percentages
texts = []
for idx, row in merged_data.iterrows():
    x = row['geometry'].centroid.x
    y = row['geometry'].centroid.y
    name = row['NAME']
    feature_percent = row[feature_percent_column_name]
    text = ax.text(x, y, name, fontsize=9, ha='center', va='center', color='black')
    texts.append(text)
    text = ax.text(x, y-0.5, f"{feature_percent:.2f}%", fontsize=9, ha='center', va='bottom', color='black')
    texts.append(text)

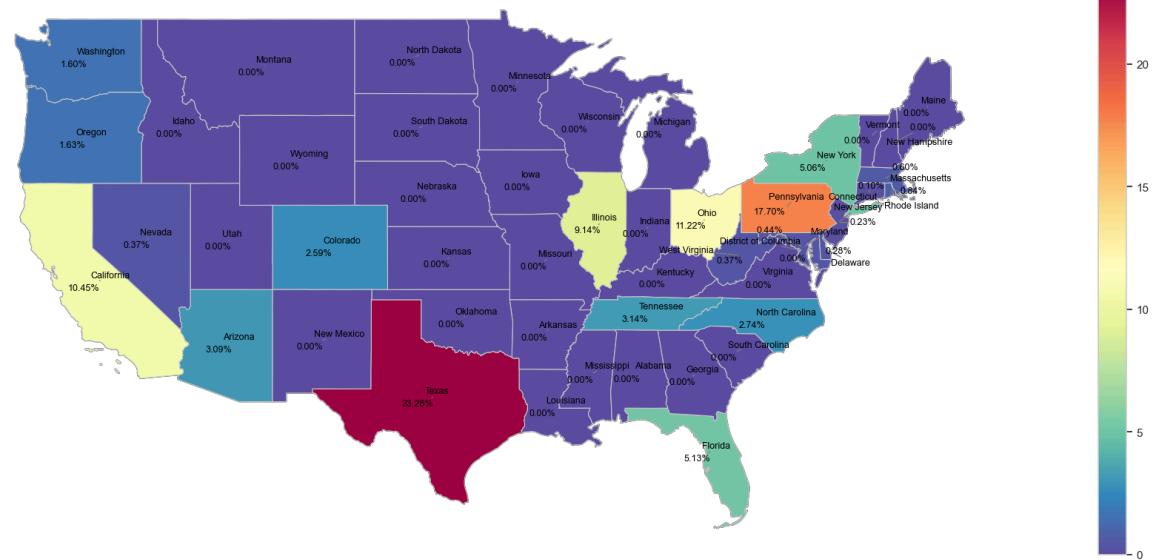
# Adjust text labels to avoid overlaps
adjust_text(texts, arrowprops=dict(arrowstyle='-', color='grey'))

ax.axis('off')
ax.set_title(f'{feature_column} percent by US State', loc='center', fontsize=20)
plt.show()

```

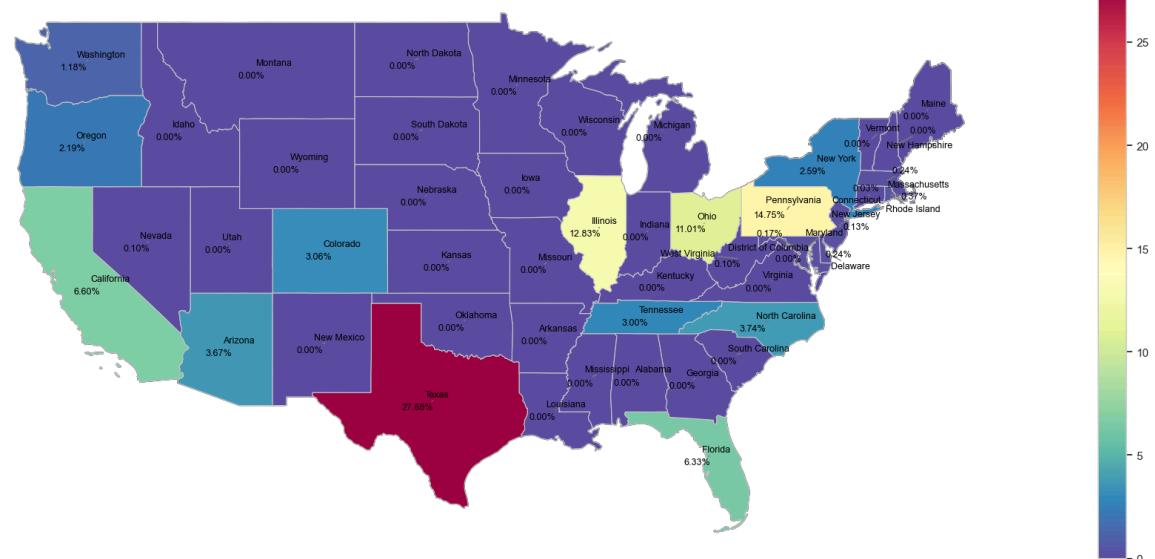
## Sales % based on US States

```
In [ ]: geo_analysis_by_state(global_super_store_df, 'Sales')
```



## Quantity % based on US States

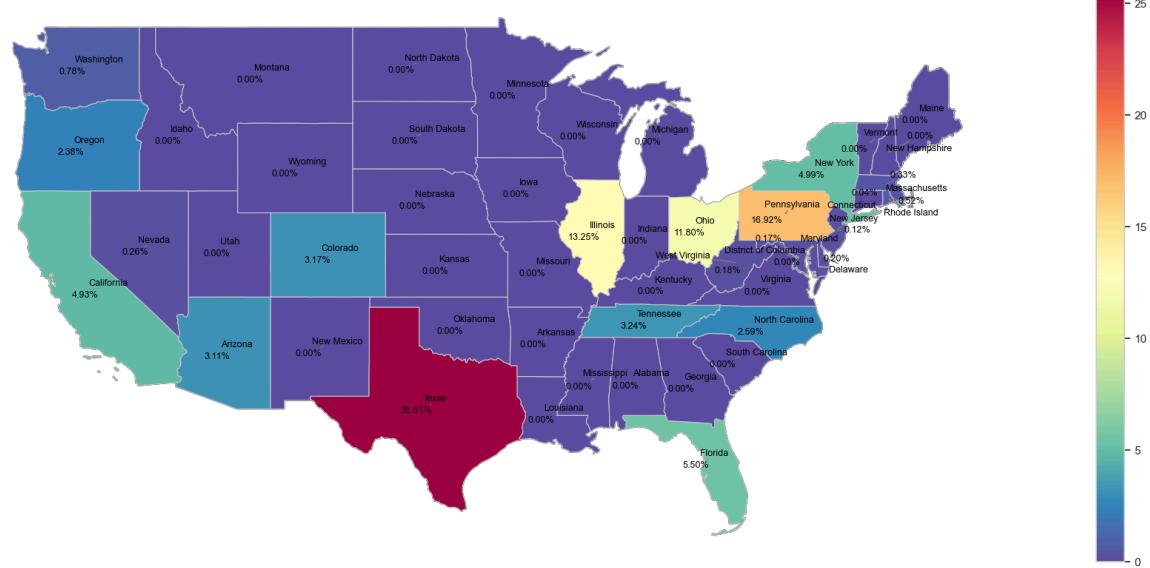
```
In [ ]: geo_analysis_by_state(global_super_store_df, 'Quantity')
```



## Loss % based on US States

```
In [ ]: geo_analysis_by_state(global_super_store_df, 'Loss')
```

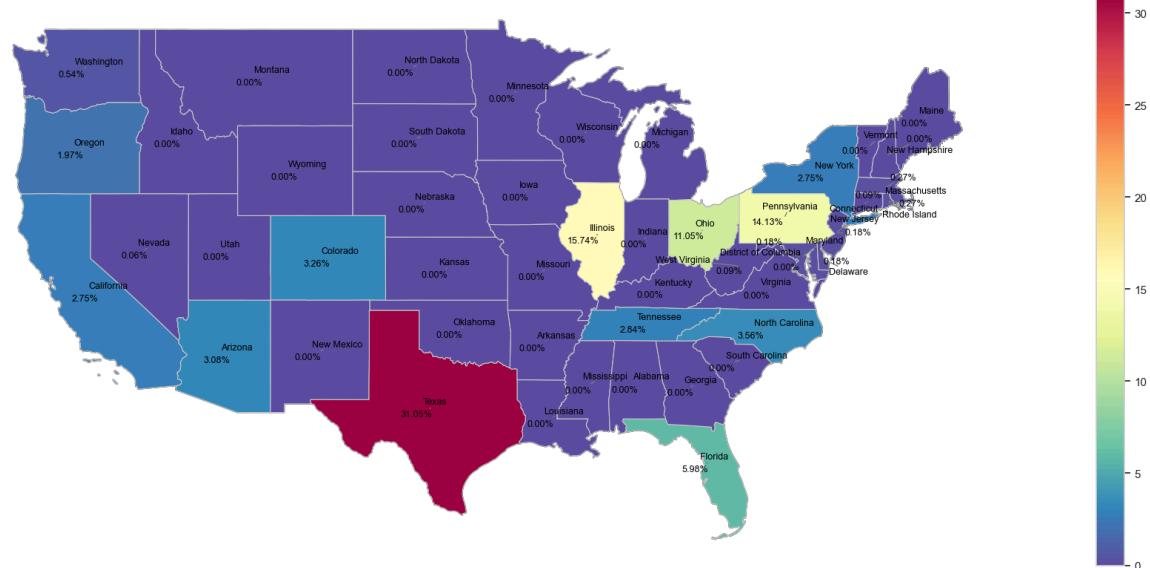
Loss percent by US State



### Discount % based on US States

```
In [ ]: geo_analysis_by_state(global_super_store_df, 'Discount')
```

Discount percent by US State



- Sales conclusion:

- Texas** has the most percentage of Sales with ~23%.
- The Central and South regions of the country collectively could not match the sales numbers of Texas.

- Quantity conclusion:

- Texas** sold ~28% of products.

- Losses conclusion:

- Texas** despite of making ~23% of Sales, has grossed an ~26% of losses.

## Conclusion

- Categorical Vs. Categorical Conclusion
  - Chi-square test of independence pointed us that the following categories has a significant association:
    - *Category Vs. Sub-Category*
    - *Region Vs. Category*
    - *Region Vs. Sub-Category*
  - Function `categorical_analysis` plotted cross-tabulation and stacked bar plot on the choose categories. Then `bivariate_categorical_figure_analysis` function would plot the pivot-table and heatmaps to given categorical features. In summary:
    - *Category Vs. Sub-Category*
      - Binders Sub-Category appeared on most number of Sales, close next is Chairs.
      - But Chairs has the highest sales value, followed by Tables.
      - Binders, and Chairs made almost the same value of loss.
    - *Region Vs. Category*
      - Central Region has made highest number of sales on Furniture Category, second close is Office Supplies Category, also from Central Region.
      - Same goes to the sales values, Central Region made the highest sales on Furniture, followed by Furniture category in the East Region.
      - Furniture from Central Region marks the highest loss as well.
    - *Region Vs. Sub-Category*
      - Binders Sub-Category on Central Region appeared on most number of Sales, close next is Phones on East Region.
      - But Chairs from Central Region has the highest sales value, followed by Phones on East Region.
      - Tables from East Region marks the highest loss, close second will be Phones on East Region.
- Numerical Vs. Numerical Conclusion
  - Sales and Discount correlation of -0.58
    - Indicates a moderate negative relationship between sales and discounts.
    - Meaning that when discounts increase, sales tend to decrease, and vice versa.
    - Since it is a moderate negative relationship, does not imply causation.
  - Sales and Loss correlation of 0.40
    - Indicates a moderate positive relationship between sales and loss.
    - Meaning that when sales increase, loss tend to increase, and vice versa.
    - Since it is a moderate positive relationship, does not imply causation.
  - The other numerical variable has either a weak positive/negative relationship, we cannot take fruitful decisions, especially causation.
- Categorical Vs. Numerical Conclusion
  - Consumer Segment has ~56% of the sales value, but Corporate Segment has the highest loss percentages.
  - Eastern Region is responsible for ~37% of the sales, most of it coming from Texas. Central Region marks the highest loss percentage.

- Furniture Category has the highest percentage of sales with ~53% and also the losses at ~39%.
- Appliance Sub-category made the highest loss at 241.38%.