Group 12 CIND830 Teamwork Assignement (Assignement 2):

Roles:

1. **Aida Judaki:** The Data Engineer (OOP Specialist) will be responsible for designing classes and objects to handle data cleaning and transformation.
2. **Stephanie Boissonneault:** The Algorithm Developer (Data Structures and Algorithms Specialist) will be responsible for implementing efficient algorithms to analyze the data and extract meaningful insights.
3. **Maliheh Garoosiha:** The Visualization Expert will be responsible for creating visual representations of the analyzed data using standard or third-party Python libraries.

```
#Upload the SuperStoreOrders.csv dataset to the session storage
#Run the following code to import the file
from google.colab import drive
import pandas as pd
df = pd.read_csv('SuperStoreOrders.csv')
df
```

	order_id	order_date	ship_date	ship_mode	customer_name	segment
0	AG-2011-2040	1/1/2011	6/1/2011	Standard Class	Toby Braunhardt	Consumer
1	IN-2011-47883	1/1/2011	8/1/2011	Standard Class	Joseph Holt	Consumer
2	HU-2011-1220	1/1/2011	5/1/2011	Second Class	Annie Thurman	Consumer
3	IT-2011-3647632	1/1/2011	5/1/2011	Second Class	Eugene Moren	Home Office
4	IN-2011-47883	1/1/2011	8/1/2011	Standard Class	Joseph Holt	Consumer
...	...	...	...	...	...	...
51285	CA-2014-115427	31-12-2014	4/1/2015	Standard Class	Erica Bern	Corporate

Data Processing and Cleaning (by Aida Judaki):

1. Design classes to read the dataset from a CSV file.
2. Implement methods to handle missing values and outliers, especially in the "Sales" column.
3. Transform data types, especially the "Date" column, to datetime or epoch timestamp format if necessary.

```python
# 1. Design classes to read the dataset from a CSV file.
import pandas as pd

# Read the CSV File
class DataReader:
  def read(self,file_name):
    raise NotImplementedError("Subclass must implement this method")

class CSVReader(DataReader):
  def read(self, file_name):
    return pd.read_csv(file_name)

def load_data(reader, file_name):
    return reader.read(file_name)

data_reader = CSVReader()
data = load_data (data_reader, 'SuperStoreOrders.csv')
df
```

	order_id	order_date	ship_date	ship_mode	customer_name	segment
0	AG-2011-2040	1/1/2011	6/1/2011	Standard Class	Toby Braunhardt	Consumer
1	IN-2011-47883	1/1/2011	8/1/2011	Standard Class	Joseph Holt	Consumer
2	HU-2011-1220	1/1/2011	5/1/2011	Second Class	Annie Thurman	Consumer
3	IT-2011-3647632	1/1/2011	5/1/2011	Second Class	Eugene Moren	Home Office
4	IN-2011-47883	1/1/2011	8/1/2011	Standard Class	Joseph Holt	Consumer
...	...	...	...	...	...	...
51285	CA-2014-115427	31-12-2014	4/1/2015	Standard Class	Erica Bern	Corporate

```
# 2. Implement methods to handle missing values and outliers, especially in the "Sales" column.
import pandas as pd
import numpy as np

class DataProcessor:
    def __init__(self, file_path):
         self.file_path = file_path
         self.df = pd.read_csv(file_path)

    def handle_missing_values(self):    #replace missing value in "Sales" with mean value
         self.df['Sales'].fillna(self.df['Sales'].mean(), inplace=True)

    def handle_outliers(self, threshold=3):    # Remove outliers in the "Sales" column using z-score
         z_scores = np.abs ((self.df['Sales'] - self.df['Sales'].mean()) / self.df['Sales'].std())
         self.df = self.df[(z_scores < threshold)]

    def save_processed_data(self, New_File_Name):   # Save the processed data to a new CSV file
         self.df.to_csv(New_File_Name, index=False)

file_name = "SuperStoreOrders.csv"
New_file_name = "New CSV File.csv"
processor = DataProcessor(file_name)

    # Handling missing values
processor.handle_missing_values()

    # Handling outliers
processor.handle_outliers()

    # Save the processed data
processor.save_processed_data(New_file_name)
df = pd.read_csv(New_file_name)
df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Se
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Con
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Con
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Cor
		US-			Standard		Sean	

```
# 3. Transform data types, especially the "Date" column, to datetime or epoch timestamp format if necessa
import pandas as pd

class CSVTransformer:
    def __init__(self, file_path):
        self.file_path = file_path
        self.df = pd.read_csv(file_path)

    def transform_date_column(self, Order_Date):
        try:
            # Convert the specified "Order_date" column to datetime format
            self.df[Order_Date] = pd.to_datetime(self.df[Order_Date], errors='raise')
            self.df[Order_Date] = self.df[Order_Date].astype(int) / 10**9
            self.df.to_csv(self.file_path, index=False)
            print(f"Transformation successful. Check the updated file at: {self.file_path}")

        except Exception as e:
            print(f"Error occurred during transformation: {e}")

#Create new CVS file.csv
file_path = 'SuperStoreOrders.csv'
transformer = CSVTransformer(file_path)
transformer.transform_date_column('Date')
df = pd.read_csv(New_file_name)
df
```

```
Error occurred during transformation: 'Date'
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Se
0	1	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Cons
1	2	CA-2017-152156	08/11/2017	11/11/2017	Second Class	CG-12520	Claire Gute	Cons
2	3	CA-2017-138688	12/06/2017	16/06/2017	Second Class	DV-13045	Darrin Van Huff	Corp
		US-			Standard		Sean	

Data Analysis (by Stephanie Boissonneault):

The iDataAnalysis class contains a methods for each quesitons:

1. Implement an algorithm to identify the top 10 sub-categories by sales
2. Design a data structure to store the monthly sales trend for the entire superstore.
3. Implement a search algorithm to find all days where a promo led to sales above a certain threshold.

```python
#Convert 'sales' to int type
df['sales'] = df['sales'].str.replace(',', '').astype(int)
#display(df.info())
```

```python
#Extract month from order date and add year/month column
from re import M
import datetime
df['month'] = pd.to_datetime(df.order_date, dayfirst= True)
#print(data.month)
df['month'] = df['month'].dt.month
df['month_year'] = pd.to_datetime(df.order_date).dt.to_period('M')
```

```
<ipython-input-25-644cee1abd01>:7: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) wa
  df['month_year'] = pd.to_datetime(df.order_date).dt.to_period('M')
```

```
class iDataAnalysis:
    '''Algorithms for Data Analysis on input dataframe (df)'''
    def __init__(self, dataframe):
        self._dataframe = dataframe

#1. Implement an algorithm to identify the top 10 subcategories by sales

    #Algorithm top_ten was made generalizable for the user to be able to group the top 10 of the defined
    #To answer question 1, the user could call the method using the dataset's sales column as variable x,
    def top_ten (self, x, group):
        '''Summation grouping of "x" by "group".
        Returns a dictionary of the top ten results of x for each group'''
        #Initializing list of dictionary key values pairs of key("group") : value("x")
        group = tuple(group)
        x = list(x)
        my_list = []
        for i in range(len(group)):
            my_dict = {"group": group[i], "x": x[i]}
            my_list.append(my_dict)
        #Group and sum values("x") by key("group")
        group_keys = "group"
        sum_keys = "x"
        result = {}
        for i in my_list:
            subgroup = i[group_keys]
            if subgroup not in result:
                for x in sum_keys:
                    result[subgroup] = int(0)
            for n in sum_keys:
                result[subgroup] += int(i[n])
        #Sort results in descending order and return the top 10 results
        #Python's sorted() function uses Timsort which is a hybrid of merge sort and insertion sort
        name = lambda x: (x[1], x[0])
        sorted_result = sorted(result.items(), key = name, reverse = True)
        return (sorted_result[0:10])

    #Algorithm top_sales_by_subcategory is another sample algorithm but made specific to grouping sales b
    def top_sales_by_subcategory (self):
        '''Summation grouping of sales by subcategory.
        Returns the top ten sales by group'''
        #Group by sub category an sum number of sales
        sumSales = df.groupby(['sub_category'])['sales'].sum().sort_values(ascending = False)
        return(sumSales[0:10])

#2. Design a data structure to store the monthly sales trend for the entire superstore.
    def store_monthly_sales(self):
        '''Stores the monthly sales trend (both "sales" and "profit") into a list of lists'''
        #Store month/year values into list
        month_year = df.month_year
        month_year_list = []
        for i in range(len(month_year)):
            if month_year[i] not in month_year_list:
                month_year_list.append(month_year[i])
        #Group and sum monthly sales and profit and store values into lists
        monthlySales = df.groupby(['year','month'])['sales', 'profit'].sum()
        sales_list = monthlySales['sales'].tolist()
        profit_list = monthlySales['profit'].tolist()
        #Store values in a list of lists
        col_list = []
        row_list = []
        for i in range(len(month_year_list)):
            col_list = month_year_list[i], sales_list[i], profit_list[i]
            row_list.append(col_list)
        return row_list

#3. Implement a search algorithm to find all days where a promo led to sales above a certain threshold.
    def searchDays(self, treshold):
        '''Search for days where a promotional item led for sales above selected sales "treshold".
        Returns tuple(days, item, sales, and discount) above selected sales treshold'''
        treshold = int(treshold)
        #Linear Search for position of treshold and store sales above treshold
        above_treshold_position = []
        position = 0
        while position < len(df.sales):
            if treshold < df.sales[position]:
                if df.discount[position] > 0:
                    above_treshold_position.append(position)
            position += 1
        my_lyst = []
        for i in above_treshold_position:
            my_tuple = (df.order_date[i], df.product_name[i], df.sales[i], df.discount[i])
            my_lyst.append(my_tuple)
        return my_lyst
```

The following are samples as to how to call the methods using the iDataAnalysis class

```
data_analysis = iDataAnalysis(df)
```

```
#1. Implement an algorithm to identify the top 10 subcategories by sales
#Algorithm option 1
data_analysis.top_ten(df.sales, df.sub_category)
#help(analysis.top_ten)
```

```
[('Phones', 1706874),
 ('Copiers', 1509439),
 ('Chairs', 1501682),
 ('Bookcases', 1466559),
 ('Storage', 1127124),
 ('Appliances', 1011081),
 ('Machines', 779071),
 ('Tables', 757034),
 ('Accessories', 749307),
 ('Binders', 461952)]
```

```
#Algorithm option 2
data_analysis.top_sales_by_subcategory()
```

```
sub_category
Phones         1706874
Copiers        1509439
Chairs         1501682
Bookcases      1466559
Storage        1127124
Appliances     1011081
Machines        779071
Tables          757034
Accessories     749307
Binders         461952
Name: sales, dtype: int64
```

```
#2. Design a data structure to store the monthly sales trend for the entire superstore.
a = data_analysis.store_monthly_sales()
print(a)
```

```
[(Period('2011-01', 'M'), 98902, 8321.80096), (Period('2011-02', 'M'), 91152, 12417.90698), (Period('2011-03', 'M'), 14
<ipython-input-26-7c165e19f570>:55: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys
  monthlySales = df.groupby(['year','month'])['sales', 'profit'].sum()
```

```
#3. Implement a search algorithm to find all days where a promo led to sales above a certain threshold.
#Here we calling the method using a sales treshold of 5000 to return only the days and items that had ove
data_analysis.searchDays(5000)
```

```
[('18-03-2011',
  'Cisco TelePresence System EX90 Videoconferencing Unit',
  22638,
  0.5),
 ('10/8/2011', 'Cisco Smart Phone, Cordless', 5277, 0.1),
 ('8/9/2011', 'Lexmark MX611dhe Monochrome Laser Printer', 8160, 0.4),
 ('2/8/2012', 'Hon Executive Leather Armchair, Black', 5760, 0.1),
 ('24-05-2013', 'Canon imageCLASS 2200 Advanced Copier', 8400, 0.4),
 ('4/6/2013', 'Samsung Smart Phone, VoIP', 5726, 0.1),
 ('17-10-2013', 'Nokia Smart Phone, with Caller ID', 5175, 0.1),
 ('24-06-2014', 'Sauder Classic Bookcase, Metal', 5487, 0.1),
 ('14-10-2014',
  'Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind',
  5084,
  0.2),
 ('23-10-2014', 'Canon imageCLASS 2200 Advanced Copier', 11200, 0.2),
 ('5/11/2014', 'Cubify CubeX 3D Printer Triple Head Print', 8000, 0.5),
 ('19-11-2014', 'Hon Executive Leather Armchair, Adjustable', 5729, 0.1)]
```

4. Design a scenario where a Stack or Queue data structure would be helpful in processing or analyzing the dataset. Implement this scenario and explain your choice.

The Scenario: The superstore would like to evaluate it's worker's efficiency at prioritizing orders based on order_priority. Assume the superstore only has one order fullfillment center for all orders. Create a queue for the store's the first 10 orders (order_id) by priority (order_priority).

```
from queue import PriorityQueue
import queue
```

```
#Creating a new column named priority_num that contains data.order_priority as coded numbered integers
#where the lowest number (1) is associated with the highest priority
df['priority_num'] = df.order_priority
df['priority_num'] = df['priority_num'].str.replace('Critical', '1')
df['priority_num'] = df['priority_num'].str.replace('High', '2')
df['priority_num'] = df['priority_num'].str.replace('Medium', '3')
df['priority_num'] = df['priority_num'].str.replace('Low', '4')
df['priority_num'] = df['priority_num'].astype(int)
```

```
#The PriorityQueue class from queue accepts a a tuple in the form (priority_number, data).
#Creating a function to store data in a tuple in the form of (priority_number, data) which can be be acce
def data_by_priority (priority_number, data):
    '''Returns a list of tuples in the form
    (priority_number, data)'''
    a_list = []
    for i in range(len(priority_number)):
        a_tuple = (priority_number[i], data[i])
        a_list.append(a_tuple)
    return a_list
```

```
#Store priority_num and order_id columns in a list of tuples using above function
order_list = data_by_priority(df.priority_num, df.order_id)

#Initiate priority queue and give worker 10 orders from order_list.
#Will print the order in which order_items are inserted into the queue
pq = PriorityQueue(10)
for order in range(10):
    print(order_list[order])
    pq.put(order_list[order])
```

```
(3, 'AG-2011-2040')
(3, 'IN-2011-47883')
(2, 'HU-2011-1220')
(2, 'IT-2011-3647632')
(3, 'IN-2011-47883')
(3, 'IN-2011-47883')
(3, 'CA-2011-1510')
(1, 'IN-2011-79397')
(4, 'ID-2011-80230')
(2, 'IZ-2011-4680')
```

```
#Will print the order in which order_items are extracted from the priority queue
#Note that the order is different from above since the order_items have been prioritized
for order in range(10):
    print(pq.get())
```

```
(1, 'IN-2011-79397')
(2, 'HU-2011-1220')
(2, 'IT-2011-3647632')
(2, 'IZ-2011-4680')
(3, 'AG-2011-2040')
(3, 'CA-2011-1510')
(3, 'IN-2011-47883')
(3, 'IN-2011-47883')
(3, 'IN-2011-47883')
(4, 'ID-2011-80230')
```

Using a PriorityQueue is ideal for processing information such as a bussiness order items which must be prioritized since the priority queue prioritizes items of different priority levels, while treating equal priority items as first in first out. This scenario is an example of a way in which workers collected data could be compared to an algorithm to identify whether workers are correctly prioritizing their orders.

Efficiency Considerations:

1. Differentiate between writing code that is memory efficient versus code that is time efficient. Provide examples from your implementation.

Memory efficiency is concerned with the amount of storage space that is used in the memory while time efficiency is concerned with the amount of time it takes for an algorithm to complete its set tasks during runtime. There is often a tradeoff between creating code that is memory efficient compared to code that is time efficient. For example, simular to the **top_ten** method created in the iDataAnalysis class, using dictionaries to store key value pairs is a less efficient way to store data than tuples, however, using dictionary keys can help speed up the itime it takes for retrieving the data stored in a key. Memory efficiency can be Time efficiency can be measured by comparing the size of a problem to the number of steps an algorithm needs to solve the problem during runtime. To ensure that code is time efficient, it's important to consider select a type of algorithm with a the BigO notation that is deemed appropriate for the size of the problem.

2. Optimize at least one part of your code for memory efficiency and another part for time efficiency. Document the changes made and their impact.

Optimizing for memory efficiency:

Memory efficiency of the code can be optimized by storing data into data-structures and collections that are memory efficient (such as those that are) and by reducing the size of objects in the memory. Arrays are data structures which are given a determined capacity upon declaration and are allocated a block of adjacent memory cells for that capacity. Arrays can threfore be sized and resized accordingly to avoid wasting memory by checking the load factor. An array with a low load factor for example (0.25) means that the logical size of the array is much smaller than its physical size, meaning that the array is taking up more memory space but only storing a small ammount of data. To be memory efficient, we would want to create a function to resize and decrease the physical size of array so that it doesn't take up as much memory. This is another example of a tradeoff of memory and time efficiency, because the transfer of elements when resizing the arrays takes up time.

Question 2 asked that I create a data structure for storing to the monthly sales trend for the entire superstore. The original structure was a list of lists. To store the elements in a data structure that is more memory efficient, we will store the monthly sales in an array of lists. To document the changes in memory efficiency, we will check the memory size taken up by the data structure using *getsizeof* from *sys*.

```
#2. Designing an array class to store the monthly sales trend
class iArray():
    '''A representation of an array'''
    def __init__(self, capacity, fillValue = None):
        '''Capacity is the size of the array.
        None object will be placed at each position.'''
        self._items = list()
        for count in range(capacity):
            self._items.append(fillValue)

    def __len__(self):
        #Obtaining the size of the array
        return len(self._items)

    def __str__(self):
        #Representing the array as String
        return str(self._items)

    def __setitem__(self, index, newItem):
        #Replacing an element or assigning a value to an element
        self._items[index] = newItem

    def __getitem__(self, index):
        #Obtaining an element in the array
        return self._items[index]

    def __iter__(self):
        #Traversing elements with loops
        return iter(self._items)

    def loadFactor(self):
        '''Returns the load factor of the array'''
        logicalSize = len(([x for x in self if x is not None]))
        physicalSize = len(self)
        loadFactor = logicalSize/physicalSize
        return(loadFactor)
```

```
#Changing the algorithm to store the lists in an array
def store_monthly_sales_optimized():
    '''Stores the monthly sales trend into list of lists'''
    #Store month/year values into list
    month_year = df.month_year
    month_year_list = []
    for i in range(len(month_year)):
        if month_year[i] not in month_year_list:
            month_year_list.append(month_year[i])
    #Group and sum monthly sales and profit and store values into lists
    monthlySales = df.groupby(['year','month'])['sales', 'profit'].sum()
    sales_list = monthlySales['sales'].tolist()
    profit_list = monthlySales['profit'].tolist()

    #Store values in a array of lists
    col_list = []
    my_array = iArray(len(month_year_list))
    for i in range(len(month_year_list)):
        col_list = month_year_list[i], sales_list[i], profit_list[i]
        my_array[i] = col_list
    return my_array
```

```
import sys

#Documenting size of original datastructure(list of lists)
data_analysis = iDataAnalysis(df)
monthly_sales = data_analysis.store_monthly_sales()
print(monthly_sales)
sys.getsizeof(monthly_sales)
```

```
[(Period('2011-01', 'M'), 98902, 8321.80096), (Period('2011-02', 'M'), 91152, 12417.90698), (Period('2011-03', 'M'), 14
<ipython-input-26-7c165e19f570>:55: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys
  monthlySales = df.groupby(['year','month'])['sales', 'profit'].sum()
472
```

```
#Documenting size of new datastructure (array of lists)
msales = store_monthly_sales_optimized()
print(msales)
sys.getsizeof(msales)
```

```
[(Period('2011-01', 'M'), 98902, 8321.80096), (Period('2011-02', 'M'), 91152, 12417.90698), (Period('2011-03', 'M'), 14
<ipython-input-26-a9b758c3cb0a>:11: FutureWarning: Indexing with multiple keys (implicitly converted to a tuple of keys
  monthlySales = df.groupby(['year','month'])['sales', 'profit'].sum()
48
```

Optimizing for time efficiency:

Code for the **searchDays** method of the **iDataAnalysis** class uses a linear search model of O(n). The search iterates through every single item of the list to determine which item positions of 'sales' are above the treshold. The O(n) search model is less time efficient for lists with a large *n* value since the model must iterate through *n* number of items during the search. Using a binary search algorithm on a sorted list, such as using Python's bisect library, can help us identify the position of items above the treshold with a worst case scenario logarithmic O(log2(n)) search. This logarithmic search greatly reduces the amount of steps needed to conduct the search on lists or large value n.

To help document changes in time efficiency, we will time both the older version of the algorithm and the newer version using default_timer from timeit.

```
#Optimizing question number 3 for time efficiency (using Python's bisect library binary search algorithm)
import bisect

#3. Implement a search algorithm to find all days where a promo led to sales above a certain threshold.
def searchDaysOptimized (treshold):
        '''Search for days where a promotional item led to sales above "treshold".
        Returns tuple(days, item, sales, and discount)'''

        #Sort dataframe by sales
        sorted_sales = df.sort_values(['sales'], ascending= True)

        #Remove rows withouth a discount
        sorted_sales = sorted_sales[sorted_sales['discount'] > 0]

        #Store values with discounts in list to assign new index position and iterate through list:
        sales_list = list(sorted_sales.sales)
        date_list = list(sorted_sales.order_date)
        product_list = list(sorted_sales.product_name)
        promotion_list = list(sorted_sales.discount)

        #Use python bisect for binary search for position of treshold
        treshold_position = bisect.bisect_right(sales_list, treshold)

        #Store data, sales, product, and discount into new list of lists
        my_lyst = []
        i = treshold_position
        while i < len(sales_list):
            my_tuple = (date_list[i], product_list[i], sales_list[i], promotion_list[i])
            my_lyst.append(my_tuple)
            i += 1
        return my_lyst
```

```
#Import default_timer to help document the algorithm's time efficiency
from timeit import default_timer as timer
```

```
#Documenting time efficiency of original alorithm (which uses a linear search)
data_analysis = iDataAnalysis(df)
start = timer()
data_analysis.searchDays(4000)
end = timer()
print(end - start) # Time in seconds, e.g. 5.38091952400282
```

```
1.2154368649999014
```

```
#Documenting time efficiency of optimized algorithm
start = timer()
searchDaysOptimized(4000)
end = timer()
print(end - start) # Time in seconds, e.g. 5.38091952400282
```

```
0.055800958999952854
```

Data Visualization (by Maliheh Garoosiha):

```
import matplotlib.pyplot as plt
```

1. Create a bar chart showing the top 10 stores categories by sales.

```
#top_categories = df.groupby('Sub-Category')['Sales'].sum().nlargest(10)
df['sales'] = pd.to_numeric(df['sales'], errors='coerce')

# Group by category and calculate total sales for each category
category_sales = df.groupby('sub_category')['sales'].sum()

# Select the top 10 categories by sales
top_10_categories = category_sales.nlargest(10)

```

```
plt.figure(figsize=(12, 10))
top_10_categories.plot(kind='bar', xlabel='Sub-Category', ylabel='Sales', title='Comparing sale in top ten
plt.show()
```

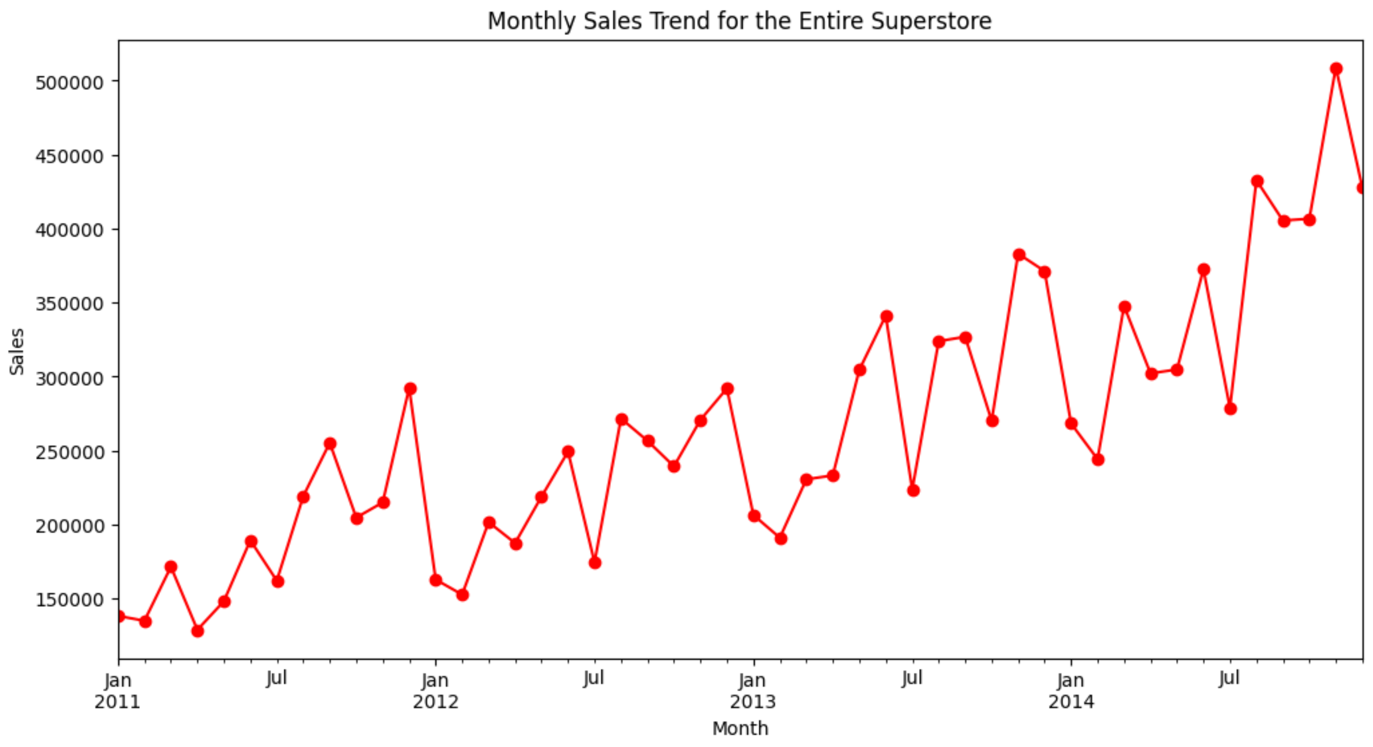
2. Plot a line chart to display the monthly sales trend for the entire superstore.

```
df['order_date'] = pd.to_datetime(df['order_date'])

# Extract month and year from 'Order Date'
df['Month'] = df['order_date'].dt.to_period('M')

# Group by month and sum the Sales for each month
monthly_sales = df.groupby('Month')['sales'].sum()

# Plotting the line chart
plt.figure(figsize=(12, 6))
monthly_sales.plot(kind='line', marker='o', color='red',xlabel='Month', ylabel='Sales', title='Monthly Sal
plt.show()
```



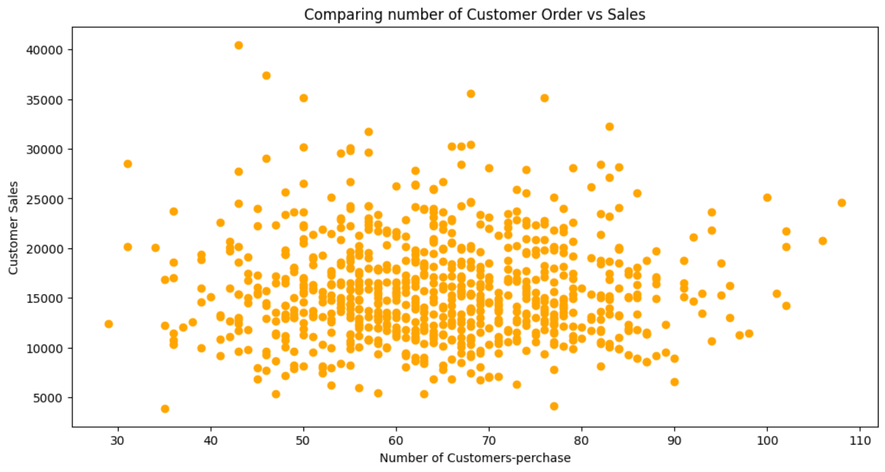
3. Design a scatter plot to visualize the relationship between the number of customers and sales for the entire dataset.

```
# Group by Customer ID and sum the Sales for each customer
customer_sales = df.groupby('customer_name')['sales'].sum()

# Group by Customer ID and count the number of orders for each customer
customer_order_count = df['customer_name'].value_counts()


# Create a scatter plot
plt.figure(figsize=(12, 6))

plt.scatter(customer_order_count, customer_sales, color='orange')
plt.xlabel('Number of Customers-perchase')  # Set the label for the x-axis
plt.ylabel('Customer Sales')  # Set the label for the y-axis
plt.title('Comparing number of Customer Order vs Sales')
plt.show()
```



```
df
```

	order_id	order_date	ship_date	ship_mode	customer_name	segment
0	AG-2011-2040	2011-01-01	6/1/2011	Standard Class	Toby Braunhardt	Consumer
1	IN-2011-47883	2011-01-01	8/1/2011	Standard Class	Joseph Holt	Consumer
2	HU-2011-1220	2011-01-01	5/1/2011	Second Class	Annie Thurman	Consumer
3	IT-2011-3647632	2011-01-01	5/1/2011	Second Class	Eugene Moren	Home Office
4	IN-2011-47883	2011-01-01	8/1/2011	Standard Class	Joseph Holt	Consumer
...	...	...	...	...	...	...
51285	CA-2014-115427	2014-12-31	4/1/2015	Standard Class	Erica Bern	Corporate
51286	MO-2014-2560	2014-12-31	5/1/2015	Standard Class	Liz Preis	Consumer