```
# Importing necessary libraries for data analysis and visualization
import pandas as pd                                # Pandas for data manipulation an
import numpy as np                                 # NumPy for numerical operations
import matplotlib.pyplot as plt                    # Matplotlib for basic plotting
import seaborn as sns                              # Seaborn for statistical data vi
from statsmodels.tsa.seasonal import seasonal_decompose   # For decomposing time
import plotly.express as px                        # Plotly Express for interactive

# Setting display options to show three decimal places for floating-point numbers
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

LOADING DATASET

```
# Display all columns without truncation
pd.set_option('display.max_columns', None)

# Load car-related dataset from URL into 'stores' DataFrame
url = 'https://raw.githubusercontent.com/ArijitDhali/PrepInsta-DA-Week-6/main/tra
stores = pd.read_csv(url, encoding='unicode_escape')

# Display first two rows of the loaded DataFrame
stores.head(2)
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	Unitec States
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	Unitec States

Next steps: Generate code with stores   View recommended plots

```
# Remove 'Row ID' column from 'stores' DataFrame
del stores["Row ID"]

# Display first two rows
stores.head(2)
```

	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	
0	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Her
1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Consumer	United States	Her

Next steps: Generate code with stores View recommended plots

Preliminary Data Inspection

```
# Display data types of columns in the 'stores' DataFrame
stores.dtypes
```

```
Order ID          object
Order Date        object
Ship Date         object
Ship Mode         object
Customer ID       object
Customer Name     object
Segment           object
Country           object
City              object
State             object
Postal Code      float64
Region            object
Product ID        object
Category          object
Sub-Category      object
Product Name      object
Sales            float64
dtype: object
```

```
# Display concise information about the 'stores' DataFrame, including data types
stores.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9800 entries, 0 to 9799
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Order ID       9800 non-null   object 
 1   Order Date     9800 non-null   object 
 2   Ship Date      9800 non-null   object 
 3   Ship Mode      9800 non-null   object 
```

```
  4   Customer ID    9800 non-null   object 
  5   Customer Name  9800 non-null   object 
  6   Segment        9800 non-null   object 
  7   Country        9800 non-null   object 
  8   City           9800 non-null   object 
  9   State          9800 non-null   object 
  10  Postal Code    9789 non-null   float64
  11  Region         9800 non-null   object 
  12  Product ID     9800 non-null   object 
  13  Category       9800 non-null   object 
  14  Sub-Category   9800 non-null   object 
  15  Product Name   9800 non-null   object 
  16  Sales          9800 non-null   float64
dtypes: float64(2), object(15)
memory usage: 1.3+ MB
```

```
stores.shape
```

```
(9800, 17)
```

```
stores.describe()
```

	Postal Code	Sales
count	9789.000	9800.000
mean	55273.322	230.769
std	32041.223	626.652
min	1040.000	0.444
25%	23223.000	17.248
50%	58103.000	54.490
75%	90008.000	210.605
max	99301.000	22638.480

Data Cleaning and Viewing

1. Handling the missing values and standardizing Date values

This code checks for duplicate rows in the 'stores' DataFrame and calculates the total number of duplicate rows.

```
# Check for duplicate rows in 'stores' DataFrame
stores.duplicated().sum()
```

```
1
```

```
# Drop duplicate rows from the 'stores' DataFrame
stores = stores.drop_duplicates()
stores.duplicated().sum()
```

```
    0
```

```
# Check if there are any missing values in the data
stores.isnull().sum()
```

```
    Order ID          0
    Order Date        0
    Ship Date         0
    Ship Mode         0
    Customer ID       0
    Customer Name     0
    Segment           0
    Country           0
    City              0
    State             0
    Postal Code      11
    Region            0
    Product ID        0
    Category          0
    Sub-Category      0
    Product Name      0
    Sales             0
    dtype: int64
```

```
stores[stores['Postal Code'].isnull()]
```

	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country
2234	CA-2018-104066	05/12/2018	10/12/2018	Standard Class	QJ-19255	Quincy Jones	Corporate	United States
5274	CA-2016-162887	07/11/2016	09/11/2016	Second Class	SV-20785	Stewart Visinsky	Consumer	United States
8798	US-2017-150140	06/04/2017	10/04/2017	Standard Class	VM-21685	Valerie Mitchum	Home Office	United States
9146	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States
9147	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States
9148	US-2017-165505	23/01/2017	27/01/2017	Standard Class	CB-12535	Claudia Bergmann	Corporate	United States
9386	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States
9387	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States
9388	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States
9389	US-2018-127292	19/01/2018	23/01/2018	Standard Class	RM-19375	Raymond Messe	Consumer	United States
9741	CA-2016-117086	08/11/2016	12/11/2016	Standard Class	QJ-19255	Quincy Jones	Corporate	United States

```
# Replace null values in the 'Postal Code' column of the 'stores' DataFrame with
stores['Postal Code'] = stores['Postal Code'].replace(np.nan, 5401)

# Check for missing values after the replacement
stores.isnull().sum()
```

```
<ipython-input-13-adc6d7488417>:2: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  stores['Postal Code'] = stores['Postal Code'].replace(np.nan, 5401)
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country          0
City             0
State            0
Postal Code      0
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
dtype: int64
```

```
stores.shape
```

```
(9799, 17)
```

```
# Convert 'Order Date' to the standard format 'YYYY-MM-DD'
stores['Order Date'] = pd.to_datetime(stores['Order Date'], format='%d/%m/%Y')
stores['Order Date']
```

```
<ipython-input-15-35c2899d2985>:2: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  stores['Order Date'] = pd.to_datetime(stores['Order Date'], format='%d/%m/%
0       2017-11-08
1       2017-11-08
2       2017-06-12
3       2016-10-11
4       2016-10-11
           ...    
9795    2017-05-21
9796    2016-01-12
9797    2016-01-12
```

```
9798   2016-01-12
9799   2016-01-12
Name: Order Date, Length: 9799, dtype: datetime64[ns]
```

```
# Convert 'Ship Date' to the standard format 'YYYY-MM-DD'
stores['Ship Date'] = pd.to_datetime(stores['Ship Date'], format='%d/%m/%Y')
stores['Ship Date']
```

```
0      2017-11-11
1      2017-11-11
2      2017-06-16
3      2016-10-18
4      2016-10-18
          ...
9795   2017-05-28
9796   2016-01-17
9797   2016-01-17
9798   2016-01-17
9799   2016-01-17
Name: Ship Date, Length: 9799, dtype: datetime64[ns]
```

```
# Calculate the days taken to ship and create a new 'Days to Ship' column in the
stores['Days to Ship'] = (stores['Ship Date'] - stores['Order Date']).dt.days
stores['Days to Ship']
```

```
0       3
1       3
2       4
3       7
4       7
       ..
9795    7
9796    5
9797    5
9798    5
9799    5
Name: Days to Ship, Length: 9799, dtype: int64
```

```
# Calculate the days taken to ship and create a new 'Days to Ship' column in the
stores['Days to Ship'] = (stores['Ship Date'] - stores['Order Date']).dt.days
stores['Days to Ship']
```

```
0       3
1       3
2       4
3       7
4       7
       ..
9795    7
9796    5
9797    5
9798    5
9799    5
Name: Days to Ship, Length: 9799, dtype: int64
```

```
# Display a random sample of one row from the 'stores' DataFrame
stores.sample()
```

	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City
1831	CA-2018-145884	2018-10-21	2018-10-21	Same Day	SL-20155	Sara Luxemburg	Home Office	United States	Muskogee

2. Viewing Order ID Dataset

This code prints the unique values and the total number of unique values in the 'Order ID' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Order ID' c
print('Unique Values: \n', stores['Order ID'].unique())
print('Total Number of Unique Values: ', stores['Order ID'].nunique())
```

```
Unique Values:
 ['CA-2017-152156' 'CA-2017-138688' 'US-2016-108966' ... 'CA-2015-127166'
 'CA-2017-125920' 'CA-2016-128608']
Total Number of Unique Values:  4922
```

3. Viewing Order Date Dataset

This code prints the unique values and the total number of unique values in the 'Order Date' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Order Date'
print('Unique Values: \n', stores['Order Date'].unique())
print('Total Number of Unique Values: ', stores['Order Date'].nunique())
```

```
Unique Values:
 ['2017-11-08T00:00:00.000000000' '2017-06-12T00:00:00.000000000'
 '2016-10-11T00:00:00.000000000' ... '2015-06-18T00:00:00.000000000'
 '2018-02-28T00:00:00.000000000' '2016-05-09T00:00:00.000000000']
Total Number of Unique Values:  1230
```

4. Viewing Ship Date Dataset

This code prints the unique values and the total number of unique values in the 'Ship Date' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Ship Date'
print('Unique Values: \n', stores['Ship Date'].unique())
print('Total Number of Unique Values: ', stores['Ship Date'].nunique())
```

```
Unique Values:
 ['2017-11-11T00:00:00.000000000' '2017-06-16T00:00:00.000000000'
 '2016-10-18T00:00:00.000000000' ... '2015-03-12T00:00:00.000000000'
 '2018-04-06T00:00:00.000000000' '2016-05-13T00:00:00.000000000']
Total Number of Unique Values:  1326
```

5. Viewing Days to Ship Dataset

This code prints the unique values and the total number of unique values in the 'Days to Ship' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Days to Shi
print('Unique Values: \n', stores['Days to Ship'].unique())
print('Total Number of Unique Values: ', stores['Days to Ship'].nunique())
```

```
Unique Values:
 [3 4 7 5 2 6 1 0]
Total Number of Unique Values:  8
```

6. Viewing Customer ID Dataset

This code prints the unique values and the total number of unique values in the 'Customer ID' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Customer ID
print('Unique Values: \n', stores['Customer ID'].unique())
print('Total Number of Unique Values: ', stores['Customer ID'].nunique())
```

```
  EH-13990   EH-13765   SC-20230   JE-15715   AC-10615   JD-16015
 'CB-12415' 'JS-16030' 'LW-17215' 'SC-20800' 'AM-10705' 'RH-19510'
 'CT-11995' 'MC-17590' 'CC-12610' 'KA-16525' 'TC-20980' 'BF-11080'
 'MM-17260' 'AH-10120' 'BW-11200' 'SW-20245' 'BS-11665' 'RF-19345'
 'WB-21850' 'TB-21625' 'AF-10870' 'RB-19435' 'KF-16285' 'JH-15820'
 'IL-15100' 'PB-18805' 'RH-19600' 'AW-10930' 'ML-17410' 'DB-13555'
 'MH-17620' 'DK-13375' 'BT-11440' 'DB-13405' 'TG-21310' 'BF-11005'
 'JM-16195' 'MZ-17335' 'MW-18220' 'MV-17485' 'SM-20320' 'TP-21415'
 'JK-15625' 'PJ-18835' 'RS-19420' 'SV-20935' 'BC-11125' 'EM-13825'
 'BM-11575' 'KN-16705' 'KW-16570' 'SC-20260' 'CV-12295' 'SG-20605'
 'TM-21010' 'EM-13810' 'ML-18040' 'CR-12580' 'AZ-10750' 'PW-19240'
 'SC-20380' 'CM-11935' 'GM-14695' 'TB-21250' 'JM-15865' 'SC-20575'
 'LS-17200' 'RR-19315' 'DB-12910' 'TT-21220' 'LO-17170' 'KD-16615'
 'NB-18580' 'BD-11635' 'CM-12235' 'EN-13780' 'KN-16450' 'BO-11350'
 'AG-10300' 'MC-17635' 'TA-21385' 'JF-15295' 'TT-21265' 'SB-20170'
 'CL-12700' 'HG-15025' 'NL-18310' 'RR-19525' 'TC-21295' 'SV-20785'
 'BE-11410' 'SC-20680' 'DF-13135' 'FH-14350' 'MS-17530' 'RH-19555'
 'GA-14515' 'JP-16135' 'Dp-13240' 'MO-17950' 'DL-13330' 'MH-18025'
 'DR-12940' 'DM-13015' 'CA-11965' 'AC-10660' 'DM-13345' 'VF-21715'
 'CC-12370' 'BF-11275' 'HG-14845' 'BP-11155' 'EM-14140' 'MA-17995'
 'AY-10555' 'GB-14575' 'JB-16045' 'MG-17875' 'SR-20425' 'JB-16000'
 'DM-12955' 'TC-21475' 'SW-20350' 'RE-19450' 'BF-11215' 'KB-16405'
 'JG-15310' 'EC-14050' 'EB-14110' 'JP-15460' 'CS-11845' 'GH-14410'
 'PT-19090' 'JL-15130' 'AH-10030' 'CC-12475' 'DW-13195' 'SJ-20215'
 'BG-11740' 'LB-16795' 'CM-11815' 'EH-14185' 'TS-21505' 'PR-18880'
 'LC-17050' 'CS-12490' 'DH-13075' 'JO-15145' 'AH-10690' 'HJ-14875'
 'MH-17455' 'RD-19930' 'SC-20020' 'SU-20665' 'FC-14335' 'RB-19330'
 'NC-18535' 'DB-12970' 'MF-17665' 'RM-19750' 'VM-21835' 'EJ-13720'
 'NC-18415' 'LS-17230' 'KE-16420' 'DH-13675' 'PF-19120' 'VT-21700'
 'MG-17695' 'SP-20920' 'CS-12175' 'DK-12895' 'KM-16660' 'AA-10645'
 'DD-13570' 'AH-10465' 'TP-21565' 'EK-13795' 'CR-12820' 'SG-20890'
 'AH-10585' 'NF-18475' 'BS-11365' 'SH-20635' 'RD-19720' 'AG-10330'
 'GR-14560' 'VS-21820' 'TM-21490' 'TS-21430' 'DB-13360' 'NR-18550'
 'CS-11860' 'MF-18250' 'LW-17125' 'AR-10345' 'KS-16300' 'AB-10015'
 'LR-17035' 'SS-20410' 'JM-15580' 'JK-15325' 'DM-13525' 'ML-18265'
 'MH-17290' 'FC-14245' 'TH-21115' 'JK-16090' 'SB-20185' 'BG-11035'
 'BW-11065' 'MG-18205' 'DO-13645' 'BP-11050' 'TS-21655' 'EM-14200'
 'AO-10810' 'MH-17440' 'SS-20515' 'LD-16855' 'VP-21760' 'TC-21145'
 'IM-15055' 'AR-10570' 'CM-12715' 'FW-14395' 'LC-16960' 'HE-14800'
 'BD-11560' 'HD-14785' 'CJ-11875' 'RS-19870' 'SC-20845' 'RE-19405'
 'SM-20905']
Total Number of Unique Values:  793
```

7. Viewing Customer Name Dataset

This code prints the unique values and the total number of unique values in the 'Customer Name' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Customer Na
print('Unique Values: \n', stores['Customer Name'].unique())
print('Total Number of Unique Values: ', stores['Customer Name'].nunique())
```

```
  'Maris LaWare' 'Dorothy Badders' 'Matt Hagelstein' 'Dennis Kane'
  'Bobby Trafton' 'Denny Blanton' 'Toby Gnade' 'Barry Franz'
  'Justin MacKendrick' 'Maria Zettner' 'Mitch Webber' 'Mark Van Huff'
  'Sean Miller' 'Tom Prescott' 'Jim Karlsson' 'Patrick Jones'
  'Ricardo Sperren' 'Susan Vittorini' 'Becky Castell' 'Elizabeth Moffitt'
  'Brendan Murry' 'Kristina Nunn' 'Kelly Williams' 'Scott Cohen'
  'Christina VanderZanden' 'Speros Goranitis' 'Tamara Manning'
  'Eleni McCrary' 'Michelle Lonsdale' 'Clay Rozendal' 'Annie Zypern'
  'Pierre Wener' 'Shahid Collister' 'Carlos Meador' 'Greg Maxwell'
  'Tim Brockman' 'John Murray' 'Sonia Cooley' 'Luke Schmidt' 'Ralph Ritter'
  'Daniel Byrd' 'Thomas Thornton' 'Lori Olson' 'Ken Dana' 'Nicole Brennan'
  'Brian Derr' 'Chris McAfee' 'Edward Nazzal' 'Kean Nguyen' 'Bill Overfelt'
  'Aleksandra Gannaway' 'Matthew Clasen' 'Tom Ashbrook' 'Jason Fortune-'
  'Tim Taslimi' 'Sarah Bern' 'Craig Leslie' 'Hunter Glantz'
  'Nancy Lomonaco' 'Rick Reed' 'Toby Carlisle' 'Stewart Visinsky'
  'Bobby Elias' 'Steve Carroll' 'David Flashing' 'Fred Harton'
  'MaryBeth Skach' 'Ritsa Hightower' 'George Ashbrook' 'Julie Prescott'
  'Dean percer' 'Michael Oakman' 'Denise Leinenbach' 'Michelle Huthwaite'
  'Daniel Raglin' 'Darrin Martin' 'Carol Adams' 'Anna Chung'
  'Denise Monton' 'Vicky Freymann' 'Christopher Conant' 'Beth Fritzler'
  'Harry Greene' 'Becky Pak' 'Eugene Moren' 'Michelle Arnett' 'Andy Yotov'
  'Giulietta Baptist' 'Julia Barnett' 'Michael Grace' 'Sharelle Roach'
  'Joy Bell-' 'Dario Medina' 'Tony Chapman' 'Sean Wendt' 'Richard Eichhorn'
  'Benjamin Farhat' 'Katrina Bavinger' 'Jason Gross' 'Erin Creighton'
  'Eugene Barchas' 'Jennifer Patt' 'Cari Sayre' 'Gary Hansen'
  'Pete Takahito' 'Jack Lebron' 'Aaron Hawkins' 'Cindy Chapman'
  'David Wiener' 'Sarah Jordon' 'Bruce Geld' 'Laurel Beltran'
  'Candace McMahon' 'Evan Henry' 'Tony Sayre' 'Patrick Ryan' 'Liz Carlisle'
  'Cindy Schnelling' 'Dave Hallsten' "Jack O'Briant" 'Anna HÃ¤berlin'
  'Heather Jas' 'Mark Hamilton' "Russell D'Ascenzo" 'Sam Craven'
  'Stephanie Ulpright' 'Fred Chung' 'Randy Bradley' 'Nick Crebassa'
  'Darren Budd' 'Maureen Fritzler' 'Roland Murray' 'Vivian Mathis'
  'Ed Jacobs' 'Nathan Cano' 'Lycoris Saunders' 'Katrina Edelman'
  'Duane Huffman' 'Peter Fuller' 'Valerie Takahito' 'Maureen Gnade'
  'Susan Pistek' 'Charles Sheldon' 'Dana Kaydos' 'Khloe Miller'
  'Anna Andreadi' 'Dorothy Dickinson' 'Amy Hunt' 'Tracy Poddar'
  'Eileen Kiefer' 'Cyra Reiten' 'Susan Gilcrest' 'Angele Hood'
  'Neil FranzÃ¶sisch' 'Bill Shonely' 'Stefanie Holloman' 'Roger Demir'
  'Alex Grayson' 'Georgia Rosenberg' 'Vivek Sundaresam' 'Tony Molinari'
  'Tom Stivers' 'Dennis Bolton' 'Nick Radford' 'Cari Schnelling'
  'Monica Federle' 'Liz Willingham' 'Alex Russell' 'Karen Seio'
  'Aaron Bergman' 'Lisa Ryan' 'Shahid Shariari' 'Jill Matthias'
  'Jason Klamczynski' 'Don Miller' 'Muhammed Lee' 'Marc Harrigan'
  'Frank Carlisle' 'Thea Hudgings' 'Juliana Krohn' 'Sarah Brown'
  'Barry Gonzalez' 'Barry Weirich' 'Mitch Gastineau' "Doug O'Connell"
  'Barry Pond' 'Trudy Schmidt' 'Evan Minnotte' "Anthony O'Donnell"
  'Mark Haberlin' 'Shirley Schmidt' 'Lela Donovan' 'Victoria Pisteka'
  'Theresa Coyne' 'Ionia McGrath' 'Anemone Ratner' 'Craig Molinari'
  'Fred Wasserman' 'Lindsay Castell' 'Harold Engle' 'Brendan Dodson'
  'Harold Dahlen' 'Carl Jackson' 'Roy Skaria' 'Sung Chung'
  'Ricardo Emerson' 'Susan MacKendrick']
Total Number of Unique Values:  793
```

8. Viewing Segment Dataset

This code prints the unique values and the total number of unique values in the 'Segment' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Segment' co
print('Unique Values: \n', stores['Segment'].unique())
print('Total Number of Unique Values: ', stores['Segment'].nunique())
```

```
    Unique Values:
     ['Consumer' 'Corporate' 'Home Office']
    Total Number of Unique Values:  3
```

9. Viewing Country Dataset

This code prints the unique values and the total number of unique values in the 'Country' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Country' co
print('Unique Values: \n', stores['Country'].unique())
print('Total Number of Unique Values: ', stores['Country'].nunique())
```

```
    Unique Values:
     ['United States']
    Total Number of Unique Values:  1
```

10. Viewing City Dataset

This code prints the unique values and the total number of unique values in the 'City' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'City' colum
print('Unique Values: \n', stores['City'].unique())
print('Total Number of Unique Values: ', stores['City'].nunique())
```

```
 'Elmhurst' 'Passaic' 'North Charleston' 'Newport News' 'Jamestown'
 'Mishawaka' 'La Quinta' 'Tallahassee' 'Nashville' 'Bellingham'
 'Woodstock' 'Haltom City' 'Wheeling' 'Summerville' 'Hot Springs'
 'Englewood' 'Las Cruces' 'Hoover' 'Frisco' 'Vacaville' 'Waukesha'
 'Bakersfield' 'Pompano Beach' 'Corpus Christi' 'Redondo Beach' 'Orlando'
 'Orange' 'Lake Charles' 'Highland Park' 'Hempstead' 'Noblesville'
 'Apple Valley' 'Mount Pleasant' 'Sterling Heights' 'Eau Claire' 'Pharr'
 'Billings' 'Gresham' 'Chattanooga' 'Meridian' 'Bolingbrook' 'Maple Grove'
 'Woodland' 'Missouri City' 'Pearland' 'San Mateo' 'Grand Rapids'
 'Visalia' 'Overland Park' 'Temecula' 'Yucaipa' 'Revere' 'Conroe'
 'Tinley Park' 'Dubuque' 'Dearborn Heights' 'Santa Fe' 'Hickory'
 'Carol Stream' 'Saint Cloud' 'North Miami' 'Plantation'
 'Port Saint Lucie' 'Rock Hill' 'Odessa' 'West Allis' 'Chula Vista'
 'Manhattan' 'Altoona' 'Thornton' 'Champaign' 'Texarkana' 'Edinburg'
 'Baytown' 'Greenwood' 'Woonsocket' 'Superior' 'Bedford' 'Covington'
 'Broken Arrow' 'Miramar' 'Hollywood' 'Deer Park' 'Wichita' 'Mcallen'
 'Iowa City' 'Boise' 'Cranston' 'Port Arthur' 'Citrus Heights'
 'The Colony' 'Daytona Beach' 'Bullhead City' 'Portage' 'Fargo' 'Elkhart'
 'San Gabriel' 'Margate' 'Sandy Springs' 'Mentor' 'Lawton' 'Hampton'
 'Rome' 'La Crosse' 'Lewiston' 'Hattiesburg' 'Danville' 'Logan'
 'Waterbury' 'Athens' 'Avondale' 'Marietta' 'Yuma' 'Wausau' 'Pasco'
 'Oak Park' 'Pensacola' 'League City' 'Gaithersburg' 'Lehi' 'Tuscaloosa'
 'Moreno Valley' 'Georgetown' 'Loveland' 'Chandler' 'Helena' 'Kirkwood'
 'Waco' 'Frankfort' 'Bethlehem' 'Grand Island' 'Woodbury' 'Rogers'
 'Clovis' 'Jupiter' 'Santa Barbara' 'Cedar Hill' 'Norfolk' 'Draper'
 'Ann Arbor' 'La Mesa' 'Pocatello' 'Holland' 'Milford' 'Buffalo Grove'
 'Lake Forest' 'Redding' 'Chico' 'Utica' 'Conway' 'Cheyenne' 'Owensboro'
 'Caldwell' 'Kenner' 'Nashua' 'Bartlett' 'Redwood City' 'Lebanon'
 'Santa Maria' 'Des Plaines' 'Longview' 'Hendersonville' 'Waterloo'
 'Cambridge' 'Palatine' 'Beverly' 'Eugene' 'Oxnard' 'Renton' 'Glenview'
 'Delray Beach' 'Commerce City' 'Texas City' 'Wilson' 'Rio Rancho'
 'Goldsboro' 'Montebello' 'El Cajon' 'Beaumont' 'West Palm Beach'
 'Abilene' 'Normal' 'Saint Charles' 'Camarillo' 'Hillsboro' 'Burbank'
 'Modesto' 'Garden City' 'Atlantic City' 'Longmont' 'Davis' 'Morgan Hill'
 'Clifton' 'Sheboygan' 'East Point' 'Rapid City' 'Andover' 'Kissimmee'
 'Shelton' 'Danbury' 'Sanford' 'San Marcos' 'Greeley' 'Mansfield' 'Elyria'
 'Twin Falls' 'Coral Gables' 'Romeoville' 'Marlborough' 'Laurel' 'Bryan'
 'Pine Bluff' 'Aberdeen' 'Hagerstown' 'East Orange' 'Arlington Heights'
 'Oswego' 'Coon Rapids' 'San Clemente' 'San Luis Obispo' 'Springdale']
Total Number of Unique Values:  529
```

11. Viewing State Dataset

This code prints the unique values and the total number of unique values in the 'State' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'State' colu
print('Unique Values: \n', stores['State'].unique())
print('Total Number of Unique Values: ', stores['State'].nunique())
```

```
Unique Values:
 ['Kentucky' 'California' 'Florida' 'North Carolina' 'Washington' 'Texas'
 'Wisconsin' 'Utah' 'Nebraska' 'Pennsylvania' 'Illinois' 'Minnesota'
 'Michigan' 'Delaware' 'Indiana' 'New York' 'Arizona' 'Virginia'
```

```
  'Tennessee' 'Alabama' 'South Carolina' 'Oregon' 'Colorado' 'Iowa' 'Ohio'
  'Missouri' 'Oklahoma' 'New Mexico' 'Louisiana' 'Connecticut' 'New Jersey'
  'Massachusetts' 'Georgia' 'Nevada' 'Rhode Island' 'Mississippi'
  'Arkansas' 'Montana' 'New Hampshire' 'Maryland' 'District of Columbia'
  'Kansas' 'Vermont' 'Maine' 'South Dakota' 'Idaho' 'North Dakota'
  'Wyoming' 'West Virginia']
 Total Number of Unique Values:  49
```

12. Viewing Postal Code Dataset

This code prints the unique values and the total number of unique values in the 'Postal Code' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Postal Code
print('Unique Values: \n', stores['Postal Code'].unique())
print('Total Number of Unique Values: ', stores['Postal Code'].nunique())
```

```
  98115. 73034. 90045. 19134. 88220. 78207. 77036. 62521. 71203.  6824.
  75051. 92374. 45011.  7090. 19120. 44312. 80219. 75220. 37064. 90604.
  48601. 44256. 43017. 48227. 38401. 33614. 95051. 55044. 92037. 77506.
  94513. 27514.  7960. 45231. 94110. 90301. 33319. 80906.  7109. 48180.
   8701. 22204. 80004.  7601. 33710. 19143. 90805. 92345. 37130. 84041.
  78745.  1852. 31907.  6040. 78550. 85705. 62301.  2038. 33024. 98198.
  61604. 89115.  2886. 33180. 28403. 92646. 40475. 80027.  1841. 39212.
  48187. 10801. 28052. 32216. 47201. 13021. 73071. 94521. 60068. 79109.
  11757. 90008. 92024. 77340. 14609. 72701. 92627. 80134. 30318. 64118.
  59405. 48234. 33801. 36116. 85204. 60653. 54302. 45503. 92804. 98270.
  97301. 78041. 75217. 43123. 10011. 48126. 31088. 94591. 92691. 48307.
   7060. 85635. 98661. 60505. 76017. 40214. 75081. 44105. 75701. 27217.
  22980. 19013. 27511. 32137. 10550. 48205. 33012. 11572. 92105. 60201.
  48183. 55016. 71111. 50315. 93534. 23223. 28806. 92530. 68104. 98026.
  92704. 53209. 41042. 44052.  7036. 93905.  8901. 17602.  3301. 21044.
  75043.  6360. 22304. 43615. 87401. 92503. 90503. 78664. 92054. 33433.
  23464. 92563. 28540. 52601. 98502. 20016. 65109. 63376. 61107. 33142.
  78521. 10701. 94601. 28110. 20735. 30076. 72401. 47374. 94509. 33030.
  46350. 48911. 44221. 89502. 22801. 92025. 48073. 20852. 33065. 14215.
  33437. 39503. 93727. 27834. 11561. 35630. 31204. 52402.  2908. 81001.
  94533. 32725. 42071.  6457. 11520. 90660. 84604. 84062. 30080. 24153.
  44134. 36608.  2740. 75061.  8360. 85301. 14304. 27360. 92683. 38301.
  75019. 91767. 89031. 18103. 19711. 85281. 92677.  8302.  2149. 13601.
  54915. 98006. 75002. 79907. 76051. 75007. 37167. 98031. 70506. 97224.
  60076. 75023. 23434. 46203.  7002. 28314. 27405. 21215. 53142. 66062.
  98002. 74133. 97756. 27604. 74403.  6450. 42104. 46614.  6010. 89015.
  99207. 76248. 45014. 32127. 97504. 22901. 59801. 33178. 29501. 97477.
  32712. 19601. 80020. 65807.  7501. 73120. 23320. 79424. 65203. 37604.
  36830. 92404.  1453. 59715. 85345. 44107.  8861. 91761. 91730. 56560.
  75150. 95207. 32174. 94086.  3820. 17403. 77840. 63116.  2169. 95336.
  44240. 76903. 84106. 35810. 37918. 72209. 48146. 43302. 80122.  5401.
```

```
 92399.  2151. 77301. 60477. 52001. 48127. 87505. 28601. 60188. 56301.
 33161. 46226. 33317. 34952. 29730. 79762. 53214. 91911. 66502. 16602.
 80229. 61821. 47401. 71854. 78539. 77520. 46142. 90712.  2895. 54880.
 76021. 98042. 74012. 33023. 33021. 77536. 67212. 78501. 52240. 83704.
  2920. 61032. 77642. 95610. 75056. 98052. 32114. 86442. 46368. 58103.
 46514. 91776. 33063. 30328. 44060. 73505. 23666. 13440. 54601. 83501.
 39401. 94526. 48858. 84321.  6708. 30605.  4240. 61832. 85323. 30062.
 85364. 54401. 99301. 60302. 32503. 77573. 20877. 84043. 35401. 92553.
 40324. 80538. 85224. 59601. 63122. 76706. 48066. 60423. 18018. 55113.
 68801. 55125. 48237. 72756. 88101. 33458. 93101. 75104. 68701. 84020.
 48104. 91941. 83201. 49423.  6460. 60089. 92630. 96003. 95928. 13501.
 72032. 82001. 42301. 83605. 70065.  3060. 38134. 94061. 37087. 93454.
 60016. 98632. 37075. 50701.  2138. 60067.  1915. 97405. 93030. 98059.
 60025. 33445. 80022. 77590. 27893. 87124. 27534. 98208. 90640. 92020.
 77705. 33407. 79605. 61761. 63301. 60174. 93010. 97123. 91505. 95351.
 67846.  8401. 80501. 95616. 26003. 95037.  7011. 53081. 30344. 57701.
  1810. 34741.  6484.  6810. 52302. 32771. 78666. 80634. 76063. 44035.
 83301. 33134. 60441.  1752. 20707. 77803. 71603. 57401. 21740.  7017.
 60004. 60543. 55433. 92672. 94568. 93405. 72762.]
Total Number of Unique Values:  627
```

13. Viewing Region Dataset

This code prints the unique values and the total number of unique values in the 'Region' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Region' col
print('Unique Values: \n', stores['Region'].unique())
print('Total Number of Unique Values: ', stores['Region'].nunique())
```

```
Unique Values: 
 ['South' 'West' 'Central' 'East']
Total Number of Unique Values:  4
```

14. Viewing Product ID Dataset

This code prints the unique values and the total number of unique values in the 'Product ID' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Product ID'
print('Unique Values: \n', stores['Product ID'].unique())
print('Total Number of Unique Values: ', stores['Product ID'].nunique())
```

```
Unique Values: 
 ['FUR-BO-10001798' 'FUR-CH-10000454' 'OFF-LA-10000240' ...
 'TEC-MA-10003589' 'OFF-AP-10003099' 'TEC-PH-10002645']
Total Number of Unique Values:  1861
```

15. Viewing Category Dataset

This code prints the unique values and the total number of unique values in the 'Category' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Category' c
print('Unique Values: \n', stores['Category'].unique())
print('Total Number of Unique Values: ', stores['Category'].nunique())
```

```
Unique Values:
 ['Furniture' 'Office Supplies' 'Technology']
Total Number of Unique Values:  3
```

16. Viewing Sub-Category Dataset

This code prints the unique values and the total number of unique values in the 'Sub-Category' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Sub-Categor
print('Unique Values: \n', stores['Sub-Category'].unique())
print('Total Number of Unique Values: ', stores['Sub-Category'].nunique())
```

```
Unique Values:
 ['Bookcases' 'Chairs' 'Labels' 'Tables' 'Storage' 'Furnishings' 'Art'
 'Phones' 'Binders' 'Appliances' 'Paper' 'Accessories' 'Envelopes'
 'Fasteners' 'Supplies' 'Machines' 'Copiers']
Total Number of Unique Values:  17
```

*### 17. Viewing Product Name Dataset *

This code prints the unique values and the total number of unique values in the 'Product Name' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Product Nam
print('Unique Values: \n', stores['Product Name'].unique())
print('Total Number of Unique Values: ', stores['Product Name'].nunique())
```

```
Unique Values:
 ['Bush Somerset Collection Bookcase'
 'Hon Deluxe Fabric Upholstered Stacking Chairs, Rounded Back'
 'Self-Adhesive Address Labels for Typewriters by Universal' ...
 'Cisco 8961 IP Phone Charcoal' 'Eureka Hand Vacuum, Bagless' 'LG G2']
Total Number of Unique Values:  1849
```

18. Viewing Sales Dataset

This code prints the unique values and the total number of unique values in the 'Sales' column of the 'stores' DataFrame.

```
# Display unique values and the total number of unique values in the 'Sales' colu
print('Unique Values: \n', stores['Sales'].unique())
print('Total Number of Unique Values: ', stores['Sales'].nunique())
```

```
Unique Values:
 [261.96  731.94   14.62  ... 235.188  26.376  10.384]
Total Number of Unique Values:  5757
```

Viewing & Saving Clean Data

Viewing the final and cleaned data, saving it into .csv format

Here, the column names in the 'stores' DataFrame are modified by replacing spaces with underscores and converting them to lowercase for consistency.

```
stores.columns = [col.replace(' ', '_').replace('-', '_').lower() for col in stor
```

```
stores.sample(5)                # Display a random sample of 5 rows from the mod
```

	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	s
4243	CA-2017-159142	2017-04-04	2017-04-05	First Class	ME-17320	Maria Etezadi	
2144	CA-2017-155481	2017-09-01	2017-09-05	Standard Class	DK-13225	Dean Katz	C
4866	CA-2018-122490	2018-11-13	2018-11-18	Standard Class	TT-21070	Ted Trevino	C
6237	US-2017-155180	2017-01-22	2017-01-29	Standard Class	TB-21280	Toby Braunhardt	C
7419	US-2017-114013	2017-03-13	2017-03-15	Second Class	SC-20770	Stewart Carmichael	C

```
print(stores.dtypes)       # Display data types of each column in the modified 's
```

```
order_id                 object
order_date       datetime64[ns]
ship_date        datetime64[ns]
ship_mode                object
customer_id              object
customer_name            object
segment                  object
country                  object
city                     object
state                    object
postal_code             float64
region                   object
product_id               object
category                 object
sub_category             object
product_name             object
sales                   float64
days_to_ship              int64
dtype: object
```

```
print(stores.isnull().sum().sum())
```

```
0
```

```
print(stores.shape)
```

```
(9799, 18)
```

```
print(stores.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9799 entries, 0 to 9799
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype         
---  ------         --------------  -----         
 0   order_id       9799 non-null   object        
 1   order_date     9799 non-null   datetime64[ns]
 2   ship_date      9799 non-null   datetime64[ns]
 3   ship_mode      9799 non-null   object        
 4   customer_id    9799 non-null   object        
 5   customer_name  9799 non-null   object        
 6   segment        9799 non-null   object        
 7   country        9799 non-null   object        
 8   city           9799 non-null   object        
 9   state          9799 non-null   object        
 10  postal_code    9799 non-null   float64       
 11  region         9799 non-null   object        
 12  product_id     9799 non-null   object        
 13  category       9799 non-null   object        
 14  sub_category   9799 non-null   object        
 15  product_name   9799 non-null   object        
 16  sales          9799 non-null   float64       
 17  days_to_ship   9799 non-null   int64         
dtypes: datetime64[ns](2), float64(2), int64(1), object(13)
memory usage: 1.4+ MB
None
```

```
stores.to_csv('stores_data_cleaned.csv', index=False)
```

Statistics & Data Vizualization

Data Analysis

1.1. Descriptive Numeric Analysis

This code generates descriptive statistics, such as mean, standard deviation, minimum, and maximum values, for numerical columns in the 'stores' DataFrame

```
# Generate descriptive statistics for numerical columns in the 'stores' DataFrame
stores.describe()
```

	postal_code	sales	days_to_ship
count	9799.000	9799.000	9799.000
mean	55218.567	230.764	3.961
std	32068.158	626.684	1.750
min	1040.000	0.444	0.000
25%	23223.000	17.248	3.000
50%	57701.000	54.480	4.000
75%	90008.000	210.572	5.000
max	99301.000	22638.480	7.000

```
# Generate descriptive statistics for numerical columns in the 'stores' DataFrame
stores.describe()
```

	postal_code	sales	days_to_ship
count	9799.000	9799.000	9799.000
mean	55218.567	230.764	3.961
std	32068.158	626.684	1.750
min	1040.000	0.444	0.000
25%	23223.000	17.248	3.000
50%	57701.000	54.480	4.000
75%	90008.000	210.572	5.000
max	99301.000	22638.480	7.000

Observation: Gives a desription about the stats on sale price which is only necessary here.

1.2. Descriptive Numeric Analysis

This code generates descriptive statistics for categorical columns in the 'stores' DataFrame, including count, unique values, top value, and frequency.

```
# Generate descriptive statistics for categorical columns in the 'stores' DataFra
stores.describe(include=['object'])
```

	order_id	ship_mode	customer_id	customer_name	segment	country	city
count	9799	9799	9799	9799	9799	9799	9799
unique	4922	4	793	793	3	1	529
top	CA-2018-100111	Standard Class	WB-21850	William Brown	Consumer	United States	New York City

Observation:

Gives an idea about the top values of every column, count and frequency of different values.

2. Top Customers:

```
Identify the top customers based on total sales or order frequency. This helps in understa
```

This code identifies and prints the top 5 customers in the 'stores' DataFrame based on the sum of their sales.

```
# Find and display the top 5 customers based on the sum of their sales
top_customers = stores.groupby('customer_name')['sales'].sum().sort_values(ascend
print("Top Customers:\n", top_customers)
```

```
Top Customers:
 customer_name
Sean Miller      25043.050
Tamara Chand     19052.218
Raymond Buch     15117.339
Tom Ashbrook     14595.620
Adrian Barton    14473.571
Name: sales, dtype: float64
```

Observation:

Someone should give them some discount for being so loyal to Superstore. Especially, Sean Miller.

3. Popular Products:

```
Find the most popular products based on sales or order quantity.
```

This code identifies and prints the top 10 popular products in the 'stores' DataFrame based on the sum of their sales.

```
# Find and display the top 10 popular products based on the sum of their sales
popular_products = stores.groupby('product_name')['sales'].sum().sort_values(asce
print("Popular Products:\n", popular_products)
```

```
Popular Products:
 product_name
Canon imageCLASS 2200 Advanced Copier
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind
Cisco TelePresence System EX90 Videoconferencing Unit
HON 5400 Series Task Chairs for Big and Tall
GBC DocuBind TL300 Electric Binding System
GBC Ibimaster 500 Manual ProClick Binding System
Hewlett Packard LaserJet 3310 Copier
HP Designjet T520 Inkjet Large Format Printer - 24" Color
GBC DocuBind P400 Electric Binding System
High Speed Automatic Electric Letter Opener
Name: sales, dtype: float64
```

Observation: Looks like Copier and Binding machine are used so roughly, that they are ordered repeatedly.

4. Customer Segmentation:

```
Explore different customer segments and their characteristics.
```

This code calculates and prints the average sales for each customer segment in the 'stores' DataFrame.

```
# Calculate and display the average sales for each customer segment
customer_segmentation = stores.groupby('segment')['sales'].mean()
print("Average Sales by Customer Segment:\n", customer_segmentation)
```

```
Average Sales by Customer Segment:
 segment
Consumer      225.066
Corporate     233.151
Home Office   243.382
Name: sales, dtype: float64
```

Observation: Looks like everyone opted for work from home.

5. Temporal Trends:

```
Analyze trends over time, such as monthly or yearly sales growth.
```

This code calculates and prints the sum of sales grouped by month, showing temporal trends in the 'stores' DataFrame.

```
# Calculate and display the sum of sales grouped by month for temporal trends
temporal_trends = stores.groupby(pd.to_datetime(stores['order_date'], format='%m/
print("Temporal Trends:\n", temporal_trends)
```

```
Temporal Trends:
 order_date
2015-01    14205.707
2015-02     4519.892
2015-03    55205.797
2015-04    27625.483
2015-05    23644.303
2015-06    34322.936
2015-07    33781.543
2015-08    27117.536
2015-09    81623.527
2015-10    31453.393
2015-11    77907.661
2015-12    68167.058
2016-01    18066.958
2016-02    11951.411
2016-03    32339.318
2016-04    34154.469
2016-05    29959.531
2016-06    23599.374
2016-07    28608.259
2016-08    36818.342
2016-09    63133.606
2016-10    31011.737
2016-11    75249.399
2016-12    74543.601
2017-01    18542.491
2017-02    22978.815
2017-03    51165.059
2017-04    38679.767
2017-05    56656.908
2017-06    39724.486
2017-07    38320.783
2017-08    30542.200
2017-09    69193.391
2017-10    59583.033
2017-11    79066.496
2017-12    95739.121
2018-01    43476.474
```

```
    2018-02    19920.997
    2018-03    58863.413
    2018-04    35541.910
    2018-05    43825.982
    2018-06    48190.728
    2018-07    44825.104
    2018-08    62837.848
    2018-09    86152.888
    2018-10    77448.131
    2018-11   117938.155
    2018-12    83030.389
    Freq: M, Name: sales, dtype: float64
```

Observation: Datewise analysis of sales happened in every month.

6. Geographical Analysis:

Examine sales performance across different regions, states, or cities.

This code conducts geographical analysis by calculating and printing the top 10 sales in regions, states, and cities in the 'stores' DataFrame.

```
# Perform geographical analysis by calculating and displaying the top 10 sales in
geographical_analysis = stores.groupby(['region', 'state', 'city'])['sales'].sum(
print("Geographical Analysis:\n", geographical_analysis)
```

```
    Geographical Analysis:
     region   state         city
    East     New York      New York City   252462.547
    West     California    Los Angeles     173420.181
             Washington    Seattle         116106.322
             California    San Francisco   109041.120
    East     Pennsylvania  Philadelphia    108841.749
    Central  Texas         Houston          63956.143
             Illinois      Chicago          47820.133
    West     California    San Diego        47521.029
    Central  Michigan      Detroit          42446.944
    South    Florida       Jacksonville     39133.328
    Name: sales, dtype: float64
```

Observation: The company should open its store in New York city and Los Angeles as the sales are higher there.

7. Shipping Analysis:

```
 Investigate the relationship between shipping mode and delivery times.
```

This code calculates and prints the average days to ship for each shipping mode in the 'stores' DataFrame.

```
# Calculate and display the average days to ship for each shipping mode
shipping_analysis = stores.groupby('ship_mode')['days_to_ship'].mean()
print("Average Days to Ship by Shipping Mode:\n", shipping_analysis)
```

```
Average Days to Ship by Shipping Mode:
 ship_mode
First Class      2.179
Same Day         0.045
Second Class     3.249
Standard Class   5.009
Name: days_to_ship, dtype: float64
```

Observation: Seems fair. Same day delivery is too impressive.

8.Correlation Analysis:

bold text

```
Explore correlations between numerical variables [ Correlation isn't necessary for postal
```

This code computes and prints the correlation matrix between the 'days_to_ship' and 'sales' columns in the 'stores' DataFrame.

```
# Calculate and display the correlation matrix between 'days_to_ship' and 'sales'
correlation_matrix = stores[['days_to_ship', 'sales']].corr()
print("Correlation Matrix:\n", correlation_matrix)
```

```
Correlation Matrix:
               days_to_ship  sales
days_to_ship         1.000 -0.006
sales               -0.006  1.000
```

Observation: Gives an analysis of days to ship to sales

9. Outliers Detection:

```
Identify and examine outliers in numerical columns.
```

```
Outliers for Sales Column
```

This code filters rows in the 'stores' DataFrame where sales are greater than the mean plus three times the standard deviation.

```
# Filter rows in the 'stores' DataFrame where sales exceed the mean plus 3 times
stores[stores['sales'] > stores['sales'].mean() + 3 * stores['sales'].std()]
```

	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	s
27	US-2016-150630	2016-09-17	2016-09-21	Standard Class	TB-21520	Tracy Blumstein	C
165	CA-2015-139892	2015-09-08	2015-09-12	Standard Class	BM-11140	Becky Martin	C
251	CA-2017-145625	2017-09-11	2017-09-17	Standard Class	KC-16540	Kelly Collister	C
262	US-2015-106992	2015-09-19	2015-09-21	Second Class	SB-20290	Sean Braxton	C
263	US-2015-106992	2015-09-19	2015-09-21	Second Class	SB-20290	Sean Braxton	C
...	...	...	...	...	...	...	
9639	CA-2016-116638	2016-01-28	2016-01-31	Second Class	JH-15985	Joseph Holt	C
9649	CA-2017-107104	2017-11-26	2017-11-30	Standard Class	MS-17365	Maribeth Schnelling	C
9660	CA-2017-160717	2017-06-06	2017-06-11	Standard Class	ME-17320	Maria Etezadi	
9741	CA-2016-117086	2016-11-08	2016-11-12	Standard Class	QJ-19255	Quincy Jones	C
9774	CA-2015-169019	2015-07-26	2015-07-30	Standard Class	LF-17185	Luke Foster	C

123 rows × 18 columns

Observation: Those days were really memorabe for company for having more sales than its used to.

```
Outliers for days_to_ship Column
```

This code filters rows in the 'stores' DataFrame where the days to ship exceed the mean plus three times the standard deviation.

```
# Filter rows in the 'stores' DataFrame where days to ship exceed the mean plus 3
stores[stores['days_to_ship'] > stores['days_to_ship'].mean() + 3 * stores['days_
```

order_id	order_date	ship_date	ship_mode	customer_id	customer_name	segme

Observation: Company is more strict to timing

10. Average Sales by Category:

```
Showcasing the average Sales by Category:.
```

This code calculates and prints the average sales for each product category in the 'stores' DataFrame.

```
# Calculate and display the average sales for each product category
average_sales_by_category = stores.groupby('category')['sales'].mean()
print("Average Sales by Category:\n", average_sales_by_category)
```

```
Average Sales by Category:
 category
Furniture          350.687
Office Supplies    119.381
Technology         456.401
Name: sales, dtype: float64
```

Observation: Seems like superstore is popular for its great technology quality.

Data Vizualization: Univariate

Here, we set the style of Seaborn plots to 'whitegrid' for a clean and simple background in visualizations.

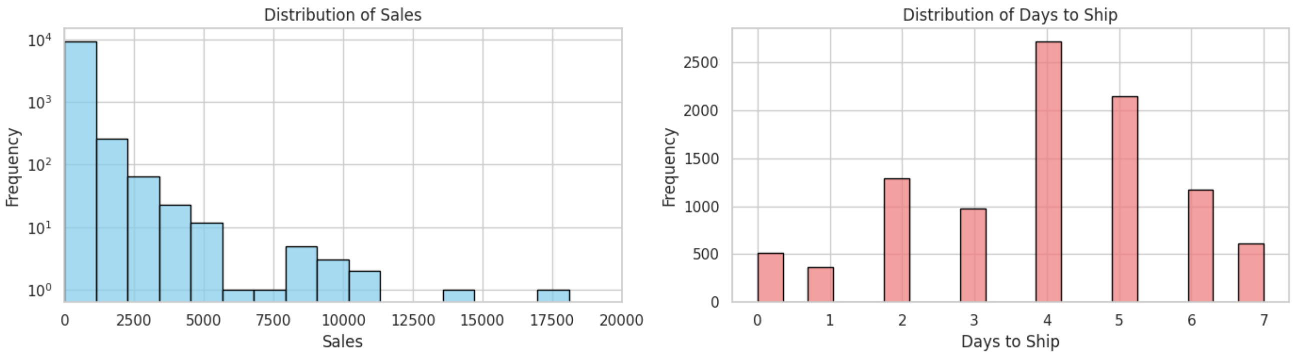
```
# Set the style of Seaborn plots to 'whitegrid'
sns.set(style='whitegrid')
```

1. Histograms

Create histograms to visualize the distribution of key numeric variables. Reference: Link

Here, we create a side-by-side histogram subplot for the distribution of sales and days to ship in the 'stores' DataFrame. The visualizations provide insights into the frequency and patterns of these two variables.

```
# Create a side-by-side histogram subplot for sales and days to ship
fig, axes = plt.subplots(1, 2, figsize=(14, 4))

# First subplot for sales distribution
sns.histplot(stores['sales'], bins=20, color='skyblue', edgecolor='black', ax=axe
axes[0].set_title('Distribution of Sales')
axes[0].set_xlabel('Sales')
axes[0].set_ylabel('Frequency')
axes[0].set_xlim(0, 20000)
axes[0].set_yscale('log')

# Second subplot for days to ship distribution
sns.histplot(stores['days_to_ship'], bins=20, color='lightcoral', edgecolor='blac
axes[1].set_title('Distribution of Days to Ship')
axes[1].set_xlabel('Days to Ship')
axes[1].set_ylabel('Frequency')

# Adjust layout for better spacing
plt.tight_layout()

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



Observation:

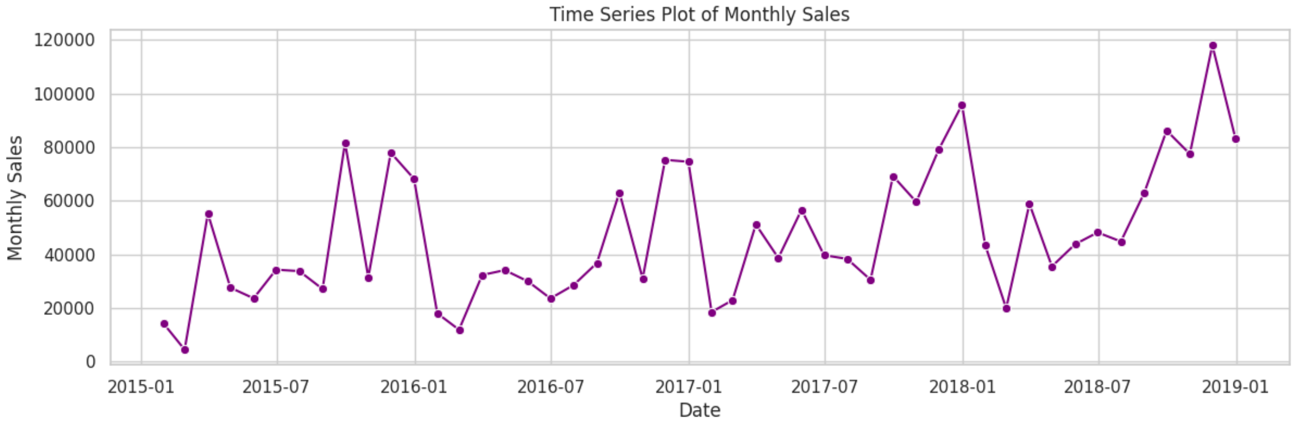
```
Sales less than 5000 are quiet common.
Most shipments take 4 days to ship
```

2. Time Series Plots

Plot time series graphs to understand the trends and patterns in sales over the 4-year period.

Here, we create a time series plot for monthly sales in the 'stores' DataFrame. The plot displays the trend in monthly sales over time using Seaborn.

```
# Create a time series plot for monthly sales
plt.figure(figsize=(14, 4))

# Convert 'order_date' to datetime format
stores['order_date'] = pd.to_datetime(stores['order_date'], format='%m/%d/%Y')

# Resample monthly and sum sales
time_series_sales = stores.resample('M', on='order_date')['sales'].sum().reset_in

# Plot the time series using Seaborn
sns.lineplot(data=time_series_sales, x='order_date', y='sales', marker='o', color
plt.title('Time Series Plot of Monthly Sales')
plt.xlabel('Date')
plt.ylabel('Monthly Sales')

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



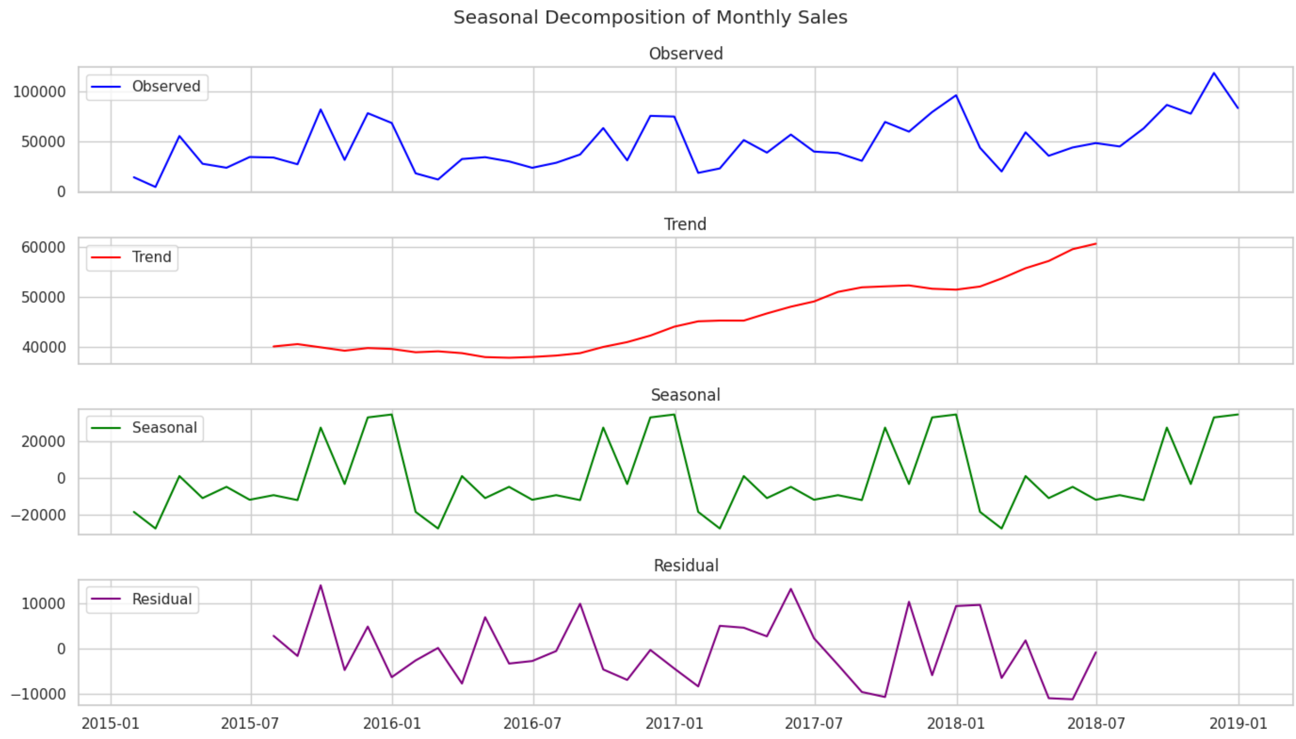
Observation: We couldn't predict, but the company seems to earn in last 2 years and tend to grow more.

3. Seasonal Decomposition

Decompose time series data into components like trend, seasonality, and residuals for deeper insights. Reference : Link

In this code, we perform seasonal decomposition on the 'sales' column in the time_series_sales DataFrame and plot the observed, trend, seasonal, and residual components separately. The resulting visualizations provide insights into the different aspects of the time series data.

```
# Perform seasonal decomposition on the 'sales' column in the time_series_sales D
result = seasonal_decompose(time_series_sales['sales'], model='additive', period=

# Plot individual components
fig, axes = plt.subplots(4, 1, figsize=(14, 8), sharex=True)

axes[0].plot(time_series_sales['order_date'], result.observed, label='Observed',
axes[0].legend(loc='upper left')
axes[0].set_title('Observed')

axes[1].plot(time_series_sales['order_date'], result.trend, label='Trend', color=
axes[1].legend(loc='upper left')
axes[1].set_title('Trend')

axes[2].plot(time_series_sales['order_date'], result.seasonal, label='Seasonal',
axes[2].legend(loc='upper left')
axes[2].set_title('Seasonal')

axes[3].plot(time_series_sales['order_date'], result.resid, label='Residual', col
axes[3].legend(loc='upper left')
axes[3].set_title('Residual')

plt.suptitle('Seasonal Decomposition of Monthly Sales')
plt.tight_layout()
plt.show()
```



Observation: Okay! Now I am sure that company is really growing!

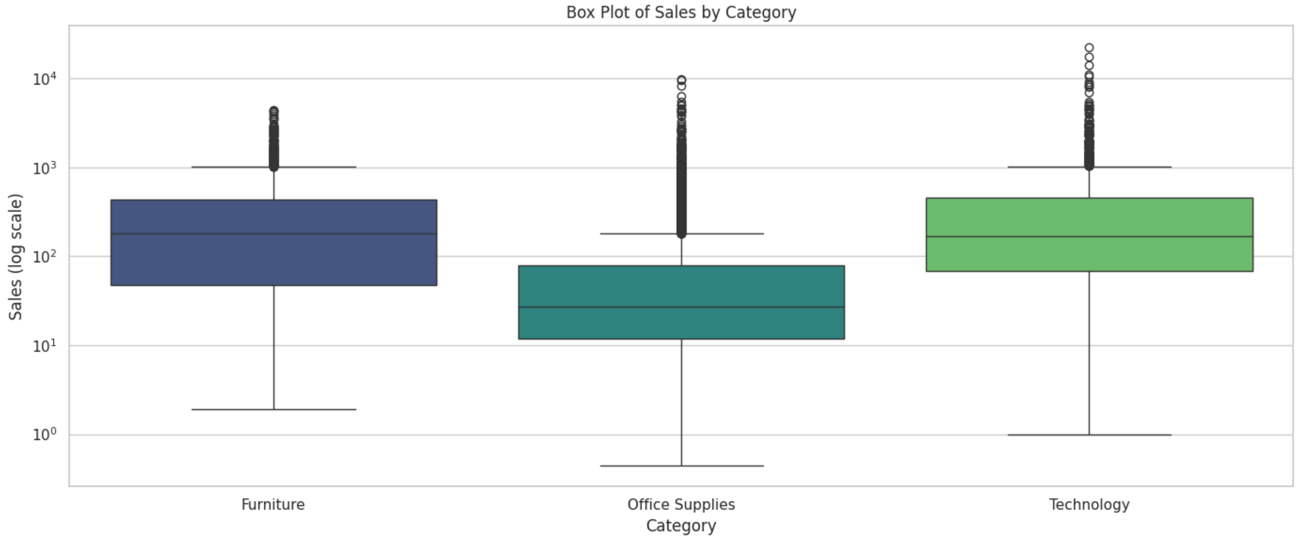
4. Box Plots

Use box plots to identify outliers and understand the distribution of numeric variables.

Here, we use Seaborn to create a box plot of sales by category in the 'stores' DataFrame. The y-axis is set to a logarithmic scale for a better representation of the data distribution.

```
# Create a box plot of sales by category using Seaborn
plt.figure(figsize=(14, 6))
sns.boxplot(x='category', y='sales', data=stores, palette='viridis')
plt.yscale('log')
plt.title('Box Plot of Sales by Category')
plt.xlabel('Category')
plt.ylabel('Sales (log scale)')
plt.tight_layout()
plt.show()
```

```
<ipython-input-61-6b93f6ee7d17>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.boxplot(x='category', y='sales', data=stores, palette='viridis')
```



Observation: Outliers seem to be strict.

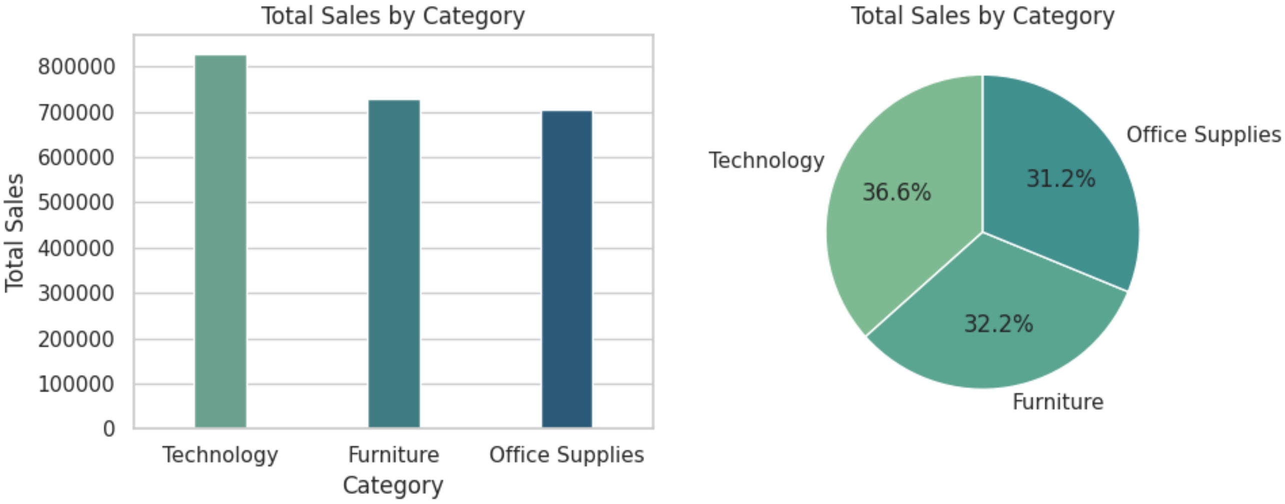
5. Sales Distribution by Category

Visualize the distribution of sales across different categories using bar charts or pie charts.

In this code, we calculate the total sales by category and create subplots with a bar plot and a pie chart to visualize the distribution of sales across categories in the 'stores' DataFrame.

```
# Calculate total sales by category
category_sales = stores.groupby('category')['sales'].sum().reset_index()

# Sort the data by sales in descending order
category_sales = category_sales.sort_values(by='sales', ascending=False)

# Create subplots with two columns
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Bar plot on the first subplot
sns.barplot(x='category', y='sales', data=category_sales, palette='crest', ax=axe
axes[0].set_title('Total Sales by Category')
axes[0].set_xlabel('Category')
axes[0].set_ylabel('Total Sales')

# Pie chart on the second subplot
axes[1].pie(category_sales['sales'], labels=category_sales['category'], autopct='
axes[1].set_title('Total Sales by Category')

# Display the plots
plt.tight_layout()
plt.show()
```

```
<ipython-input-62-7cf5ba9b34cf>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.barplot(x='category', y='sales', data=category_sales, palette='crest', 
```



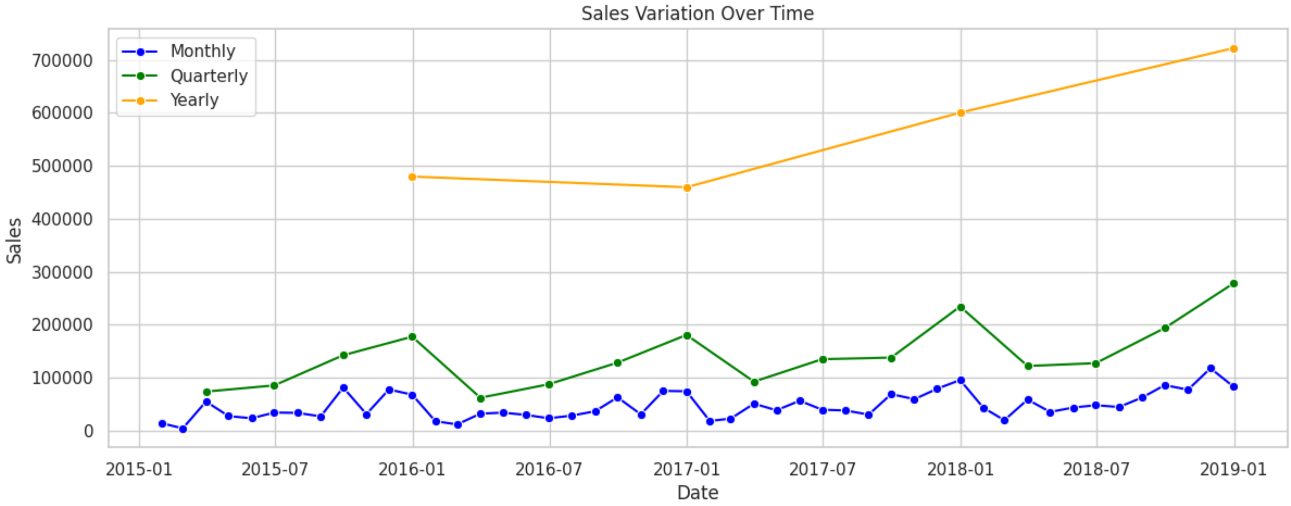
Observation: Its interesting to see that Technology has more amount of sale.

6. Sales Variation Over Time

Plot line charts to observe how sales vary over different time periods (months, quarters, years).

Here, we calculate monthly, quarterly, and yearly sales variations and create a Seaborn-style line plot to visualize how sales vary over time in the 'stores' DataFrame. The plot displays monthly, quarterly, and yearly trends with different colors and markers.

```
# Calculate monthly, quarterly, and yearly sales variations
monthly_sales_variation = stores.resample('M', on='order_date')['sales'].sum()
quarterly_sales_variation = stores.resample('Q', on='order_date')['sales'].sum()
yearly_sales_variation = stores.resample('Y', on='order_date')['sales'].sum()

# Create a Seaborn-style plot using lineplot
plt.figure(figsize=(14, 5))
sns.lineplot(data=monthly_sales_variation, marker='o', color='blue', label='Month
sns.lineplot(data=quarterly_sales_variation, marker='o', color='green', label='Qu
sns.lineplot(data=yearly_sales_variation, marker='o', color='orange', label='Year

# Set titles and labels
plt.title('Sales Variation Over Time')
plt.xlabel('Date')
plt.ylabel('Sales')

# Add legend
plt.legend()

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



Observation: For Yearly, its been growing For month and daily basis, there is a zigzag pattern with a growing trend.

Data Vizualization: Bivariate

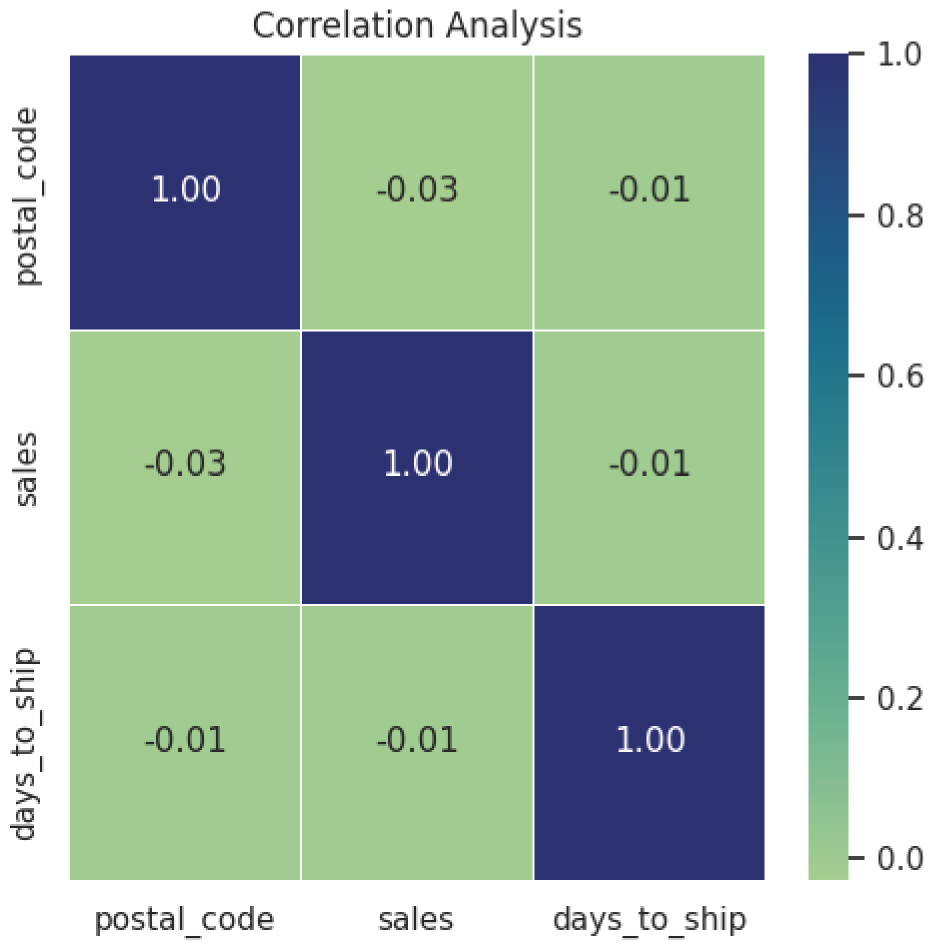
1. Correlation Analysis

Examine the correlation between sales and other relevant numeric variables.

In this code, we calculate the correlation matrix for the numeric columns in the 'stores' DataFrame and create a heatmap using Seaborn to visually represent the correlations between different variables. The heatmap includes annotations with correlation values and uses the 'crest' color map.

```
# Calculate the correlation matrix
correlation_matrix = stores.corr()

# Create a heatmap using Seaborn
plt.figure(figsize=(5, 5))
sns.heatmap(correlation_matrix, annot=True, cmap='crest', fmt=".2f", linewidths=.
plt.title('Correlation Analysis')
plt.tight_layout()
plt.show()
```

```
    <ipython-input-64-d922d7798e8c>:2: FutureWarning: The default value of numeri
      correlation_matrix = stores.corr()
```



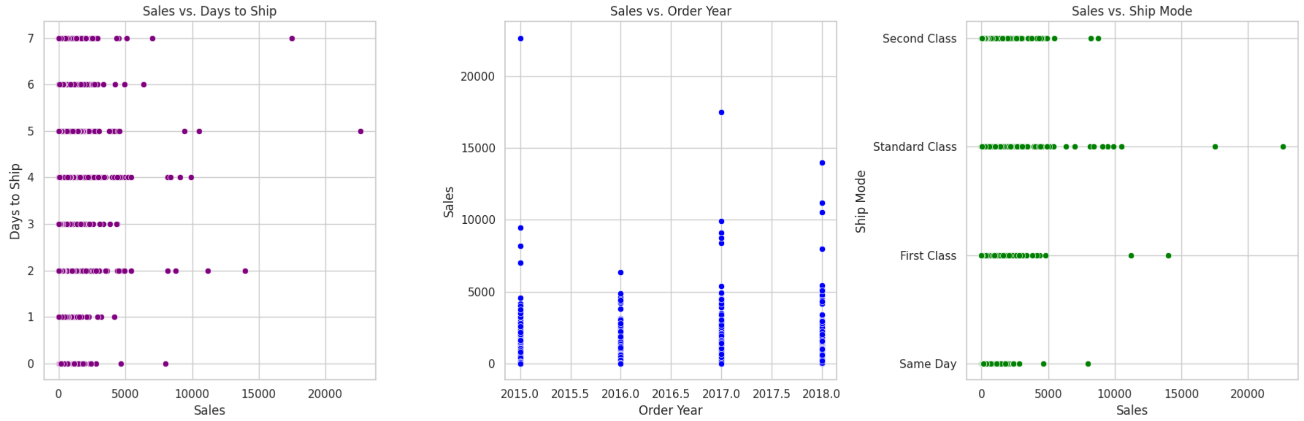
Observation: Gives an understanding to sales to shipment days

2. Scatter Plots

Plot scatter plots to explore the relationship between sales and another numeric variable.

This code creates a figure with three scatter plots side by side to compare the relationships between sales and days to ship, sales and order year, and sales and ship mode in the 'stores' DataFrame. Each subplot is labeled with the respective titles and axis labels for clarity.

```
# Create a figure with three scatter plots side by side
plt.figure(figsize=(18, 6))

# Scatter Plot: Sales vs. Days to Ship
plt.subplot(1, 3, 1)
sns.scatterplot(x='sales', y='days_to_ship', data=stores, color='purple')
plt.title('Sales vs. Days to Ship')
plt.xlabel('Sales')
plt.ylabel('Days to Ship')

# Scatter Plot: Sales vs. Order Date
plt.subplot(1, 3, 2)
sns.scatterplot(x=stores['order_date'].dt.year, y='sales', data=stores, color='bl
plt.title('Sales vs. Order Year')
plt.xlabel('Order Year')
plt.ylabel('Sales')

# Scatter Plot: Sales vs. Ship Mode
plt.subplot(1, 3, 3)
sns.scatterplot(x='sales', y='ship_mode', data=stores, color='green')
plt.title('Sales vs. Ship Mode')
plt.xlabel('Sales')
plt.ylabel('Ship Mode')

# Adjust layout
plt.tight_layout()
plt.show()
```



Observation:

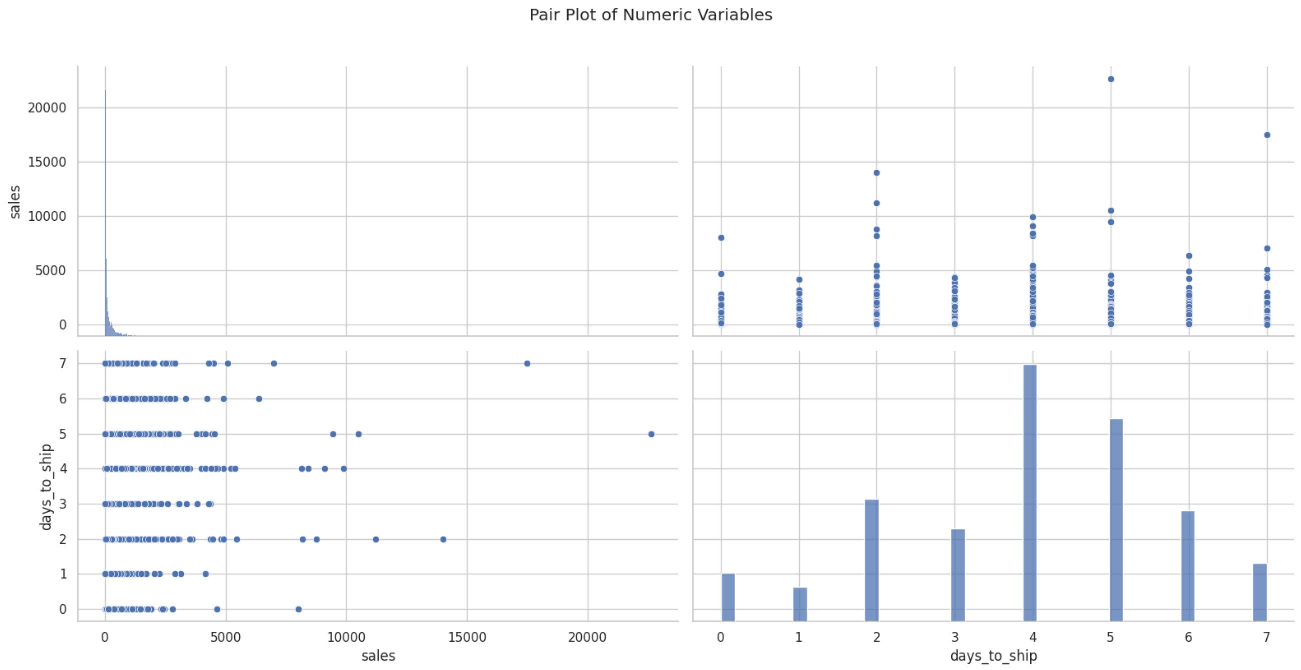
```
It seems that the company is uniform to delivery standards. Order size doesn't priortize
In 2018 seem like they are growing with more number of big sales.
Most of the sale is a Standard Class Delivery.
```

3. Pair Plots

Use pair plots for a quick overview of relationships between multiple numeric variables.

Here, we create a pair plot using Seaborn for the numeric variables 'sales' and 'days_to_ship' in the 'stores' DataFrame. The pair plot provides a visual representation of the relationships between these numeric variables.

```
# Generate a pair plot for selected numeric variables using Seaborn
numeric_columns = ['sales', 'days_to_ship']
sns.pairplot(stores[numeric_columns], height=4, aspect=2)
plt.suptitle('Pair Plot of Numeric Variables', y=1.02)
plt.tight_layout()
plt.show()
```



Observation:

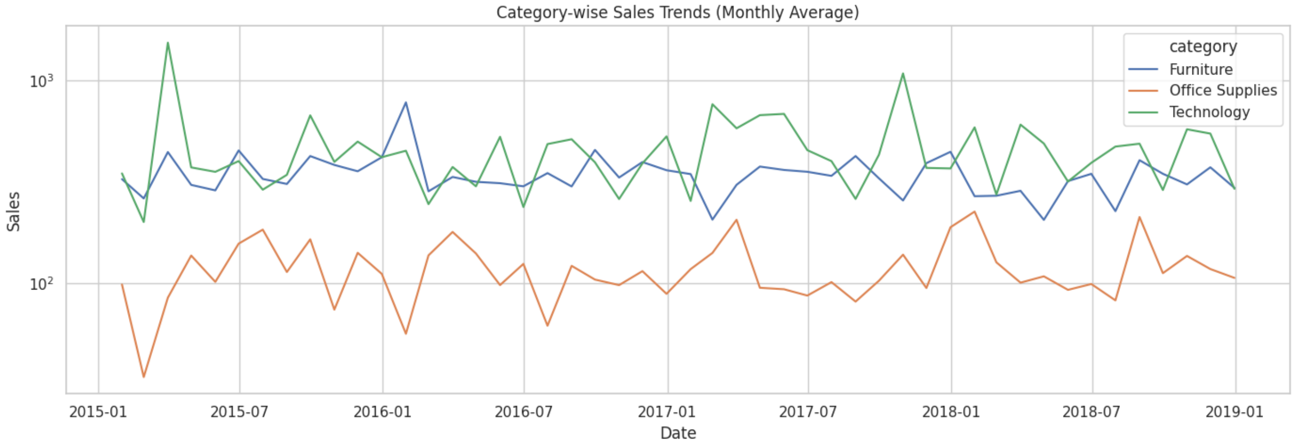
Gives analysis to days to ship and sales. They don't depend on each other.

4. Category-wise Sales Trends

Analyze how sales trends differ across different categories using line charts. Reference: pd.Grouper Saved my Day 🙂

Here, we convert the 'order_date' column to datetime format, then resample the data to calculate the monthly average sales for each category. The resulting trends are visualized using a line plot with a logarithmic scale on the y-axis for better representation.

```
# Convert 'order_date' to datetime format
stores['order_date'] = pd.to_datetime(stores['order_date'])

# Resample the data to monthly average for each category
monthly_avg_sales = stores.groupby(['category', pd.Grouper(key='order_date', freq

# Create a line plot with log scale for the y-axis
plt.figure(figsize=(14, 5))
sns.lineplot(x='order_date', y='sales', hue='category', data=monthly_avg_sales)
plt.yscale('log')
plt.title('Category-wise Sales Trends (Monthly Average)')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.tight_layout()
plt.show()
```

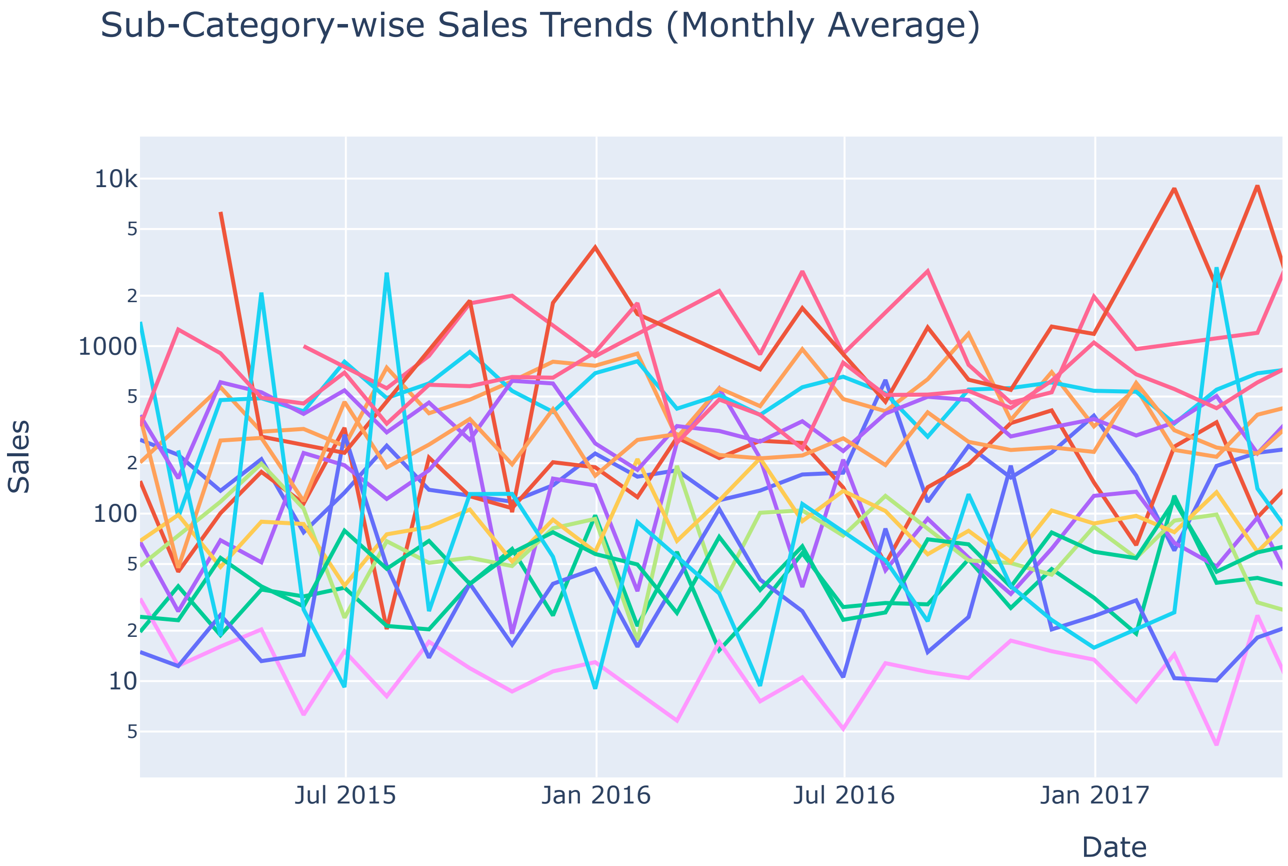


Observation:

Shows that Technology is always on demand than that of Furniture and office supplies.

Here, we convert the 'order_date' column to datetime format and resample the data to calculate the monthly average sales for each sub-category. The trends are visualized using a line plot with a logarithmic scale on the y-axis using Plotly Express for interactive and detailed exploration.

```
# Convert 'order_date' to datetime format
stores['order_date'] = pd.to_datetime(stores['order_date'])

# Resample the data to monthly average for each sub_category
monthly_avg_sales = stores.groupby(['sub_category', pd.Grouper(key='order_date',

# Create a line plot with log scale for the y-axis using Plotly Express
fig = px.line(monthly_avg_sales, x='order_date', y='sales', color='sub_category',
              labels={'sales': 'Sales', 'order_date': 'Date', 'sub_category': 'Su
              title='Sub-Category-wise Sales Trends (Monthly Average)',
              log_y=True, width=1200, height=500)

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



Observation:

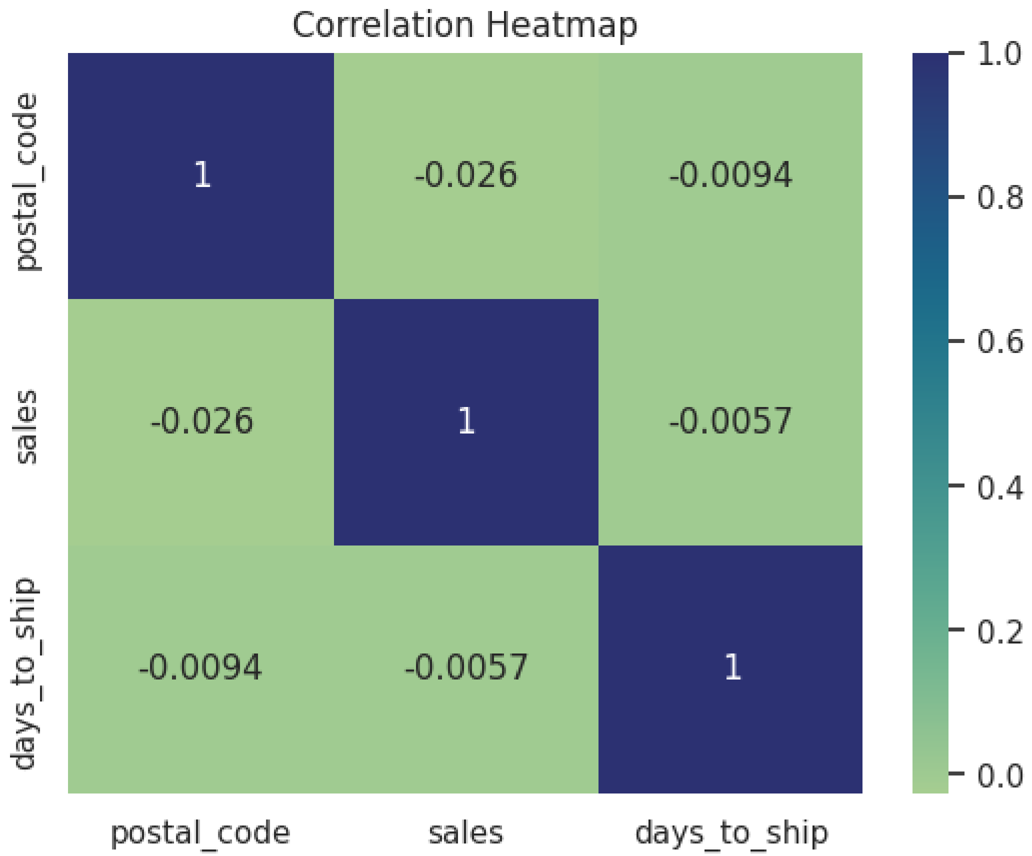
I don't know why they need sooooo many copiers and machines for binding.

5. Heatmaps

Create heatmaps to visualize the correlation matrix for better insights

In this code, we use Seaborn to generate a heatmap for the correlation matrix, visualizing the correlations between different variables. The color intensity and annotations provide insights into the strength and direction of the correlations.

```
# Create a heatmap for the correlation matrix using Seaborn
sns.heatmap(correlation_matrix, annot=True, cmap='crest')
plt.title('Correlation Heatmap')
plt.show()
```



Observation:

Gives an understanding to sales to shipment days

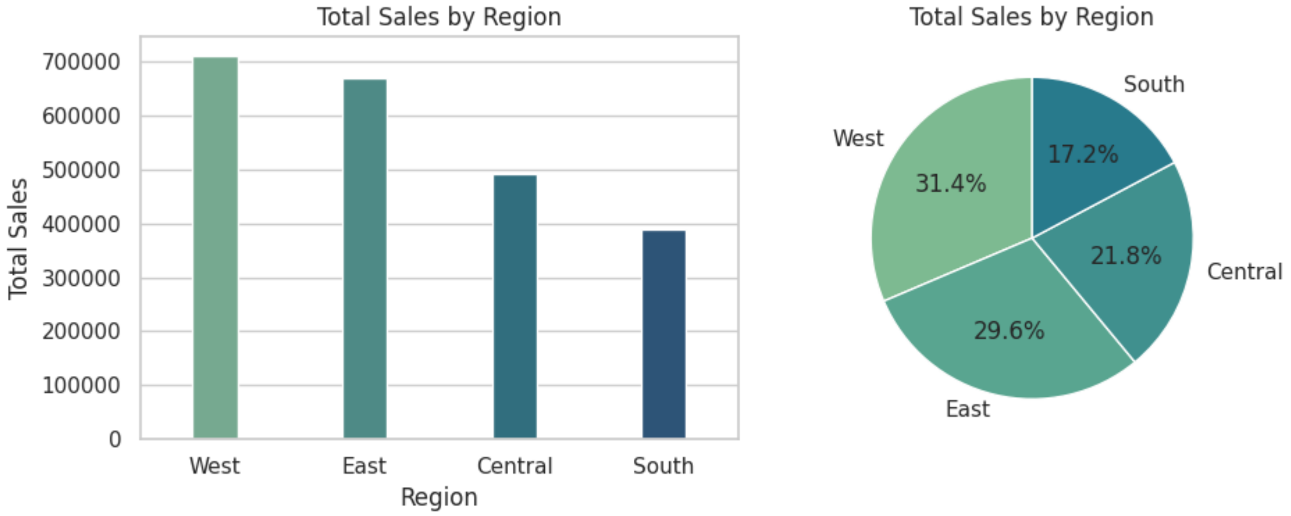
6. Sales by Region

Compare sales across different regions using bar charts or stacked bar charts.

In this code, we calculate the total sales by region and create subplots with a bar plot and a pie chart to visualize the distribution of sales across different regions in the 'stores' DataFrame.

```
# Calculate total sales by region
region_sales = stores.groupby('region')['sales'].sum().reset_index()

# Sort the data by sales in descending order
region_sales = region_sales.sort_values(by='sales', ascending=False)

# Create subplots with two columns
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Bar plot on the first subplot
sns.barplot(x='region', y='sales', data=region_sales, palette='crest', ax=axes[0]
axes[0].set_title('Total Sales by Region')
axes[0].set_xlabel('Region')
axes[0].set_ylabel('Total Sales')

# Pie chart on the second subplot
axes[1].pie(region_sales['sales'], labels=region_sales['region'], autopct='%1.1f%
axes[1].set_title('Total Sales by Region')

# Display the plots
plt.tight_layout()
plt.show()
```

```
    <ipython-input-70-e42d015deedd>:11: FutureWarning:


    Passing `palette` without assigning `hue` is deprecated and will be removed i
```



Observation:

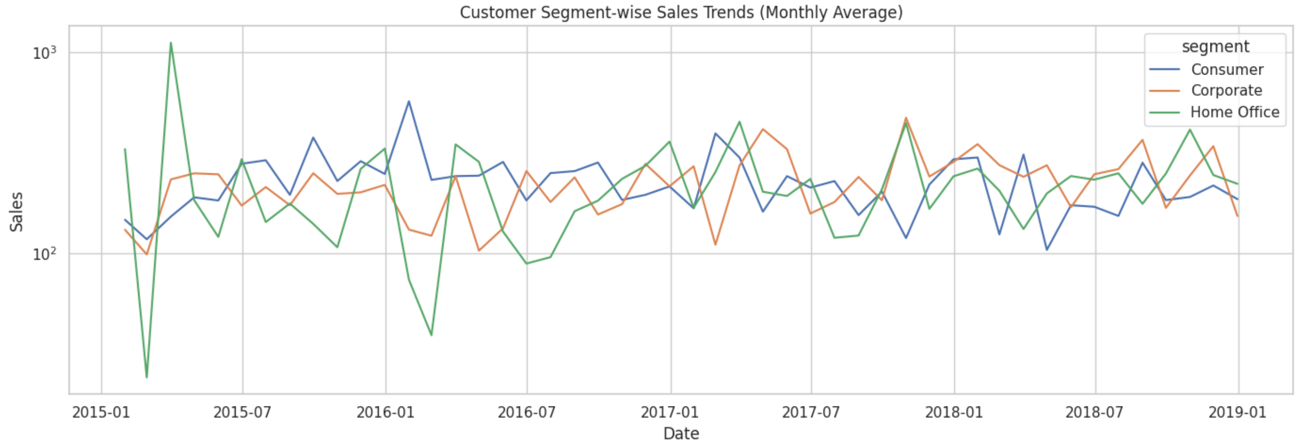
People from West region of US order more supplies from superstore then comes East then Central and South

7. Customer Segment Analysis

Explore sales trends for different customer segments using line charts or bar charts.

Here, we convert the 'order_date' column to datetime format, resample the data to calculate the monthly average sales for each customer segment, and visualize the trends using a line plot with a logarithmic scale on the y-axis.

```
# Convert 'order_date' to datetime format
stores['order_date'] = pd.to_datetime(stores['order_date'])

# Resample the data to monthly average for each segment
monthly_avg_sales = stores.groupby(['segment', pd.Grouper(key='order_date', freq=

# Create a line plot with log scale for the y-axis
plt.figure(figsize=(14, 5))
sns.lineplot(x='order_date', y='sales', hue='segment', data=monthly_avg_sales)
plt.yscale('log')
plt.title('Customer Segment-wise Sales Trends (Monthly Average)')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.tight_layout()
plt.show()
```



Observation:

I guess there is a clash between the manager who wants work from office and employees who want work from home.

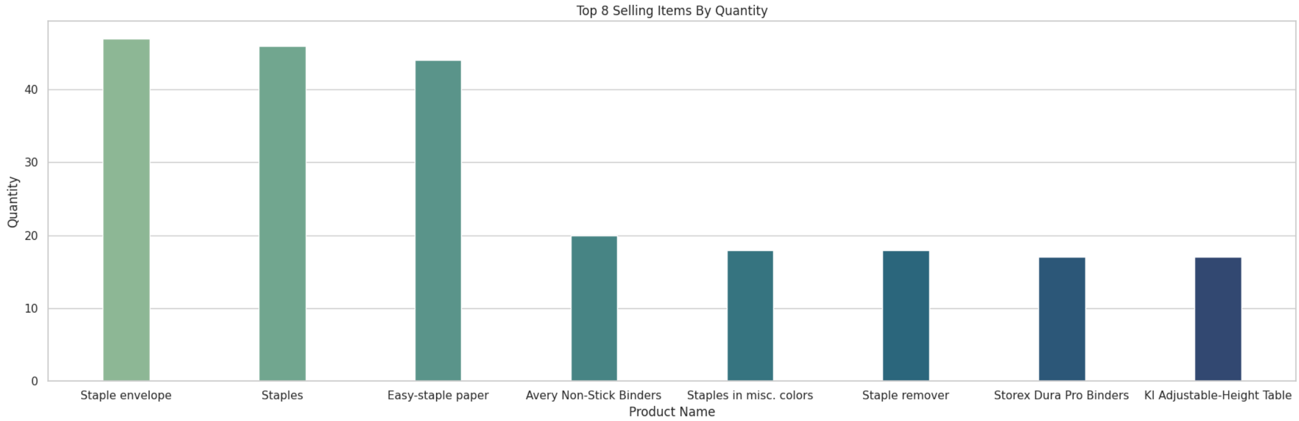
8. Sales vs. Quantity

Investigate the relationship between sales and quantity sold using scatter plots.

Here, we identify and visualize the top 8 selling items by quantity using a count plot. The figure width is increased for better presentation of the information.

```
# Identify the top 8 selling items
top_8_products = stores['product_name'].value_counts().nlargest(8).index

# Create a count plot to visualize the quantity of the top 8 selling items
plt.figure(figsize=(18, 6))  # Increase the width of the figure
sns.countplot(x='product_name', data=stores[stores['product_name'].isin(top_8_pro
plt.title('Top 8 Selling Items By Quantity')
plt.xlabel('Product Name')
plt.ylabel('Quantity')
plt.tight_layout()
plt.show()
```

```
<ipython-input-72-d221ab1bf3ac>:6: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i
```



Observation:

Envelop and staples ! I agree, to attach and hide secret documents.

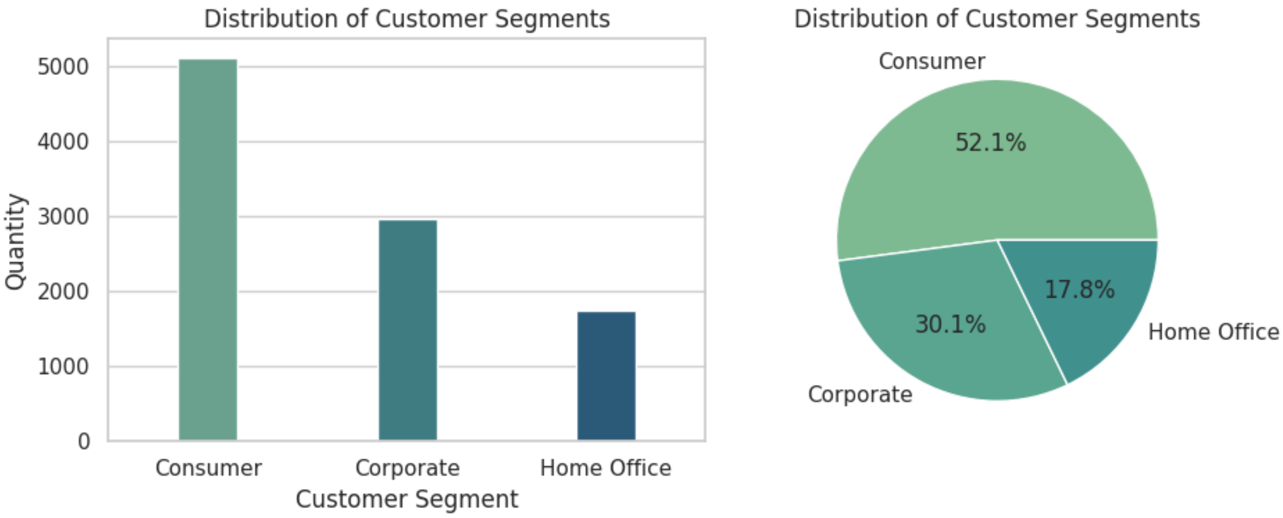
9. Customer Segment vs. Quantity

Compare the quantity sold across different customer segments using bar charts.

Here, we analyze and visualize the distribution of customer segments using a bar plot and a pie chart. The bar plot provides a detailed quantity breakdown, while the pie chart offers a proportional representation of customer segments.

```
# Segment distribution
segment_counts = stores['segment'].value_counts().reset_index()

# Create subplots with two columns
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Bar plot on the first subplot
sns.barplot(x='index', y='segment', data=segment_counts, palette='crest', ax=axes
axes[0].set_title('Distribution of Customer Segments')
axes[0].set_xlabel('Customer Segment')
axes[0].set_ylabel('Quantity')

# Pie chart on the second subplot
axes[1].pie(segment_counts['segment'], labels=segment_counts['index'], autopct='%
axes[1].set_title('Distribution of Customer Segments')

# Display the plots
plt.tight_layout()
plt.show()
```

```
<ipython-input-73-9a3ad71b9e95>:8: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i
```



Observation:

The company is mostly B2C as it is customer centric.

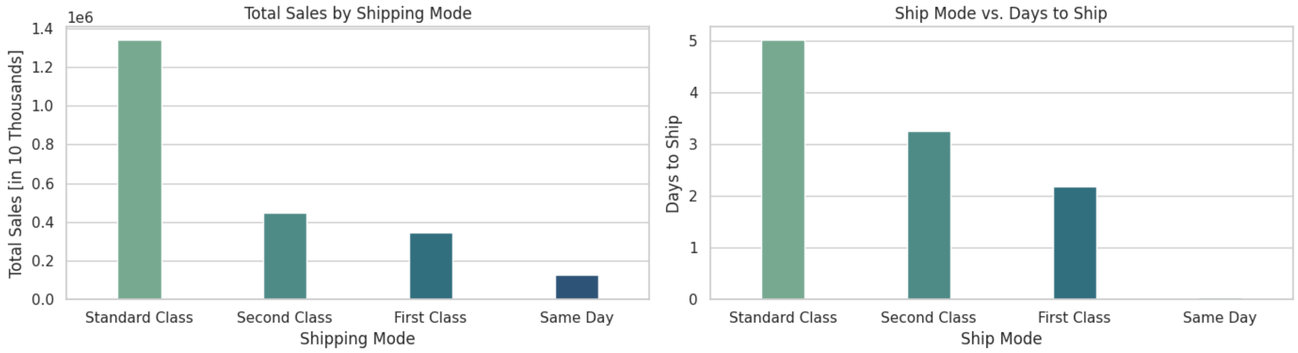
10. Shipping Mode vs. Sales

Explore the relationship between shipping modes and sales using categorical plots.

Here, we analyze and visualize total sales by shipping mode using a bar plot. Additionally, we explore the relationship between ship mode and days to ship using another bar plot on the second subplot. The second plot helps understand how different shipping modes affect the time taken to ship products.

```
# Analyze total sales by shipping mode
ship_mode_sales = stores.groupby('ship_mode')['sales'].sum().reset_index()

# Sort the data by sales in descending order
ship_mode_sales = ship_mode_sales.sort_values(by='sales', ascending=False)

# Create subplots with two columns
fig, axes = plt.subplots(1, 2, figsize=(14, 4))

# Bar plot for total sales by shipping mode on the first subplot
sns.barplot(x='ship_mode', y='sales', data=ship_mode_sales, palette='crest', ax=a
axes[0].set_title('Total Sales by Shipping Mode')
axes[0].set_xlabel('Shipping Mode')
axes[0].set_ylabel('Total Sales [in 10 Thousands]')

# Analyze the relationship between ship mode and days to ship
ship_mode_days_to_ship = stores.groupby('ship_mode')['days_to_ship'].mean().reset

# Sort the data by days to ship in descending order
ship_mode_days_to_ship = ship_mode_days_to_ship.sort_values(by='days_to_ship', as

# Bar plot for ship mode vs. days to ship on the second subplot
sns.barplot(x='ship_mode', y='days_to_ship', data=stores, ax=axes[1], palette='cr
axes[1].set_title('Ship Mode vs. Days to Ship')
axes[1].set_xlabel('Ship Mode')
axes[1].set_ylabel('Days to Ship')

# Display the plots
plt.tight_layout()
plt.show()
```

```
<ipython-input-74-3c3c9fe2a62b>:11: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i

<ipython-input-74-3c3c9fe2a62b>:23: FutureWarning:


The `ci` parameter is deprecated. Use `errorbar=('ci', False)` for the same e

<ipython-input-74-3c3c9fe2a62b>:23: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed i
```



Observation:

```
Most people prefer standard class. Who has money for same day deliver?
the comany really works hard on their delivery procedure.
```

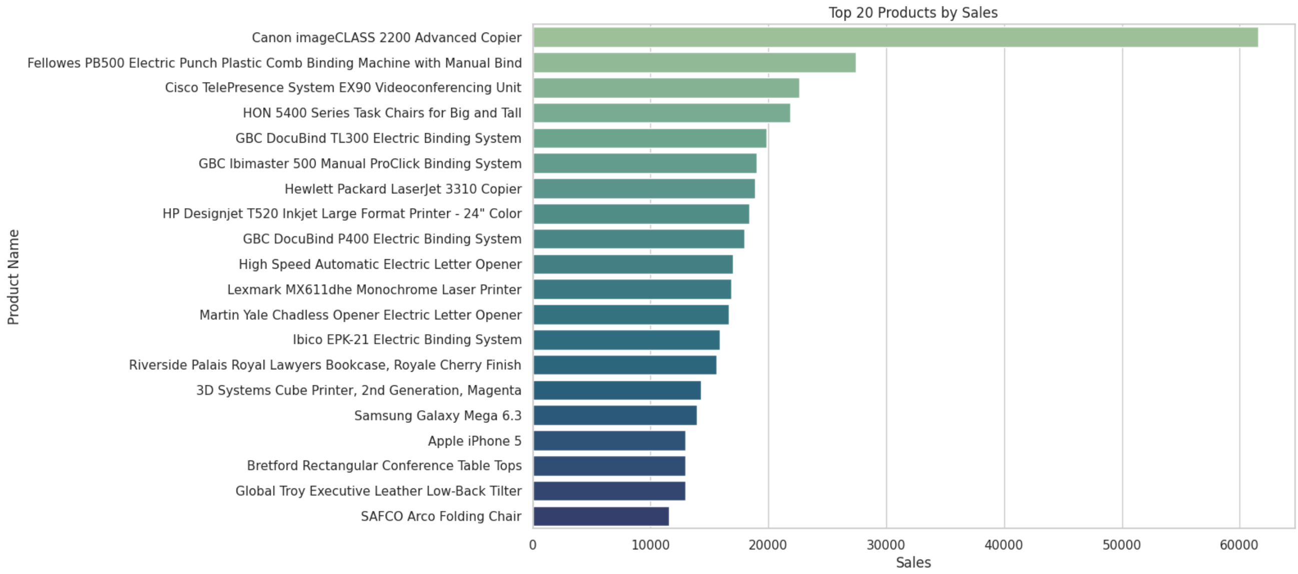
11. Product-wise Sales Analysis

Investigate how sales vary for different products or categories using bar charts or line charts.

Here, we identify and visualize the top 20 products by sales using a horizontal bar plot. The length of each bar represents the sales amount, providing a clear comparison of product performance.

```
# Identify and visualize the top 20 products by sales with a horizontal bar plot
top_products = stores.groupby('product_name')['sales'].sum().sort_values(ascendin

# Create a horizontal bar plot
plt.figure(figsize=(12, 8))
sns.barplot(y='product_name', x='sales', data=top_products, palette='crest')
plt.title('Top 20 Products by Sales')
plt.xlabel('Sales')
plt.ylabel('Product Name')
plt.show()
```

```
    <ipython-input-75-28c350cff05a>:6: FutureWarning:



    Passing `palette` without assigning `hue` is deprecated and will be removed i
```


Top 20 Products by Sales
Canon imageCLASS 2200 Advanced Copier
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind
Cisco TelePresence System EX90 Videoconferencing Unit
HON 5400 Series Task Chairs for Big and Tall
GBC DocuBind TL300 Electric Binding System
GBC Ibimaster 500 Manual ProClick Binding System
Hewlett Packard LaserJet 3310 Copier
HP Designjet T520 Inkjet Large Format Printer - 24" Color
GBC DocuBind P400 Electric Binding System
High Speed Automatic Electric Letter Opener
Lexmark MX611dhe Monochrome Laser Printer
Martin Yale Chadless Opener Electric Letter Opener
Ibico EPK-21 Electric Binding System
Riverside Palais Royal Lawyers Bookcase, Royale Cherry Finish
3D Systems Cube Printer, 2nd Generation, Magenta
Samsung Galaxy Mega 6.3
Apple iPhone 5
Bretford Rectangular Conference Table Tops
Global Troy Executive Leather Low-Back Tilter
SAFCO Arco Folding Chair
Product Name
Sales